REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre :………………
Série :………………………

Thèse

Présentée en vue de l'obtention du diplôme de docteur en sciences

Option : **Informatique**

Techniques intelligentes pour la gestion de la cohérence des Big data dans le cloud

Par :

M. KHELAIFA Abdennacer

Soutenue le : 26/01/2023

Devant le jury :

BENNOUI Hammadi	Professeur	Université de Biskra	Président
BENHARZALLAH Saber	Professeur	Université de Batna 2	Rapporteur
KAHLOUL Laid	Professeur	Université de Biskra	Co-Rapporteur
SLATNIA Siham	Professeur	Université de Biskra	Examinatrice
SEGHIR Rachid	Professeur	Université de Batna 2	Examinateur
LABOUDI Zakaria	MCA	Université d'Oum El Boughi	Examinateur

Année universitaire : 2022 – 2023

Dedication

To my late parents, may God have mercy on them,
*To my wife **IKRAM**, the highest symbol of sincerity and loyalty,*
*To my beloved sons **TAMIM** and **GHAITH** and my darling daughter **KHADIDJA**.*
To my brothers, my sisters, and their families.
To all my colleagues and friends.

Acknowledgement

First and foremost, thanks and praise go to ALLAH at the beginning and the end. I thank him for giving me the health and moral capacity that allows me to overcome all difficulties.
I would like to express my sincere gratitude to my supervisor Prof. Saber BENHARZALLAH. for the continuous support of my Ph.D study and related research, and for his patience and motivation. His guidance helped me in all the time of research and writing of this thesis.
Many thanks go to Prof. Laid KAHLOUL, Dr. Abdelkader LAOUID, Dr Ahcène BOUNCEUR and Prof. Reinhardt EULER. Their help and advice carried me through all the stages of writing my project.
I'd like to give special thanks to the HasLab/INESCTEC professors in Braga Portugal mainly Prof. Ali Shocker for assistance with particular techniques and methodologies. Many thanks go also to Dr. Peter ZELLER and Dr. Annette BIENIUSA from TU Kaiserslautern for their technical support.
My sincere appreciation needs to be addressed to the honourable board of examiners, Prof. Hammadi BENNOUI, Prof. Siham SLATNIA, Prof. Rachid SEGHIR, and Dr. Zakaria LABOUDI whose insightful remarks during the viva will certainly enrich this work.

Abstract

This thesis tackles the problem of data consistency of Big data in the cloud. Indeed, our research focuses on studying different adaptive consistency approaches in the cloud and proposing a new approach for the Edge computing environment. Managing consistency has major consequences for distributed storage systems. Strong consistency models require synchronization after each update which affects significantly the system's performance and availability. Conversely, models with low consistency provide better performance as well as better data availability. However, these latter models can tolerate too much temporary inconsistency under certain conditions. Therefore, an adaptive consistency strategy is needed to tune, during run-time, the consistency level depending on the criticality of the requests or the data items. This thesis provides two contributions. In the first contribution, a comparative analysis of existing adaptive consistency approaches is made according to a set of defined comparative criteria. This kind of survey provides the user/researcher with a comparative performance analysis of the existing approaches. Moreover, it clarifies the suitability of these approaches for candidate cloud systems. In the second contribution, we propose MinidoteACE, a new adaptive consistency system which is an improved version of Minidote a causally consistent system for edge applications. Unlike Minidote which supports only causal consistency, our model allows applications to run also queries with stronger consistency guarantees. Experimental evaluations show that throughput decreases only by 3.5% to 10% when replacing a causal operation with a strong operation. However, update latency increases significantly for strong operations up to three times for 25% update workload.
Keywords: Big data, cloud computing, data consistency, adaptive consistency, adaptive policy, MinidoteACE, causal consistency.

