CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Highly Available Data Replication Strategies Exploiting Data Semantics, Operation Types and Finite State Space

Dissertation zur Erlangung des Grades und Titels eines
Doktors der Naturwissenschaften

vorlegt von

M.Sc. Awais Usman

Gutachter:
Prof. Dr.-Ing. Oliver Theel
Prof. Dr. Oliver Kramer

Tag der Disputation:
19.09.2023

It is because of the love and prayers of my beloved mother and father, that I am able to write this thesis. There were difficult times during the PhD with a lot of pressure and uncertainty, but my loving wife – Tayyaba – and children always stood beside me and encouraged me to keep going till the end. It was not possible without the support of my family to complete the PhD. I am really grateful to them.

I would like to sincerely acknowledge the guidance of Prof. Dr.-Ing. Oliver Theel throughout the PhD. He gave me the confidence and freedom to research and experiment on complex ideas. His critical analytical skills persuaded me to bring in novelty and quality. I am also very thankful to Prof. Dr. Ernst-Rüdiger Olderog to provide me the opportunity to pursue my PhD in his SCARE research group. SCARE turned out to be an excellent platform for me, where I can collaborate with my fellow researchers. It was a wonderful time at Carl von Ossietzky University of Oldenburg and it left permanent pleasant marks in my memories.

– Awais

Abstract

In this highly dynamic era of the technology, most of the data intensive applications are designed to target an optimal combination of high operation availabilities, scalability, data consistency, fault tolerance, and operation costs. To accomplish this the research community introduced many valuable techniques based on data replication or data distribution. This work is among the continuations of the research focused on data replication. Data replication is an important research area, as reliable access to data makes up the base of most of the information technology services. In data replication multiple and identical copies of the data item are managed. These multiple copies are known as data replicas. High operation availabilities, low operation costs, and data consistency are the major conflicting targets in almost every research on data replication strategies. Operation availability means the availability of operation in the presence of failure. The number of replicas needed to execute the operation determine the operation cost. The Data consistency is the property of a data replication strategy to keep the data replicas consistent, i.e., the impact of operation executions on a set of replicas is the same as the operations are executed on a single data item. This is a particular notion of data consistency and is known as one-copy serializability which is generally ensured with the help of two phase locking protocol. If replicated data is consistent, then data consistency is satisfied and vice versa.

The classic design considerations for data replication strategies are: i) those which focus on high operation availabilities and strong data consistency result in higher cost, ii) those which focus on high operation availabilities and low operation costs cannot guarantee strong data consistency, and iii) those which focus towards low operation costs and strong data consistency compromise on high operation availabilities. In this work, we researched on the development of data replication strategies which provide high operation availabilities and try to achieve strong data consistency with low operation costs. In the scope of this work, three data replication strategies are proposed and discussed: i) Semantic data replication (SDR), ii) the Component-based highly available replication strategy (CbHaRS), and iii) the Data replication strategy for replicated service registry ($\mathcal{DRSR}$). Next, these three strategies are briefly introduced.

Semantic data replication Most of the state-of-the-art data replication strategies exploit read and write operations. The data replicas are manipulated with the help of read quorums and write quorums. A quorum is a set of replicas. The read operation and write operation are executed by read quorum and write quorum, respectively. A quorum can be made of a subset of the replicas or it can contain all the replicas. A quorums intersection property is used to ensure data consistency. In semantic data replication we adopted a different technique for data consistency . The SDR's mechanism is to exploit data semantics and encoding techniques to achieve high operation availabilities. It is done by figuring out the replicas to write based on the input – for a write operation, input means the data to be written, e.g., write

value of a variable $x = 5$, and for a read operation, input means to data to be read, e.g., read value of variable x. In SDR, the data to write is encoded into codes. These codes are then redundantly distributed among the replicas. The encoding technique makes it possible to deterministically identify the replicas to be written based on the codes of data. It is done by exploiting the structure of the codes. This helps SDR to provide high operation availabilities with low operation costs.

Component-based highly available replication strategy In the past, the research community had introduced many valuable techniques based on data replication and data distribution. However, in the ongoing research there is a paradigm shift: the research community is targeting to minimize the inter-application coordination – the coordination required to ensure data consistency – to achieve high operation availabilities. Two-phase locking and distributed locking are examples of coordination protocols required to ensure data consistency. In this work, we present, a *component-based highly available replication strategy* which exploits operation types and a hybrid communication method – synchronous and asynchronous communication methods – to achieve high operation availabilities. Distributed locking protocol is enforced for scenarios identified by the application. CbHaRS is highly scalable as it utilizes *data components* as the building blocks for the replication strategy. Causal data consistency in CbHaRS is ensured by a so-called *component administrator*. The application knowledge is used to overcomes the limitations of causal data consistency. The communication type between the data component and the component administrator depends upon the state of the data component. Additionally, the state of the data component depends upon the operation type. We further extend the concept of client specific on-demand replication to general component-based replication. CbHaRS is a project in progress. To prove the effectiveness of CbHaRS, we have implemented the CbHaRS prototype and discussed the achieved results.

Data replication strategy for the replicated service registry The Internet market is highly competitive and is influenced by high expectation levels of internet users, continuous advancement in the information technology, and high processing and storage capabilities of the hardware. In this work, we focus on the design and development of a replicated and highly available service registry for the microservice architecture. The service registry key-value store comprises of six nodes and supports a total of $\mathbf{2^{16}}$ microservices. Existing replicated service registries, like ZooKeeper and ETCD are based on majority consensus strategies. Moreover, if these strategies fail to achieve majority consensus, then they are bound to provide limited functionality. As part of this research, we propose a highly available data replication strategy for the replicated service registry. In order to overcome the limitations faced by the existing strategies, the a data replication strategy for a replicated service registry exploits i) a simple encoding scheme; and ii) a mapping method for efficient distribution of the encoded values to the service registry nodes.

The objective – provide high operation availabilities with low operation costs – for each of the replication strategies is same. However, they distinguish from each other in terms of the methodology that they use to achieve their objective and their application. Each of the proposed data replication strategies focuses on a particular set of problems. SDR is suitable for write intensive applications e.g. recording event trail or logs. CbHaRS can be applied to those applications where data consistency is guaranteed by enforcement of application specific rules. $\mathcal{DRSR}$ focuses on the applications having a maximum of $\mathbf{2^{16}}$ different values e.g a service registry.

Zusammenfassung

In diesem hoch dynamischen Feld der Technologie sind die meisten Applikationen mit hohem Datendurchsatz darauf ausgelegt, eine optimale Kombination einer hohen Operationsverfügbarkeit, Skalierbarkeit, Datenkonsistenz, Fehlertoleranz und Operationskosten anzustreben. Um dies zu erreichen, hat die Gemeinschaft der Forschenden viele nützliche Techniken basierend auf Datenreplikation oder Datenverteilung vorgestellt. Diese Arbeit setzt auf Datenreplikation fokussierte Forschung fort. Datenreplikation ist ein wichtiges Forschungsfeld, weil ein zuverlässiger Zugriff auf Daten die Basis der meisten informationstechnischen Dienste ist. Durch die Datenreplikation werden viele und identische Kopien einer Dateneinheit verwaltet. Diese Kopien werden Replikate genannt. Hohe Operationsverfügbarkeit, niedrige Operationskosten und Datenkonsistenz sind die größten konfliktären Ziele in nahezu jeder Forschung bezüglich Datenreplikationsstrategien. Operationsverfügbarkeit bedeutet die Verfügbarkeit der Operation in Anwesenheit eines Ausfalls. Die Anzahl an Replikaten, welche mindestens benötigt wird, um die Operation auszuführen, bestimmt die Operationskosten. Die Datenkonsistenz ist die Eigenschaft einer Datenreplikationsstrategie, welche die Daten konsistent hält. Der Effekt der Ausführung von Operationen auf eine Sammlung von Replikaten ist der selbe, wenn die Operationen auf eine einzelne Dateneinheit ausgeführt werden. Das ist eine bestimmte Idee von Datenkonsistenz und ist als Ein-Kopien-Serialisierbarkeit bekannt, welche im Allgemeinen mithilfe des Zwei-Phasen-Sperrprotokolls gewährleistet wird. Wenn replizierte Daten konsistent sind, dann ist die Datenkonsistenz gewährleistet und vice versa.

Die klassischen Designüberlegungen für Datenreplikationsstrategien sind: i) hohe Operationsverfügbarkeit und starke Datenkonsistenz resultieren in hohen Kosten, ii) hohe Operationsverfügbarkeit und niedrige Operationskosten können eine starke Konsistenz nicht gewährleisten und iii) niedrige Operationskosten und starke Datenkonsistenz verhindern eine hohe Operationsverfügbarkeit. In dieser Arbeit untersuchen wir die Entwicklung von Datenreplikationsstrategien, welche eine hohe Operationsverfügbarkeit bieten und versuchen dabei, eine hohe Datenkonsistenz bei niedrigen Operationskosten zu erreichen. Drei Datenreplikationsstrategien werden vorgeschlagen und diskutiert: i) Semantische Datenreplikation (SDR), ii) komponentenbasierte hoch verfügbare Replikationsstrategie (CbHaRS) und iii) Datenreplikationstrategie für replizierte Serviceverzeichnisse (DRSR). Nachfolgend werden diese Strategien kurz vorgestellt.

Semantische Datenreplikation Die meisten der etablierten Datenreplikationsstrategien nutzen Lese- und Schreiboperationen. Diese Datenreplikate werden mithilfe von Lese- und Schreibquoren modifiziert. Ein Quorum ist eine Menge an Replikaten. Die Leseoperation wird mit einem Lesequorum ausgeführt, eine Schreiboperation wird mit einem Schreibquorum ausgeführt. Ein Quorum kann aus einer Untermenge aller Replikate oder allen Replikaten insgesamt bestehen. Die Schnittmenge aus Quoren wird verwendet, um Datenkonsistenz zu gewährleisten. Bei der Semantischen Datenreplikationsstrategie haben wir eine andere Technik zur Wahrung der

Datenkonsistenz eingeführt. Der SDR-Mechanismus nutzt die Semantik von Daten und Kodierungstechniken, um eine hohe Operationsverfügbarkeit zu erreichen. Dafür werden die Replikate ermittelt, welche bei einer Eingabe beschrieben werden müssen. Dann bedeutet eine Eingabe, dass Daten geschrieben werden müssen (z.B. das Schreiben eines Wertes in eine Variable ($x = 5$)). Für eine Leseoperationen bedeutet eine Eingabe, dass Daten gelesen werden müssen (z.B. lese den Wert der Variablen x). Bei SDR werden die zu schreibenden Daten in Codes kodiert. Diese Codes werden dann redundant über die Replikate verteilt. Die Kodierungstechnik ermöglicht die zu beschreibenden Replikate deterministisch zu ermitteln, basierend auf der Kodierung der Daten. Dafür wird die Struktur der Codes genutzt. Das hilft SDR hohe Operationsverfügbarkeiten bei geringen Operationskosten zu erreichen.

Komponentenbasierte hoch verfügbare Replikationsstrategie In der Vergangenheit hat die Gemeinschaft der Forschenden viele wertvolle Techniken basierend auf Datenreplikation und Datenverteilung vorgestellt. Aktuelle Forschung durchläuft allerdings einen Paradigmenwechsel: Die Forschungsgemeinschaft versucht den Koordinationsaufwand zwischen Applikationen zu minimieren, um hohe Operationsverfügbarkeiten zu erreichen. Koordination ist allerdings erforderlich, um Datenkonsistenz zu gewährleisten. Zwei-Phasen-Sperrung und verteiltes Sperren sind Beispiele von Koordinationsprotokollen. In dieser Arbeit präsentieren wir eine komponentenbasierte hoch verfügbare Replikationsstrategie, welche die Operationstypen und eine hybride Kommunikationsmethode (synchrone und asynchrone Kommunikationsmethoden) nutzt, um hohe Operationsverfügbarkeiten zu erreichen. Das Verteilte Sperrprotokoll ist beschränkt auf Szenarien, welche durch die Applikationen vorgegeben werden. CbHaRS ist hoch skalierbar, weil es die Datenkomponenten als Grundlage für die Replikationsstrategie nutzt. Kausale Datenkonsistenz ist in CbHaRS durch einen sogenannten Komponentenadministrator sichergestellt: Das Wissen über die Anwendung wird verwendet, um die Limitationen kausaler Datenkonsistenz zu überwinden. Die Art der Kommunikation zwischen den Datenkomponenten und den Komponentenadministrator hängt vom Status der Datenkomponente ab. Zusätzlich hängt der Status der Datenkomponente vom Operationstyp ab. Wir erweitern das Konzept der klientenspezifischen Replikation auf Nachfrage zu einer generellen, komponentenbasierten Replikation. CbHaRS ist ein laufendes Projekt. Um die Wirksamkeit von CbHaRS zu beweisen, haben wir einen CbHaRS-Prototypen implementiert und diskutieren die Ergebnisse.

Datenreplikationstrategie für replizierte Serviceverzeichnisse Der Markt des Internets ist hoch kompetitiv und wird von hohen Erwartungen der Nutzer, kontinuierlicher Weiterentwicklung der Informationstechnologie und hohen Berechnungs- und Speicherkapazitäten der Hardware beeinflusst. In dieser Arbeit fokussieren wir uns auf das Entwerfen und Entwickeln einer Replikation für hoch verfügbare Serviceverzeichnisse für die Mikroservice-Architektur. Der Schlüssel-Wert-Speicher eines Serviceverzeichnisses umfasst sechs Knoten und unterstützt eine Anzahl von $\mathbf{2^{16}}$ Mi-

kroservices. Existierende, replizierte Serviceverzeichnisse wie ZooKeeper und ETCD basieren auf der Majority-Consensus-Strategie. Wenn diese Strategien in der Menge der Replikate keine Mehrheit finden, dann sind sie nur eingeschränkt funktionsfähig. Als Teil dieser Arbeit stellen wir eine hoch verfügbare Datenreplikationsstrategie für Serviceverzeichnisse vor. Um die Grenzen existierender Strategien zu umgehen, nutzt die Datenreplikationsstrategie für Serviceverzeichnisse i) ein simples Kodierungsschema und ii) eine Zuordnungsmethode für effiziente Verteilung kodierter Werte zu den Knoten des Serviceverzeichnisses.

Das Ziel – hohe Operationsverfügbarkeiten bei niedrigen Operationskosten – ist für jede der Replikationsstrategien dasselbe. Sie unterscheiden sich untereinander in der Methodik, welche sie zum Erreichen dieses Ziels verwenden und in dem Anwendungsfall. Jede der vorgestellten Datenreplikationsstrategien fokussiert eine bestimmte Menge an Problemen. SDR ist für schreibintensive Anwendungen, wie Logging oder das Aufzeichnen von Änderungen, geeignet. CbHaRS kann angewendet werden, wenn Datenkonsistenz durch anwendungsspezifische Regeln gewährleistet werden kann. $\mathcal{DRSR}$ fokussiert sich auf Anwendungen, welche maximal $\mathbf{2^{16}}$ unterschiedliche Werte benötigen, wie ein Serviceverzeichnis.

Contents

List of Figures xiv

List of Tables xvi

List of Algorithms xvii

Acronyms xxi

1 Introduction 1
- 1.1 Data Partitioning . . . 2
- 1.2 Data Replication . . . 2
- 1.3 Quorum Systems . . . 4
  - 1.3.1 Quorum Intersection Property . . . 4
- 1.4 Trade-off Parameters for Data Replication . . . 6
- 1.5 Classification of Data Replication Strategies . . . 9
- 1.6 Thesis Contributions . . . 10
  - 1.6.1 Semantic data replication . . . 10
  - 1.6.2 Component-based highly available replication strategy . . . 11
  - 1.6.3 Data replication strategy for the replicated service registry . . . 12

2 A Novel Data Replication Strategy exploiting Data Semantics and a Coding Technique 15
- 2.1 Introduction . . . 15
- 2.2 State of the Art . . . 21
  - 2.2.1 Syntactic data replication . . . 21
  - 2.2.2 Semantic data replication . . . 21
  - 2.2.3 Erasure Codes . . . 22
- 2.3 Problem Statement . . . 23
- 2.4 Replication Strategy . . . 23
  - 2.4.1 Functional Model . . . 23
  - 2.4.2 SDR Replication Strategy . . . 28
  - 2.4.3 Implementation . . . 30
  - 2.4.4 Availability Analysis . . . 34
  - 2.4.5 Results . . . 38

2.5 Conclusion & Future work 40

3 Component-Based Data Replication Strategy 41
3.1 Literature Review 41
3.2 Introduction 45
3.3 Problem Statement 47
3.4 Replication Strategy 47
3.4.1 RI-related operation execution in a synchronized state 49
3.4.2 RI-related operation execution in an unsynchronized state . . 49
3.4.3 GI-related operation execution by CA 52
3.5 Conclusion & Future Work 54

4 A Data Replication Strategy for A Replicated Services Registry 57
4.1 Introduction 57
4.2 Related Work 61
4.3 Microservices Architecture 62
4.3.1 μSA without a $S\Re$ 63
4.3.2 μSA with a $S\Re$ 63
4.4 A Data replication strategy for a replicated service registry 64
4.4.1 The Functional Model 64
4.4.2 Replication Strategy 67
4.4.3 Analysis 69
4.4.4 Analytical Results 72
4.5 Conclusion & Future Work 75

5 Outlook 77

References 82

Publications 93

List of Figures

1.1 Pros and cons of a non-replicated system 2
1.2 Pros and cons of a replicated system 3
1.3 Example of write quorum and read quorum 5
1.4 Quorums intersection example 6
1.5 Data replication strategy trade-off parameters 7

2.1 Classification of data replication strategies 16
2.2 Quorum intersection graph for *queue* data type[Her1984] 17
2.3 ENQ and DEQ operation example 18
2.4 SDR complete state space 20
2.5 Quorum intersection graph for *PagedFile* data type[Her1984] 22
2.6 SDR encoding example 24
2.7 Finite state space for D 25
2.8 Timestamp generation for r("1A\$") having $r\_id = 250$ 29
2.9 SDR write example 30
2.10 State space for even and odd numbers 36
2.11 State space for prime and not prime numbers 37
2.12 SDR read operation availability 38
2.13 SDR write operation availability 39

3.1 CAP Theorem 43
3.2 CbHaRS components overview 46
3.3 Replica invariant-related operation exeuciton 48
3.4 Global invariant-related operation execution 53
3.5 Comparison of replica invariant-related and global invariant-related operation executions 55

4.1 Internet connectivity bandwidth growth 58
4.2 Supercomputer computing abilities trend 58
4.3 Hard drive storage-cost trend 59
4.4 μSA containing replicated μS instances 62
4.5 A simple reservation business process example 63
4.6 μSA containing multiple μSs and a $S\Re$ 64

4.7 $\mathcal{DRSR}$ node model 66
4.8 Read operation availabilities 73
4.9 Write-operation availabilities 74
4.10 $\mathcal{DRSR}$ read operation operation cost comparision 74
4.11 $\mathcal{DRSR}$ write operation operation cost comparision 75

List of Tables

2.1 SDR codes and semantic classes extraction 28
2.2 SDR example for write and read operations for $r(A)$. 30

List of Algorithms

1 SDR Write Operation 31
2 SDR Read Operation 33

3 Replica invariant related operation execution 50
4 Global invariant related operation execution 51

5 $\mathcal{DRSR}$ $I_{\mathrm{S}\Re}$ operation 68
6 $\mathcal{DRSR}$ $U_{\mathrm{S}\Re}$ operation 70

List of Acronyms

1SR	one-copy serializability
2PL	two phase locking protocol
A	architecture. description of architecture
ADC	ASCII decimal codes
B	billion
C16	Code16
CA	component administrator
CBHARS	component-based highly available replication strategy
CMRDTS	commutative replicated data types
CRDTS	conflict-free replicated data types
CVRDTS	convergent replicated data types
DBS	database system
DC	data consistency
DCM	data component
DDBS	distributed database system
DEQ	Dequeue operation
DI	data item
DR	data replication
DRSR	a data replication strategy for a replicated service registry
DRST	data replication strategy
DS	distributed system
DSEM	data semantics
E_n	enough nodes
ENP	EvenNotPrime
ENQ	Enqueue operation

EP	EvenPrime
EXU	execute-update operation
f	NIndices mapping to nodes
f^{-1}	nodes mapping to NIndices
FQ	Final write quorum
FSS	finite state space
FT	fault-tolerant
FTC	fault tolerance
$\mathbb{G}$	complete mapping
GI	global invariant
GP	Grid protocol
GTQP	Generalized tree quorum protocol
HA	highly available
HGP	Hierarchical grid protocol
HOA	high operation availability
HQC	Hierarchical quorum consensus
$I_{\mathrm{S}\Re}$	inquire service
IC	inconsistent
INV.-LOC.	invocation-location
IQ	Initial read quorum
KVS	key-value store
LNT	logical network topology
LOC	low operation cost
M	million
μ	micro
MCS	majority consensus strategy
MQB	multimedia quorum based protocol
MS	microservice
MSA	microservice architecture
NHA	not highly available
NI	NIndices
OA	operation availability
OB-CRDTS	operation-based conflict-free replicated data types
OC	operation cost
ONP	OddNotPrime

OP	OddPrime
PIT	partition-intolerant
PITC	partition intolerance
PRU	prepare-update operation
PT	partition-tolerant
PTC	partition tolerance
$\Re$	registry
RI	replica invariant
RO	read operation
ROWA	Read-One Write-All
ROWAA	Read-one write-all available
RST	replication strategy
RWO	read and write operation
S	service
SB-CRDTS	state-based conflict-free replicated data types
SD	service discovery
SDR	semantic data replication
SI	microservice's key
SNST	synchronized state
SR	service registry
SRK	service registry key
SS	state space
SSS	sub-state space
STDR	syntactic data replication
SYNTH	synchronization threshold
TLP	triangular lattice protocol
TQP	Tree quorum protocol
TSR	timestamped reference
$U_{S\Re}$	update service
UDP	update package
UNSNST	unsynchronized state
WO	write operation
WS	web service
WVS	weighted voting strategy
YCSB	Yahoo Cloud Serving Benchmark

1 Introduction

There is an old Greek saying, interpreted as, "change is the only constant in life," and it is definitely true from the perspective of technology. From the last couple of decades the technological advancements are immense and there is a lot that has been changed. However, the changes are outcome of the continuous improvements. Presently, the online application giants like Facebook, Google, Amazon etc. support millions of a continuously growing number of users. These applications rely on strong distributed backend infrastructures [DHJ+2007, BAC+2013, CDE+2013]. One of the primary reasons of having such a strong backend infrastructures is to provide enhanced user experience. The system should provide high operation availabilities to accomplish this. The higher the operation availabilities are the richer is the user experience [BDF+2013, BFF+2014, UST2017].

In this work, we focus on *data replication*, one of the concepts that contributes towards achieving high operation availabilities. In data replication, high operation availability is the high probability that the operation can successfully be performed at an arbitrary point in time. E-commerce, online booking, search engine, and social networking are some classes of highly available systems. These systems manage multiple copies of their data items called *data replicas*. These systems continue to perform in the presence of k[1] data replica failures. Several hundreds or thousands of users simultaneously access these systems. For example, in November 2017, amazon.com had around 2.9 billion (B) visits, facebook.com had approximately 30.66B visits, youtube.com had nearly 24.47B visits, expedia.com had around 51.45 million (M) visits, booking.com had approximately 369.77M visits, and google.com had nearly 43.43B visits [Sim2017]. The internet market is highly competitive and is influenced by high expectation levels of the internet users, continuous advancement in the information technology, and high processing and storage capabilities

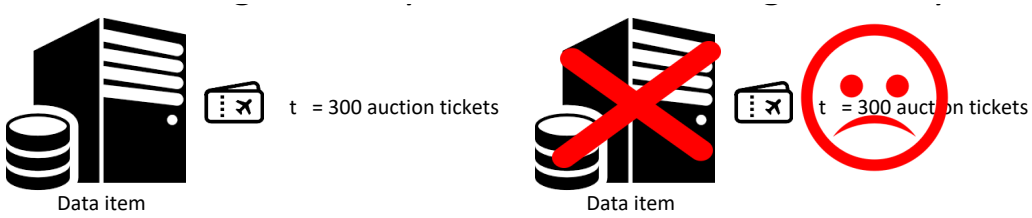
[1] $k < n$ and n is total number of replicas. k varies depending upon the system requirements

of the hardware [McA2010]. However, to survive in such a highly competitive environment, these systems are primarily focusing on high operation availabilities [DHJ+2007, CDE+2013, BAC+2013].

1.1 Data Partitioning

Data replication is the primary focus of this work however, data partitioning is also briefly explained in this section. Data partitioning is a concept where a data item is split into parts.

Example Let us consider a record of a person as a data item. This record contains fields like *ID*, *name*, *street address*, *city*, and *bank balance*. There are two write operations namely: *updateAddress(ID)* and *updateBankBalance(ID)*. *updateAddress(ID)* writes new value of *street address* and *city* against the search value *ID*. *updateBankBalance(ID)* writes new value of *bankbalance* against the search value *ID*. Both of the write operations write the new value of person data item and are mutually exclusive to each other. However, this mutual exclusion can be avoided if the data item is split into parts known as data partitions. For example, one data partition – *partA* – consists of fields *ID*, *name*, *street address*, and *city*. The other partition – *partB* – consists of fields ID and bank balance. Now *updateAddress(ID)* writes on *partA* and *updateBankBalance(ID)* writes on *partB* simultaneously. This approach is known as horizontal partitioning. The split can be done by a horizontal method, vertical method, or a hybrid method [Run2008, Kha2010]. Along with the infrastructure a variable percentage of the data is redundantly distributed to different sites.



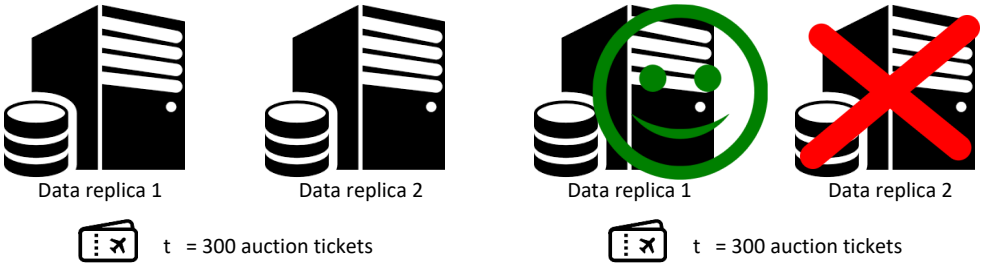
(a) A simple non-replicated system with one running instance of a data item. It is easy to manage, and it is cost-effective as read-cost and write-cost will always be 1.

(b) A simple non-replicated system with failed instance of a data item. In case of failure, it does not provide a successful operation execution at that time.

Figure 1.1: Pros and cons of a non-replicated system

1.2 Data Replication

Most of the time-critical business, e-Commerce, and social networking applications depend heavily upon a strong backend infrastructure. From the design and architecture point of view, usually, the backend infrastructure is logically or physically



(a) A replicated system with two replicas of the data item. The replicated system increases operation costs as compared to non-replicated system, but it may provides higher operation availabilities depending on data replication strategy and fault model adopted.

(b) A replicated system can bear replica failure. If a replica fails, then the other replica is available for operation executions – depending upon the data replication strategy. Therefore, the addition of a replica increases the operation availabilities of the system.

Figure 1.2: Pros and cons of a replicated system

distributed across multiple sites [CDK2005]. A logically distributed backend infrastructure is mostly supported by virtual machines or a physically distributed backend infrastructure. The virtual machines are hosted by powerful servers [TVS2007]. The physically distributed backend infrastructure is either supported by the heterogeneous collection of computing resources in the form of a grid [JBFT2005] or the homogeneous collection of computing resources in the form of a cluster [Pra2008]. In contrast to data partitioning, multiple copies of data item are created. These copies are known as data replicas. Data replication focuses on high operation availabilities and data distribution increases performance of system by providing the user localized access to the system with respect to its geographical distribution space.

Example In Figure 1.1 we explain a centralized non-replicated online booking system. In this case, the system has one data item. This kind of systems is easy to manage and operate. There is only one data item, and for this reason it is cost-effective as the operational costs for one data item are low. However, it suffers from low operation availabilities because in case of data item failure, there are no available operations.

In Figure 1.2, the data item is replicated into two data replicas. This kind of systems requires a strategy to manage the data consistency among the replicas. The operation costs on two replicas also increases as compared to the operation cost of single data item. However, it will increase the operation availability of the system, and the system will be able to execute operations in presence of a single failure.

With having the data replicated among multiple sites arose the requirement to keep the replicated data consistent. To deal with this requirement, the replicas of data items are created and the strategies which are used to manipulate the data replicas are provided by so-called data replication strategies [Iak2012]. A very important aspect of data replication is the enhancement of the system capabilities to be fault-tolerant. Data replication also enables the system to execute concurrent opera-

tions resulting in high performance and/or throughput and improved responsiveness. Concurrent operation execution depends upon operation type and data replication strategy. Classical data replication strategies use two types of operations: i) read operation and write operation. These operations are executed via the use of a concept called quorums. In the next section we briefly elaborate on quorum system.