Résumé

Cette thèse aborde le problème de cohérence des données de Bigdata dans le cloud. En effet, nos recherches portent sur l'étude de différentes approches de cohérence adaptative dans le cloud et la proposition d'une nouvelle approche pour l'environnement Edge computing. La gestion de la cohérence a des conséquences majeures pour les systèmes de stockage distribués. Les modèles de cohérence forte nécessitent une synchronisation après chaque mise à jour, ce qui affecte considérablement les performances et la disponibilité du système. À l'inverse, les modèles à faible cohérence offrent de meilleures performances ainsi qu'une meilleure disponibilité des données. Cependant, ces derniers modèles peuvent tolérer trop d'incohérences temporaires sous certaines conditions. Par conséquent, une stratégie de cohérence adaptative est nécessaire pour ajuster, pendant l'exécution, le niveau de cohérence en fonction de la criticité des requêtes ou des données. Cette thèse apporte deux contributions. Dans la première contribution, une analyse comparative des approches de cohérence adaptative existantes est effectuée selon un ensemble de critères de comparaison définis. Ce type de synthèse fournit à l'utilisateur/chercheur une analyse comparative des performances des approches existantes. De plus, il clarifie la pertinence de ces approches pour les systèmes cloud candidats. Dans la seconde contribution, nous proposons MinidoteACE, un nouveau système adaptatif de cohérence qui est une version améliorée de Minidote, un système de cohérence causale pour les applications Edge. Contrairement à Minidote qui ne fournit que la cohérence causale, notre modèle permet aux applications d'exécuter également des requêtes avec des garanties de cohérence plus fortes. Des évaluations expérimentales montrent que le débit ne diminue que de 3,5 % à 10 % lors du remplacement d'une opération causale par une opération forte. Cependant, la latence de mise à jour augmente considérablement pour les opérations fortes jusqu'à trois fois pour une charge de travail où le taux des opérations de mise à jour est de 25 %.
Mots Clés: Big data, cloud computing, cohérence des données, cohérence adaptative, politique adaptative, MinidoteACE, cohérence causale.

الملخص

تتناول هذه الأطروحة مشكلة تناسق البيانات الخاصة بالبيانات الضخمة في بيئة الحوسبة السحابية. في الواقع، يركز بحثنا على دراسة مختلف مناهج التناسق التكيفي للبيانات في السحابة واقتراح منهج جديد لبيئة حوسبة الحافة. إدارة التناسق لها عواقب وخيمة على أنظمة التخزين الموزعة. تتطلب نماذج التناسق القوي المزامنة بعد كل تحديث مما يؤثر بشكل كبير على أداء النظام وتوافره. على العكس من ذلك ، توفر النماذج ذات التناسق المنخفض أداءً أفضل بالإضافة إلى توفر بيانات أفضل. ومع ذلك ، يمكن لهذه النماذج الأخيرة أن تتسامح مع الكثير من التناقض المؤقت في ظل ظروف معينة. لذلك ، هناك حاجة إلى استراتيجية تناسق تكيفية لضبط مستوى التناسق، أثناء وقت التشغيل، إعتماداً على أهمية الإستعلام أو البيانات في حد ذاتها. تقدم هذه الأطروحة مساهمتين. في المساهمة الأولى ، يتم إجراء تحليل مقارن لمناهج التناسق التكيفية الموجودة وفقاًا لمجموعة من المعايير المقارنة المعرفة من طرفنا. يوفر هذا النوع من الدداسات الشمولية للمستخدم \ الباحث تحليلاً مقارناًا لأداء المناهج. علاوة على ذلك ، فإنه يوضح مدى ملاءمة هذه الأساليب لأنظمة السحابة المرشحة. في المساهمة الثانية ، نقترح MinidoteACE ، نظام تكيفي جديد نظام التناسق وهو نسخة تحسين من Minidote وهو نظام متناسق سببياًا لتطبيقات الحافة. على عكس Minidote الذي يدعم الاتساق السببي فقط ، يسمح نموذجنا للتطبيقات بتشغيل استعلامات بضمانات تناسق أقوى أيضاًا. تظهر التقييمات التجريبية أن الإنتاجية تقل فقط بنسبة 3.5% إلى 10% عند استبدال العملية السببية بعملية قوية. ومع ذلك ، يزداد زمن انتقال التحديث بشكل كبير للعمليات القوية حتى ثلاث مرات بالنسبة لحجم عمل تكون فيه عمليات التحديث بنسبة 25%.

كلمات مفتاحية: البيانات الضخمة ، الحوسبة السحابية، تناسق البيانات ، التناسق التكيفي ، السياسة التكيفية ، MinidoteACE ، التناسق السببي.

Contents

List of Figures

List of Tables

List of Algorithms

Acronymes

MinidoteACE:	Minidote Adaptive Consistency in the Edge.
CRDTs:	Conflict Replicated Data Type.
ACID:	Atomicity, Consistency, Isolation, and Durability.
NoSQL:	Not Only Structured Query Language.
MEC:	Mobile Edge Computing.
RAID:	Redundant Array of Independent Disks.
CAP:	Consistency, Availability and Partition Tolerance.
SDN:	Software Defined Networking.
SLAs:	Service Level Agreement.
MRFU:	Most Recent and Frequently Used.
IDEA:	Infrastructure for Detection-based Adaptive consistency guarantees.
ICG:	Incremental Consistency Guarantees.
ABARC:	Application-Based Adaptive Replica Consistency.
FHBC:	File Heat-Based Consistency.
CPQ:	continuous partial quorums.
SACRFCG:	Self-adaptive conflict resolution with flexible consistency guarantee.
SCM:	Selective Consistency Model.
ACPK:	Adaptive consistency policy for kafka.
4S:	Safe Serializable Secure Scheduling.
CMS:	Consistency Management System.
ACSM:	Additional Client-Side Module.
Camus:	CAusal MUlticast Service.
DpGraph:	Dependency Graph.
Vc:	Vector Clock.

List of Publications

International Journals

- Abdennacer Khelaifa, Saber Benharzallah, Laid Kahloul, Reinhardt Euler, Abdelkader Laouid and Ahcène Bounceur – "A comparative analysis of adaptive consistency approaches in cloud storage", Journal of Parallel and Distributed Computing 129 (2019), p. 36–49 (Indexed ISI Thomson IF=4.542).
- Abdennacer Khelaifa, Saber Benharzallah, Laid Kahloul – "A new adaptive causal consistency approach in edge computing environment", International Journal of Computing and Digital Systems 12 (2022), p. 945–960 (Indexed SCOPUS).

General Introduction

Nowadays, Social media, smartphones, tablets, GPS devices, sensors, log files, and a variety of other devices and sources generate a tremendous amount of unstructured data every second. Furthermore, the amount of data created each year is far greater than it has ever been, which is why our era is known as the Information Age. This data deluge, known as Big Data, adds a slew of problems to many facets of data storage and administration. These issues arise not only because of the enormous sizes, but also because of the required velocity and the complexity of data arriving from several sources with varying requirements, all while dealing with significant load variability. In order to deal with the related challenges, many Big Data systems rely on large and novel infrastructures, as well as new platforms and programming models that have become cheaper and more powerful than ever before due to the fast development of processing and storage technologies and the success of the Internet. In this context, the emerging paradigm of Cloud Computing [3] offers excellent means for Big Data. Within this paradigm, users can lease on-demand computing and storage resources in a Pay-As-You-Go manner [4]. Thereby, corporations can acquire the resources needed for their Big Data applications at a low cost when needed [5, 6]. Meanwhile, they avoid large investments on physical infrastructures that need huge efforts for building and maintaining them, which, in addition, requires a high level of expertise.

In distributed systems, replica technology [7] is a key technology in enhancing the system's performance. With the assistance of replica and redundancy of data in several domains [8, 9, 10, 11, 12, 13], the distributed file system is able to reduce access time lag, network bandwidth consumption and system unreliability. However, this also leads to the problem of replica consistency management. Namely, when data is accessed and read by multiple users, inconsistency occurs in replicas, and as a result, the system's consistency and accuracy are directly influenced. The study of replica consistency aims to achieve synchronism among multiple replicas. The most popular strategies provided by storage systems are strong consistency and eventual consistency [14]. Strong consistency ensures all of the replicas to be updated immediately. There is no difference between replicas. Therefore, every access to replicas will get fresh data. But the cost of maintaining the freshness of replicas increases significantly, which cuts down the availability of replicas and the system's performance. For instance, Google BigTable [15], Microsoft Azure Storage [16] and Apache HBase [17] provide strong consistency. Eventual consistency doesn't update all the replicas immediately, so it tolerates replica divergence. But it promises all the replicas to be consistent at a specific time. For instance, Amazon Dynamo [18], Cassandra [19] and MongoDB [20] provide eventual consistency.

Providing one consistency strategy is only suitable for particular scenes since the clients of cloud storage are multifarious and not all the applications need the same level of consistency. In addition, the required consistency strategy of an application is variable at run-time. The management of consistency heavily impacts storage systems. Furthermore, with Big Data scales, the management of consistency is critical to meet performance, availability, and monetary cost requirements. Traditional storage systems and databases that implement rigorous models such as strong consistency have shown their limitations in meeting the scalability demands and the performance requirements of nowadays Big Data applications. In this context, researchers have introduced many improvements where they used intelligent techniques to resolve the problem of consistency. Flexible and adaptive consistency approaches are considered intelligent solutions since they inspect the application requirements and the system context in order to provide adequate guarantees.