1.3 Quorum Systems

A Quorum is a set of replicas that work together to achieve a common goal. For example, in case of a write operation on a quorum to be successful, each replica of the respective quorum must commit to perform the write operation otherwise the write operation will not be performed. This means that, to perform a certain operation, a group of replicas come into an agreement to perform that operation. In [Gif1979], voting method is exploited to obtain the agreement. Each replica is given a vote. These votes are used when forming a read quorum V_r or write quorum V_w to perform a read operation or a write operation respectively. The following two rules are enforced to ensure one-copy serializability.

- $V_r + V_w > V$ where $V \in \mathbb{N}$ is the number of votes of data item
- $V_w \geq V/2 + 1$

In distributed control [Tho1979], majority consensus protocol, and voting method a site initiates the voting. It then collects the responses and if the majority of replicas votes in the favor of it, then it submits the transaction.

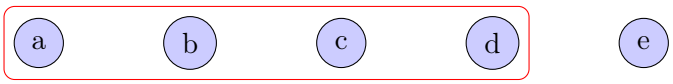
The data replication strategy defines a quorum system. The size of the quorum – the number of replicas in a quorum – depends upon the data replication strategy. Most of the data replication strategies define two types of quorums: i) a write quorum and ii) a read quorum. A write quorum is used to execute the write operation, i.e., write quorum updates value of the data item. Similarly, a read quorum is used to execute the read operation, i.e., read quorum reads value of the data item. To ensure one-copy serializability, a read quorum must intersect with a write quorum. And there must exist an intersection between the write quorums. This intersection is guaranteed with the help of quorum intersection property, that is discussed in the proceeding section.

1.3.1 Quorum Intersection Property

Along with providing high operation availabilities, it is also important to keep the data consistent across the replicas. In this section we focus on one-copy serializability, which is a notion of data consistency. It is ensured when a read operation returns the last written value of the data item. This is achieved with the help of a so called quorum intersection property. Let us suppose a replicated system made up of five replicas as shown in Figure 1.3. A write quorum Q_w containing the replicas $\{a, b, c, d\}$

- Quorum: A a set of replicas that work together to achieve a common goal

- A write quorum Q_w:



- A read quorum Q_r:

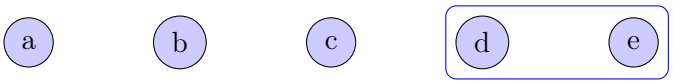


Figure 1.3: Example of a write quorum $Q_w = \{a, b, c, d\}$ and a read quorum $Q_r = \{d, e\}$.

is formed. A read quorum Q_r is formed having the replicas $\{d, e\}$. According to the quorum intersection property:

$$Q_w \cap Q_r \neq \{\}$$

These read and write quorums fulfill the intersection property of the quorum systems

$$\{a, b, c, d\} \cap \{d, e\} = \{d\}$$

because replica d ensures that the read quorum will always return the last written value by the write quorum.

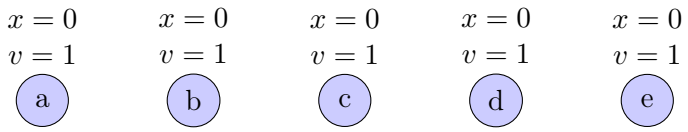
$$\begin{aligned} &Q_{w_i}, Q_{w_j} \in \{Q_{w_1}, Q_{w_2}, Q_{w_3}, \ldots, Q_{w_n}\} \text{ where } n \in \mathbb{N} \text{ number of write quorums} \\ &Q_{r_k} \in \{Q_{r_1}, Q_{r_2}, Q_{r_3}, \ldots, Q_{r_m}\} \text{ where } m \in \mathbb{N} \text{ number of read quorums} \\ &Q_{w_i} \cap Q_{w_j} \neq \emptyset \text{ where } i \neq j, i \in \mathbb{N}, j \in \mathbb{N} \\ &Q_{w_i} \cap Q_{r_k} \neq \emptyset \text{ where } i \in \mathbb{N}, k \in \mathbb{N} \end{aligned} \tag{1.1}$$

Equation 1.1 explains the quorum intersection property. Most of the replication strategies ensure the following rules:

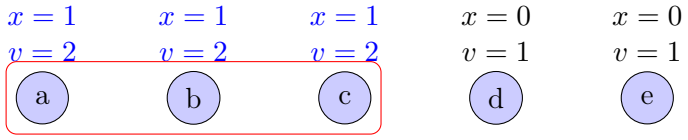
- Read quorums must intersect with write quorums
- Write quorums must intersect with write quorums

In Figure 1.4, we explain the read and write operation executions with the help of an example. The initial version (v) of the replicas $a, b, c, d,$ and e is $v = 1$ and the value of data item (x) is $x = 0$. First, a write operation $write(x, 1)$ selects the write quorum $Q_w = \{a, b, c\}$ and updates the value of x to 1. Also, the highest version among the replicas is picked, incremented by 1, and then set as $v = 2$ for all the replicas in the write quorum. The next write operation $write(x, 0)$ selects

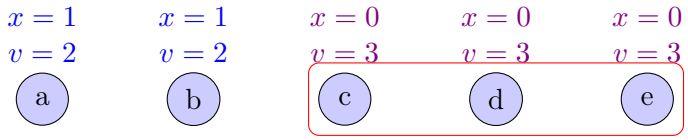
- Check for quorum availability



- $write(x, 1)$ by acquiring a write quorum $Q_{w1} = \{a, b, c\}$



- $write(x, 0)$ by acquiring a write quorum $Q_{w2} = \{c, d, e\}$



- $read(x)$ by acquiring a read quorum $Q_{r1} = \{b, c, d\}$

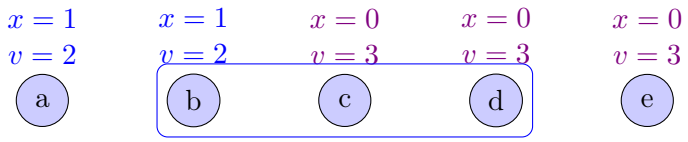


Figure 1.4: Read and write quorum intersection example. Write quorums Q_{w1} and Q_{w2} intersect with each other. Read quorum Q_{r1} also intersects with write quorums Q_{w1} and Q_{w2}. This quorum intersection property ensures data consistency.

the write quorum $Q_w = \{c, d, e\}$ and updates the value of x to 0. Replica c has the highest version $v = 2$ among the quorum. It is incremented by 1 and then set as $v = 3$ for $c, d,$ and e. The next operation $read(x)$ is a read operation to get the *last written* value of x. The operation is carried out via read quorum $Q_r = \{b, c, d\}$. The replica with the highest version in the quorum is searched and value of the x is read from that replica. In this example, last written value of x can be read from replicas c and d. Both of the replicas have $v = 3$ and $x = 0$.

1.4 Trade-off Parameters for Data Replication

To have gains in high operation availabilities, researchers have provided a number of data replication strategies focusing on this matter. Every data replication strategy

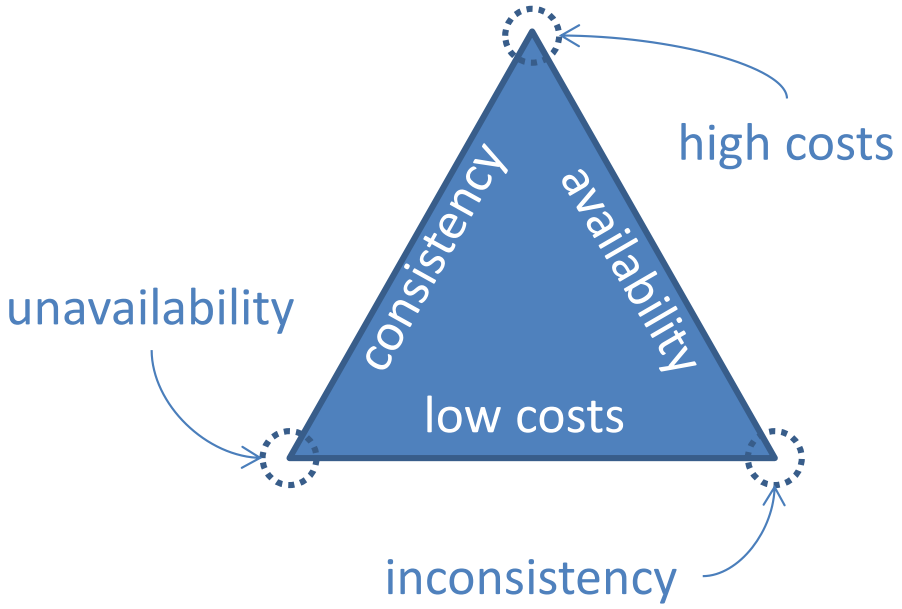


Figure 1.5: Data replication strategy trade-off parameters. High operation availability and low operation cost leads towards data inconsistency. High operation availability and data consistency results in higher costs. Data consistency with low operation cost causes operation unavailability.

revolves around the trade-off factors like high operation availabilities, data consistency, fault tolerance, and costs [His1989, GL2002, Aba2012, UST2017]. There exist diverse data consistency models in literature for data replication strategies ranging from stricter data consistency notions and then gradually moving towards weaker notions of data consistency like one-copy serializability, sequential consistency, causal consistency, and eventual consistency etc. [BHG1987, Lam1979]. The selection of an appropriate data consistency notion for a data replication strategy is made in accordance to the trade-off factors [His1989]. Brewer's CAP theorem demonstrates that the high operation availabilities are impacted by strong data consistency and partitioning intolerance [GL2002]. To achieve high operation availabilities, strong data consistency or partitioning intolerance need to be relaxed [DHJ+2007].

In Figure 1.5, we explain the trade-off between data consistency, operation availabilities, and operation costs. Data replication strategies focusing towards high operation availabilities and strong data consistency results in *high operation costs*. Because, this case enforces quorum intersection between read quorums and write quorums, and between write quorums and write quorums. The size of the quorum depends upon the data replication strategy, for example, in case of Read-One Write-All the read quorum size is one, on the other hand the write quorum size is n where

n is total number of replicas. The operation cost is directly proportional to the size of quorum. The data replication strategies trying to achieve high operation availabilities with low operation costs may suffer from *data inconsistency*. In this case small read and write quorums are used to reduce operation cost. However, there is a high probability that small quorums do not fulfill the intersection property which leads to data inconsistency. Lastly, the data replication strategies which lean towards strong data consistencies and low operation costs are impacted by *unavailability*. This is the case of not-fully distributed quorum system where the failure of certain replica(s) impact the availability of the quorum(s).

We also experience the same in case of transaction execution. To ensure ACID properties [HR1983] two transactions cannot execute simultaneously, if both try to update a common data item. In this case, one of the transactions stalls for the availability of the resources acquired by the already running transaction. And, in case of network partitioning the stalling transaction can stall for much longer or until restart, because the required resource is no longer available due to network partitioning [BDF+2013]. In [BFF+2014, BDF+2015] the authors have exploited application-specific rules for high operation availabilities, and the authors have introduced the concept of minimizing coordination – coordination required between replicas to ensure consistency – among the data replicas. The coordination minimization majorly depends upon the application invariants – correctness predicates for the replicas – and transactions' isolation levels [BDF+2013, BFF+2014, BDF+2015]. The application operations are categorized into two broad categories:

- *conflicting operations*: operations of conflicting categories cannot execute simultaneously, because, their simultaneous execution may invalidate a global application invariant – these invarinats are a set of data consistency rules that are enforced on all the replicas.

- *non-conflicting operations*: operations of non-conflicting categories can execute simultaneously. These operations depend primarily on replica-level application invariants – these invarinats are a set of data consistency rules that are enforced on a single replica independent of other replicas. These operations can be concurrent read-read, concurrent read-write and concurrent write-write.

The main idea behind the operation categorization is to ensure the validity of global application invariants during simultaneous execution of replica-level application invariants. We have exploited this approach for one of our data replication strategies presented in chapter 3. So far we had discussed about the need for data replication, and elaborated on the benefits of quorum system. We also explain briefly the three factors impacting design and development of the data replication strategy. In the next section we focus on the classification of them.

1.5 Classification of Data Replication Strategies

The data replication strategies, based on their type, can be classified into two main groups: i) Syntactic data replication (StDR) and ii) Semantic data replication (SDR) [Iak2012] as shown in Figure 1.6. Data replication strategies like Read-One Write-All (ROWA) [BG1984], the Tree quorum protocol (TQP) [AA1990], the majority consensus strategy (MCS) [Tho1979], and the weighted voting strategy (WVS) [Gif1979] belong to the class of syntactic data replication [HHB1996]. These strategies consider the execution mechanism required for read and write operations. The main focus is to develop strategies which address "how read and write operations will be carried out on the replicas". These strategies depend upon syntactic information about the transaction and its operations. For example, the signature – operation execution call – of the read and write operation describes the number and type of parameters for the operation [Iak2012]. The input[2] to the operations is evaluated against some syntax rules or some grammatical structure defined as the input validation expressions, and the actual data semantics is not considered. In majority of the cases, syntactic data replication makes use of quorums to carry out read and write operations and they guarantee one-copy serializability (1SR) by exploiting quorum intersections between read and write quorums and between write and write quorums [Iak2012].

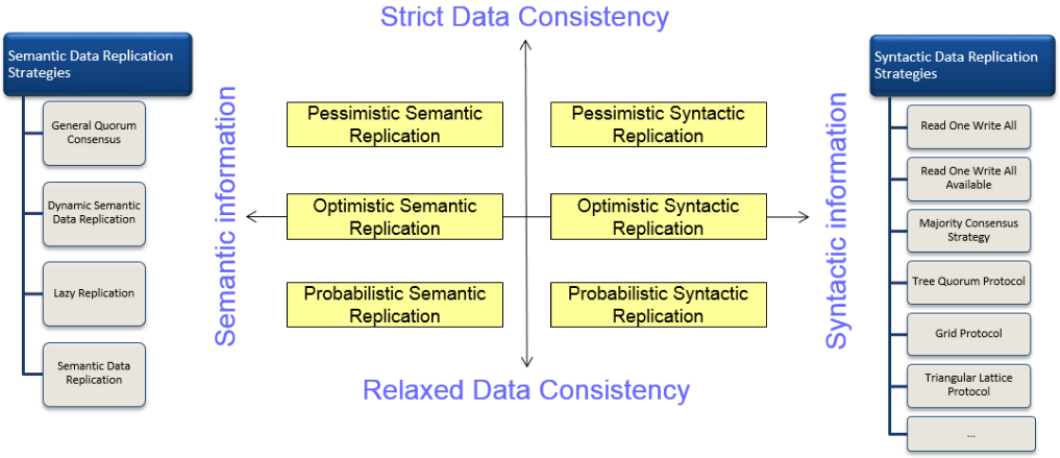


Figure 1.6: Classification of data replication strategies

The other category of data replication strategies, namely semantic data replication strategies, takes benefit by exploiting semantics of the application domain, application data, and the data type used to store and manipulate application data [Iak2012]. In [Her1985] and [Her1986], the author focuses on object's data type to improve the operation availabilities, and introduces strategies having robust configurations and high operation availabilities. The author uses timestamps and logs instead of ver-

[2]For a write operation, input means the data to be written, e.g., write value of a variable $x = 5$, and for a read operation, input means to data to be read, e.g., read value of variable x

sion numbers. The benefit of using timestamps and logs is that the write quorum intersection is no loner required to find highest version number among the replicas. Instead, in a syntactic data replication strategy, write quorum intersection is always required to find the value written by the last write operation to guarantee data consistency. However, in semantic data replication strategy, a write operation appends write quorum logs with the newest timestamp and updates the data item value. Refer to Figure 2.3 for an example. Data consistency is guaranteed by constructing a view of the replicated object from these replicated logs [Her1986]. Considering the effectiveness of this strategy, we also utilize a timestamp-based approach for SDR (Chapter 2).

1.6 Thesis Contributions

This thesis focuses on an important research area which is to provide high operation availabilities, achieve strong data consistency, and reduce operation costs. Our effort results in three data replication strategies namely i) Semantic data replication (SDR), ii) Component-based highly available replication strategy (CbHaRS), and iii) Data replication strategy for a replicated service registry ($\mathcal{DRSR}$). A brief overview about these data replication strategies is given in Sections 1.6.1, 1.6.2, and 1.6.3 respectively. For detailed discussion on SDR please refer to Chapter 2. CbHaRS is elaborated in Chapter 3. In Chapter 4, we explain $\mathcal{DRSR}$. And finally, in Chapter 5, we summarize our research and share future directions.

1.6.1 Semantic data replication

Semantic data replication [UST2017] is an enhancement of the research done by Gifford [Gif1979] and Herlihy [Her1985, Her1986]. However, the distinctive features of our work in data semantics is the utilization of a coding scheme in a replicated environment to achieve high operation availabilities. Semantic data replication has six replicas. The replicas are divided into two groups. One group contains semantic information of the data and the other group contains the codes of the data as shown in Figure 1.7. For complete details of the replication strategy please refer to Section 2.4. The feature set of SDR is made up of data semantics, finite state space, and coding scheme distinguishes our work from existing data replication strategies. Now, we briefly explain the underlying logic of our feature set.

1. *Finite state space (Θ):* SDR defines a Θ around the semantics of the data[3]. For the scope of our work, we utilize ASCII codes [ANS1986] to define the Θ. The Θ is represented in the form of a connected directed graph. Its vertices represent four nodes of the distributed systems and each edge represents a digit of the ASCII code (we explain this in detail in Section 2.4.1). This Θ is then used in read and write operations to identify the subset of replicas to be read and written. The Θ is the key feature of SDR and it distinguishes SDR from existing replication

[3]Data is the input processed by the system

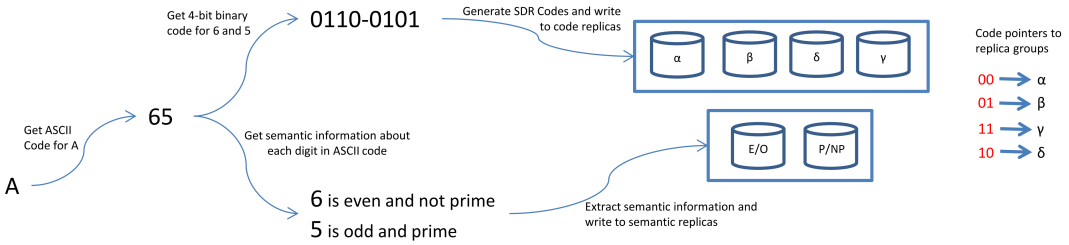


Figure 1.7: A visual representation of different steps of SDR. This example shows how the input value is processed to obtain its semantic information, and how SDR codes are generated from it.

strategies [HHB1996] as the Θ guides the write operation to select the replicas based on the input. For read operation, it determines the next replica to read based on the value of the current replica. In Section 2.4.3, we give an example of read and write operation executions. In case of strategies relying on quorum consensus and voting, the operation execution depends upon a search-and-select mechanism – for predefined quorum or participation from a certain percentage of replicas to form a dynamic quorum – to make a decision [Tho1979, Gif1979]. However incase of SDR, replica selection is made with the help of Θ.

2. *Coding scheme:* We introduce a light-weight coding scheme in SDR. Our coding scheme encodes the input value according to its semantic information. The write operation utilizes the encoded value of the data to identify the replicas which will take part in operation execution. In case of the read operation, the semantic information and Θ guide the read operation to a reduced set of replicas to get the latest value based on the timestamp. The encoded values are then decoded to regenerate the actual value. Section 2.4.1.3 elaborates the coding scheme in detail.

3. *Replica roles:* We bind the replicas into groups. Each group is tagged with the role to store unique code value[4]. One of the groups is dedicated to store the semantic information about the data, and the other group is used to store the encoded values of the data. In [KBMP1996] replica roles can be switched as a result of a reconfiguration. However in our case, replica roles are not switched because of their distinct role. SDR is designed to be fault-tolerant without reconfiguration. It takes advantage of the Θ knowledge and the redundantly distributed codes. The redundancy is incorporated in the codes by the encoding scheme. We gave an example of SDR replication in Figure 1.7. In the upcoming section we discuss our component-based data replication strategy.

1.6.2 Component-based highly available replication strategy

In [UZT2017a], we present component-based highly available replication strategy. The focus of this research work is to develop application-specific highly available data replication strategy. In case of application-specific data replication strategies,

[4]The code value is part of the codes which are generated via SDR coding scheme

the data consistency rules are defined by the application semantics. Component-based highly available replication strategy (CbHaRS) focuses towards achieving high operation availabilities by utilizing the states of the data components (DCMs) – a replica and the data item hosted by that replica collectively are represented as data component (DCM)[5] – and by exploiting the semantics of the data component's operations. CbHaRS uses DCMs as the building blocks for the data replication strategy. The DCMs exist at different abstraction levels. From the application's perspective, there exists a DCM for each client. And, from the client's perspective there exist multiple copies of its DCM [BCD+2000]. For the scope of our work, we elaborate on the client's perspective which is the basic unit for our data replication strategy. CbHaRS is scalable and at the same time it is fault-tolerant. Plugging in a data component contributes towards scalability because then there will be one more component for concurrent operation execution, for more information see Section 3.4.

Among the data components, one of the data component plays the role of component administrator (CA). Application invariants – data consistency rules defined by the application semantics – are categorized into replica invariants (RIs) and global invariants (GIs). Refer to Figure 1.8 for an overview of the CbHaRS. In [BDF+2015] the concept of global and local application-level constraints is defined with the help of a middleware running on top of a geo-replicated date store. In CbHaRS, the RIs are defined on DCMs other than the CA and GIs are defined only on the CA. The RIs allow simultaneous local-operation executions on multiple DCMs targeting the same data item. In case of GIs, the operation executions are managed by the CA. To ensure data consistency of the global state of the system, the operation executions that involve GIs may result in a change to the RIs. These operations are executed in synchronous communication mode. On the other hand, the DCMs communicate with each other via the CA asynchronously. The following are the main contributions of our work:

- exploit operation types and a hybrid communication model to achieve high operation availabilities
- utilization of data components for scalability and high operation availabilities
- non-blocking concurrent operation executions while ensuring strong data consistency

In the proceeding section we provide a brief overview of our final data replication strategy and afterwards close this chapter.

1.6.3 Data replication strategy for the replicated service registry

In [UZT2018], we focus on high operation availabilities from the perspective of a microservice architecture [DGL+2017]. We designed a highly available and fault-tolerant service registry for a fixed number i of microservices, $0 \leq i \leq 2^{16}$. We achieve

[5] Here, the assumption is that, each replica hosts only one data item.

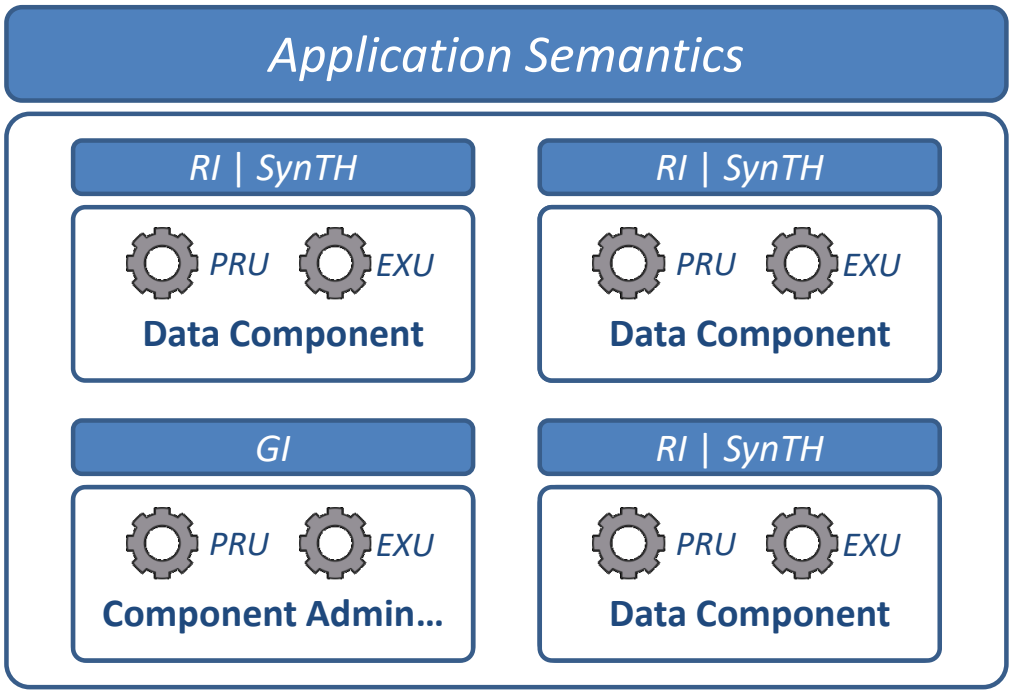


Figure 1.8: The application semantics layer defines the rules to implement the required data consistency. The replica invariant enforces the data consistency at replica level. That means the replica must hold the invariant defined for it. Prepare-update operation (PRU) and execute-update operation (EXU) are used to manipulate the data item. Global invariant is enforced by the component administrator.

these properties by replicating the service registry on six independent nodes. In a replicated environment, the independent nodes work together to achieve the common goal [The1993]. A replicated service registry is more fault-tolerant as compared to a non-replicated centralized service registry. Because, there exist n replicas of $S\Re$ and it can bear m faults, where $m < n$ and $m, n \in \mathbb{N}$ and value of m depends upon the data replication strategy. To support operations execution on the replicated service registry, we design a data replication strategy which we call *a data replication strategy for a replicated service registry* ($\mathcal{D}\Re\mathcal{S}\Re$). High operation availability is among the primary objectives of a replicated environment [BDF+2013, BFF+2014]. The key contributions of our work are:

- the development of a replicated and highly available service registry for a microservice architecture
- development of a simple and efficient encoding scheme for a finite number $i, 0 \leq i \leq 2^{16}$ of microservices

- and, a mapping method for efficient distribution of encoded values to service registry nodes

In this section, we briefly explain the three data replication strategies and their key characteristics. The way we exploited simple coding schemes and finite state space is the differentiating factor between our work and the existing data replication strategies.

2

A Novel Data Replication Strategy exploiting Data Semantics and a Coding Technique

Data replication is an important research area, as reliable access to data makes up the base of most of IT services. High operation availabilities, low operation costs, and data consistency are major target conflicts in almost every data replication research. In this chapter, we discuss a data semantics and data encoding-based data replication strategy called the *Semantic data replication (SDR)*. SDR exploits data semantics, coding schemes, and efficient search algorithms to come up with a strategy whose goal is to provide high operation availabilities with low operation costs. SDR primarily takes advantage of the finite state space introduced by the coding scheme for the replicated objects, and it uses the *a-priori knowledge* of the finite state space to know right away which replicas to contact for an operation. This feature distinguishes SDR from existing infinite state space replication strategies. Exploitation of the data coding schemes is a significant contribution in SDR, as the coding scheme helps in fast execution of read and write operations. It provides consistent access to the state of the replicated object for the read operations.

2.1 Introduction

Most of the time-critical business, e-Commerce and social networking applications depend heavily upon a strong backend infrastructure. From the design and architecture point of view, usually the backend infrastructure is logically or physically distributed across multiple sites [CDK2005]. A logically distributed backend infrastructure is mostly supported by virtual machines, which are hosted by powerful

servers [TVS2007], and a physically distributed backend infrastructure is either supported by a heterogeneous collection of computing resources in the form of a grid [JBFT2005] or a homogeneous collection of computing resources in the form of a cluster [Pra2008]. Along with the infrastructure, a variable percentage of the data, the data files, and the metadata is also distributed and replicated to different sites. One of the reasons behind this distribution is to increase the performance of the system by providing localized access to the system with respect to its geographical distribution space. With having the data distributed among multiple sites arose the requirement to keep the distributed or replicated data consistent and up to date. To deal with the above mentioned requirement, replicas of the data items are created. The strategies which are used to manipulate the data replicas are provided by the so-called *data replication strategies* [Iak2012].