First Contribution

To develop a robust adaptive protocol, it is necessary to make a comparative study of the proposed approaches by taking into consideration all the elements around consistency in the cloud. All contributions that we have found described few proposed approaches as related works but none, to our knowledge, established an exhaustive analysis of the existing adaptive consistency approaches. In the first contribution [21], we evaluate the most popular adaptive techniques in cloud systems using a set of criteria proposed for this purpose. To achieve our goal, we first give the main concepts of consistency in distributed systems including adaptive consistency. Then, we define a set of criteria that describe the design or the behavior of such an approach like: architectural model, operation level, granularity of consistency strategy, adaptive policy, statistical based or predictive policy, conflict detection and resolution, monetary cost consideration, threshold definition, security and privacy and implementation tools. Thereafter, we review 19 proposed works by describing their main contributions and then summarizing their behaviors towards the defined criteria in a table. The table-based analysis gives us a global view and allows us to discuss different aspects of adaptive consistency.

Second Contribution

In the second contribution, we propose MinidoteACE [22] a new adaptive consistency strategy that enables the client to choose the level of consistency for each update he sends either causal or strong. For this purpose, we improve Minidote system [23] to become able to support adaptive consistency. We keep the existing behavior for causal operations, however we use the implementation of causal stability [24] to carry out strong operations. An operation is stable if it is delivered to all the node of the cluster. Hence, a strong operation will not be executed until it will be stable. Towards that end, we delay the execution of strong queries until they arrive at the other nodes. The new consistency level is weaker than the typical strong consistency as it just ensures the arrival of queries to the system nodes, not their execution. Our model provides stronger consistency guarantees to ensure high availability. If we assume that operations will be executed correctly, replicas will have all recent updates so clients will be able to regain their updates when they switch from one node to another. Hence, in addition to causal consistency, clients can also perform updates with stronger guarantees. Providing adaptive consistency allows application designers to choose the convenient consistency level for each operation according to application needs and the system's context. The application can specify which operations should be run under causal consistency and which others should have stronger consistency guarantees.

Thesis structure

This thesis was divided into three chapters to address the issues raised. The first chapter introduces basic definitions of big data and different computing paradigms: Cloud, Fog, Edge and Cloudlet. In the same chapter, we present challenges and issues of Big data within Cloud Computing, Edge and Fog.
The second Chapter gives a comparative study of adaptive consistency approaches in the cloud. It tackles the concept of data consistency and its different models. It proposes also a set of comparative criteria and discusses the existing adaptive consistency solutions according to the proposed criteria. Then, it outlines challenges and future research directions.
The third Chapter is dedicated to the proposed adaptive consistency approach in Edge Computing environments. It gives a background to understand the context of the work such as : Causal consistency, CRDTs and Causal Stability. Then, It details the solution's architecture and algorithms as well as its implementation and obtained results.

The thesis is concluded with a general conclusion that summarises all the work mentioned in the thesis and the effectiveness of the achieved results. It presents also the prospects of the proposed solutions.

Chapter 1

General concepts and Context

Introduction

The last decade has known an exponential growing of data sizes within many corporations and organizations. According to IDC, the overall global datasphere reached 40 zettabytes by the end of 2019 [25]. while it is predicted to reach a huge 175 zettabytes by 2025. This fast growth of data, known as Big Data, introduces many issues and challenges related to storing, managing, processing and querying the huge amount of data. Big companies such as Facebook, Google, Amazon, and Microsoft have tackled these issues by relying on novel large-scale infrastructures and software platforms. However, small companies had struggled to keep pace with the fast growth rate of data size and variety which prove the persistent need for efficient solutions that enable them to address the emerging challenges. In this context, Cloud Computing provides excellent tools for dealing with Big Data's most difficult features. This chapter introduces the Big Data phenomena, as well as platforms and infrastructures for dealing with the issues it poses. Then, as an effective platform for exploiting Big Data management, we concentrate in on Cloud Computing. Finally, we go through Big Data difficulties and challenges, as well as how key cloud companies approach them. As a result, we highlight replication in the context of Cloud Computing, as well as its benefits and drawbacks.