A very important aspect of data replication is the enhancement of the system capabilities to be fault-tolerant. Data replication also enables the system to process concurrent requests resulting in high performance and/or throughput and improved responsiveness. Currently, there are numerous data replication strategies available, and the choice among them is made by considering the trade-off between the factors like operation availability, data consistency, and read-write operation cost [His1989]. In literature, there exist diverse data consistency classes for data replication strategies ranging from stricter data consistency notions and then gradually moving towards weaker notions of data consistency like one-copy serializability (1SR), sequential consistency, causal consistency, and eventual consistency etc [BHG1987, Lam1979]. The selection of an appropriate data consistency notion for a data replication strategy is made in accordance to trade-off factors [His1989]. We describe the data consistency notion for our data replication strategy in Section 2.4.3. The data replication

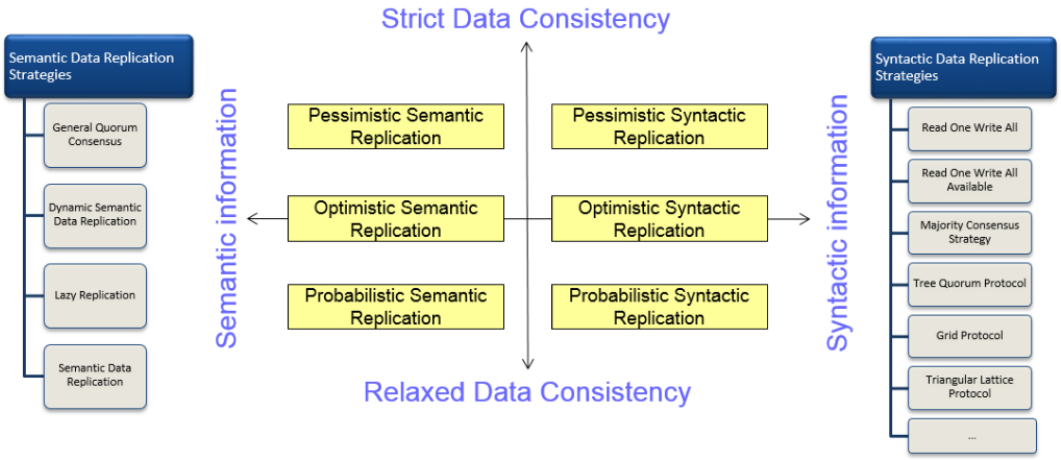


Figure 2.1: Classification of data replication strategies

strategies can be categorized into two broad categories as shown in Figure 2.1. One of the categories belongs to syntactic data replication strategies [Iak2012] like the Read-One Write-All (ROWA) strategy [BG1984], the Tree quorum protocol (TQP)

[AA1990], the majority consensus strategy (MCS) [Tho1979], and the weighted voting strategy (WVS) [Gif1979] etc. [HHB1996]. The strategies that fall under the umbrella of syntactic data replication only consider the execution mechanism required for read and write operations. The main focus is on developing strategies which address "how read and write operations will be carried out on the replicas?" These strategies depend upon syntactic information about the transaction and its operations. For example, the signature – operation execution call – of the read and write operation describes the number and type of parameters for the operation. The input to the operations is evaluated against some syntax rules or some grammatical structure defined by the input validation expressions, and the actual data semantics is not considered. In majority of the cases, syntactic data replication makes use of quorums[1] to carry out read and write operations and they guarantee 1SR by exploiting quorum intersections between read and write quorums, and between write and write quorums.

The other category of data replication strategies, namely semantic data replication strategies, takes benefit by exploiting the semantics of the application domain, application data, and the data type used to store and manipulate application data [Iak2012]. In [Her1984, Her1985, Her1986], the author focuses on the object's data type to improve the operation availabilities, and introduces strategies having robust configurations and high operation availabilities. He defined different operation types for each data type. For example, an enqueue operation (ENQ) and a dequeue operation (DEQ) are defined for a Queue data type. Each operation is supported by an initial read quorum (IQ) and final write quorum (FQ). Figure 2.2 shows a

Operation	(IQ, FQ)		
ENQ	(0,1)	(0,2)	(0,3)
DEQ	(5,1)	(4,2)	(3,3)

Figure 2.2: Quorum intersection graph for five replicas of *queue* data type[Her1984].

possible quorum intersection graph for the Queue data type. The arrow represents the intersection-relationship between the IQ and FQ. The intersection-relationship is necessary to ensure data consistency. The arrow from ENQ to DEQ elaborates that FQ of ENQ must intersect with IQ of DEQ. The arrow from DEQ to DEQ

[1] A quorum is the minimum number of participants that should agree to perform an operation.

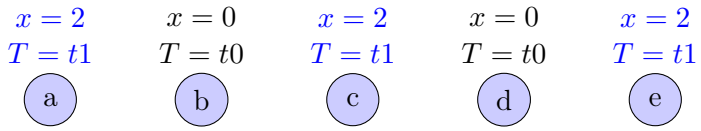
elaborates that FQ of DEQ must intersect with IQ of DEQ.

Enqueue operation and Dequeue operation operation example: From Figure 2.2 let us take the following configuration:

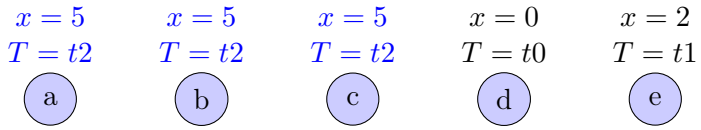
- ENQ$(0, 3)$ means that the size of IQ is 0 and of FQ is 3.
- DEQ$(3, 3)$ means that the size of IQ is 3 and FQ is 3.

Now we explain how ENQ$(x = 2)$, ENQ$(x = 5)$, and DEQ(x) are performed. The

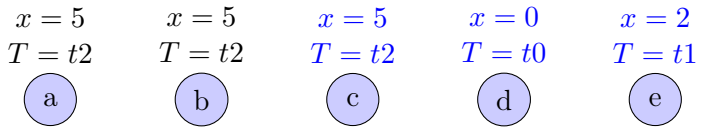
- ENQ$(x = 2)$: Timestamp $T = t1$ via FQ=$\{a, c, e\}$



- ENQ$(x = 5)$: Timestamp $T = t2$ via FQ=$\{a, b, c\}$.



- DEQ(x): IQ=$\{c, d, e\}$ found $t2$ as the highest Timestamp.



- DEQ(x): FQ=$\{a, c, e\}$ set $x = 5$ with Timestamp $T = t3$.

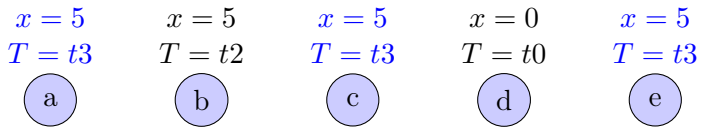


Figure 2.3: ENQ and DEQ operation example

author uses timestamps and logs instead of version numbers and versions, respectively. The benefit of using timestamps and logs is that the write quorum intersection is no loner required to find the highest version number among the replicas. We explained this with help of an example in Figure 2.3. In syntactic data replication strategies, write quorum intersection is always required to find the value written by the last write operation to guarantee data consistency. However, in a semantic data replication strategy, a write operation appends to the write quorum logs with the

newest timestamp and updates the data item value, and data consistency is guaranteed by constructing a view of the replicated object – with the help of IQ and FQ – from these replicated logs [Her1986]. This ensure that for 5 replicas the FQ of ENQ operation will always intersect with IQ of DEQ operation. Also, the IQ of its DEQ operations will always intersect with FQ of its DEQ operation. Considering the effectiveness of this strategy, we also utilize a timestamp-based approach for semantic data replication.

Up to this point, we briefly discussed the two categories of data replication strategies. Now, in a nutshell, we illustrate our motivation to come up with a data replication strategy belonging to the category of semantic data replication. Researcher are paying a lot of attention towards data semantics in the context of data analysis and knowledge discovery, by actually understanding the *meaning* of data. And here, in our work, we insist on the utility of the data semantics in the context of distributed systems – online transaction processing systems (OLTP) – which can provide high operation availabilities at low costs. The syntactic data replication strategies provide high operation availabilities by exploiting the logical network topology (LNT) of the distributed system, whereas, semantic data replication strategies not only utilize the underlying LNT, but also exploit the semantics of the data to provide better operation availabilities, as compared to syntactic data replication strategies. Our work is a continuation of the research done by Gifford [Gif1979] and Herlihy [Her1985, Her1986]. However, the distinctive features of our work is the exploitation of data semantics, and utilization of a coding scheme in a replicated environment to have high operation availabilities. This feature set, made up of data semantics, finite state space, and coding scheme, distinguishes our work from existing data replication strategies. Now, we briefly explain the idea behind using a 1) finite state space, 2) coding scheme, and 3) replica group roles:

1. *Finite state space (Θ):* SDR defines a Θ around the semantics of the data.[2] For the scope of our work, we utilize ASCII codes [ANS1986] to define the Θ. The Θ is represented in the form of a connected directed graph as shown in Figure 2.4. Its vertices represent four nodes of the distributed systems and each edge represents a decimal of the ASCII code (we explain this in detail in Section 2.4.1). This Θ is then used in read and write operations to identify the subset of replicas to be read and written. The Θ is the key feature of SDR and it distinguishes SDR from existing replication strategies [HHB1996] as the Θ guides the write operation to select the replicas based on the input. For a read operation, it predicts the next replica to read based on the value of the current replica. In Section 2.4.3 we give an example of read and write operation execution in SDR. In SDR, replica selection is made on the basis of Θ. In case of strategies relying on quorum consensus and voting, the operation execution depends upon the search-and-select mechanism – for a predefined quorum or participation from a certain percentage of replicas to form a dynamic quorum – to make a decision [Tho1979][Gif1979].

2. *Coding scheme:* We introduce a simple coding scheme in SDR. Our coding

[2]Data is the input processed by the system

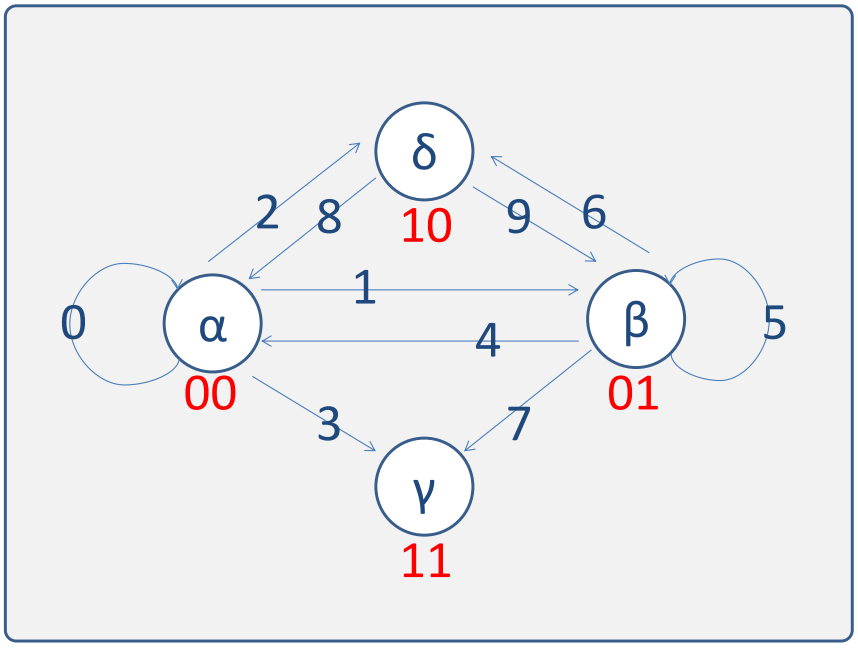


Figure 2.4: SDR complete state space

scheme encodes the data according to its semantic information. The write operation utilizes the encodes value to identify the replica(s), which will take part in the operation execution. In case of a read operation, the semantic information and Θ guide the read operation to a reduced set of replicas to get the latest value based on the timestamp. The encoded values are then decoded to regenerate the actual value (Section 2.4.1.3 elaborates the coding scheme in detail).

3. *Replica roles:* We grouped the replicas into groups and each group is tagged with the role to store unique code value.[3] One of the groups is dedicated to store the semantic information about the data, and the other group is used to store the encoded values of the data. In [KBMP1996], replica roles can be switched as a result of a reconfiguration, but in our case, replica roles are not switched because of their distinct role. SDR is designed to be fault-tolerant without reconfiguration by taking advantage of the Θ knowledge and the redundant information. The redundancy is incorporated in the codes by the encoding scheme.

The rest of the chapter is organized as follows. Section 2.2 sheds light on the existing literature regarding data replication strategies. In Section 2.4, we explain our approach SDR in detail, and we elaborate SDR's functional model followed by its implementation details. In Section 2.4.5, we present the results obtained so far.

[3]The code value is part of the codes which are generated via SDR coding scheme

At the end in Section 2.5, we conclude our work by sharing the achievements we have made and the future directions of our work.

2.2 State of the Art

Data Replication is among the research fields addressing the domain of distributed database system (DDBS). These systems manage n copies of a data item x hosted at m different sites. Each site k_m is an independent entity. All sites work together to achieve the common global goal. We refer to a site with a data item as the *replica*. In this section, we shed light on existing work in the context of data replication.

2.2.1 Syntactic data replication

As discussed in Section 2.1, syntactic data replication focuses on operation execution mechanism for the read and write operations. A number of strategies have been proposed in this category. One of the aspects that differentiate these strategies is their underlying logical network topology (LNT). LNT is the logical representation of interconnections between the replicas on top of a physical network, i.e. how the replicas are logically connected with each other on top of a physical network. ROWA, ROWAA [BG1984], MCS [Tho1979], and WVS [Gif1979] don't have any LNT. These strategies assumes a fully connected network topology. TQP [AA1990] and its variation Generalized tree quorum protocol (GTQP) [AEA1992], Hierarchical quorum consensus (HQC) [Kum1991] arrange replicas in the form of logical trees. Grid protocol (GP) and its variation Hierarchical grid protocol (HGP) [KC1991], and trinagular lattice protocol (TLP) [WB1992] follow a grid-like LNT. These data replication strategies e.g. [KA2011b],[KA2011a], and [SKB2004] exploit the logical structuring of the replicas to achieve high operation availabilities, with a focus on high write operation availabilities at low operation costs.

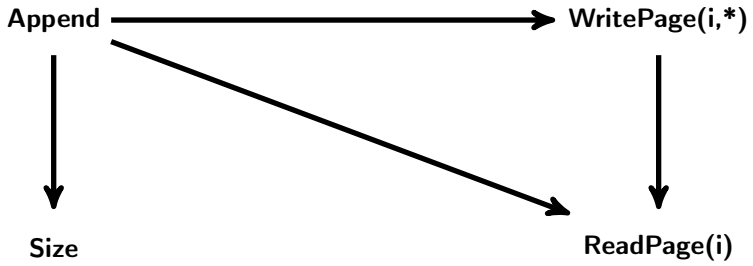
The strategies discussed so far are based on the read and write operation execution mechanism on top of a LNT. In the next section, we discuss strategies which utilize semantic information for high operation availabilities.

2.2.2 Semantic data replication

In [Her1984, Her1986, Her1985] M. Herlihy exploited type-specific properties of abstract data types[4] (ADT) to obtain high operation availabilities. These availability gains are facilitated by type-specific operations which are the only means of manipulating the object of their respective data types [Mey1988]. Another distinct characteristics of this strategy is the replication of the timestamped events instead of version numbers. These events collectively represent the complete history of all the events that occurred on the data item. As shown in Figure 2.5, Append, WritePage,

[4] the data structure used to store the data.

ReadPage, and Size operations are defined for a *PagedFile* data type. Each operation defines an initial read quorum and a final write quorum. Depending upon the system configurations, the number of replicas in the initial read quorum and the final write quorum varies. In Section 2.1, we explained how the arrows represent intersection-relationship between FQ and IQ of dependent operations. For thorough study kindly refer to [Her1984].



Operation	(IQ,FQ)					
Append	(0,5)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)
Size	(1,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)
ReadPage	(1,0)	(3,0)	(3,0)	(3,0)	(4,0)	(5,0)
WritePage	(1,5)	(1,3)	(2,3)	(3,3)	(4,2)	(5,0)

Figure 2.5: Quorum intersection graph for *PagedFile* data type[Her1984].

In [OAET2013], the multimedia quorum based protocol (MQB) is presented which is specifically designed to deal with distributed multimedia objects in an overlay network. MQB introduces two types of write operations named *Enriching* and *Impoverishing*. In case of the enriching operations execution, the replicas involved in the operation execution are updated immediately and move to the updated state. In case of the execution of the impoverishing operation, the respective replicas are not updated and the write operation is recorded in the replicas for later execution. When a read operation is executed on a replica which is previously updated by an impoverishing write operation, then, first the recorded write operations is executed to change the state of the replica and then the read operation is executed to get the updated state. In this strategy, the author exploits the semantics of the write operation to reduce the processing overhead in case of multimedia files as the write updates are deferred until the execution of a read operation. This strategy is suitable for environments where there are relatively less read operations then the write operations.

2.2.3 Erasure Codes

H. Weatherspoon and D. Kubiatowicz [WK2002] investigated data replication strategies based on erasure codes and non-erasure codes. In erasure codes, a data item x is divided into i parts and then these i parts are encoded into j encoded parts such

that $j > i$. The original data item can be reconstructed from any i encoded parts. For example, if x is divided into 4 parts and afterwards these 4 parts are encoded into 12 encoded parts, then to reconstruct x any of 4 encoded parts will be required. The rate of encoding r is defined as $r = \frac{i}{j} < 1$. The storage cost overhead due to erasure code is determined by factor $\frac{1}{r}$. For our example the storage cost overhead is 3 ($r = \frac{4}{12}, \frac{1}{r} = 3$). As per the analyses in [WK2002], erasure coded systems perform far better than their counter-part (replication strategies without erasure codes), and also are efficient in terms of network bandwidth, storage requirements, and I/O. Also, a recent study [GIM2016] on erasure codes and data replication has demonstrated that combining the two strategies – erasure codes and non-erasure code based data replication strategies – helps to develop a system with high operation availabilities.

So far we discussed the state-of-the-art data replication strategies. In the next Section 2.3 we present our problem statement and after that we explain in detail our data replication strategy in Section 2.4.

2.3 Problem Statement

In the data replication strategies proposed by Gifford [Gif1979] and Herlihy [Her1985], they utilized timestamps and histories inorder to provide high operation availabilities. They also described how to make use of different operation types to avoid write quorum intersection. Influenced by these approached, we thought, how to exploit the semantics of data, encoding techniques and a finite state space to achieve high operation availabilities and low operation costs? The basic idea is to instruct the write operation to select the replicas based on the input values, e.g., if the input value is A, then replica k_α and replica k_β will be written. And for the read operation, how to know in advance the value by contacting few number of replicas, and to guide the read operation which replica to read based on the value of the current replica, e.g., if a read operation reads replica k_ϵ and replica k_ρ then which next replica to read will be identified by the values of these replicas.

2.4 Replication Strategy

In this section, we first explain the SDR model and its corresponding characteristics. After that, we elaborate its design and implementation details. At the end, we highlight important achievements as result of our analyses.

2.4.1 Functional Model

We can call data the unprocessed or not refined form of information. When data is expressed in an organized manner, it is known as a data item (X). It is the unit of replication [WPS+2000]. The collection of data in a structural manner is known as a database (DB). A database is managed by a database management system ($DBMS$). In replicated databases, there exist multiple physical copies X_i of data

item X. Data replication strategies abstract these multiple copies to a single copy presented to the client. In SDR, we represent the input value of a data item as X. Now, we discuss in detail the important parts of the SDR model.

2.4.1.1 SDR Replica Model

The input value of the data item is encoded into *replication pointers* as codes X'. X' are the 2-bit binary codes – $00_2, 01_2, 10_2, 11_2$ – which determine the replica group to be read or written. These codes are distributed across the replicas $\Re$. Replicas are divided into 2 groups in total, one group with $2x$ replicas for semantic replicas $\Re_\sigma$ and one group with $4x$ replicas for code replicas $\Re_\omega$. Each replica group is assigned a dedicated role.

Semantic information about the input value X is stored in replica group $\Re_\sigma$. Encoded information about the input value X is saved in replica group $\Re_\omega$. Section 2.4.2 elaborates the need and utilization of the semantic information and the codes. Equation 2.1 explains how the replicas are grouped together. Among replica group $\Re_\omega$, there exist 4 replicas defined as α, β, γ and δ. These four replicas are responsible to save encoded value X' of X. The replica group $\Re_\sigma$ has 2 replicas defined as ϵ and ρ. The replica ϵ is dedicated to save information about two semantic properties: i) whether the code value of the input X is *even*, ii) or whether the code value is *odd*. The replica ρ is dedicated to save information about two more semantic properties: i) whether the code value of the input X is *prime*, ii) or whether the code value is $not - prime$. These semantic properties are discussed in detail in upcoming Section 2.4.1.2 about semantic modeling.

$$\begin{aligned} \Re_\omega &= \{\alpha, \beta, \gamma, \delta\} \\ \Re_\sigma &= \{\epsilon, \rho\} \\ \Re &= \Re_\omega \cup \Re_\sigma \end{aligned} \tag{2.1}$$

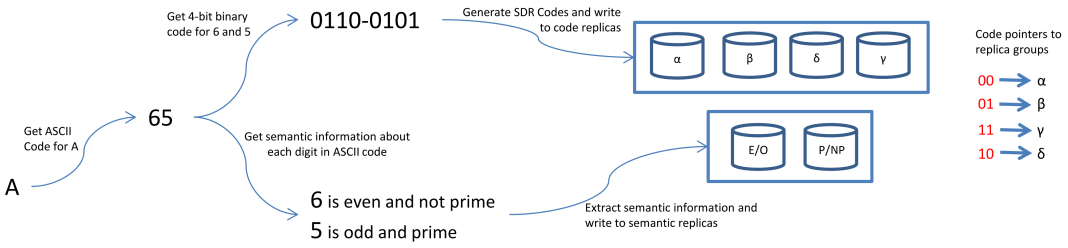


Figure 2.6: A visual representation of the steps involved in SDR. This example shows how the input value is processed to obtain its semantic information and how SDR codes are generated from it. Once we have both then those are written to their respective replicas. SDR codes are saved on α, β, γ and δ. Semantic information is saved on ϵ and ρ.

2.4.1.2 SDR Semantic Model

In ASCII code [ANS1986], every character is represented by numbers in some format like decimal, octa, hexa etc. For the SDR semantic model, we utilized ASCII decimal codes for character encoding. These decimal codes are made up of decimals ranging from 0 to 9 (D).

$$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The input value X is first decoded into the ASCII decimal codes (ADC) which is expressed in the form of decimal decimals. Refer to the example given in Figure 2.6, the input value X is an alphabet "A" and it is expressed as 65 which is its ADC. In proceeding steps, the two semantic properties – refer to Equation 2.2 and Equation 2.3 – are identified for each decimal of the ADC.

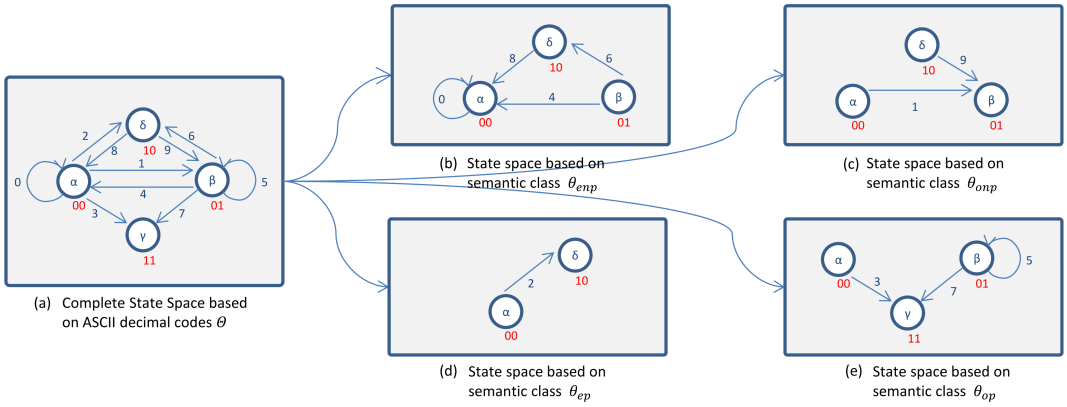


Figure 2.7: Subfigure (a) is the finite state space. It contains all the decimals from 0 to 9. Subfigures (b),(c),(d), and (e) explain sub-state spaces. The 4-bit binary code of the decimals are mapped to the replicas. Subfigures (b),(c),(d), and (e) are derived from subfigure (a). Semantic properties of the decimals are used to derive these sub-state spaces.

As the first step of the replication process, a finite state space (Θ) based on D is generated. In Figure 2.7a, we describe the Θ with the help of a directed graph. The graph for Θ is defined by using the code replicas $\Re_\omega$. The vertices of the graph are composed of $\alpha, \beta, \gamma,$ and δ. Each edge on and between the vertices represents a decimal d, $d \in D$.

$$\Theta = \{0000_2, 0001_2, 0010_2, 0011_2, 0100_2,$$
$$0101_2, 0110_2, 0111_2, 1000_2, 1001_2\}$$

Θ utilizes the 4-bit binary code of d for the decimal representation on the edges. The direction of the edge determines the *from replica* and *to replica* to regenerate the 4-bit binary code of decimal d. For example, as shown in Figure 2.7a the edge between α and γ determines $d = 0011_2$ which is the code for 3. The process to regenerate decimal d is explained in Section 2.4.1.3. The SDR semantic model exploits the following two semantic properties of each decimal d_i:

1. Whether d_i is prime p or not prime np?

$$\begin{aligned} &d_i \in p, p \subset D, p = \{2,3,5,7\} \\ &\text{or} \\ &d_i \in np, np \subset D, np = \{0,1,4,6,8,9\} \end{aligned} \tag{2.2}$$

2. Whether d_i is even e or odd o?

$$\begin{aligned} &d_i \in e, d_i \subset D, d_i = \{0,2,4,6,8\} \\ &\text{or} \\ &d_i \in o, d_i \subset D, d_i = \{1,3,5,7,9\} \end{aligned} \tag{2.3}$$

Based on the two semantic properties, four semantic classes θ_i for D are derived, namely EvenNotPrime (θenp), OddNotPrime (θonp), EvenPrime (θep)and OddPrime (θop). Refer to Sub-Figures b,c,d, and e in Figure 2.7.

$$\begin{aligned} &\theta ep = \{0010_2\} \\ &\theta op = \{0011_2, 0101_2, 0111_2\} \\ &\theta enp = \{0000_2, 0100_2, 0110_2, 1000_2\} \\ &\theta onp = \{0001_2, 1001_2\} \\ &\Theta = \theta ep \cup \theta op \cup \theta enp \cup \theta onp \end{aligned}$$

The semantic class is expressed in the form of sub-state space (θ). There exists a non-intersecting property between the semantic classes, which states that each of θ_x is a subset of Θ such that:

$$\begin{aligned} &\theta ep \cap \theta op = \phi \\ &\theta ep \cap \theta enp = \phi \\ &\theta ep \cap \theta onp = \phi \\ &\theta op \cap \theta enp = \phi \\ &\theta op \cap \theta onp = \phi \\ &\theta enp \cap \theta onp = \phi \\ &\Theta = \theta ep \cup \theta op \cup \theta enp \cup \theta onp \end{aligned} \tag{2.4}$$

Equation 2.4 present the non-intersecting properties of θ_x. In Section 2.4.2, we elaborate the effectiveness of these non-intersecting properties and we explain how the semantic classes θ_x play their role in increasing the operation availabilities. These classes help to know in advance which replicas will be involved in an operation.

2.4.1.3 SDR Encoding Model

In the preceding Section 2.4.1.2, we explained the semantic model of SDR which is based on the ADC. We also explained how the semantic model makes use of the semantic properties of the decimals d. In this section, we elaborate the encoding

model. Along with the semantic model, it sets the the foundation for our replication strategy.

First, the input value X is encoded into an ADC. Afterwards, each decimal d_i of the ADC is converted to a 4-bit binary code. The purpose of using 4-bit binary code is to utilize a minimum number of bits required to encode the decimals from 0 to 9, because the ADC is made up of decimals from 0 to 9. The 4-bit binary code is splited into two 2-bits parts. One part is made up of the lower order 2-bits and the other part is made up of the higher order 2-bits. We refer to these 2-bit binary codes as X'. X' is a replication pointer. It determines the replica group to be read or written. By utilizing the ADC, the binary code for input value X is calculated. The binary code is then splited into X'. Based on 2-bits, the only four possible combination for X' are:

$$\{00_2, 01_2, 11_2, 10_2\} \ni X' \tag{2.5}$$

Here, we focus on the mapping part that exists between X' and the replica group $\Re_\omega$. Each of the 2-bit combination X'_i is mapped to a specific replica in the replica group $\Re_\omega$. This "X' to replica mapping" guides the write operation to identify the target replicas in $\Re_\omega$. This is a very important characteristic of SDR. It uses the knowledge obtained as a result of semantic analysis and binary coding to know exactly which replicas to update. It eliminates the need to use replica searching and selection algorithms, resulting in high operation availabilities. For example, in a quorum system, a replica commit to an operation execution request with the help of its vote. These votes are then collected by the requesting replica to determine whether it has enough votes to perform the operation execution or not. However, in our approach, there is no search for votes. It is the input value that identifies which replicas will participate in the operation execution. In Section 2.4.4, we explain in detail the high operation availability mechanism. Details of the mapping are:

$$\begin{aligned} 00_2 &\mapsto \alpha \\ 01_2 &\mapsto \beta \\ 11_2 &\mapsto \gamma \\ 10_2 &\mapsto \delta \end{aligned} \tag{2.6}$$

For the read operation, sub-state space θ_x is used to identify the target replicas to read the codes. It also provides information about the read order, i.e., the order in which the replicas should be read. Afterwards, the two 2-bit code pairs are combined to get the actual value. Table 2.1 illustrates an example of the SDR encoding model. We demonstrate the process of encoding the input value X into X' and extraction of the semantic classes. So far, we have studied the modeling part of the SDR. We discuss the replicas model, the data semantics model, and the data encoding model. In the proceeding Section 2.4.2, we explain the SDR replication strategy in detail based on SDR functional model.