1.1 Big data

1.1.1 Big Data Definitions

According to [26], Big data is "data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the structures of your database architectures". Big Data doesn't mean only the large amount of data, it adresses also the complexity generated by the growing sizes of datasets regarding to different aspects of data handling. This opinion is also shared by [27], [28] and [29]. In [30], Stonebraker was the first who used the "3Vs" model to introduce Big Data. The "3Vs" refer to a set of attributes identified as defining characteristics of Big Data, according to their common name initial. Big Data refers to data sets that are of a big volume, need big velocity, or exhibit big variety. [1] considers two additional dimensions of Big data: Value and Veracity which design the "5Vs" Model of Big data As shown in Figure 1.1.

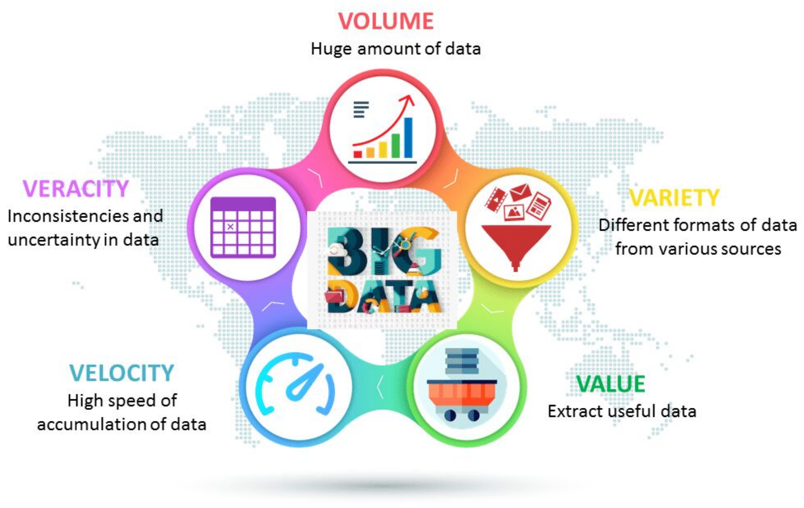
VOLUME
Huge amount of data
VERACITY
Inconsistencies and uncertainty in data
VARIETY
Different formats of data from various sources
BIG DATA
VELOCITY
High speed of accumulation of data
VALUE
Extract useful data

Figure 1.1: 5 V's of Big Data [1].

Volume: data sizes have known an exponential increasing in the last decade due to Data Deluge phenomenon and the variety of data sources, including business transactions, smart devices, industrial equipment, videos, social media and more. This phenomenon is a result of lower storage costs and the analytical models that need large data set to provide more accurate results. In fact, the size dimension of Big Data represents the primary challenge to the existing data management systems.

Velocity: the speed of Data collection and its flow into the processing engines make data velocity as importance as data volume [31]. At first, specific kinds of applications have been characterized by velocity, but it becomes a larger issue with the growth in the Internet of Things. Therefore, A wide range of applications require fast data processing in near real-time manner. For instance, electronic trading, Sensors and smart meters fit in this class of applications.

Variety: Data has many forms and rarely come in a perfectly ordered form and ready for processing. These data are available in a variety of formats, ranging from text to video and audio, relational databases to semi-structured data, and massive binary objects. The major problem is to effectively integrate and manage all of these different forms of data. According to SAS [31], 80 percent of created data is not numerical. Those data, on the other hand, should be used in analytics and decision–making.

Value: Value means big data has a very high value if it is processed in an appropriate way or it can also be said how valuable or meaningful data is. For example, the biodata of employees of a company selling food raw materials will not be of value for the benefit of predictive analysis of sales of raw materials to customers. The data may be unimportant and of no value in one way, but it can be very important and very valuable in another. Data that has no value in any part will not be filtered in the Big data analytics application system.

Veracity: Veracity [31] refers to the quality of data mainly accuracy and precision. The big variety of data sources makes linking, matching, cleansing and transforming data across systems so difficult. Businesses need to connect and correlate relationships, hierarchies and multiple data linkages. Otherwise, their data can quickly become uncontrollable.

1.1.2 Big Data Platforms

Since Database Management Systems and traditional data querying paradigms cannot fulfill the increasing requirements of Big data, multiple storage methods and data processing approaches have been proposed to overcome this issue.

- Parallel File Systems: To address centralized file system scalability and fault tolerance limits parallel file systems have been created. They depend on decentralized storage, which allows for scalable performance and quick access. The enormous parallel architecture of this file system class makes it possible to distribute working loads over several servers, which may be extended across large areas, to access the data and occasionally to access metadata. In addition, replication allows file systems to have faster access to data from closer replicas. Most parallel file system developers are unwilling to adhere to POSIX semantics [32] for the provision of such scaling capabilities. Like the ACID criteria, the POSIX standard enforces a strong semantic and restricts information processing which penalizes system performance and is a key bottleneck for scalability. However,many file systems such as GPFS [33] feel that POSIX compatibility is highly important in spite of the performance penalty. Other systems prefer to provide a basic POSIX-compliant data access interface even when the system itself is not completely POSIX-compliant, like Luster file system [34]. In the meanwhile, POSIX compatibility with contemporary file systems for increased performance, scalability and availability, such as PVFS [35], Ceph [36], Google File System [37] and HDFS [38].

- NoSQL Datastores: RDBMS has over the past several years proven that its scalability constraints are overwhelmingly high for Big Data. Strong ACID semantics (Atomicity, Consistency, Isolation and Durability) are used in RDBMS designs and query models (primarily SQL) which were deemed necessary for many years in order to offer a "correct" model. However, enforcing such semantics offers new constraints with the increasing size of applications, in the age of big data [35]. To offer scalability, availability and excellent speed, NoSQL rejects part or all ACID semantics. NoSQL datasets often rely on considerably simpler data queering models than SQL, with key-based access and key/value scheme-based data. Data may be stored and disseminated massively based on appropriate algorithms and data structures. NoSQL data storage and storage systems such as Amazon Dynamo [18], Google Big Table [15] and Cassandra [19] have now proved to be very efficient in the use of global data. Usually, these systems are developed for a certain application class (e.g. Web applications, document storage, etc.). Therefore, they rely on concepts that might loosen or sacrifice part of the strong semantics, in order to make them more efficient, better available and more scalable.

- MapReduce: MapReduce [39] is a Google-inspired programming technique that offers a massively parallel processing of huge data collections. Two primary phases comprise the MapReduce frameworks: the Map Phase and the Reduce Phase. The users provide a map function, dividing the incoming into sub-problems and generating interim data as key-value pairs. These intermediate data are also transmitted to a user-specified reduction function. In the Reduce step all values with the same intermediate key are combined. MapReduce has become an essential paradigm for Big Data data processing because it offers scalability, availability, performance and reliability whereas traditional RDBMS (with their SQL-based query architecture) struggle to fulfill needs for scalability. MapReduce frameworks like Hadoop [38] can most of the time efficient processing of data of broad size to substitute complicated SQL queries. In this context, a large range of MapReduce applications has been created including applications for distributed data querying, data analytics, parallel sorting, data clustering, and machine learning.

1.1.3 Big Data Infrastructures

The remarkable rise of Big Data demands efficient infrastructures and innovative computing paradigms in order to handle large amounts of data and their big velocity. Hereafter, we show three typical infrastructure models that give great ways to manage Big Data.

- Clusters: Cluster computing involves linking several computers over a local area network (LAN) to their normal software stack (e.g. operational systems), to construct a global system. One of the key inspirations for effective distributed computing was cluster computing. This paradigm is primarily aimed at providing improved performance and low cost availability. During the Big Data era, specialized clusters are generally constructed by companies consisting of commodity hardware. These clusters operate Big Data (such as Hadoop and NoSQL systems) platforms to efficiently hold enormous amounts of data. Simultaneously, they offer real-time treatment and high availability. These cluster sizes can range from tens to tens of thousands of nodes [40].
- Grids: By definition in [41], the term "the Grid" refers to *a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver non-trivial qualities of service.* A grid is therefore a system which:
  - Coordinates users without centralized control over resources.
  - Offers standard, open protocols and interfaces for broad use.
  - Providing non-trivial service characteristics.

  Grid Computing covers in general the federation and coordination of heterogeneous sub-organizations and resources which might be distributed over remote locations. It tries to hide from users and reveals to them the illusion of a global system that the coordination of subsystems is complicated.
- Clouds: Over the years, parallel and distributed computing has evolved to ensure greater service quality and efficiency in performance, cost and fault tolerance. Cloud Computer is a rapidly emerging computing paradigm. Software and/or hardware is provided through a computer network, often the internet, as a service under this paradigm. In their abstraction level, cloud services differ. They may be classified into three levels: software, infrastructure and platform. Since it came into being, Big Data applications have taken over cloud computing quickly. For many companies it was long-lastingly necessary to acquire resources in Pay-as-You-Go ways to expand their big data platforms, and cloud providers supply this. In addition, Big Data platforms are offered to the most cloud service providers. They let customers to execute their apps without worrying about the costly administration and upkeep of infrastructure.