- Get ASCII codes

Input	ASCII	Binary codes
1	49	$4_{10} = 0100_2, 9_{10} = 1001_2$
A	65	$6_{10} = 0110_2, 5_{10} = 0101_2$
$	36	$3_{10} = 0011_2, 6_{10} = 0110_2$

- Generate SDR codes

Input	ASCII	Binary codes	SDR codes
1	49	$4_{10} = 0100_2, 9_{10} = 1001_2$	$(01_2 \mapsto \beta\ ,\ 00_2 \mapsto \alpha)$,$(10_2 \mapsto \delta\ ,\ 01_2 \mapsto \beta)$
A	65	$6_{10} = 0110_2, 5_{10} = 0101_2$	$(01_2 \mapsto \beta\ ,\ 10_2 \mapsto \delta)$,$(01_2 \mapsto \beta\ ,\ 01_2 \mapsto \beta)$
$	36	$3_{10} = 0011_2, 6_{10} = 0110_2$	$(00_2 \mapsto \alpha\ ,\ 11_2 \mapsto \gamma)$,$(01_2 \mapsto \beta\ ,\ 10_2 \mapsto \delta)$

- Extract SDR classes

Input	ASCII	Binary codes	SDR codes	SDR semantic classes
1	49	$4_{10} = 0100_2, 9_{10} = 1001_2$	$(01_2 \mapsto \beta\ ,\ 00_2 \mapsto \alpha)$,$(10_2 \mapsto \delta\ ,\ 01_2 \mapsto \beta)$	$4 \mapsto \theta_{enp}$,$9 \mapsto \theta_{onp}$
A	65	$6_{10} = 0110_2, 5_{10} = 0101_2$	$(01_2 \mapsto \beta\ ,\ 10_2 \mapsto \delta)$,$(01_2 \mapsto \beta\ ,\ 01_2 \mapsto \beta)$	$6 \mapsto \theta_{enp}$,$5 \mapsto \theta_{op}$
$	36	$3_{10} = 0011_2, 6_{10} = 0110_2$	$(00_2 \mapsto \alpha\ ,\ 11_2 \mapsto \gamma)$,$(01_2 \mapsto \beta\ ,\ 10_2 \mapsto \delta)$	$3 \mapsto \theta_{op}$,$6 \mapsto \theta_{enp}$

Table 2.1: SDR codes and semantic classes extraction

2.4.2 SDR Replication Strategy

In this section, we describe how we develop the replication strategy based on SDR model. First, we describe the request resolution process. After that, we explain how the sub-state space θ is selected and how the 2-bit binary codes X' are distributed on the target replicas. Then, we elaborate on the implementation mechanism for SDR. At the end, we describe our analysis, and finally, we state our results.

2.4.2.1 Request Resolution

A unique request id (r_id) is generated for each client request. In case of the write operation, the value X of the input request (r) is decoded to get its ADC. Refer to Figure 2.6, for the request r, the value to be replicated is "A". The ADC for "A" is 65. As per our semantic model, each decimal d_i of the ADC is analyzed separately against the two semantic properties – refer to Formulas 2.2 and 2.3. The analysis result for the ADC's first decimal 6 is that: it is even e and not a prime number np, therefore its semantic class is θenp. For the second decimal 5 is: it is odd o and a prime number p, therefore its semantic class is θop. Once the semantic analysis is done, the 4-bit binary code of each decimal d_i is obtained. In our example, the 4-bit binary codes for 6 and 5 are 0110_2 and 0101_2, respectively. In Table 2.1 we explained these steps.

The SDR generates a timestamped reference for each request, based on Lamport clocks [Lam1978]. The timestamped reference is made up of $system\_timestamp$, r_id, ch_index and d_i_index. The $system\_timestamp$ contains the datetime value of the replica. It is incremental and unique for each write operation. During a read operation it is used to oder the timestamped reference in ascending order to decode the written value from them. r_id is the request identifier. There are multiple write operations for each write request and r_id is same in each write operation. r_id is used to group the write operations of a request. ch_index represents the character index in the overall request, e.g., for request r("1A\$"), ch_index for $1 = 0, A = 1, \$ = 2$. d_i_index contains the index of decimal d_i in ADC. The structure of the

timestamped reference is as follows:

$$\overbrace{\square\square\square\square\square\square\square\square\square\square\square}^{system\_timestamp}\,|\,\overbrace{\square\square\square\square\square}^{r\_id}\,|\,\overbrace{\square\square\square\square\square}^{ch\_index}\,|\,\overbrace{\square\square\square}^{d_i\_index}\,|\,\overbrace{\square\square}^{od\_index} \tag{2.7}$$

In Figure 2.8, we give an example of timestamped reference for input request "1A\$". There are separate write operations for every $d_i \in$ ADC. During a read operation, correctly identifying the index position of decimal d_i is important, as incorrect index positions can lead to different character codes. For example the ADC for "A" is 65, and if we switch the position of 6 and 5 then it will become 56 which is the ADC for 8.

$/ *$ ASCII code for 1 is 49 $* /$
Timestamp for $4 \Rightarrow system\_timestamp, 250, 0, 0$
Timestamp for $9 \Rightarrow system\_timestamp, 250, 0, 1$
$/ *$ ASCII code for A is 65 $* /$
Timestamp for $6 \Rightarrow system\_timestamp, 250, 1, 0$
Timestamp for $5 \Rightarrow system\_timestamp, 250, 1, 1$
$/ *$ ASCII code for \$ is 36 $* /$
Timestamp for $3 \Rightarrow system\_timestamp, 250, 2, 0$
Timestamp for $6 \Rightarrow system\_timestamp, 250, 2, 1$

Figure 2.8: Timestamp generation for r("1A\$") having $r\_id = 250$

SDR replicates the X' instead of the actual value. Referring to Figure 2.9, the input value to be replicated is "A", SDR transforms the input value into its ADC (which is 65). After that, it gets the semantic information of each decimal d_i of the ADC (6 and 5), and then gets the 4-bit binary code of each decimal d_i of the ADC (which is 0110_2 and 0101_2, respectively). Finally, with the help of Formula 2.6, SDR distributes these codes and the semantic information on to the replicas.

2.4.2.2 θ_x Selection and X' Distribution

After the request resolution step, the next step is the selection of sub-state space θ_x from the finite state space (Θ) for a particular operation. The θ_x selection is based on the analysis result of the two semantic properties 2.2 and 2.3. θ_x selection is a repetitive step and it is performed for each d_i in ADC for request r. That is, for the input request r("A"), its extracted ADC is 65. The θ_x selection is first performed for 6 and then for 5. θenp (EvenNotPrime) is the outcome of the analysis result of 6, and θop (OddPrime) is the outcome for 5. Figure 2.7b shows the sub-state space θenp based on the semantic class EvenNotPrime, and Figure 2.7e shows the

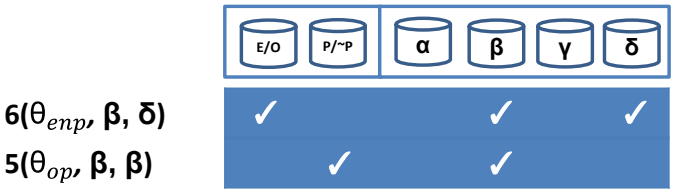


Figure 2.9: This figure shows how replicas are updated by the write operation for input value A. On top six replicas are shown. On left side, semantic classes and NIndices are given. The white tick marks tells which replicas are updated.

sub-state space θop based on the semantic class OddPrime. Once θ_x is selected, the next step is to split the 4-bit binary code of d_i into two 2-bit binary codes X', and then map those to replicas of group $\Re_\omega$. In Figure 2.7b, 6 is represented as the edge between replicas β and δ, because, the 2-bit binary code assigned to β is 01_2 and the 2-bit code assigned to δ is 10_2. The 4-bit binary code for 6 is $0110_2(\beta\delta)$ that means, the 4-bit binary code for 6 is distributed as two 2-bit binary codes X' on the replicas β and δ. The direction of the edge determines the *from* and *to* replicas to regenerate the 4-bit binary code of the decimal d. That is, the binary code for $6_{10} = 0110_2$ splits into 01_2 and 10_2 and is then mapped to replica β and replica δ, respectively, as per our mapping Formula 2.6. In the next section, we explain the implementation of SDR.

2.4.3 Implementation

Until now we explained the key ideas for the SDR replication strategy, now, we discuss the write and read operation execution mechanisms. As an example, assume that for request $r(\text{“}A\text{”})$, the request id r_id is 250. A timestamped reference (tr) $T250$ is generated against the r_id. Timestamped reference represents all the records that have the same r_id. There exist multiple write operations against a r_id, because all the ADC codes are made up of more than one decimal.

	$\Re_\sigma$		$\Re_\omega$			
	ϵ	ρ	$\alpha(00_2)$	$\beta(01_2)$	$\gamma(11_2)$	$\delta(10_2)$
$6_{10} = 0110_2 \mapsto (\delta, \beta), 6_{10} \mapsto \epsilon$	$T250-0$	–	–	$T250-0$	–	$T250-0$
$5_{10} = 0101_2 \mapsto (\beta, \beta), 5_{10} \mapsto \rho$	–	$T250-1$	–	$T250-1$	–	–

Table 2.2: SDR example for write and read operations for $r(A)$.

Algorithm 1: SDR Write Operation

```
   /* Input value to be written, e.g., A                                  */
   Input  : input
   /* Write operation execution response.                                 */
   Output: success_message or error_message
 1 r_id ← getRequestID(Input)
 2 tr ← getTimeStampRef(r_id)
 3 ch_idx ← 0
   /* Iterate for each character in the input, e.g., if the input is
      AB then perform the below steps for A and B.                        */
 4 foreach character ch ∈ input do
 5     codeASCII ← getASCIICode(ch)
 6     foreach integer d ∈ codeASCII do
 7         d_i_idx ← getDigitIndex(codeASCII,d)
           /* Apply the two semantic properties                           */
 8         sc ← getSemanticClass(d)
 9         bc ← get4bitBinaryCode(d)
           /* E.g.  from2bits for first decimal (6) of A are 01_2.        */
10         from2bits ← get2bitCode(bc,higher)
           /* Get the mapping from Mapping Formula 2.6                    */
11         rm ← extReplicaMapping(from2bits)
           /* Higher order bits                                           */
12         od_idx ← higherOrderBits
13         ts ← genTmStamp()
14         tr ← prepareTmStamp(ts, ch_idx, d_i_idx, od_idx)
           /* Write the first timestamp of the decimal                    */
15         writeToReplicas(rm, sc, tr)
           /* E.g.  to2bits for first decimal (6) of A are 10_2.          */
16         to2bits ← get2bitCode(bc,lower)
           /* Get the mapping from Mapping Formula 2.6                    */
17         rm ← extReplicaMapping(to2bits)
18         od_idx ← lowerOrderBits
           /* Write the second timestamp of the decimal                   */
19         ts ← genTmStamp()
20         tr ← prepareTmStamp(ts, ch_idx, d_i_idx, od_idx)
21         writeToReplicas(rm, sc, tr)
22     end
23     ch_index ← ch_index +1
24 end
```

2.4.3.1 SDR Write

In our example, two records are generated for $r(\text{“}A\text{”})$, one for 6 and one for 5, and both of those are represented by $T250-0$ and $T250-1$ respectively, where $0,1 \in d_i\_index$. Table 2.2 explains how the write operation writes the tr among replicas. The values represented as "–" are null values. SDR only writes to a replica if the evaluate-write function returns true. The evaluate-write function is comprised of the two semantic properties discussed in the Formulas 2.2 and 2.3, and the 2-bit binary code X' to replica mapping discussed in Formula 2.6. For the decimal 6 of the request $r(\text{“}A\text{”})$ the responses from the evaluate-write function are:

- 6 is even, then write tr to replica ϵ.
- 6 is not prime, then do not write anything on replica ρ.
- write tr to replicas β and δ and don't write anything on replicas α and γ.

Similarly, based on responses from the write validity predicates for the second decimal 5 of the request $r(\text{“}A\text{”})$, replica ϵ is not written because 5 is not even and ρ is written because 5 is prime, and from the replica group $\Re_\omega$ only replica β is written because decimal 5 is represented as a repetitive loop[5] on replica β, and rest of the replicas are not written. Algorithm 1 gives the pseudo code for the SDR write operation.

2.4.3.2 SDR Read

In the case of a read operation, the replica group $\Re_\sigma$ is queried first prior to replica group $\Re_\omega$, to check for the records with the latest tr. We ensured sequential data consistency by this tr mechanism, as in tr the unique r_id is used to order the occurrence of the read or write operation in the complete system, and ch_index along with d_i_index are used for total ordering of the independent actions performed as part of the write operation.

The read operation first extracts the semantic information about the ADC d_i. After that, the semantic class is identified based on the semantic information. The semantic class helps to select the sub-state space θ_x which determines its respective replicas from $\Re_\omega$. The sub-state space θ_x also describes the order to read from the replicas. The read operation utilizes the sub-state space θ_x to extract the two 2-bit binary codes X' from the replicas $\Re_x \in \Re_\omega$. It then combines the two X', in the order specified by the sub-state space θ_x, to get the 4-bit binary code and this 4-bit binary code is the binary code for the decimal d_i. One important point to remember here is that the read operation always extracts two 2-bit binary codes against the ADC decimals $1,2,3,4,6,7,8,$ and 9, and one 2-bit code in the case of ADC decimals 0 and 5 (because of the repetitive loop, refer to Figure 2.7). Once, all the ADC decimals are extracted, the actual ADC is generated by arranging the decimals in the order specified by the d_i_index part of the timestamped reference. Finally, the generated ADC is decoded back into its actual input value.

[5]For the decimal d_i, the two lower order 2-bits and the two higher order 2-bits are same, e.g., 0000_2 and 0101_2. Refer to Figure 2.7.

Algorithm 2: SDR Read Operation

Input : input
Output: *success_message* or *error_message*

```
1  output as string
   /* Get all records of the latest timestamp                          */
2  allColl ← getTStampInfoFrom$\Re_\sigma$or$\Re_\omega$()
3  chIdxColl ← getAllDistictChIndex(allColl)
   /* Sort the records based on character index.  The reason for
      sorting is to ensure that the result of read operation is the
      same as the actual input in case of multicharacter input.      */
4  chIdxColl ← sortByDesc(chIdxColl)
   /* Iterate over the records                                        */
5  foreach integer chIdx ∈ chIdxColl do
6      dColl ← empty
       /* Get records for a ch_index                                  */
7      foreach integer d ∈ getSubColl(chIdx) do
           /* Get semantic information from $\Re_\epsilon$            */
8          scInfo ← extSCInfo(allColl)
           /* Use the semantic information to know in advance which
              replicas to read from $\Re_\omega$.  With the help of $\theta_x$ identify
              the from and to replicas.  After that identify the
              decimal.                                                */
9          d ← applySCInfoOn$\Re_\omega$(scInfo)
           /* Save the decimal in d                                   */
10         dColl ← addtoASCIICode(d)
11     end
       /* Combine the decimals and then decode them to get the written
          value.                                                      */
12     ch ← genChFromASCIICode(dColl)
13     output ← output + ch
14 end
15 success_message ← output
```

Algorithm 2 gives the pseudo code for the SDR read operation. For explaining the algorithm, we assume that all the replicas are available. At line number 2 of the algorithm two records with the same $tr = 250$ were read. For the first entry, θenp is selected – as replica ϵ has the tr and replica ρ do not have tr therefor the semantic class EvenNotPrime is selected – for the first record against the tr $T250$. θenp explains that, to extract the decimal d, reading can be started from any of the replicas α, β and δ (refer to Figure 2.7b), and it also elaborates the reading order (edge direction between vertices). When reading is started from replica α, for $T250$ there is no record in α, which means that the only possible replicas to check to extract the value for d are β and δ. Here, there are two ways to extract d: d can either be predicted to be 6 because the only possible iteration is from β to δ which make up the 4 bit binary code 0110, or read the other two replicas (β and δ) and then combine their mapped 2-bit binary values to extract the 4-bit binary code. Both options lead to same result, but the choice among the options is made according to the availability of the replicas at that time. Then the same process is applied to the second record which extracts the decimal d as 5, and then the d_i_index part of the timestamped reference is used to order both decimals d_i $(6, 5)$ to get the ADC which is 65. ADC is then decoded back to get the actual character which is "A". At line number 3 and 4, ch_index is used to sort the records if the input has multiple character e.g. "ABC". The reason for sorting is to ensure that the result of read operation is the same as the actual input. From line 5 to 14 the encoded values are read and then decoded to get the written value. In the upcoming Section, we discuss the scenarios in detail where θ is used to correctly predict the 4-bit binary code in the presence of replica failures.

2.4.4 Availability Analysis

We analyzed the operation availability of SDR by putting the replicated system into different states based on different available replica group combinations. Our assumption is that the replicas perform independently and the replica failures are independent of each other. Atleast three of six replicas are needed to ensure strong data consistency. We deduced that all the possible replica combnation can be grouped into four possible scenarios. For each scenario, we explain with the help of an example, how the read and write operations are performed. We also explain, how the finite state space is used to know in advance the replicas to write, and in case of read operation how finite state space is utilized to correctly guess the value. In this section, we explain four possible replica availability scenarios. Afterwards, we discuss the write operation OpW and read operation OpR executions. For each OpW, X' – these are the 2-bit binary codes which determine the replica group to be read or written 2.4.1.1 – guides the OpW to select one or two replicas from $\Re_\omega$ for tr replication. Refer to Section 2.4.2.2 for details on tr, and Formula 2.6 for details on X' to replica mapping.

2.4.4.1 Availability scenario 1: When replicas ϵ and ρ from replica group $\Re_\sigma$, and any two or more replicas from replica group $\Re_\omega$ are available

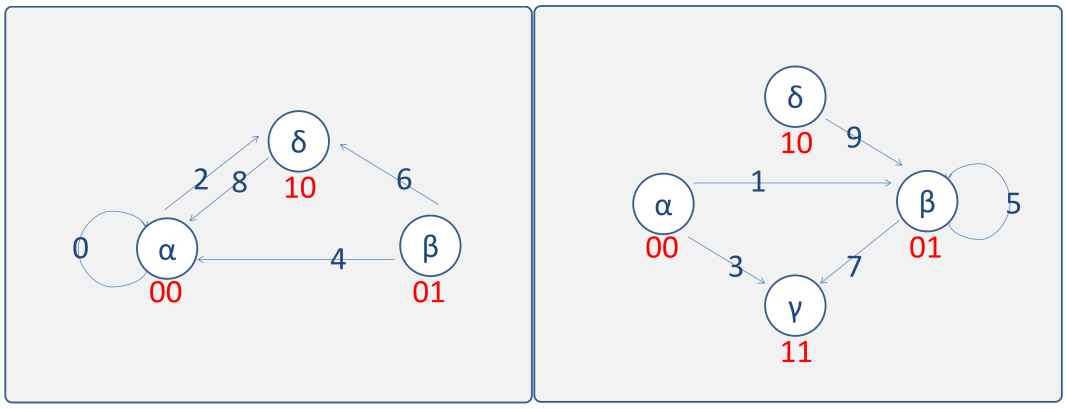
In this case, the complete replica group $\Re_\sigma$ is available along with two or more replicas from the replica group $\Re_\omega$. The OpW is able to replicate the tr on the complete $\Re_\sigma$ group along with either partial or complete tr replication on the $\Re_\omega$ group. Both, sub-state space θ_x and X' to replica mapping facilitates OpW execution. Refer to Section 2.4.3.1 for an OpW example.

OpR reads the tr with the latest timestamped reference from the $\Re_\sigma$ group. First, it extracts the sub-state space θ_x and then it reads the tr records from the $\Re_\omega$ group. It utilizes the information provided by the sub-state space θenp about the replica and the read order. OpR requires at least two replicas from the replica group $\Re_\omega$ to generate the correct 4-bit binary code. For example, for the first record in Table 2.2, if replica γ and replica δ are not available, then OpR checks for tr records on replicas β and α. But, it only finds tr on replica β, and then, it predicts the 4-bit binary code as 0110. As, replica α is empty, which mean that replica α was not written for this tr. Referring to sub-state space θenp (refer to Figure 2.7b), the edge from replica β to replica α is not applicable for this case. The only possibility left is the edge from replica β to replica δ which actually represents the 4-bit binary code 0110 ($\beta\delta$) which is the 4-bit binary code for 6.

2.4.4.2 Availability scenario 2: When replicas ϵ and ρ are not available but any three or more replicas from replica group $\Re_\omega$ are available

In this case, the replica group $\Re_\sigma$ is not available but three or more replicas from the replica group $\Re_\omega$ are available. OpW is guided by X' to replica mapping (refer to Formula 2.6) to select the respective replicas for tr replication.

OpR reads tr from replica group $\Re_\omega$ and then utilizes the information provided by the finite state space for the ADC (refer to Figure 2.7a). To correctly generate the 4-bit binary code for d_i, OpR requires three or more replicas from replica group $\Re_\omega$. For example, for the first record in Table 2.2, if the replica β is not available, then OpR needs to predict the binary code by considering the values from the rest of the three replicas i.e. α, γ, and δ. OpR only finds the tr record in replica δ. By analyzing the contents of tr, OpR finds out that tr at replica δ has od_idx (refer to line numbers 13 and 18 in Algorithm 1) for the lower order 2-bits of the 4-bit binary code, i.e. ??10. Now, as per Θ, the higher order 2-bits can only be 01??. They cannot be 00?? or 10?? or 11??, because 00_2 is mapped to replica α and it contains null ("–"), and 10_2 is mapped to replica δ but there is no repetitive loop on replica δ, and 11_2 is mapped to replica γ which also contains null ("–"). So, OpR utilizes the Θ to decode the value of the decimal. It then generates a complete 4-bit binary code 0110_2 which is actually the binary code for 6.



(a) State space based on semantic class θ_e

(b) State space based on semantic class θ_o

Figure 2.10: State space for even and odd numbers

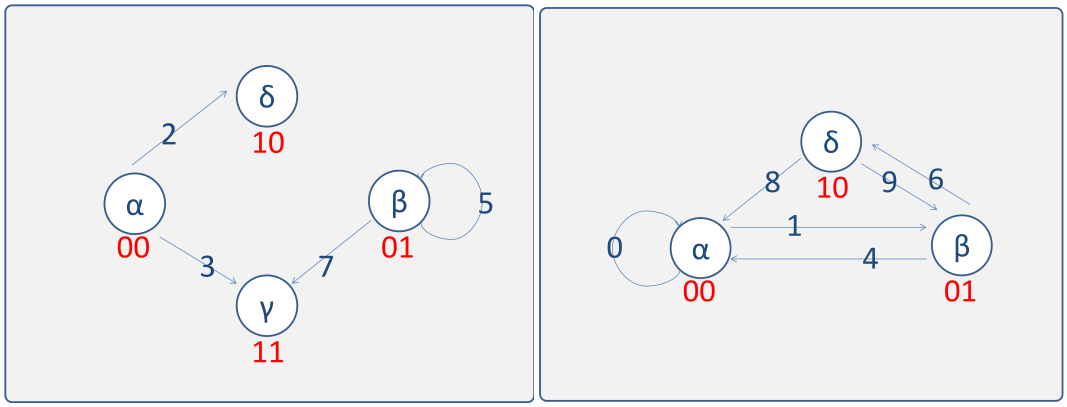
2.4.4.3 Availability scenario 3: When only replica ϵ and two or more replicas from replica group $\Re_\omega$ are available

In this case, the replica ϵ and two or more replicas from the replica group $\Re_\omega$ are available. Two sub-state spaces – θe and θo – from the Θ are extracted, as shown in Figure 2.10. The reason behind the extraction of two new sub-state spaces is that the availability of only one semantic replica, which is replica ϵ, and replica ϵ represents the even/odd semantic property. θe (Figure 2.10a) and θo (Figure 2.10b) represent the two semantic classes of even and odd numbers from ADC, respectively.

$$\theta e = \{0000_2, 0010_2, 0100_2, 0110_2, 1000_2\}$$
$$\theta o = \{0001_2, 0011_2, 0101_2, 0111_2, 1001_2\}$$
$$\theta e \cap \theta o = \phi$$
$$\Theta = \{\theta e \cup \theta o\}$$

OpW replicates tr on replica ϵ and at least two of the available replicas in replica group $\Re_\omega$. od_idx of tr written on replica ϵ is set to 0 for θe and 1 for θo. OpW is facilitated by X' to replica mapping to identify the replicas to be written.

OpR reads the tr from replica ϵ and determines the respective newly derived sub-state space. It can be either θe or θo. For example, for the first record in Table 2.2, OpR determines θe, and then, as per definition of θe, the binary code (to be decoded) can only be of $d_i \in$ even numbers. Now suppose only replicas α and γ are available. Then as per θe, the number can not be $0, 2, 4$, and 8 because replica α has null tr. Therefore, the only possibility is 6. At the end, OpR follows the same process of generating the binary code as explained in the previous Section 2.4.4.2.



(a) State space based on semantic class θ_p

(b) State space based on semantic class θ_{np}

Figure 2.11: State space for prime and not prime numbers

2.4.4.4 Availability scenario 4: When only replica ρ and two or more replicas from replica group $\Re_\omega$ are available

In this case, the replica ρ and two or more replicas from replica group $\Re_\omega$ are available. Two sub-state spaces – θp and θnp – from the Θ are extracted, as shown in Figure 2.11. The reason behind the extraction of two new sub-state spaces is that the availability of only one semantic replica, which is replica ρ, and replica ρ represents the prime/not-prime semantic property. θp (Figure 2.11a) and θnp (Figure 2.11b) represent the two semantic classes of prime and not-prime numbers from ADC, respectively.

$$\theta p = \{0010_2, 0011_2, 0101_2, 0111_2\}$$
$$\theta np = \{0000_2, 0001_2, 0100_2, 0110_2, 1000_2, 1001_2\}$$
$$\theta p \cap \theta np = \phi$$
$$\Theta = \{\theta p \cup \theta np\}$$

OpW replicates tr on replica ρ and at least two of the available replicas in replica group $\Re_\omega$. od_idx of tr written on replica ρ is set to 1 for θp and 0 for θnp. OpW is facilitated by X' to replica mapping to identify the replicas to be written.

OpR reads the tr from replica ρ and determines the respective newly derived sub-state space θ_x. It can be either θp or θnp. For example, for the second record in Table 2.2, OpR determines θp, and then, as per definition of θp, the binary code can only be of $d_i \in$ prime numbers. Now suppose the replicas α and γ are available. Both the replicas have null tr. Which means the value can not be $2, 3$, and 7. Therefore the value is 5. OpR follows the same process of generating the binary code as explained in Section 2.4.4.2.

In this section we elaborated on how the finite state space is used to correctly identify the value when some replicas are not availble. For the rest of the configuration,

we consider the operations being unavailable, since they compromise on strong data consistency. If we relax the strong data consistency level, then it further increases the operation availabilities for SDR.

2.4.5 Results

What is actually the role of θ, X', and X' to replica group $\Re_\omega$ mapping, and how do these SDR attributes increase the operation availabilities? As shown in Figure 2.6, the 4-bit binary code of each decimal $d_i \in D$ is mapped among the replica group $\Re_\omega$ as a combination of two states or one repetitive state.[6] For the read operation, θ helps to select a sub-state space and to identify the replicas from replica group $\Re_\omega$. It also determines the sequence to read from the same set of replicas. For the write operation, based on the input value, the X' to replica mapping determines the replicas to write.

This approach of replica selection based on semantic classes and the sub-state space is more write-efficient as compared to existing syntactic data replication strategies like quorums consensus or voting protocols. It utilizes the a prior knowledge in the form of the sub-state space complimented by the X' coding techniques. It is important for us to come up with a highly available data replication strategy, particularly the high write operation availability, which is a motivation behind SDR.

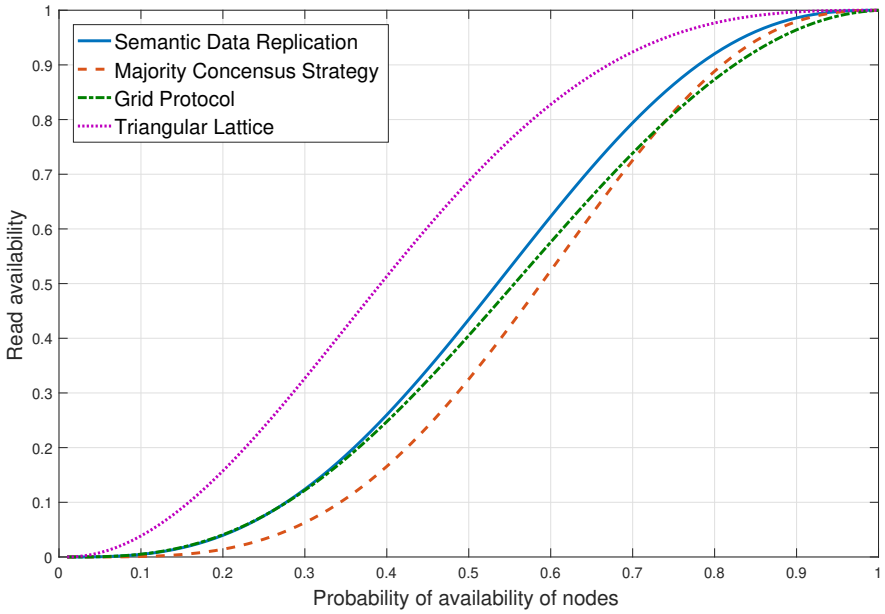


Figure 2.12: Read operation availability

[6] it means, for the respective decimal d, the two lower order 2-bits and the two higher order 2-bits are same e.g. 0000 and 0101.

In Figure 2.12, we demonstrate the read operation availability comparison results. The details about the write operation availability comparison results are discussed in Figure 2.13. The read operation and write operation availabilities are compared against the availability probability of nodes (replicas). Availability probability of a replica can be either 0 or 1. Our assumption was that replicas are independent of each other. Failure of one replica do not have any impact on any other replica.

These comparison are done between our data replication strategy and some of state-of-the-art highly available syntactic data replication strategies. For our analysis we used six replicas. We compared our strategy with the majority consensus strategy [Tho1979], the Grid protocol [CAA1992] and the trinagular lattice protocol [WB1992] for the same number of replicas. These state-of-the-art data replication strategies are selected for comparison because they provide high operation availabilities, and are mostly referenced in literature on data replication strategies. During the analysis, we found that the write availability of our data replication strategy is higher than the other data replication strategies. Our approach is best suitable for the applications which have higher percentage of write operations, for example, writing logging trails. In the case of a read operation, the triangular lattice protocol demonstrates better results than all the data replication strategies, but our approach is still able to achieve higher read availability than the rest of the data replication strategies.

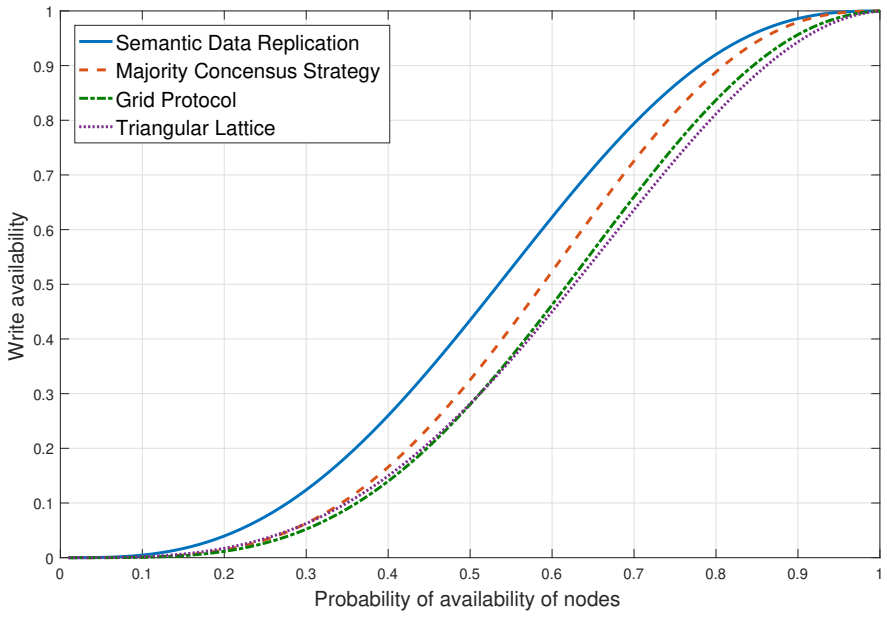


Figure 2.13: Write operation availability

2.5 Conclusion & Future work

In this work, we presented a novel highly available semantic data replication strategy which is empowered by *coding techniques, finite state spaces based on ASCII decimal codes* as *a priori knowledge, semantic classes,* and *codes to replica mapping.* We have elaborated in detail, how the input value is transformed using the ADC, and how we established a finite state space based on ADC. Then, we explained how to extract the semantic classes from the ADC finite state space, by classifying the decimals of the ADC against the two semantic properties. Afterwards, we discussed, how we transformed the 4-bit binary code of the ADC decimal into two 2-bit binary codes, and then mapped these 2-bit binary codes to the code replicas. We focused primarily on achieving *high operation availability for write operations.* We have shown that the finite state space and semantic classes guides the write operation, based on the input value, to the exact replicas to write. This is in contrast to syntactic data replication strategies, where every write operation *has to look* among the complete population to lock the resources by running different replica searching and selection algorithms. Our results show that for 6 replicas, we achieve a better write availability then the state-of-the-art syntactic data replication strategies. For the read operation availability, our approach stood second best in the comparison.

The constraint of having 6 replicas is due to ASCII decimal codes. We need two replicas to map semantic information (explained in Section 2.4.1.2) and four replicas to map the ASCII decimal codes (explained in Section 2.4.1.3). There are fewer research contributions in the field of semantic data replication as compared to syntactic data replication. That is reason there exist a number of syntactic data replication strategy like majority consensus strategy [Tho1979], the Grid protocol [CAA1992] and the trinagular lattice protocol [WB1992]. However, there still exists a lot of research potential in semantic data replication, and can be exploited for *specific* applications, because, then we can design the strategy as per the needs.

Currently, our approach is utilizing ASCII decimal codes as its coding technique. We will explore more on the coding techniques as a next step towards semantic data replication. Furthermore, we will investigate options to utilize the semantic properties of the underlying data structure, specifically hash table like data structure. We will also concentrate to come up with more semantic properties about the data and the application domain. In this work, we primarily focused on the semantic aspect of the replication strategy. However, as part of our future work, we will be demonstrate on the other features of this semantic replication strategy like network latency and storage consumption.

3 Component-Based Data Replication Strategy

In this highly dynamic era of technology, most of the data-intensive applications are designed to target a good combination of high operation availabilities, scalability, data consistency etc. To address this issue, the research community has introduced many valuable techniques based on data replication or data distribution [Tho1979, CAA1992, UST2017]. However, in ongoing research there is a paradigm shift: the research community is targeting to minimize the inter-application coordination – the coordination required to ensure data consistency – to achieve high operation availabilities. In this chapter, we present a *component-based highly available replication strategy (CbHaRS)* which exploits operation types and a hybrid communication method to achieve high operation availabilities. CbHaRS is highly scalable. It utilizes data components as building blocks for the replication strategy. Data consistency in CbHaRS is ensured by a so-called *component administrator*. The communication mechanism between the data components and the component administrator depends upon the state of the data components. Additionally, the state of the data component is manipulated by the operation type. We further extend the concept of client specific on-demand replication to general component-based replication. To prove the effectiveness of CbHaRS, we have implemented the CbHaRS prototype and discuss the achieved results. In the next section, we discuss the related literature.

3.1 Literature Review

There is an old Greek saying from Heraclitus (quoted in Robinson, 1968: 90), interpreted as, "The only thing that is constant is change," and it is definitely true from

the perspective of technology. From the last two decades, there is a lot that has been changed. However, these changes are the outcome of the continuous improvements. Presently, the online hi-tech firms like Facebook, Google, Amazon etc. support millions of a continuously growing number of users. These applications rely on strong distributed backend infrastructures [DHJ+2007, BAC+2013, CDE+2013]. The primary reason for having such strong backend infrastructures is to ensure enhanced user experience in terms of responsiveness and to do routine tasks, and a system should provide high operation availabilities to accomplish this. The higher the operation availabilities are, the richer is the user experience [BDF+2013, BFF+2014, UST2017].

To have gains in high operation availabilities, researchers have provided a number of data replication strategies. Every data replication strategy revolves around the trade-off factors like high operation availabilities, data consistency, and operation costs [GL2002, Aba2012, UST2017]. Data consistency is an important aspect of the research on distributed systems. There exist different types of distributed systems with different data consistency requirements. In [ABCH2013], authors discussed different approaches of data consistency ranging from strong data consistency to week data consistency. They summarized that the choice of data consistency can be made according to the application requirements. For a global scale system, adaptation of the traditional strong data consistency impacts the operation availabilities of the system at a higher extent as compared to a small scale system [BCvR2009].

In [DPS+1997], authors discuss application-specific conflict resolution for weakly consistent replicated databases. In [GDN+2003], authors make use of application-specific distributed objects – an encapsulation of business logic and data – to provide high operation availability. They designed separate data consistency model for each distributed object, and they make use of persistent-asynchronous communication between the shared objects to ensure data consistency. In [BDF+2013], authors discuss highly available transactions. They exploit transaction isolation levels and data consistency to introduce application-level data consistency which is the base for highly available transactions. One of the facts is that inter-component coordination is required to ensure data consistency. The research community highlights that an increase in coordination results in an increase in operation costs. Minimized coordination has a positive impact on the distributed system especially from the perspective of operation availabilities. In [BFF+2014, BDF+2015], the authors exploit application-specific rules for high operation availabilities, and they introduced the concept of minimizing the coordination – coordination required between replicas to ensure consistency – among the data replicas.

In [SPBZ2011b], conflict-free replicated data types (CRDTs) are introduced. As the name suggests, these data structures are used to minimize conflict – the conflict which arise as a result of simultaneous operation executions. CRDTs are distributed among the replicas. With the help of CRDTs, replicas can be updated independently without the need of coordination with each other. Optimistic data replication makes use of CRDTs, it allows the conflicts to occur and later resolve the conflicts by merging the updates from different replicas. Currently, there are several implementation of CRDTs like maps, sets, counters, registers, and flags [RED, RAIa, RAIb].

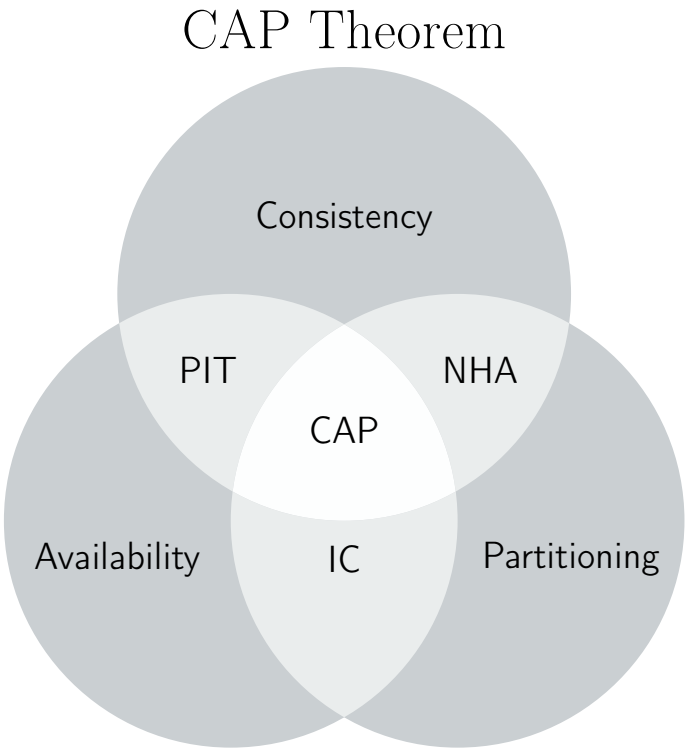


Figure 3.1: CAP Theorem [GL2002] explains how availability, data consisteny, and partitioning influence the desing of a data replication strategy.

There are two types of CRDTs i) operation-based conflict-free replicated data types (OB-CRDTs) and ii) state-based conflict-free replicated data types (SB-CRDTs) [SPBZ2011a]. Both of the types offer strong eventual consistency. OB-CRDTs are commutative in nature, that is why are also know as *commutative replicated data types (CmRDTs)*. They communicate with each other by sending operation requests. CmRDTs depend on the guarantee that each replica receives the operation request only once. However, the order in which the replica executes the operations is not important. SB-CRDTs are convergent in nature, that is why they are also known as *convergent replicated data types (CvRDTs)*. SB-CRDTs differs from OB-CRDTs by the way the replicas communicate with each other. SB-CRDTs send their complete state to other replicas where they are joined and merged to resolve the conflicts. Like OB-CRDTs, the order in with the states are merged is idempotent.

Brewer's CAP theorem demonstrates that the high operation availabilities are impacted by strong data consistency and partition intolerance [GL2002]. To achieve high operation availabilities, strong data consistency and partition intolerance need to be relaxed [DHJ+2007]. The data replication strategy which is highly available and provide strong data consistency is Partition-intolerant (PIT). The data replication strategy designed to bear partitioning and is highly available can't guarantee strong

data consistency and is Inconsistent (IC). Lastly, the data replication strategy which is partition-tolerant and ensures data consistency are Not highly available (NHA). We also experience the same in case of transaction execution. To ensure ACID properties [HR1983], two transactions cannot execute simultaneously if both try to update a common data item. In this case, one of the transactions is blocked until the resources which are locked by the ongoing transaction are not unlocked. And, in case of network partitioning the blocked transaction remains blocked for much longer or until restart, because the required resource is no longer available due to network partitioning [BDF+2013].

The coordination minimization mainly depends upon the application invariants – the data consistency rule defined by the application – and the transactions' isolation levels [BDF+2015, BFF+2014, BDF+2013]. The application invariants also influence operation execution. The application operations are categorized into two broad categories:

- *conflicting operations*: operations of conflicting categories cannot execute simultaneously, because, their simultaneous execution may invalidates a global application invariant.[1]
- *non-conflicting operations*: operations of non-conflicting categories can execute simultaneously. These operations depend primarily on replica-level application invariants.[2] These can be concurrent read-read, read-write, or write-write operations.

Data consistency rules defined by the application semantics are enforced by the application invariants. There are two types of application invariants: replica invariant (RI) and global invariant (GI). For example, in an ecommerce retail store, a GI is enforced at the application level such that:

$$0 < \sum_{n=1}^{N} Sold\_Items(n) \leq Total\_Items \leq 3000$$

and an RI is enforced on a single replica such that:

$$0 < \sum_{n=1}^{N} Sold\_Items(n) \leq Allocated\_Items \leq 500.$$

In this example *Total_Items* are distributed among the replicas as *Allocated_Items* such that:

$$\sum_{n=1}^{N} Allocated\_Items(n) = Total\_Items$$

When each replica enforces its RI then the data consistency – for this application invariant – is ensured as the sum of *Allocated_Items* can not be more than *Total_Items*.

[1] A data consistency rule applicable to the complete application e.g. $0 < \text{count}(x) \leq 100, x \in \mathbb{N}$

[2] A data consistency rule applicable to the respective replica only e.g. $0 < \text{count}(x) \leq 20, x \in \mathbb{N}$

We elaborate in detail in Section 3.4 on the communication mechanism and how the invariants are enforced. The main idea is to ensure the validity of global invariants during simultaneous execution of replica invariants. In the next section we introduce in detail our data replication strategy.

3.2 Introduction

In this chapter, we present the *component-based highly available replication strategy (CbHaRS)* [UST2017]. It is continuation of research work for designing application-specific highly available data replication strategies. CbHaRS focuses towards achieving high operation availabilities by utilizing the states of *data components (DCMs)* – a replica and the data item hosted by that replica collectively are represented as a data component (DCM)[3] – and by exploiting the the data component's operation execution mechanism. One of the data components plays the role of *component administrator (CA)*. The operation mechanism describe the mode of operation execution. One mode is the *local* operation execution by the DCM. In this mode, replica invariants are enforced by the DCM, and no communication is required between DCM and CA. The other mode is the *global* operation execution by DCM and CA. In this mode, replica invariants are ensured by the DCM and global invariants are ensured by the CA. Communication between DCM and CA is required to enforce global application level invariants.

CbHaRS uses DCMs as the building blocks for the data replication strategies. The DCMs exist at different abstraction levels. From the application's perspective, there exists a DCM for each client. And, from the client's perspective, there exist multiple copies of its DCM [BCD+2000]. For the scope of this chapter, we elaborate on the client's perspective which is the basic unit for our data replication strategy. CbHaRS is scalable and at the same time it is fault-tolerant. Plugging in a DCM contributes towards scalability. For example, during a campaign when high user flux is expected on the system then we can add DCMs and when the campaign is over we can remove the not needed DCM. Addition and removal of DCM requires reconfiguration. However, the achieved benefits are much more as compared to the reconfiguration overhead.

Application invariants are categorized into replica invariant (RI) and global invariant (GI). In [BDF+2015], the concept of global and local application-level constraints is defined with the help of a middleware running on top of a geo-replicated data store. The RIs are defined on DCMs. The GIs are defined only on the CA. The RIs allow simultaneous local-operation executions on multiple DCMs targeting the same data item. In case of GIs, the operation executions are managed by the component administrator. To ensure the consistency of the global state of the system, the operation executions that involves GIs may result in a change to the RIs. These operations are executed in synchronous communication mode. On the other hand, the DCMs

[3]Here, the assumption is that each replica hosts only one data item

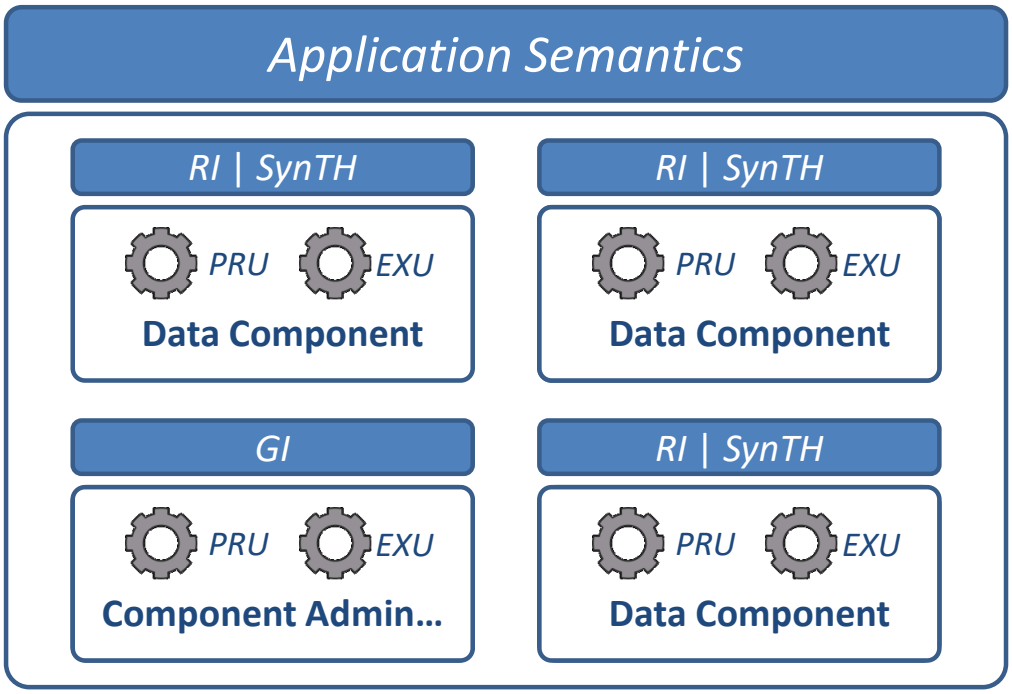


Figure 3.2: The application semantics layer defines the rules to the implement required data consistency. Replica invariant is enforced by the data consistency. Prepare-update operation and execute-update operation operations used to manipulate the data item. Global invariant is enforced by the component administrator.

communicate with each other via the CA asynchronously. The following are the main contributions of our work:

- we exploit operation types and a hybrid communication method to achieve high operation availabilities
- we utilize data components for scalability and fault tolerance
- we allowed non-blocking concurrent operation executions while ensuring causal data consistency

In Figure 3.2, we provide an overview of our data replication strategy. There is one CA and three DCMs. All of them have two types of operation namely prepare-update operation (PRU) and execute-update operation (EXU). These operations are used to synchronize with CA. Along with RI the DCMs also have a synchronization threshold (SynTH). It is used to notify CA that the DCM requires reconfiguration. We elaborate in detail in Section 3.4. Afterwards, in Section 3.5, we summarize our recent progress and elaborate on the future directions of our work.

3.3 Problem Statement

In most of the data replication strategies data consistency is ensured by controlled execution of mutually exclusive operations. Read-Write and Write-Write operations are mutually exclusive when the target is the common data item. For these mutually exclusive operations, when a write operation updates a data item, then the read operation should not read the value of that data item until the write operation execution completes. Same is the case for Write-Write operations execution scenarios. In this work, we further extended the concept of coordination avoidance – the communication between the replicas to mutually agree on an operation execution – between the replicas to achieve high operation availability. Our contribution focuses on the data consistency rules defined by an application. We exploit application rules to allow simultaneous execution of mutually exclusive operations. The application rules are divided into two categories. One category allows simultaneous execution of mutually exclusion operations and the other category do not allow simultaneous execution.

3.4 Replication Strategy

The component-based highly available replication strategy (CbHaRS) to be presented is a component-based data replication strategy. CbHaRS focuses on a specific type of applications. This type of applications defines data consistency criteria based on its rules. If the application-defined rules are enforced, then data consistency is ensured. In case of CbHaRS, operations are defined on so-called data components (DCMs). The DCMs use these operations to communicate with each other and to update their data item. We define two types of operations: i) the prepare-update operation (PRU) and ii) the execute-update operation (EXU).

GI and RI are defined according to application rules. In some scenarios, a GI is distributed among the DCMs as RI. For the DCMs, a synchronization threshold (SynTH) is defined against the RI. When the DCM's data item value reaches the SynTH, then the DCM notifies the CA to update the respective RI across all the DCMs. Because, once the DCM reaches a SynTH, it cannot process further request against the respective RI. There is a possibility that a DCM reaches its SynTH while other DCMs don't. This situation is monitored by CA. We explain these concept with the help of below example.

Example Let us take an example of an event management system. For this application, some rules are defined, e.g., number of participants are limited, every participant must register with a unique email address, and ticket reservations must be less than or equal to the number of available tickets. Based on this set of rules, we define an application invariant *AVAILABLE_TICKETS* as GI. A validation condition for this GI is defined as:

$$RESERVATIONS \not> AVAILABLE\_TICKETS$$

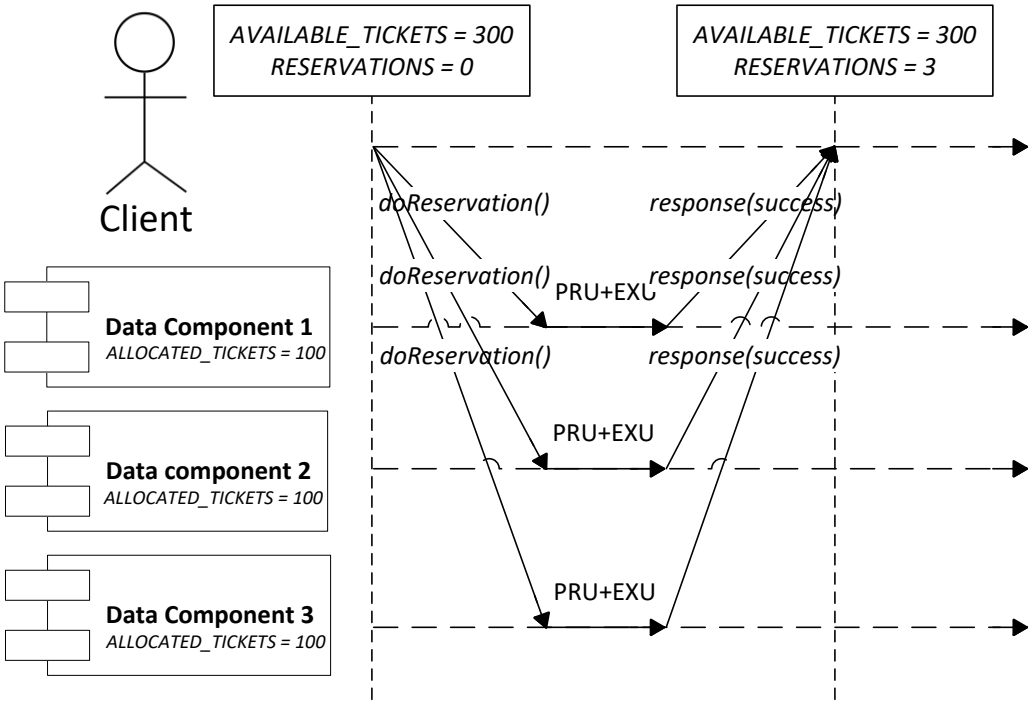


Figure 3.3: Data components execute concurrent replica invariant-related operations. A RI named *ALLOCATED_ TICKETS* validates operation executions at replica level. This RI ensures that a data component can do a maximum of 100 reservations. Collectively, all the data components can do 300 reservations.

To enable coordination-less concurrent executions of the operations for DCMs, the GI is divided among the DCMs. Each DCM now has a RI called *ALLOCATED_ TICKETS*. The DCM takes the responsibility to enforce its RI. The combine affect of all the DCMs ensures the validity of the GI, that is:

$$\sum_{n=1}^{N} \text{ALLOCATED\_TICKETS}(n) \not> \textit{AVAILABLE\_ TICKETS} \tag{3.1}$$

where N is the total number of DCMs. As shown in Figure 3.3, the DC executes a PRU, when it receives a client's request to update the data item. A PRU operation prepares an update package (UDP). The UDP contains: i) the value to be updated, ii) the steps to perform the update, and iii) the state information about the respective RI. The DCM applies the UDP to perform the actual update with the help of EXU. The DCMs manipulate their states depending upon: i) SynTH status and ii) RI validation. Based on the DCM's current state, the communication mechanism – coordinated communication or coordination-less communication – is selected. The CA maintains a key-value store to record the current state of each of the DCM.

The role of CA is to maintain consistency of the RIs across all the DCMs. It manipulates the DCM states and performs synchronization within the DCMs in response to the RI update requests. The DCM can be in two states: i) the synchronized state and ii) the unsynchronized state. A DCM is in the synchronized state when the SynTH for any of its RIs is not reached, and for the unsynchronized state it is vice versa. In context of data replication – for some quorum-based data replication strategies for example MCS and TLP – the coordination among replicas is required to ensure data consistency. In CbHaRS, when DCMs are in a synchronized state, then DCM allow concurrent execution of the mutually exclusive operations. However, the data consistency is ensured by the RIs – as explained in Equation 3.1. We will discuss more on the DC states, operation executions, and communication mechanisms in the following Sections 3.4.1, 3.4.2, and 3.4.3 respectively.

3.4.1 RI-related operation execution in a synchronized state

In CbHaRS, a DCM is in a synchronized state, when for its RI the limit is not reached. For example, if the limit for *ALLOCATED_TICKETS* is 100 and the DCM has done less than 100 *RESERVATIONS*, then the DCM can execute an operation without coordinating with other DCMs. We explain the execution steps in Algorithm 3 from Line number 6 to Line number 14. In this case, operation cost = 1, because a single DCM is involved in an operation execution. Referring to Figure 3.3, concurrent seats reservations are carried out simultaneously by the DCMs against their *ALLOCATED_TICKETS* RI. There are three DCMs. For each DCM a RI *ALLOCATED_TICKETS* is defined like:

$$ALLOCATED\_TICKETS = 100$$
$$0 \leq RESERVATIONS \leq 100$$
$$RESERVATIONS \leq ALLOCATED\_TICKETS$$

No coordination is required among the DCMs during these operation executions. The data consistency at DCM is ensured by *ALLOCATED_TICKETS* RI. The operation availability depends upon the availability of the DCM, and a quorum formation is not needed. The *AVAILABLE_TICKETS* GI is ensured because, in total, all the DCMs cannot do more than 300 seat reservations.

3.4.2 RI-related operation execution in an unsynchronized state

When against a RI, a DCM reaches its limit, then the DCM switches its state to unsynchronized state. For example, in case of *ALLOCATED_TICKETS* RI, if a DCM has done 100 *RESERVATIONS*, then the DCM change its state to unsynchronized state – line number 12 to 14 in Algorithm 3. The unsynchronized state means that the DCM requires coordination with other DCMs and CA. In this case, operation cost $> 1 \leq N$ where N is total number of DCM.

Once a DCM reaches it SynTH, it informs the CA. At this point, the CA collects the RI utilization status from all the DCMs and performs one of the following actions:

Algorithm 3: Replica invariant related operation execution

```
   Input  : input
   Output: response
   /* Get DCM state:  synchronized or unsynchronized                    */
1  stateFlag ← getCurrentState()
   /* Prepare update package                                            */
2  UDP ← prepareUpdateRequest(input)
3  if stateFlag = unsynchronized then
   |  /* Forward the request to CA to process the request              */
4  |  response ← putRequestToCA(UDP)
5  else
   |  /* Is the request GI manipulative?                                */
6  |  requestType ← getRequestType(UDP)
7  |  if requestType = nonManipulative then
   |  |  /* Process request by DCM                                      */
8  |  |  response ← executeUpdateRequest(UDP)
   |  |  /* Check if DCM threshhold reached                            */
9  |  |  if DCMThreshholdReached() = true then
   |  |  |  /* Notify CA to trigger RI configuration                   */
10 |  |  |  notifyCA()
11 |  |  end
   |  |  /* Check if RI limit reached                                  */
12 |  |  if DCMLimitReached() = true then
13 |  |  |  stateFlag ← unsynchronized
14 |  |  end
15 |  else
   |  |  /* Forward the request to CA to process the request          */
16 |  |  response ← putRequestToCA(UDP)
17 |  end
18 end
```

- it recomputes the RI and its SynTH. Afterwards, it distributes the updated value of RI to the DCMs and reset their states to synchronized states.
- if the RI limit for most of the DCM is reached, then the CA defer the action. For example, if two of the three DCMs have done 200 *RESERVATIONS* and third has done 90 *RESERVATIONS*, then in this case no action will be taken. Because the overall cost of reconfiguration is higher than the remaining *RESERVATIONS*.