1.2 Cloud Computing

Cloud Computing is a rising paradigm that has become increasingly popular in the last decade, particularly in IT and economy community. Many authors have tried to give a clear definition of cloud computing. While various definitions exist, they have common characteristics. [42, 43, 44, 45]. Cloud computing is defined by the capability to supply software and hardware in a scalable manner. The basic component of Cloud is architecturally a data center that includes raw hardware processing and storage for renting in conjunction with software in a pay-as-you-go manner. Buyya et al. [44] define a cloud as *"a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers".* In [45], Berkeley RAD lab defined Cloud computing as follows: *"Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. The services themselves have long been referred to as Software as a Service (SaaS). The data center hardware and software is what we will call a Cloud. When a Cloud is made available in a pay-as-you-go manner to the general public, we call it a Public Cloud; the service being sold is Utility computing. We use the term Private Cloud to refer to internal data centers of a business or other organization, not made available to the general public. Thus, Cloud computing is the sum of SaaS and Utility computing, but does not include Private Clouds. People can be users or providers of SaaS, or users or providers of Utility computing"* From a hardware perspective, Cloud computing characterizes itself from traditional computing paradigms in three areas.

- Users of Cloud computing do not need to prepare ahead for future IT infrastructure expansion. Cloud computing gives customers the illusion of having boundless computing resources.
- Cloud customers can start using a few computing resources of Cloud resources and then scale up as their needs grow.
- Cloud providers give short-term payment capabilities. Processors and storage resources, for example, can be freed if they are no longer needed in order to keep money. Cloud computing matches Big Data perfectly as it delivers nearly unlimited resources on request. Moreover, it provides the Big Data processing capability for any user who cannot create its own infrastructure, like clusters, networks, and so on. Cloud customers might potentially conduct their big data processing economically by leasing resources.

1.2.1 Cloud Service Levels

Cloud Computing services differ in their levels and can be categorized as seen in Figure 1.2 into three tiers. Cloud customers now have multiple types of resources to acquire from different suppliers. A customer may Cloud clients nowadays can purchase resources of different natures from various providers. A client can lease infrastructure resources, platform resources, or software resources, possibly all three types simultaneously.

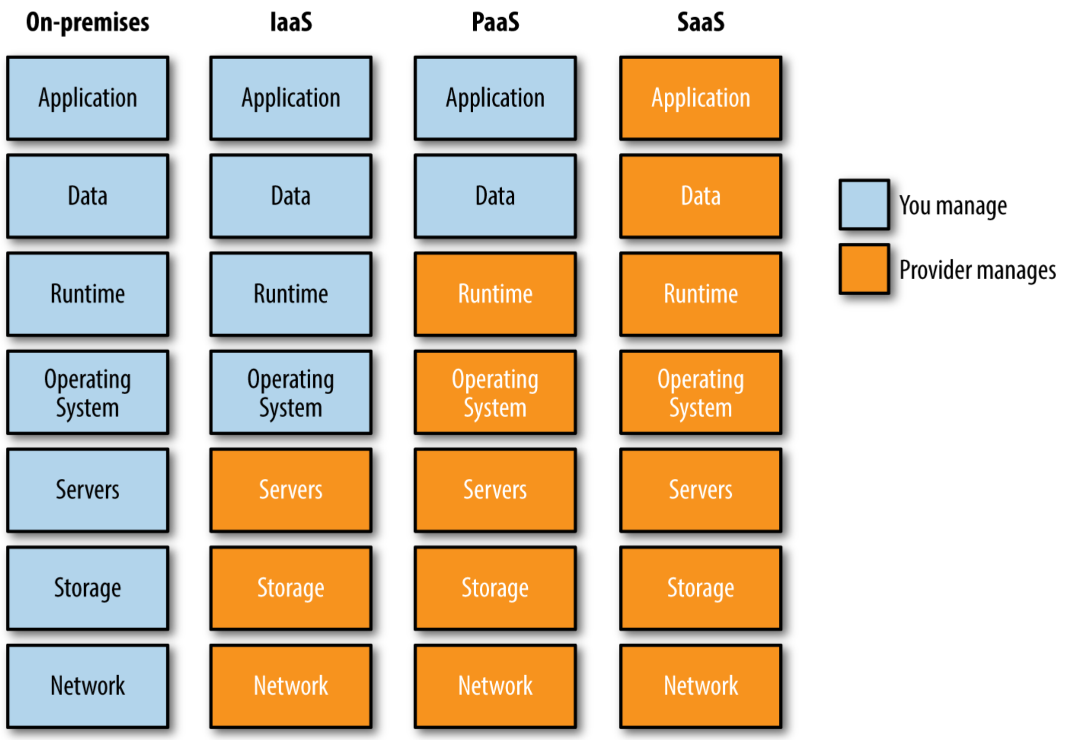

Figure 1.2: Infrastructure layers of Cloud. [2].