For example, the SynTH for DCMs in Figure 3.3 is set to 80. When a DCM has done 80 reservations, then it notifies CA about the status of *ALLOCATED_ TICKETS* RI. After 100 reservations, the DCM switches its state to unsynchronized state state. Meanwhile the CA receives another notification from the second DCM. At this point in time, the CA collects the values of *ALLOCATED_ TICKETS* RI from the three DCMs. It will then reconfigure the *ALLOCATED_ TICKETS* RI by increasing the *ALLOCATED_ TICKETS* values for first DCM and vice versa for third DCM. The important point is that Equation 3.1 holds.

Algorithm 4: Global invariant related operation execution

```
   Input  : input
   Output: response
   /* Configuration could be pull or push mechanism                    */
1  configurationFlag ← getConfigurationFlag()
   /* Prepare update request package                                   */
2  UDP ← prepareUpdateRequest(input)
3  if configurationFlag = pullMechanism then
   |  /* For pull mechanism, all GI related requests are executed by
   |     the CA                                                        */
4  |  response ← putRequestToCA(UDP)
5  else
   |  /* For push mechanism, all GI-manipulative operation requests
   |     are executed by CA                                           */
6  |  requestType ← getRequestType(UDP)
7  |  if requestType = nonManipulative then
   |  |  /* GI-nonmanipulative operation request is executed by DCM */
8  |  |  response ← executeUpdateOperation(UDP)
9  |  else
   |  |  /* GI-manipulative operation requests are executed by CA.
   |  |     After the execution, the updated GI value is pushed to all
   |  |     DCMs                                                      */
10 |  |  response ← putRequestToCA(UDP)
11 |  end
12 end
```

3.4.3 GI-related operation execution by CA

Let us define *UNIQUE_EMAIL* as GI for our event management system. According to this GI, an email address can only be used for a single registration. Now, as per CbHaRS, this application invariant cannot be divided among DCMs. Because simultaneous registrations by multiple DCMs invalidate this GI. To ensure the validity of the GIs, the GIs are managed by the CA. It is the responsibility of the CA to propagate the updated information about GIs to the DCMs. For example, once an email address is registered by the CA, then this information should also be communicated to the DCMs. For the CA, there are two mechanisms to manage the GIs: i) a pull mechanism and ii) a push mechanism.

3.4.3.1 Pull mechanism – On-request information retrieval

Using the pull mechanism, the DCMs always coordinate with the CA for operation executions involving GI. For example, if a DC receives a request to inquire about the registration status of an email address, then the DC coordinates with the CA to determine the registration status – line number 3 and 4 in Algorithm 4. After that, the DC returns back the registration status received against the inquiry from the CA. This approach always returns the consistent information, but it puts an extra burden on the CA to also address the get-value operations. We named these get-value operations as GI-nonmanipulative operations because they only get value of the GI and donot perform any update on it.

3.4.3.2 Push mechanism – Propagate updated information

Using the push mechanism, the DCMs address the GI-nonmanipulative operations. However, in case of GI-manipulative operations, the DCMs require coordination with the CA, and the DCM forward these operation requests to the CA – line number 6 to 10 in Algorithm 4. The GI-manipulative operations are the set-value operations and are used to update the value of GI. The CA first executes a PRU to prepare the UDP. Afterwards, it applies the UDP with the help of EXU. As soon as the execution of the EXU is completed, the CA pushes the UDP to all the DCMs. The DCMs upon reception of the UDP from the CA, apply the UDP with the help of EXU.

In push mechanism, the prime responsibility of the CA is to manage GI-manipulative operations. To ensure consistency, the CA executes this type of operations in serial order. Referring to Figure 3.4, for the *UNIQUE_EMAIL* GI, a DCM addresses the *isRegistered()* inquiry. Simultaneously, CA performs the *deRegister()* update for the same GI. The DC responded that a registration for the email exists, but – in parallel – the CA cancels the registration which result in temporary data inconsistency. The data inconsistency is eliminated as soon as the the DC receives the UDP from the CA about its GI. As shown in Figure 3.4, the CA propagates UDP to DCMs after the update.

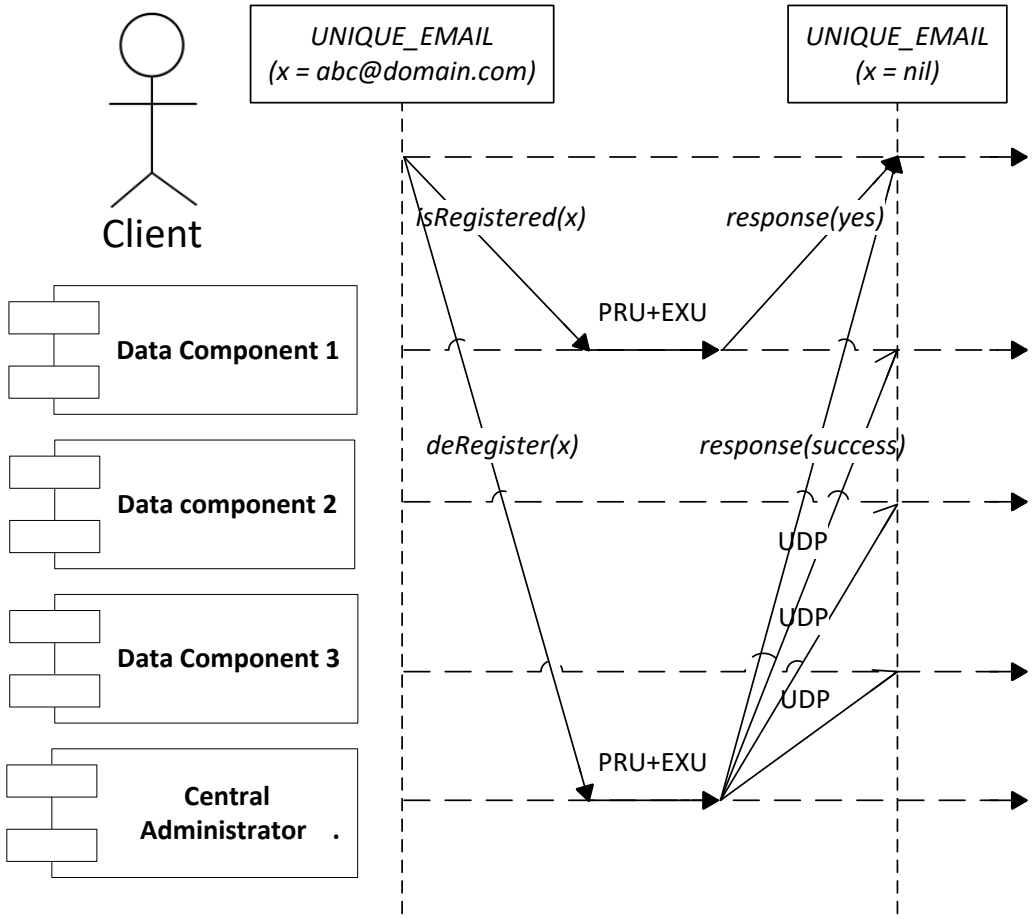


Figure 3.4: Push mechanism scenario. The data component responds to inquiry *isRegistered()*. The component administrator addresses the manipulate registration *deRegister()* request, and afterwards propagates asynchronously the update package to the data components.

3.4.3.3 Overheads

We identified are few overheads in CbHaRS. First of all, to select an CA, we need to do an election [Lam1998]. Secondly, after every GI-manipulative operations the CA sends an UDP to each DCM, and the cost of each GI-manipulative operation is equal to the number of DCMs, as shown in Figure 3.4. Lastly, when a DCM is in unsynchronized state, then the cost of RI-manipulative operation is higher, because it requires more DCMs to execute the request. However, the gains in operation availabilities and low operation costs are of more value than the associated overheads. If the consistency rules defined by the application can only be categorized as global invariants, then it is not recommended to use CbHaRS. Because in this case the operation cost will be same as of write operation cost of Read-One Write-All [BG1984]

data replication strategy. CbHaRS is best suited where we can extract more replica invariants than global invariants from the application-defined data consistency rules.

3.5 Conclusion & Future Work

Our prototype implementation of the CbHaRS has one CA and ten DCMs. We implemented CbHaRS in Java on a Linux Arch machine. We executed four test runs with $1000, 5000, 10000$ and 15000 transactions. Each transaction is composed of a read operation followed by a write operation. We implemented the *RESERVATIONS* process by the following two strategies: i) GI implementation by CA ii) RI implementation by DCMs. In case of the GI-implemented strategy, each DC coordinates with the CA for invariant validation. Here, we set $15ms$ for the GI validation-and-communication time between the CA and the DCMs. However, in case of the RI implemented strategy, the GI is distributed by the CA among the DCMs as RI. The DCMs – in parallel – perform independent local validations. The validation time is also set as $15ms$ for all the DCMs. The RI strategy – for this particular application scenario – exploits the RI, and allows simultaneous read-write and write-write operation executions. Refer to Figure 3.3 for details, where *reservations* are made simultaneously in compliance with the $ALLOCATED\_TICKETS$ RI. While, the GI strategy – for the same scenario – allows operation executions in a serial order. Because, the invariant validation is done via $AVAILABLE\_TICKETS$ GI.

In Figure 3.5, we present the time difference between both of the strategies to execute the same number and type of transactions. Invariant validations facilitates both of the strategies to focus towards stronger levels of data consistencies. Considering the outcome of the prototype, we foresee to achieve inspiring results in the future.

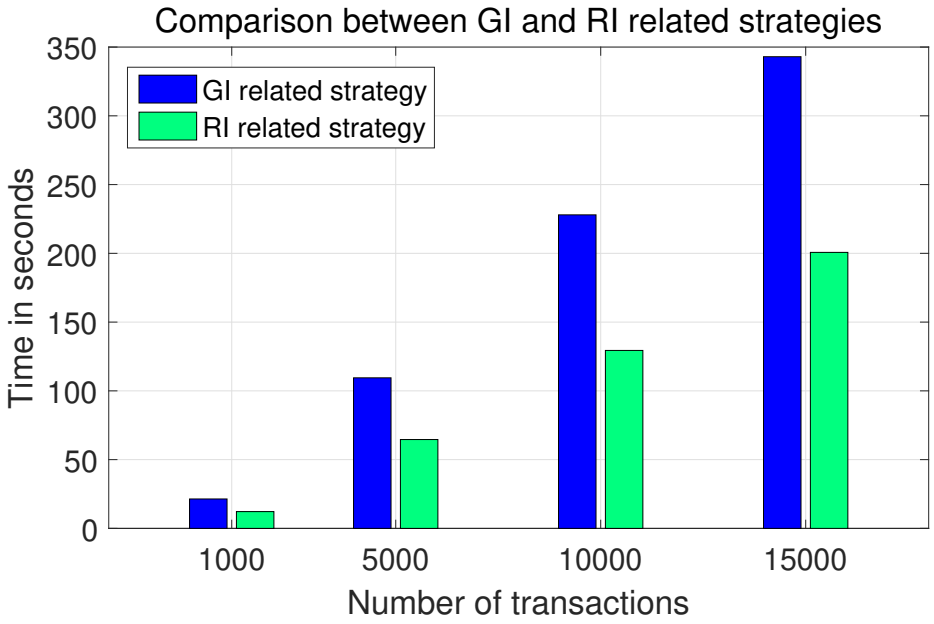


Figure 3.5: A comparison of the GI and the RI mechanisms. *Operations execution with GI* demonstrates the invariant enforced by the CA. *Operations execution with RI* demonstrates the invariant enforcement by the DCMs.

4

A Data Replication Strategy for A Replicated Services Registry

The Internet market is highly competitive and is influenced by high expectation levels of internet users, continuous advancement in the information technology, and high processing and storage capabilities of the hardware. In this chapter we focus on the design and development of a replicated and highly available service registry for microservice architectures. The service registry key-value store comprises of six nodes and supports a total of $\mathbf{2^{16}}$ microservices. Existing replicated service registries, like ZooKeeper [HKJR2010] and ETCD [ETC2017] are based on majority consensus strategies. If these strategies fail to achieve majority consensus, then they are bound to provide limited functionality. Neither a new service can be added not an existing service can be removed. Because, to change the cluster state a majority consensus is required. As part of this research, we propose a highly available data replication strategy for replicated service registry [UZT2018]. The a data replication strategy for a replicated service registry ($\mathcal{DRSR}$) exploits: i) a simple encoding scheme; and ii) a mapping method for efficient distribution of the encoded values to the service registry nodes.

4.1 Introduction

E-commerce, online booking, search engine, and social networking are some classes of highly available systems. Several hundreds or thousands of users simultaneously access these systems. For example, in November 2017, amazon.com had around 2.9 billion (B) visits, facebook.com had approximately 30.66B visits, youtube.com had nearly 24.47B visits, expedia.com had around 51.45 million (M) visits, booking.com had approximately 369.77M visits, and google.com had nearly 43.43B visits

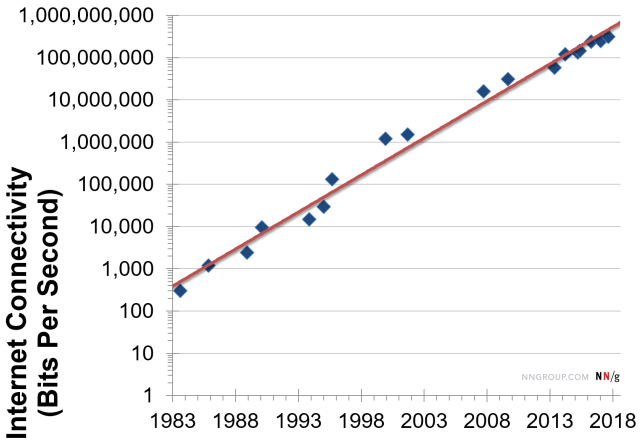


Figure 4.1: Internet connectivity bandwidth growth [NNG2019]

[Sim2017]. The internet market is highly competitive and it is influenced by high expectation levels of the internet users, continuous advancement in the information technology, and high processing and storage capabilities of the hardware [McA2010]. From Figure 4.1 we can see that there is a consistent increase in internet connectivity bandwidth. Same is the case with computing power. The steady increase in computing power can be noticed in Figure 4.2. We also experience similar kind of

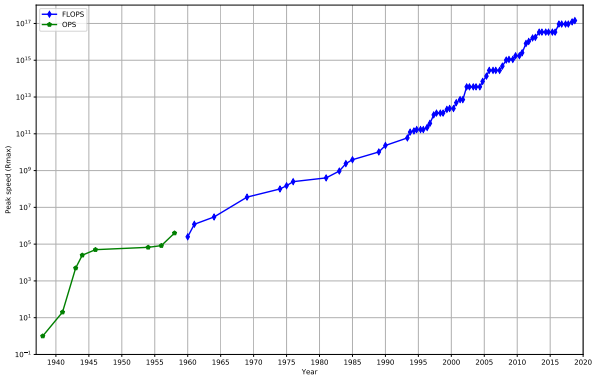


Figure 4.2: Supercomputer computing abilities trend [Wik2019]

tread about storage. The storage cost per 1 Terabyte of storage is decreasing day by day. Also the storage devices are becoming compact over time. There is a lot of innovation that results in such achievements as described in [BS2003]. Figure 4.3 provides an overview on storage cost over time.

Now we get an idea about technology advancements in the last couple of decades. These are tremendous. These advancements enabled us to rethink – how to exploit

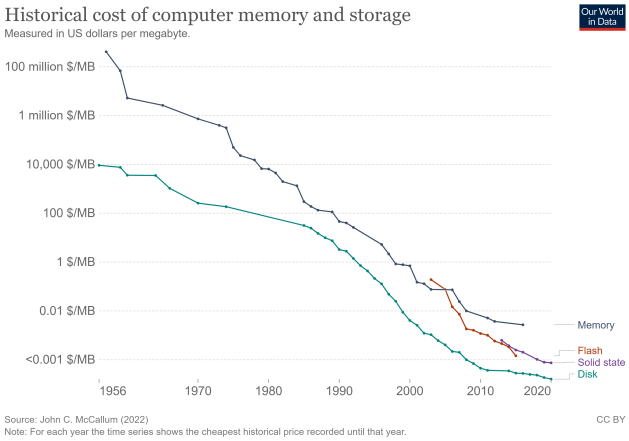


Figure 4.3: Hard drive storage-cost trend [Ove2022]

the technology to take maximum benefit – the way the systems can be designed. The internet capabilities also increased a lot as shown in Figure 4.1. We can now design wide-area network-based distributed systems with little or less worry about the network bandwidth and its unavailability. Amazon web services [Aws2017] is one of the good examples. However, to survive in such a highly competitive environment, these systems are primarily focusing on high operation availabilities [CDE+2013, DHJ+2007, BAC+2013].

In this work, we focus on high operation availabilities from the perspective of a microservice architecture (μSA) [DGL+2017]. We proposed a highly available service registry ($S\Re$) for a fixed number i of microservices (μSs), $0 \leq i \leq 2^{16}$. In the scope of this work, our encoding scheme supports a maximum of 2^{16} values. The μSA recommends to define μSs as granule independent *packages*, that can be deployed in isolation, equipped with their own persistence mechanism, and are able to communicate with each other via messages [DGL+2017]. A microservice (μS) is expected to perform a small set of closely related atomic operations [Str2003]. In case of distinct operations – the operations belonging to different modules which cannot be grouped together – a separate μS is defined for each of the operations. Taking the example of an e-commerce system, if there exists a $manage\_shopping\_cart$ μS, then it is expected to manage operations related to the shopping carts only. On the other hand, the analytical functionality to recommend products to a user based on her/his shopping behavior is provided by an $analyze\_and\_predict$ μS.

One of the important design time consideration for a μSA is *service discovery* (SD) [RIN+2017]. It is a process to find the instances of a μS in a distributed infrastructure. The distributed infrastructure can be logical or physical, and to provide high operation availabilities, it is expected to be fault-tolerant. Nowadays, it is possible to have a μSA based on a cloud platform [SRD2016, HKF2017]. In this case the cloud-services provider provides the infrastructure to design, deploy, and maintain the μSs. Some of the popular cloud-services providers are Amazon Web

Services [Aws2017], Azure [Azu2022], and Google cloud [Goo2022].

To utilize a μS, the client[1] should have the invocation-location information of the μS. Depending upon the system configuration and load balancing requirements, there can exist multiple instances of a μS. Furthermore, to incorporate fault tolerance, the μSs can be migrated from a problematic node to a functional node, thus resulting in a change of the system configuration. In these situations, all clients utilizing those μSs need to update their service invocation-location information. However, it is a tedious, error-prone, and inefficient process. To deal with this problem, a *service registry* ($S\Re$) is utilized [BHJ2016, YSS2016, MRSU2016, BCD$^+$2000].

According to [RIN$^+$2017], the $S\Re$ acts as a database of the μSs instances. μSs invocation-location information is saved in this database. In [BHJ2016], the authors viewed $S\Re$ as having the main supporting role in the μSAs. The client queries $S\Re$ to obtain the μS's data – in particular μS's invocation-location information. The client maintains information of the $S\Re$. Moreover, considering the importance of $S\Re$ in a μSA, the $S\Re$ must be highly available and fault-tolerant.

In this work, we primarily focus on designing a highly available and fault-tolerant $S\Re$. We achieve these properties by replicating the $S\Re$ on six independent nodes. In a replicated environment, the independent nodes work together to achieve the common goal [The1993]. A replicated $S\Re$ is more fault-tolerant as compared to a non-replicated centralized $S\Re$. Because, there exist n replicas of $S\Re$ and it can bear m faults, where $m < n, m, n \in \mathbb{N}$ and the value of m depends upon the data replication strategy. To support operation executions on the replicated $S\Re$, we design a data replication strategy which we call *a data replication strategy for a replicated service registry* ($\mathcal{DRSR}$). High operation availability is among the primary objectives of a replicated environment [BFF$^+$2014, BDF$^+$2013]. In Section 4.4, we explain how $\mathcal{DRSR}$ provides high operation availabilities. The key contributions of this chapter are:

- the development of a replicated and highly available service registry for a microservice architecture;
- the development of a simple and efficient encoding scheme for a finite number $i, 0 \leq i \leq 2^{16}$ of microservices; and
- a mapping method for efficient distribution of encoded values to service registry nodes.

For an overview of related work, refer to Section 4.2. In Section 4.3, we emphasize the importance of a highly available $S\Re$. We elaborate the $\mathcal{DRSR}$ in detail in Section 4.4. The functional model, implementation, analysis, and results of $\mathcal{DRSR}$ are discussed in Section 4.4.1, Section 4.4.2, Section 4.4.3, and Section 4.4.4, respectively. Finally, in Section 4.5, we conclude this part.

[1] Client can be, by itself, a μS or an application utilizing the μS.

4.2 Related Work

A μSA is an architectural type for developing applications. In this architecture type, a large problem is divided into sub-problems. For each sub-problem a μS is developed. However, the actual art lies in defining *small, granule,* and *independent* sub-problems. This definition is done in such a way that the μS *packages* – solutions to sub-problems along with the storage and persistence mechanisms – can be deployed and maintained independently [Ric2017, BWZ2017, DGL$^+$2017, New2015, MCL2017, FM2017]. The μSs tend to grow in large numbers due to their small and limited scope. The loose coupling at the granule level results in more complexity and increased communication among the packages. However, the achieved gains are much larger as compared to additional complexities [MRSU2016, BWZ2017, HKF2017].

A classic approach to keep track of μSs in a μSA is via a $S\Re$ [BHJ2016, YSS2016, MRSU2016, BCD$^+$2000]. A $S\Re$ is a repository of the μSs. Along with other important information, it contains invocation-location information of μSs [RIN$^+$2017]. Every functional – non-obsolete – μS publishes its existence by registering to the $S\Re$ [Str2003]. A μS can register itself to $S\Re$, or it can be the responsibility of some other service(s) to discover functional-unregistered μSs and then register those to $S\Re$. The same process is followed by a μS to cancel its registration. The client gets to know about functional μS via SD – it is a process of locating the μS by querying the $S\Re$ [Ric2017, RIN$^+$2017, SGT2011, WCD2017].

The important role of a $S\Re$ requires it to be highly available and fault-tolerant [MIB2009]. In the literature, there exists a number of techniques dealing with the importance, reliability, and fault tolerance of $S\Re$. For example, Netflix Eureka [Eur2017] based on AWS [Aws2017], ETCD [ETC2017], and ZooKeeper [HKJR2010]. Netflix Eureka 2.0 supports full replication. This is achieved by some broadcast mechanisms. However, Netflix Eureka requires continuous configuration and maintenance needs to manage the read-only and write-only clusters of the $S\Re$. ZooKeeper is well-known because of its reliability, high operation availabilities, and data consistency. It is a replicated system and is supported by consensus algorithms like majority consensus strategy [Tho1979]. ETCD is based on Raft [OO2014] – also, a consensus algorithm. Most of the consensus-based techniques, like ZooKeeper and ETCD, require a majority of nodes to progress. In case of state-machine-based replicated systems, the progress refers to a state transformation [KKW2013]. If a majority is not available, then there is no progress, but in some scenarios read-operation execution is supported. Another highly available data replication strategy is the trinagular lattice protocol [WB1992]. It depends upon logical arrangements of nodes in the form of a lattice. Depending on nodes arrangement, the operation executions require a combination – that is, a quorum – of adjacent nodes. For write operation, the trinagular lattice protocol requires a quorum containing node(s) from every column and every row. For read operation, it requires a quorum containing node(s) either from every column or from every row. Our approach deals with the problems related to majority consensus and logical arrangements of nodes.

In [SPSPMJP2006], Salas J. et al. presented a framework for highly available web

service (WS). The framework exploits full replication based on a reliable multi-cast approach for message delivery. However, this approach requires each WS instance of the group to know about all members of its replication group. This approach faces an overhead of reconfiguration and a lack of robustness in case of highly dynamic environments [RKUP2017]. However, to minimize the impacts of reconfiguration, our approach encapsulates only the information about $S\Re$ in the μSs and the client.

4.3 Microservices Architecture

In [Ric2017], the author proposed various μSAs. Based on these μSAs, we proposed a highly available $S\Re$. Before going into details, it is important to understand the role and importance of a highly available $S\Re$. Let us consider the example of a bookings management system. Such type of systems is designed to be highly available and primarily focuses on the operation availabilities aspect of the CAP theorem [GL2002]. According to the CAP theorem, there always exist trade-offs among operation availabilities, data consistency, and partition tolerance.

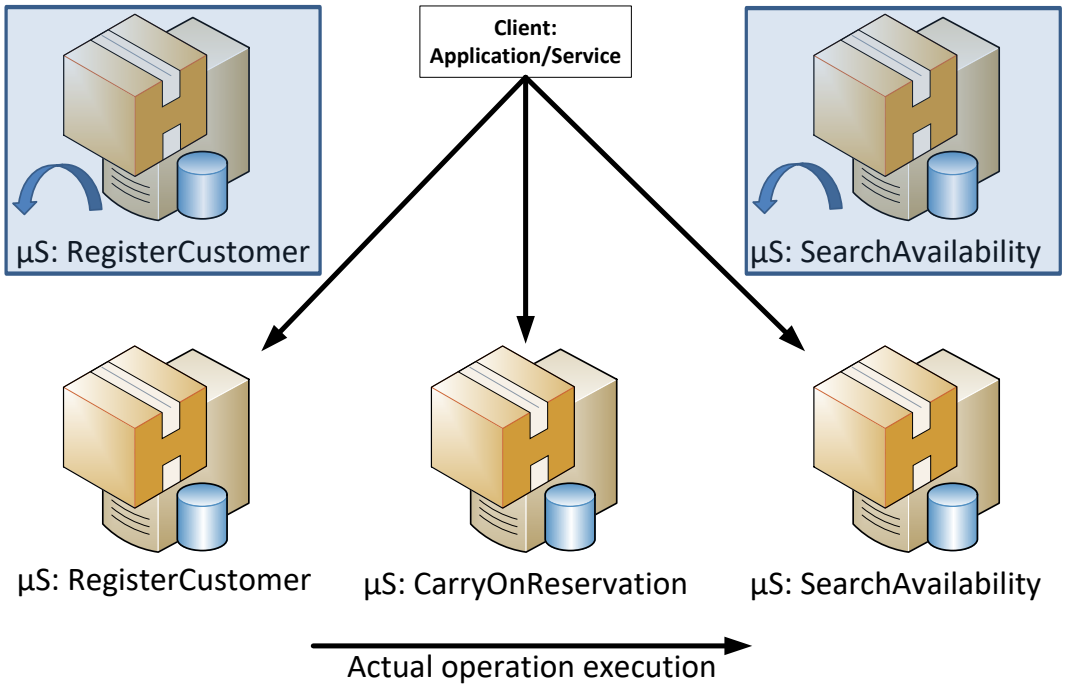


Figure 4.4: A μSA containing replicated μS instances. The μS in grey-coloured boxes are the new instances. The curved arrow means that the μS is not configured yet and necessary configuration of the client and other μSs is required to utilize the new μSs.

Figure 4.4 shows a simple μSA for a bookings management system example. There

exist three services, namely: i) a *RegisterCustomer* μS which is responsible to execute operations related to customer registration only; ii) a *CarryOnReservation* μS which is responsible to execute operations related to booking confirmations; and, iii) a *SearchAvailability* μS which is designed to execute query-operations as per customer preferences. There exists a client, which requires one or more μSs to complete an operation. As shown in Figure 4.5, to perform booking, the client first checks seat availability via *SearchAvailability* μS and, if the seat is available then the client will do the booking via *CarryOnReservation* μS. Otherwise, client get a seat unavailability response.

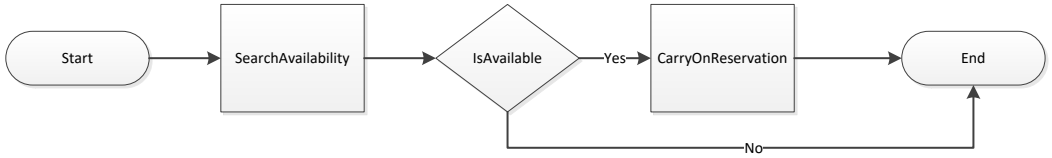


Figure 4.5: A simple reservation business process example.

4.3.1 μSA without a $S\Re$

In a simple scenario, there exists single instance of each μS. Only the client and the μS contain invocation-location information of each other. When a configuration change – the deployment of μS on another node, due to a node failure or maintenance – occurs, the client and the existing μS are updated with the latest invocation-location information of each other. For a few μS, such tight coupling is somewhat manageable. However, it gets complicated to deal with configuration changes with increased number or replicated instances of the μSs. As shown in Figure 4.4, to load-balance *SearchAvailability* and *RegisterCustomer* μSs, one more instance of each μS is brought into the system. At this point in time, the client and the existing μSs do not have any information of the two new μS instances. This change needs to be communicated to all components of the system.

4.3.2 μSA with a $S\Re$

Figure 4.6 shows a μSA including a $S\Re$. The $S\Re$ contains information, noticeably the invocation-location information, about all components of the system. When the components – μS and client – want to communicate with each other, they get the latest invocation-location information from the $S\Re$. This type of μSA reduces the complexity associated with configuration changes. On the other hand, it increases dependency on the $S\Re$. Having the motivation to address the configuration-related challenges, in the next section, we present a highly available replication strategy for a replicated $S\Re$.

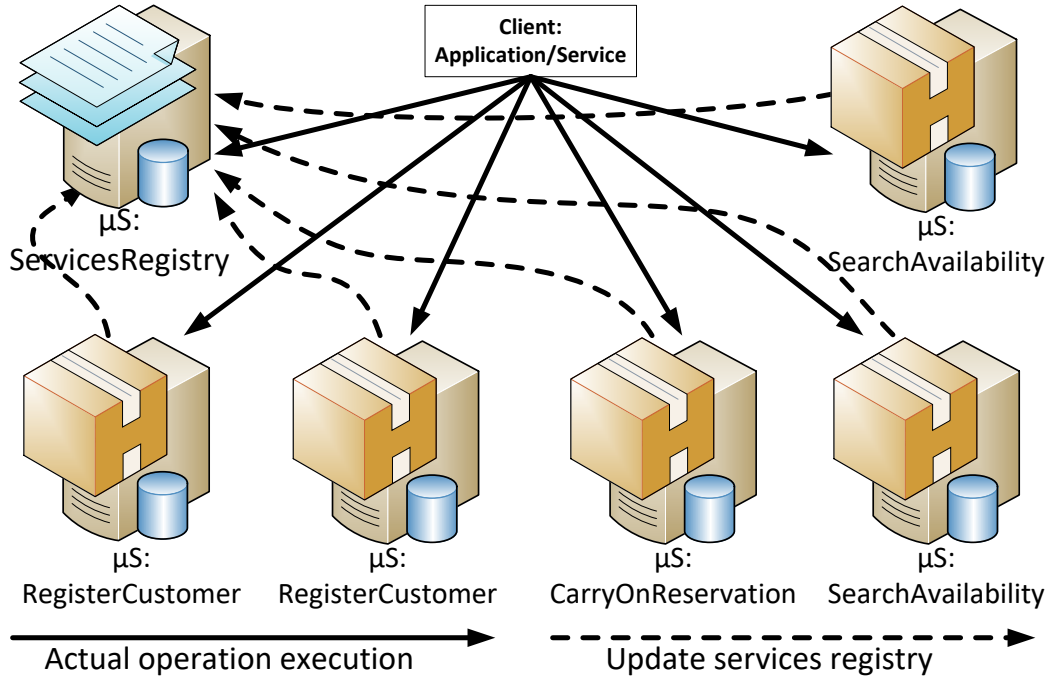


Figure 4.6: A μSA with multiple μSs and a $S\Re$. Addition and removal of μSs require configuration change at the service registry. The client and other μSs do not require any change, because they get information from service registry.

4.4 A Data replication strategy for a replicated service registry

This section elaborates the a data replication strategy for a replicated service registry ($\mathcal{DRSR}$) [UZT2018]. It is a highly available data replication strategy, and to achieve high operation availabilities, $\mathcal{DRSR}$ exploits the following:

- a simple coding scheme –named as Code16 (C16) – for transforming service registry key ($S\Re[S]$) into redundant codes;
- a $S\Re$ in the form of a key-value store with the capacity to address 2^{16} μSs; and
- a mapping method for efficient distribution of redundant codes to six $S\Re$ nodes.

4.4.1 The Functional Model

First of all, we explain the modeling foundations of $\mathcal{DRSR}$. This section is divided into three subsections: i) Subsection 4.4.1.1 discusses the model for Code16, ii)

Subsection 4.4.1.2 elaborates the service registry nodes model, and iii) Subsection 4.4.1.3 sheds light on the validation model of $\mathcal{DRSR}$.

4.4.1.1 The Code16 Model

$\mathcal{DRSR}$ maintains the $S\Re$ in the form of a key-value store.[2] The coding scheme transforms the input[3] service registry key ($S\Re[S]$) to a 16-bit binary code.

$$S := \{i \mid i \in \mathbb{N}, 0 \leq i \leq 2^{16}\} \tag{4.1}$$

For example, the 16-bit binary code of a μS having the $S = 26806_{10}$ is 0110100010110110_2. The binary code is then divided into eight equal parts starting from lower order bits to higher order bits. Each part is made up of two bits as shown in the following:

$$\overbrace{01}^{I_8}\ \overbrace{10}^{I_7}\ \overbrace{10}^{I_6}\ \overbrace{00}^{I_5}\ \overbrace{10}^{I_4}\ \overbrace{11}^{I_3}\ \overbrace{01}^{I_2}\ \overbrace{10}^{I_1}$$

The four possible distinct two bits binary codes are: $00, 01, 10$, and 11. Where $I_1, I_2, \ldots, I_8$ are their index positions. The binary code and their index positions have the following relationships:

$$\begin{aligned}
&I = \{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8\} \\
&A, B, C, D \subseteq I, A \cup B \cup C \cup D = I \\
&A = \{I_k \mid 00_2 = S \wedge (11_2(4^{k-1})), 1 \leq k \leq 8\} \\
&B = \{I_k \mid 01_2 = S \wedge (11_2(4^{k-1})), 1 \leq k \leq 8\} \\
&C = \{I_k \mid 10_2 = S \wedge (11_2(4^{k-1})), 1 \leq k \leq 8\} \\
&D = \{I_k \mid 11_2 = S \wedge (11_2(4^{k-1})), 1 \leq k \leq 8\}
\end{aligned} \tag{4.2}$$

The two bits binary codes and their indices are referred as *NIndices*. The NIndices are distributed among $S\Re$ nodes. The 16-bit binary code of $S = 26806_{10}$ in the form of NIndices, utilizing the Constraints 4.2, is represented as:

$$\begin{aligned}
&\mathbb{M} = \{A, B, C, D\} \\
&\text{NIndices} : S \mapsto \mathbb{M} \times \mathbb{M} \times \mathbb{M} \times \mathbb{M} \\
&\text{NIndices}(26806) = (A, B, C, D) \\
&= (\{I_5\}, \{I_2, I_8\}, \{I_1, I_4, I_6, I_7\}, \{I_3\})
\end{aligned} \tag{4.3}$$

In this section we explained, how the $S\Re[S]$ is transformed into NIndices. In the next section we elaborate on the mechanism to distribute the NIndices on the service registry nodes.

4.4.1.2 The S$\Re$ Model

The $\mathcal{DRSR}$ takes six nodes into account to form a highly available $S\Re$. As shown in

[2] A key-value store is a collection of keys along with their values. The keys are unique.
[3] The input represents a μS's key in the $S\Re$.

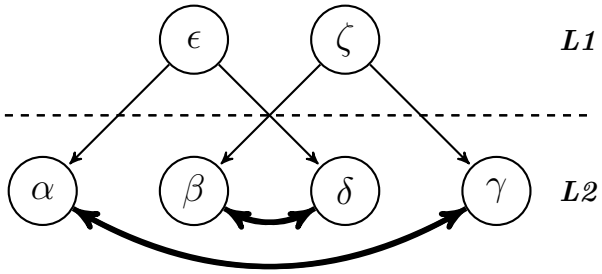


Figure 4.7: $\mathcal{DRSR}$ node model. Dotted line is used to separate $L1$ and $L2$ group. Thick arrows represent complete redundancy of NIs. Thin arrows show partial redundancy of NIs.

Figure 4.7, the $S\Re$ nodes are divided into two levels: L_1 and L_2. There are two nodes on L_1. They are denoted as ϵ and ζ. At level L_2, there are four nodes. They are denoted as α, β, γ, and δ. While updating $S\Re$, the $S\Re[S]$ is distributed in form of the NIndices to a subset of $S\Re$ nodes. At the time of reading from $S\Re$, the NIndices are read from a subset of $S\Re$ nodes to regenerate the $S\Re[S]$. A pair of NIndices is mapped to each $S\Re$ node. The NIndices mapping to nodes (f) is as follows:

$$\begin{aligned} &f : \mathbb{M} \times \mathbb{M} \mapsto \{\ \epsilon, \zeta, \alpha, \beta, \gamma, \delta\} \\ &f_{con} = \{(A, C, \epsilon), (B, D, \zeta), (A, D, \alpha), \\ &(B, C, \beta), (D, A, \gamma), (C, B, \delta)\} \end{aligned} \tag{4.4}$$

f is the function which relates NI and the nodes. The f_{con} explains the actual relationship between NIndices and the nodes. For example, $(A, C) \mapsto \epsilon$ and $(B, D) \mapsto \zeta$. For our node setting, as shown in Figure 4.7, this mapping fulfills our needs.

From the Equation 4.4, we can observe that a pair of the same NIndices – A, D – is mapped to α and γ. Another pair of the same NIndices – B, C – is mapped to β and δ. The reason to keep redundant NIndices is to enhance the fault tolerance capability of the $\mathcal{DRSR}$. Referring to Figure 4.7, the arrows between the nodes demonstrate the *redundancy relationship*. The thick arrows between the $S\Re$ nodes on L_2 show the complete redundancy of the NIndices. For example, node β and node δ have a thick arrow between them, because both of them have the same pair of NIndicess – B, C – mapped to them. However, the thin arrows directing from the $S\Re$ nodes from L_1 to L_2 highlight the partial redundancy of the NIndices. For instance, node ϵ and node α have one common NIndices – A – mapped to them.

4.4.1.3 The Validation Model

The $\mathcal{DRSR}$ validation model defines the correctness conditions for the two $S\Re$ operations: i) update service ($U_{S\Re}$) and ii) inquire service ($I_{S\Re}$) operations. The correctness conditions are derived from the following two definitions.

Definition 1. *The minimum number of unique nodes required by a* S$\Re$ *operation is called* enough nodes *(E_n).*

$$E_n \subset \{\epsilon, \zeta, \alpha, \beta, \gamma, \delta\}$$

Definition 2. *The nodes mapping to NIndices (f^{-1}), such that I is mapped to enough nodes is known as* complete mapping.

$$\begin{aligned} & f^{-1} : \{\ \epsilon, \zeta, \alpha, \beta, \gamma, \delta\} \mapsto \mathbb{M} \times \mathbb{M} \\ & f_{con}^{-1} = \{(\epsilon, A, C), (\zeta, B, D), (\alpha, A, D), \\ & (\beta, B, C), (\gamma, D, A), (\delta, C, B)\} \\ & \mathbb{G} = \{b \mid b \in \mathbb{F}_1 \cup \mathbb{F}_2 \wedge a \in E_n \\ & \wedge f_{con}^{-1}(a) = (\mathbb{F}_1, \mathbb{F}_2) \wedge \mathbb{F}_1, \mathbb{F}_2 \in \mathbb{M}\} \\ & (\mathbb{G} = \mathbb{M}) \equiv \text{complete mapping} \end{aligned} \tag{4.5}$$

As per Equation 4.4, the NIndices distributed redundantly among nodes. The NIndices are mapped to nodes in such a way that fewer nodes are required to complete an operation. Because of redundancy, we need a subset of nodes to collect all NIndices. f^{-1} is the function which explains the relationship between nodes and NIndices. However, f_{con}^{-1} is the function that explains which nodes contains which NIndices. For example, node ζ has NIndices B and D, and node δ has NIndices C and B. In Equation 4.5, $\mathbb{G}$ represents *complete mapping*. When we are able to map all the NIndices to a subset of nodes – known as enough nodes (E_n) – then we say we have complete mapping ($\mathbb{G}$).

The $U_{\text{S}\Re}$ operation is executed for the μS's registration and cancel-registration request. In general, the $U_{\text{S}\Re}$ is a write operation (WO). The $I_{\text{S}\Re}$ is executed on the $S\Re$ to determine the μS's invocation-location information. In general, it is a read operation (RO). The operation execution – for $U_{\text{S}\Re}$ and $I_{\text{S}\Re}$ – is correct, if the following holds:

- according to Equation 4.4, while an $U_{\text{S}\Re}$ operation execution, enough nodes are available to receive their mapped NIndices; and
- during an $I_{\text{S}\Re}$ operation execution, enough nodes are available and it is possible to extract complete mapping.

In Section 4.4.3, we discuss the operation execution in detail.

4.4.2 Replication Strategy

The replication strategy is based on $\mathcal{D}\Re\mathcal{S}\Re$'s functional model described in Section 4.4.1. This section elaborates on the $\mathcal{D}\Re\mathcal{S}\Re$ data replication strategy. It discusses in detail the mechanism used to replicate NIndices to $S\Re$ nodes. We explain the replication strategy from the perspective of $I_{\text{S}\Re}$ and $U_{\text{S}\Re}$ requests. The important

Algorithm 5: $\mathcal{DRSR}$ $I_{\mathrm{S}\Re}$ operation

```
   /* inquire service request                                    */
   Input  : S
   /* microservice's location or failure_message                 */
   Output: response
   /* extract NIndices of request                                */
 1 request ← extractServiceDetails(Input)
   /* determine availability scenario                            */
 2 availableNodes ← findNodes(request)
   /* valid scenario available                                   */
 3 if !enoughNodes(availableNodes) then
 4 |   response ← failure_message
 5 |   return response
 6 end
   /* determine minimum nodes                                    */
 7 tNodes ← determineNodes(availableNodes)
 8 NIndices ← readIndices(tNodes)
   /* generate code from NIndices                                */
 9 binary16 ← generateBinaryCode(NIndices)
   /* get service location information                           */
10 response ← getServiceLocationInfo(binary16)
11 return response
```

steps for $I_{S\Re}$ and $U_{S\Re}$ operations are explained in Algorithm 5 and Algorithm 6, respectively.

One of the important focus areas of a data replication strategy is its high operation availability. As per the CAP theorem [GL2002], there always exist trade-offs among operation availabilities, data consistency, and partition tolerance. In a replicated environment, the operation availability is mainly affected by the number of replicas required to execute an operation. The fewer the number of replicas required by an operation, the higher are the operation availability [UST2017, UZT2017a, ST2017, BDF$^+$2013, DHJ$^+$2007, AEA1992, Tho1979, Gif1979]. We also describe Algorithms 5 and 6 from the perspective of high operation availabilities. As explained in Section 4.4.1.1 that the $S\Re$ is a key-value store where the key is a unique microservice's key (S) and the value is its invocation-location information. In context of this work, $S\Re$ nodes are considered as replicas of $S\Re$ key-value store.

In Algorithm 5, we explain the $I_{S\Re}$ operation. It is a RO executed on the $S\Re$ for the discovery of μS's invocation-location. The first step is to extract S from the request. After that, a search among the $S\Re$ nodes is made to check their availability. If E_n are not available, then the $I_{S\Re}$ operation is not available and a $failure\_message$ is returned to the source. Otherwise, E_n are selected and the $I_{S\Re}$ operation execution extracts the NIndices which are later used to regenerate the 16-bit binary code of the S. Finally, $S\Re$ utilizes S to extract the μS's invocation-location information $S\Re[S]$.

In Algorithm 6, we explain the $U_{S\Re}$ operation. It is a WO executed on $S\Re$ to register a new μS or to delete the registration of an existing μS. The first step is to extract S from the request. The next step is to get the 16-bit binary code of S. After that, the binary code is used to generate NIndices. The next step is to run a search across the $S\Re$ to check the nodes' availability. If E_n nodes are available – in compliance with Equation 4.4, and a complete mapping exists – as per Definition 2, then the $U_{S\Re}$ operation is executed successfully along with the latest timestamp. Otherwise, the operation execution fails and a $failure\_message$ is returned to the source. The timestamp is unique for every $U_{S\Re}$ operation and is used to find out the latest written value.

4.4.3 Analysis

We unveil the high operation availability of the $\mathcal{D}\Re\mathcal{S}\Re$ by analytically evaluating it and then comparing it with state-of-the-art data replication strategies. We also describe, how Code16 model 4.4.1.1 and $S\Re$ model 4.4.1.2 play their important roles to achieve high operation availabilities, and how the validation model 4.4.1.3 helps to ensure data consistency. There are four possible availability scenarios. Each of the scenarios refers to μS's example from Section 4.4.1.

Algorithm 6: $\mathcal{DRSR}$ $U_{\mathrm{S}\Re}$ operation

```
   /* update service request                                          */
   Input  : S
   /* success_message or failure_message                              */
   Output: response
   /* extract NIndices of request                                    */
 1 request ← extractServiceDetails(Input)
   /* generate 16-bit binary serviceID                               */
 2 binary16 ← extractBinaryCode(request)
   /* extract indices for 00_2, 01_2, 10_2, 11_2                     */
 3 NIndices ← extractIndices(binary16)
   /* check if nodes are available                                   */
 4 avaiablelNodes ← findNodes(NIndices)
 5 if !enoughNodes(availableNodes) then
 6 |   response ← failure_message
 7 |   return response
 8 end
   /* get nodes w.r.t.  indices and nodes' availability              */
 9 tNodes ← determineNodes(availableNodes)
10 timstamp ← generateTimestamp(request)
   /* replicate indices with timestamp                               */
11 if distributeIndices(tNodes,timestamp) then
12 |   response ← success_message
13 else
14 |   response ← failure_message
15 end
16 return response
```

4.4.3.1 Availability scenario 1: When the complete $L1$ group is available

NIndices for μS having S equal to 26806 are presented by Equation 4.3. According to Equation 4.4, node ϵ has the NIndices A and C, therefore, has the following mapping.

$$f_{con}^{-1}(\epsilon) = (A, C) = (\{I_5\}, \{I_1, I_4, I_6, I_7\})$$

Furthermore, the NIndices for node ζ are B and D, hence, has the given below mapping.

$$f_{con}^{-1}(\zeta) = (B, D) = (\{I_2, I_8\}, \{I_3\})$$

The nodes ϵ and ζ are enough nodes, and according to Definition 2, they represent a complete mapping of NIndices.

$$\mathbb{G} = \{A, C, B, D\} = \{\{I_5\}, \{I_1, I_4, I_6, I_7\}, \{I_2, I_8\}, \{I_3\}\} = \mathbb{M}$$

When the nodes ϵ and ζ are available, then the $U_{S\Re}$ and $I_{S\Re}$ operations for all the possible values of S are available. Therefore, the scenario is in compliance with $\mathcal{D}\Re\mathcal{S}\Re$ validation model.

4.4.3.2 Availability scenario 2: When ϵ and two nodes from the $L2$ group are available

In this scenario, in the $L1$ group, node ϵ is available but node ζ is not available. However, in the $L2$ group, two or more nodes are available. To comply with the $\mathcal{D}\Re\mathcal{S}\Re$ validation model, E_n are required to have a complete mapping. For example, some of the possible cases are $\{\epsilon, \alpha, \delta\}$, $\{\epsilon, \beta, \gamma\}$, and $\{\epsilon, \alpha, \beta\}$. Let us examine the first case – $\{\epsilon, \alpha, \delta\}$. As per Equation 4.4, NIndices for the node ϵ are A and C, hence, has the following mapping:

$$f_{con}^{-1}(\epsilon) = (A, C) = (\{I_5\}, \{I_1, I_4, I_6, I_7\})$$

The NIndices for the node α are A and D. Hence, it has the given below mapping:

$$f_{con}^{-1}(\alpha) = (A, D) = (\{I_5\}, \{I_3\})$$

Lastly, the NIndices for the node δ are C and B. It, therefore, has the following mapping:

$$f_{con}^{-1}(\delta) = (C, B) = (\{I_1, I_4, I_6, I_7\}, \{I_2, I_8\})$$

As per the Definition 1 and Definition 2, the nodes ϵ, α, and δ are enough nodes and they have a complete mapping, respectively.

$$\mathbb{G} = \{A, C, D, B\} = \{\{I_5\}, \{I_1, I_4, I_6, I_7\}, \{I_3\}, \{I_2, I_8\}\} = \mathbb{M}$$

Thus, this case also complies with $\mathcal{D}\Re\mathcal{S}\Re$ validation model.

4.4.3.3 Availability scenario 3: When ζ and two nodes from the $L2$ group are available

In this scenario, in the $L1$ group, node ζ is available but node ϵ is not available. However, in the $L2$ group, two or more nodes are available. As described in Section 4.4.3.2, if there exist enough nodes, and they have a complete mapping, then the scenario is in compliance with the $\mathcal{DRSR}$ validation model. For example, some of the possible cases of E_n having a complete mapping are $\{\zeta, \delta, \gamma\}$, $\{\zeta, \beta, \gamma\}$, and $\{\zeta, \alpha, \delta\}$.

4.4.3.4 Availability scenario 4: When the $L1$ group is not available

In this scenario, only the nodes in $L2$ are available. To comply with the $\mathcal{DRSR}$ validation model, three out of four nodes in $L2$ are required. And, if three nodes are available, then they are enough nodes and have a complete mapping. For example, some of the possible cases of E_n are $\{\alpha, \beta, \gamma\}$, $\{\gamma, \beta, \delta\}$, and $\{\alpha, \gamma, \delta\}$. Let us evaluate one of the cases – $\{\gamma, \beta, \delta\}$. According to Equation 4.4, NIndices for the node γ are D and A. This results in the following mapping:

$$f_{con}^{-1}(\gamma) = (D, A) = (\{(I_3)\}, \{I_5\})$$

The NIndices for the node β are B and C. Therefore, it has the given below mapping:

$$f_{con}^{-1}(\beta) = (B, C) = (\{I_2, I_8\}, \{I_1, I_4, I_6, I_7\})$$

Finally, the NIndices for the node δ are C and B, and, it results in the following mapping:

$$f_{con}^{-1}(\delta) = (C, B) = (\{I_1, I_4, I_6, I_7\}, \{I_2, I_8\})$$

As per the Definition 1 and Definition 2, the nodes γ, β and δ are enough nodes and they have a complete mapping, respectively.

$$\mathbb{G} = \{D, A, B, C\} = \{\{I_3\}, \{I_5\}, \{I_2, I_8\}, \{I_1, I_4, I_6, I_7\}\} = \mathbb{M}$$

4.4.4 Analytical Results

In Sections 3.2 and 4.3, we motivated the need of having a highly available service registry and the important role that it plays in a microservice architecture, respectively. In this section, we present the results of our work. We compare the $\mathcal{DRSR}$ with baseline data replication strategies: i) the majority consensus strategy (MCS) [Tho1979] and ii) the trinagular lattice protocol (TLP) [WB1992]. For the analysis, we consider *operation availabilities* and *operation cost* as the comparison characteristics. We calculate the access cost by the method proposed in [ST2017]. Refer to [Koc1994] for details on the method to calculate the operation availabilities. Furthermore, our assumptions for the analysis are six independent nodes, a fully connected

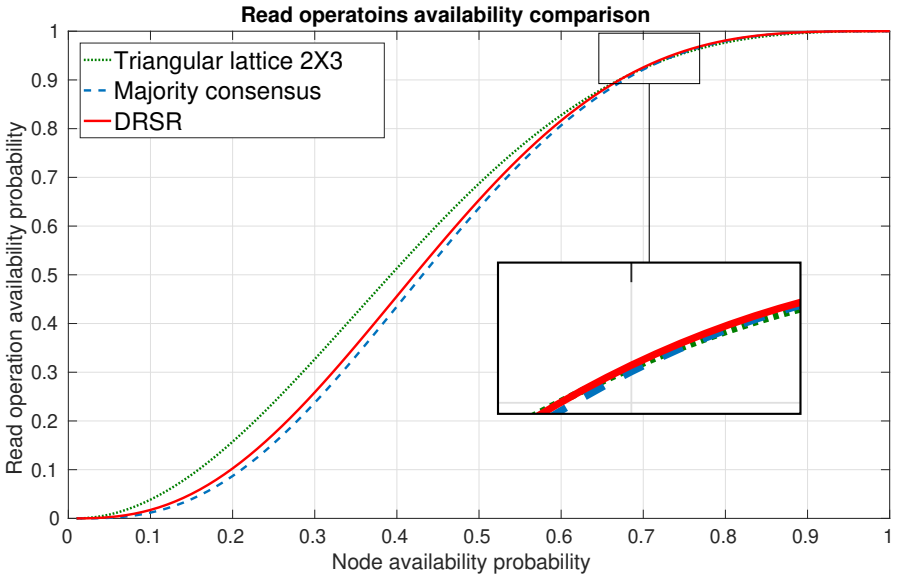


Figure 4.8: Read operation availabilities

network, and a fixed number i of microservices (μSs), $0 \leq i < 2^{16}$. The TLP, MCS, and $\mathcal{DRSR}$ are analyzed on the same set of assumptions.

We present in Figure 4.8 and Figure 4.9 the operation availability comparison results. Figure 4.8 provides our analysis results for read operation ($I_{S\Re}$) availabilities. We can observe that, for a node-availability of p between $0 < p < 0.6$, the TLP has a higher read operation availability than MCS and $\mathcal{DRSR}$. However, for p between $0.7 \leq p < 1.0$, the $I_{S\Re}$ availabilities for the three data replication strategies, at a higher level, are the same. When we look closely then, we see that $\mathcal{DRSR}$ availability is higher than the others. Figure 4.9 elaborates our analysis results with respect to the write operation ($U_{S\Re}$) availabilities. In this case, $\mathcal{DRSR}$ provides higher write operation availabilities than MCS and TLP.

We present operation cost comparison results in Figure 4.10 and Figure 4.11. Figure 4.10 provides our analysis results with respect to the read costs. For six nodes, we can observe that MCS has the highest access cost, that is 3, for all values of p. The read cost for $\mathcal{DRSR}$ is between 2 and 3. TLP has the minimum read cost around 2. Figure 4.11 provides our analysis results with respect to the write costs. For six nodes, we can observe that MCS has the highest access cost , that is 4, for all values of p. The write cost for TLP is between 3 and 4. $\mathcal{DRSR}$ has the minimum write cost between 2 and 3.

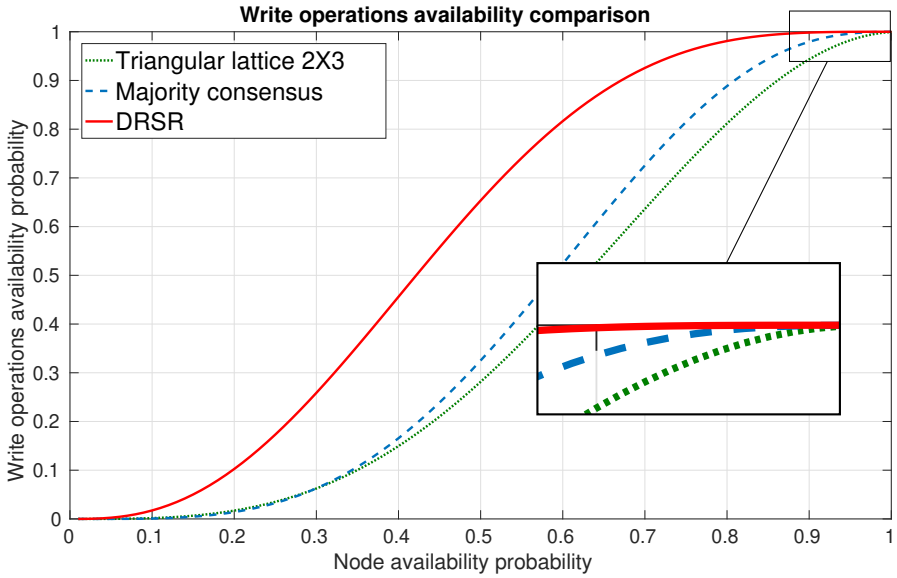


Figure 4.9: Write-operation availabilities

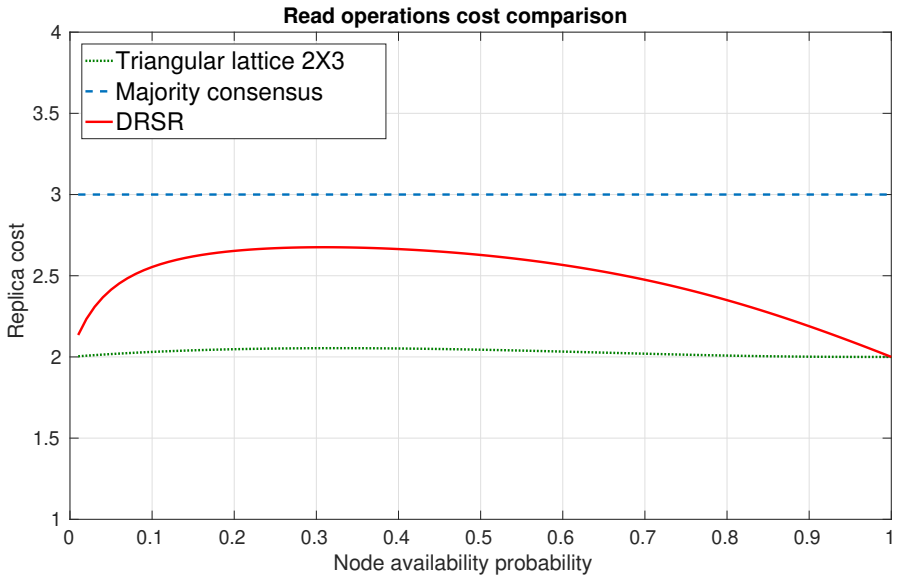


Figure 4.10: $\mathcal{DRSR}$ read operation operation cost comparision

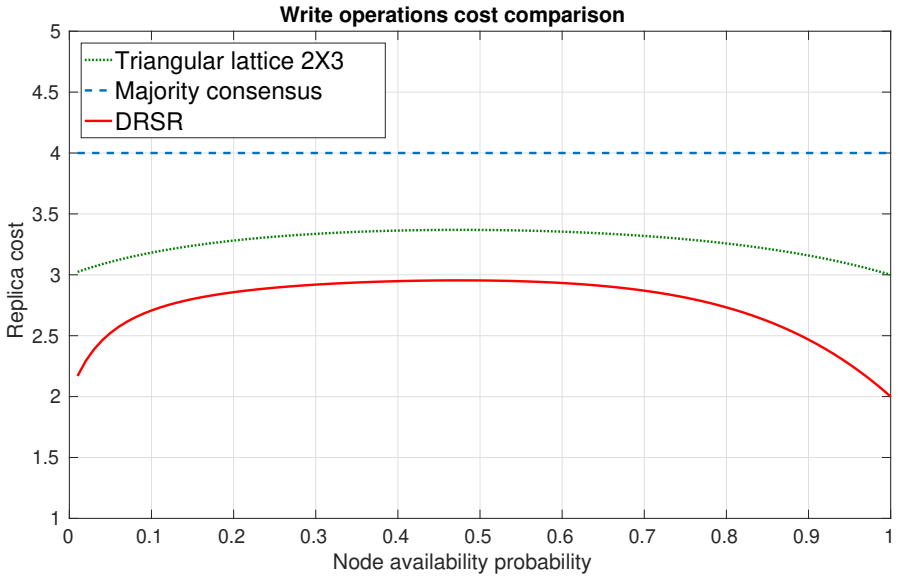


Figure 4.11: $\mathcal{DRSR}$ write operation operation cost comparision

4.5 Conclusion & Future Work

As we have discussed in Section 4.2, the service registries like [HKJR2010] and [ETC2017] are developed on top of a majority consensus algorithm. If a majority of the nodes is not available, then the system cannot progress, i.e., a $U_{S\Re}$ is not available. But, in some scenarios, $I_{S\Re}$ can still be performed. Similarly, for the TLP [WB1992], the $U_{S\Re}$ is affected by the availability of complete rows or columns – depending upon the lattice configurations. In this work, we presented $\mathcal{DRSR}$ – a highly available and fault-tolerant data replication strategy for $S\Re$ in a μSA. It exploits a simple encoding scheme, called Code16, which transforms microservice's key into NIndices. The $\mathcal{DRSR}$ then replicates these NIndices onto six $S\Re$ nodes with the help of an efficient NIndices mapping to nodes method. Our analytical results showed that with a slightly higher costs, the $\mathcal{DRSR}$ achieves competitive operation availabilities for $I_{S\Re}$. While, for $U_{S\Re}$, our strategy bears minimum cost and achieves higher operation availabilities. Also notably, $\mathcal{DRSR}$ does not require a nodes majority nor adjacent nodes availability. For our future work, we will enhance $\mathcal{DRSR}$ to adopt multiple encoding schemes. The plan is come up with a replication strategy which will distribute the μSs keys into equal sized buckets. The sorting of μSs keys will be done on the basis of semantic similarity. A variable number of replicas and a particular encoding scheme will be linked to each of the buckets. We are hopeful, that by this bucketing strategy, we will be able to achieve even higher operation availabilities at lower operation costs.

5 Outlook

Our work emphasized on the techniques to develop highly available data replication strategies with low operation costs. We took into account a distinctive set of features, such as predictability, data semantic, encoding, application semantic, and operation types, to design highly available data replication strategies. Most of the baselined data replication strategies, like Read-One Write-All [BG1984], majority consensus strategy [Tho1979], and trinagular lattice protocol [WB1992] depends upon read quorums and write quorums. They exploit the quorums intersection property to ensure strong data consistency. We discussed, how we can utilize encoding along with redundancy to provide high operation availabilities. Also, the intention was to ensure data consistency and low operation costs. Having this motivation, we presented three data replication strategies namely: Semantic data replication (SDR)[UST2017], the component-based highly available replication strategy (CbHaRS) [UZT2017a], and a data replication strategy for a replicated service registry ($\mathcal{DRSR}$) [UZT2018]. These data replication strategies followed different methodologies but the primary objective behind them is the same – provide high operation availabilities with low operation costs.

SDR is a novel highly available semantic data replication strategy which is empowered by coding techniques, a finite state space based on ASCII decimal codes, apriori knowledge, semantic classes, and codes to replica mapping. We elaborated in detail, how the input value is transformed using the ASCII decimal codes and, how we established a finite state space based on these codes. We defined two semantic properties. These properties were used to classify the ASCII decimal code digits into semantic classes. We discussed how to transform the 4-bit binary code of each of the ASCII decimal codes digits into two 2-bit binary codes. Afterwards, we explained the mechanism to map these 2-bit binary codes to the code replicas. In SDR, we focused primarily to achieve high operation availabilities for write operations. We showed

that the finite state space and semantic classes guide the write operation – based on the input value – to the exact replicas to write. This is in contrast to syntactic data replication strategies, where every write operation has to look among the complete population to lock the resources by running different replica searching and selection algorithms. Our results show that for multiple of six replicas, we achieve a better write availability then the baselined syntactic data replication strategies. For the read operation availability, our approach stood second best in the comparison. SDR is best suited for write-intense application types. In these application types activity logging is important for example blockchain and financial services.

Currently, SDR coding technique utilizes only ASCII decimal codes. We will explore more on the coding techniques as a next step towards SDR improvements. Furthermore, we will investigate options to utilize the semantic properties of the underlying data structure, specifically hash table-like data structure. In this work, we primarily focused on the semantic aspect of the data. We will also concentrate to come up with more semantic properties about the data and the application domain. For SDR, we presented a generalized approach irrespective of any application domain. We did not considered the data consistency and high operation availability requirements of a specific application. As part of our future work, we will research on the mechanism to link application-specific requirements to the design considerations of data replication strategies. We will explore on, how we can enhance SDR capabilities to incorporate application-specific requirements.

CbHaRS is our first attempt to design a data replication strategy which is influenced by application-specific requirements. CbHaRS ensures data consistency with the help of rules – these rules are derived from the application-specific requirements. The prototype implementation of the CbHaRS has one component administrator (CA) and ten data components (DCMs). Component-level data consistency is ensured by the replica invariant (RI) constraints defined for each DCM. Application-level data consistency is ensured by the global invariant (GI) constraints defined for the CA. We executed multiple test runs with varying number of transactions. Each transaction is composed of a read operation followed by a write operation. We tested the *RESERVATIONS* process with two data replication strategies – one with GI and RI mechanism, and the other with a GI-only mechanism. In first case, the GI was distributed by the CA among the DCMs as RI. The data consistency was ensured by implementation of threshold mechanism by each replica. We observed that, the replicas can make a decision about some of operation executions itself due to presence of RI. It omits the need to form a quorums for some operations. Because of that, different operations were executed in parallel by different replicas until the threshold was reached. After that, the operation executions were carried out with the help of component administrator – which enforces global invariant to ensure data consistency. On the other hand, for GI-only mechanism, each DC coordinates with the CA to ensure data consistency. Here we observed, that the GI-only strategy allows operation executions in a serial order.

Invariant validations facilitates both types of the strategy to focus towards stronger levels of data consistencies. By delegating the data consistency requirements in

the form of RI constraints to DCs, we can achieve high operation availabilities as compared to quorums based data replication strategies. DCMs coordinates with each other and CA for a limited set of scenarios. These scenarios are not general and may vary from application to application. Considering the outcome, we foresee to achieve inspiring results in the future. For our future work – on a larger scale, we will implement CbHaRS on top of TPC-C benchmark, and will utilize Yahoo Cloud Serving Benchmark (YCSB) for the performance evaluation.

Some online business applications – mostly from the E-commerce domain and cloud services providers – exploit a microservice architecture (μSA) [Ric2017] to provide high operation availabilities. μSA is designed on top of microservices (μSs). A μS is a granule-functional unit which can execute its tasks on its own, and can be deployed independent of other μSs. However, the implementation of a business process may require multiple μSs. The number of μSs tends to grow large for complex systems. A service registry ($S\Re$) is often used to keep track of all the μSs. Whenever a μS wants to communicate with another μS, it gets the required information about the other μS from the $S\Re$. A $S\Re$ introduces simplicity and liberty to design a highly available μSA but it increases the importance of its availability. The failure of the $S\Re$ may results in operation unavailability, which is a big risk. The service registries like [ETC2017] and [HKJR2010] are developed on top of a majority consensus algorithm. If a majority of the nodes is not available, then the system cannot progress, i.e., a write operation is not available. But, in some scenarios, read operation can still be performed. In Chapter 4, we presented $\mathcal{D}\Re\mathcal{S}\Re$ – a highly available and fault-tolerant data replication strategy for $S\Re$ in a μSA. It exploits a simple encoding scheme, called as Code16, which transforms a microservice's key into NIndices. The $\mathcal{D}\Re\mathcal{S}\Re$ then replicates these NIndices onto six $S\Re$ nodes with the help of an efficient f-mapping-to-nodes method. Our analytical results showed that with a slightly higher cost, the $\mathcal{D}\Re\mathcal{S}\Re$ achieves competitive operation availabilities for the read operation. However, for the write operation, our strategy bears minimum cost, and achieves higher operation availabilities. Also notably, $\mathcal{D}\Re\mathcal{S}\Re$ does not require nodes majority and adjacent nodes availability. For our future work, we will enhance $\mathcal{D}\Re\mathcal{S}\Re$ to adopt multiple encoding schemes. The plan is come up with a replication strategy which will distribute the μSs keys into equal sized buckets. The sorting of μSs keys will be done on the basis of semantic similarity. A variable number of replicas and a particular encoding scheme will be linked to each of the buckets. We are hopeful, that by this bucketing strategy, we will be able to achieve higher operation availabilities at lower operation costs.

Semantic data replication is an interesting research area. There is more complexity inherent in it as compared with syntactic data replication. That is the reason there are fewer research efforts in the area of semantic data replication as compared to syntactic data replication. However, having the belief in and to highlight the potential of semantic data replication, we decided to explore this area. We demonstrated that high operation availabilities with reduced operation costs can be achieved by incorporating the concepts like encoding schemes, application specific rules, and selection of replicas based on input values.

Publications

a) A. Usman, R. Schadek and O. Theel. A Novel Highly Available Data Replication Strategy exploiting Data Semantics, Coding Techniques and Prior At-Hand Knowledge. In *Proceedings of the 22nd IEEE Pacific Rim International Symposium on Dependable Computing*, pages 301–310, Christchurch, New Zealand, January 2017. IEEE

b) A. Usman, P. Zhang and O. Theel. A Component-Based Highly Available Data Replication Strategy Exploiting Operation Types and Hybrid Communication Mechanisms. In *Proceedings of the 14th IEEE International Conference on Services Computing*, pages 495–498, Honolulu, HI, USA, June 2017. IEEE

c) A. Usman, P. Zhang and O. Theel. An Efficient and Updatable Item-to-item Frequency Matrix for Frequent Itemset Generation. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, pages 85:1–85:6, New York, NY, USA, 2017. ACM

d) A. Usman, P. Zhang and O. Theel. A Highly Available Replicated Service Registry for Service Discovery in a Highly Dynamic Deployment Infrastructure. In *Proceedings of the 15th IEEE International Conference on Services Computing*, pages 265–268, San Francisco, CA, USA, July 2018. IEEE

Bibliography

[AA1990] Divyakant Agrawal and Amr El Abbadi. The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB '90, pages 243–254, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[Aba2012] Daniel Abadi. Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *IEEE Computer Magazine*, 45(2):37–42, February 2012.

[ABCH2013] Peter Alvaro, Peter Bailis, Neil Conway and Joseph M. Hellerstein. Consistency Without Borders. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 23:1–23:10, New York, NY, USA, 2013. ACM.

[AEA1992] Divyakant Agrawal and Amr El Abbadi. The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. *ACM Transactions on Database Systems (TODS)*, 17(4):689–717, December 1992.

[ANS1986] X3 ANSI. 4: Coded character set–7-bit american national standard code for information interchange. *Am. Nat'l Standards Inst., New York*, 1986.

[Aws2017] Aws. AWS. `https://aws.amazon.com/`, 2017. [accessed 13-December-2017].

[Azu2022] Azure. Azure. `https://azure.microsoft.com/en-us/`, 2022. [accessed 12-September-2022].

[BAC+2013] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song and Venkat Venkataramani. TAO: Facebook's Distributed Data Store for the Social Graph. In *Proceedings of the USENIX Annual Technical Conference*, pages 49–60, San Jose, CA, 2013. USENIX.

[BCD+2000] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of the Ninth International Symposium on High-Performance Distributed Computing*, pages 51–59, Pittsburgh, PA, USA, 2000. IEEE.

[BCvR2009] Ken Birman, Gregory Chockler and Robbert van Renesse. Toward a Cloud Computing Research Agenda. *SIGACT News*, 40(2):68–80, June 2009.

[BDF+2013] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein and Ion Stoica. Highly Available Transactions: Virtues and Limitations. *Proc. VLDB Endow.*, 7(3):181–192, November 2013.

[BDF+2015] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh and Marc Shapiro. Putting Consistency Back into Eventual Consistency. In *Proceedings of the 10th European Conference on Computer Systems*, EuroSys '15, pages 6:1–6:16, New York, NY, USA, 2015. ACM.

[BFF+2014] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein and Ion Stoica. Coordination Avoidance in Database Systems. *Proc. VLDB Endow.*, 8(3):185–196, November 2014.

[BG1984] Philip A. Bernstein and Nathan Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Database Systems (TODS)*, 9(4):596–615, December 1984.

[BHG1987] Philip A. Bernstein, Vassco Hadzilacos and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[BHJ2016] Armin Balalaie, Abbas Heydarnoori and Pooyan Jamshidi. *Migrating to Cloud-Native Architectures UsingÂ Microservices: An Experience Report*, pages 201–215. Springer International Publishing, Cham, 2016.

[BS2003] Richard Bradshaw and Carl Schroeder. Fifty years of ibm innovation with information storage on magnetic tape. *IBM Journal of Research and Development*, 47(4):373–383, 2003.

[BWZ2017] Justus Bogner, Stefan Wagner and Alfred Zimmermann. Automatically Measuring the Maintainability of Service- and Microservice-

based Systems: A Literature Review. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pages 107–115, New York, NY, USA, 2017. ACM.

[CAA1992] Shun Yan Cheung, Mostafa H Ammar and Mustaque Ahamad. The grid protocol: a high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):582–592, Dec 1992.

[CDE+2013] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang and Dale Woodford. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013.

[CDK2005] George F Coulouris, Jean Dollimore and Tim Kindberg. *Distributed systems: concepts and design*. Pearson Education, 2005.

[DGL+2017] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham, 2017.

[DHJ+2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[DPS+1997] Alan J Demers, Karin Petersen, Michael J Spreitzer, Douglas B Terry, Marvin M Theimer and Brent B Welch. Application-specific conflict resolution for weakly consistent replicated databases, February 11 1997. US Patent 5,603,026.

[ETC2017] Team ETCD. ETCD. `https://coreos.com/etcd/`, 2017. [accessed 13-December-2017].

[Eur2017] Team Netflix Eureka. Netflix Eureka. `https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance`, 2017. [accessed 13-December-2017].

[FM2017] C. Y. Fan and S. P. Ma. Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. In *Proceedings of the IEEE International Conference on AI Mobile Services*, pages 109–112, Honolulu, HI, USA, June 2017. IEEE.

[GDN+2003] Lei Gao, Mike Dahlin, Amol Nayate, Jiandan Zheng and Arun Iyengar. Application Specific Data Replication for Edge Services. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 449–460, New York, NY, USA, 2003. ACM.

[Gif1979] David K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM.

[GIM2016] Marco Gribaudo, Mauro Iacono and Daniele Manini. Improving Reliability and Performances in Large Scale Distributed Applications with Erasure Codes and Replication. *Future Generation Computer Systems*, 56:773–782, March 2016.

[GL2002] Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, 33(2):51–59, June 2002.

[Goo2022] Google. Google Cloud. `https://cloud.google.com/`, 2022. [accessed 12-September-2022].

[Her1984] Maurice Herlihy. General quorum consensus: a replication method for abstract data types. 01 1984.

[Her1985] Maurice Herlihy. Atomicity vs. Availability: Concurrency Control for Replicated Data. 1985.

[Her1986] Maurice Herlihy. A Quorum-consensus Replication Method for Abstract Data Types. *ACM Transactions on Computer Systems (TOCS)*, 4(1):32–53, February 1986.

[HHB1996] Abdelsalam A. Helal, Abdelsalam A. Heddaya and Bharat B Bhargava. *Replication Techniques in Distributed Systems*, volume 4 of *Advances in Database Systems*. Springer US, 1996.

[His1989] Hisgen, Andy and Birrell, Andrew and Mann, Timothy and Schroeder, Michael and Swart, Garret. Availability and Consistency Tradeoffs in the Echo Distributed File System. In *Proceedings of the Second Workshop on Workstation Operating Systems*, pages 49–54, Pacific Grove, CA, USA, 1989. IEEE.

[HKF2017] A. Habl, O. Kipouridis and J. Fottner. Deploying Microservices for a Cloud-based Design of System-of-Systems in Intralogistics. In *Proceedings of 15th IEEE International Conference on Industrial Informatics*, pages 861–866. IEEE, July 2017.

[HKJR2010] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira and Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the 2010 USENIX Annual Technical Conference*, USENIXATC'10, Berkeley, CA, USA, 2010. USENIX Association.

[HR1983] Theo Haerder and Andreas Reuter. Principles of Transaction-oriented Database Recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.

[Iak2012] K Kiss Iakab. *Probabilistic Quorum Systems for Dependable Distributed Data Management.* OlWIR Oldenburger Verlag für Wirtschaft, Informatik und Recht, Oldenburg, 2012.

[JBFT2005] Bart Jacob, Michael Brown, Kentaro Fukui and Nihar Trivedi. *Introduction to Grid Computing.* IBM RedBooks, 2005.

[KA2011a] Vinit Kumar and Ajay Agarwal. Generalized Grid Quorum Consensus for Replica Control Protocol. In *Proceedings of the International Conference on Computational Intelligence and Communication Networks*, pages 395–400, Gwalior, India, Oct 2011. IEEE.

[KA2011b] Vinit Kumar and Ajay Agarwal. *Multi-dimensional Grid Quorum Consensus for High Capacity and Availability in a Replica Control Protocol*, pages 67–78. Springer Berlin Heidelberg, 2011.

[KBMP1996] Karama Kanoun, Marie Borrel, Thierry Morteveille and Alain Peytavin. Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System. In *Proceedings of Annual Symposium on Fault Tolerant Computing*, pages 106–115, Sendai, Japan, June 1996. IEEE.

[KC1991] Akhil Kumar and Shun Yan Cheung. A High Availability $\sqrt{N}$ Hierarchical Grid Algorithm for Replicated Data. *Information Processing Letters*, 40(6):311–316, December 1991.

[Kha2010] Khan, Shahidul and Latiful Haque, Abu. A New Technique for Database Fragmentation in Distributed Systems. *International Journal of Computer Applications*, 5, 08 2010.

[KKW2013] T. Kobus, M. Kokocinski and P. T. Wojciechowski. Hybrid Replication: State-Machine-Based and Deferred-Update Replica-

tion Schemes Combined. In *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems*, pages 286–296, Philadelphia, PA, USA, July 2013. IEEE.

[Koc1994] Hans-Henning Koch. *Entwurf und Bewertung von Replikationsverfahren (In German).* PhD thesis, Technische Hochschule Darmstadt, Germany, 1994.

[Kum1991] Akhil Kumar. Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data. *IEEE Transactions on Computers*, 40(9):996–1004, 1991.

[Lam1978] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam1979] Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, 28(9):690–691, September 1979.

[Lam1998] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133â169, may 1998.

[McA2010] Andrew McAfee. The Weird and Wonderful Economics of Digitization. `https://hbr.org/2010/03/the-weird-and-wonderful-econom.html`, March 2010. [Online; accessed 13-December-2017].

[MCL2017] G. Mazlami, J. Cito and P. Leitner. Extraction of Microservices from Monolithic Software Architectures. In *Proceedings of the 24th IEEE International Conference on Web Services*, pages 524–531, Honolulu, HI, USA, June 2017. IEEE.

[Mey1988] Bertrand Meyer. *Object-Oriented Software Construction*, volume 2. Prentice Hall New York, 1988.

[MIB2009] Microsoft, IBM and BEA. WS-Coordination/WS-Transaction Specification, February 2009.

[MRSU2016] Antonio Messina, Riccardo Rizzo, Pietro Storniolo and Alfonso Urso. A Simplified Database Pattern for the Microservice Architecture. In *Proceedings of the 8th International Conference on Advances in Databases, Knowledge, and Data Applications*. IARIA XPS Press, June 2016.

[New2015] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 1st. edition, February 2015.

[NNG2019] NNGROUP. NNGROUP. `https://www.nngroup.com/articles/law-of-bandwidth/`, 2019. [accessed 17-July-2019].

[OAET2013] Tadateru Ohkawara, Ailixier Aikebaier, Tomoya Enokido and Makoto Takizawa. Quorum-Based Synchronization Protocols for Multimedia Replicas. *Cluster Computing*, 16(4):979–988, December 2013.

[OO2014] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proceedings of the USENIX Annual Technical Conference*, pages 305–319, Philadelphia, PA, 2014. USENIX.

[Ove2022] Team OverWorldInData. OverWorldInData. `https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage?country=~OWID_WRL`, December 2022. [accessed 05-September-2022].

[Pra2008] CSR Prabhu. *Grid and Cluster Computing*. PHI Learning Pvt. Ltd., 2008.

[RAIa] Team RAIK. RAIK CRDTs. `https://docs.riak.com/riak/kv/2.2.3/developing/data-types/`. [accessed 28-April-2019].

[RAIb] Team RAIK. RAIK CRDTs Guide. `https://gist.github.com/russelldb/f92f44bdfb619e089a4d/`. [accessed 28-April-2019].

[RED] Team REDIS. REDIS CRDTs. `https://redislabs.com/blog/getting-started-active-active-geo-distribution-redis-applications-crdt-conflict-free-replicated-data-types/`. [accessed 28-April-2019].

[Ric2017] Chris Richardson. Microservice Architecture. `http://microservices.io/`, 2017. [Online; accessed 13-December-2017].

[RIN+2017] C. Rotter, J. IllÃ©s, G. NyÃri, L. Farkas, G. CsatÃ¡ri and G. Huszty. Telecom Strategies for Service Discovery in Microservice Environments. In *Proceedings of the 20th Conference on Innovations in Clouds, Internet and Networks*, pages 214–218, Paris, France, March 2017. IEEE.

[RKUP2017] D. Richter, M. Konrad, K. Utecht and A. Polze. Highly-Available Applications on Unreliable Infrastructure: Microservice Architectures in Practice. In *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security Companion*, pages 130–137, Prague, Czech Republic, July 2017. IEEE.

[Run2008] Adrian Runceanu. Fragmentation in distributed databases. In Khaled Elleithy, editor, *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, pages 57–62, Dordrecht, 2008. Springer Netherlands.

[SGT2011] Mohamed Sellami, Walid Gaaloul and Samir Tata. An Implicit Approach for Building Communities of Web Service Registries. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, pages 230–237, New York, NY, USA, 2011. ACM.

[Sim2017] Team SimilarWeb. SimilarWeb. `https://www.similarweb.com/`, December 2017. [accessed 13-December-2017].

[SKB2004] Bujor Silaghi, Pete Keleher and Bobby Bhattacharjee. Multi-Dimensional Quorum Sets for Read-Few Write-Many Replica Control Protocols. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pages 355–362, Chicago, IL, USA, April 2004. IEEE, IEEE.

[SPBZ2011a] Marc Shapiro, Nuno Preguiça, Carlos Baquero and Marek Zawirski. *A comprehensive study of Convergent and Commutative Replicated Data Types*. PhD thesis, Inria–Centre Paris-Rocquencourt; INRIA, 2011.

[SPBZ2011b] Marc Shapiro, Nuno Preguiça, Carlos Baquero and Marek Zawirski. Conflict-Free Replicated Data Types. In Xavier Défago, Franck Petit and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[SPSPMJP2006] Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez and Ricardo Jiménez-Peris. WS-replication: A Framework for Highly Available Web Services. In *Proceedings of the 15th International Conference on World Wide Web*, pages 357–366, New York, NY, USA, 2006. ACM.

[SRD2016] G. Sousa, W. Rudametkin and L. Duchien. Automated Setup of Multi-cloud Environments for Microservices Applications. In *Proceedings of the 9th IEEE International Conference on Cloud Computing*, pages 327–334. IEEE, June 2016.

[ST2017] R. Schadek and O. Theel. Increasing the Accuracy of Cost and Availability Predictions of Quorum Protocols. In *Proceedings of the 22nd IEEE Pacific Rim International Symposium on Dependable Computing*, pages 98–103, Christchurch, New Zealand, January 2017. IEEE.

[Str2003] Thomas Strang. *Towards Autonomous Services for Smart Mobile Devices*, volume 2574, pages 279–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[The1993] O. Theel. General structured voting: a flexible framework for modelling cooperations. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 227–236, Pittsburgh, PA, USA, May 1993. IEEE.

[Tho1979] Robert H. Thomas. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems (TODS)*, 4(2):180–209, June 1979.

[TVS2007] Andrew S Tanenbaum and Maarten Van Steen. *Distributed Systems*. Prentice-Hall, 2007.

[UST2017] A. Usman, R. Schadek and O. Theel. A Novel Highly Available Data Replication Strategy exploiting Data Semantics, Coding Techniques and Prior At-Hand Knowledge. In *Proceedings of the 22nd IEEE Pacific Rim International Symposium on Dependable Computing*, pages 301–310, Christchurch, New Zealand, January 2017. IEEE.

[UZT2017a] A. Usman, P. Zhang and O. Theel. A Component-Based Highly Available Data Replication Strategy Exploiting Operation Types and Hybrid Communication Mechanisms. In *Proceedings of the 14th IEEE International Conference on Services Computing*, pages 495–498, Honolulu, HI, USA, June 2017. IEEE.

[UZT2017b] A. Usman, P. Zhang and O. Theel. An Efficient and Updatable Item-to-item Frequency Matrix for Frequent Itemset Generation. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, pages 85:1–85:6, New York, NY, USA, 2017. ACM.

[UZT2018] A. Usman, P. Zhang and O. Theel. A Highly Available Replicated Service Registry for Service Discovery in a Highly Dynamic Deployment Infrastructure. In *Proceedings of the 15th IEEE International Conference on Services Computing*, pages 265–268, San Francisco, CA, USA, July 2018. IEEE.

[WB1992] Chienwen Wu and Geneva G Belford. The Triangular Lattice Protocol: A Highly Fault Tolerant and Highly Efficient Protocol for Replicated Data. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 66–73, Houston, TX, USA, Oct 1992. IEEE, IEEE.

[WCD2017] Lucas Waye, Stephen Chong and Christos Dimoulas. Whip: Higher-order Contracts for Modern Services. *Proceedings of the ACM on Programing Languages (PACMPL)*, 1(ICFP):36:1–36:28, August 2017.

[Wik2019] Wikipedia. SupercomputerWiki. `https://en.wikipedia.org/wiki/Supercomputer`, 2019. [accessed 17-July-2019].

[WK2002] Hakim Weatherspoon and John D Kubiatowicz. *Erasure Coding vs. Replication: A Quantitative Comparison*, pages 328–337. Springer Berlin Heidelberg, 2002.

[WPS+2000] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme and Gustavo Alonso. Understanding Replication in Databases and Distributed Systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, pages 464–474, Taipei, Taiwan, 2000. IEEE.

[YSS2016] Yale Yu, H. Silveira and M. Sundaram. A Microservice Based Reference Architecture Model in the Context of Enterprise Architecture. In *Proceedings of the IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*, pages 1856–1860, Xi'an, China, October 2016. IEEE.

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.
Ebenso versichere ich, dass ich diese Disseration nur in diesem Promotionsverfahren eingereicht habe und dass diesem Promotionsverfahren keine anderen endgültig nicht bestandenen Promotionsverfahren vorausgegangen sind.

Awais Usman, Bochum, den 15.11.2023