- Software as a Service (SaaS): Software as a service is the most visible layer for end-users. software is delivered as a service over a network, Internet typically, by allowing customers to directly use applications deployed on Cloud infrastructure. With recent developments in network hardware and the high bandwidth, customers may run applications and software over the Internet, which are previously running on local machines. Such a service is advantageous to customer organizations, since it streamlines application management. Moreover, it enables improved version homogenization between application users, and gives global access and collaboration. In general, any Web browser can be used as an interface of Cloud applications. Examples of SaaS Clouds are Google Docs [46] and Microsoft Office Live [47], etc.

- Platform as a Service (PaaS): PaaS sits on the bottom of SaaS to offer a high-level computing platform that often includes Operating System, database, programming and execution environment. Using PaaS, software developers can design their applications using specific frameworks provided by the platform without the need to control the underlying hardware infrastructure (IaaS).

In the last few years, PaaS has become very popular. The customers export platform and infrastructure management to a third party with all its complexity, which is generally better experienced and better suited to face the accompanying infrastructure and platform difficulties. Google App Engine [48], Microsoft Azure [49], and ConPaaS [50] are examples of PaaS.

- Infrastructure as a Service (IaaS): IaaS provides raw hardware resources on demand in the form of virtualized resources, such as computing power, network and storage. Infrastructure pools including storage, computer Servers, network, and other hardware components are hosted, managed, and maintained by the cloud provider. Customers have to manage and maintain the software stack. Fees are levied on a model that reflects the real level of utilization of crude resources: Volume storage, hourly cycles of CPU, etc. Examples of IaaS Cloud platforms include: Eucalyptus [51], OpenNebula [52], and Amazon Elastic Compute Cloud [53].

1.2.2 Cloud Computing Models

Many cloud models have been introduced to meet the needs of their targeted users [54]. There are three main models :

1.2.2.1 Public Clouds

Public clouds are the most common type of cloud computing deployment. The cloud resources (such as servers and storage) are owned, maintained, and distributed via the Internet by a third party cloud service provider. With a public cloud, the cloud provider owns and manages all hardware, software, and other supporting infrastructure. Major Cloud providers like Amazon, Microsoft and Google target a wide range of customers with reliable and effective services at reasonable pricing. The concept of public cloud offers several benefits over other paradigms. Public clouds cut financial costs dramatically, in particular for new projects, as the charge comprises only the amount of resources usage. Public cloud customers access services and manage their accounts using a web browser and share the same hardware, storage, and network devices with other organizations or cloud tenants.

1.2.2.2 Private Clouds

Private clouds are platforms designed specifically to serve one organization. The cloud platform might be built internally by the same organization or by a third party. The versatility of the private cloud can help organizations since the platform can be developed to meet the organization's specific requirements. However, the organization is obliged to handle (by itself or by delegating to a third party) the infrastructure and platform management and maintenance, which can be both complicated and costly. Since only customers belonging to the organization are allowed to access the cloud, private clouds are characterized by their high security and privacy. However, they might suffer from limited scalability and elasticity as the needs grow over the years. Private clouds are often used by government agencies, financial institutions, any other mid- to large-size organizations with business-critical operations seeking enhanced control over their environment.

1.2.2.3 Hybrid Clouds

A hybrid cloud is a combination of two or more clouds of different nature (public and private). Hybrid cloud computing allows companies to seamlessly expand their local infrastructure to the public cloud When the need for more computing and processing resources rises. Such a solution allows companies to handle any overflow — without access to the whole of their data by data centers. The heterogeneity of resources in several clouds, however, can lead to an additional overall management and fluctuation in performance. In spite of this, virtualization techniques were mature enough to handle physical resource heterogeneity. In addition, in some circumstances a combination of public and private resources can offer a vulnerability to private data safety. Hybrid clouds remove huge investments in the management of short-term demand growth as well as in the freely-funding of local resources to more sensitive data or applications. They also enable organizations to scale up computer resources. Instead of buying, programming and maintaining new resources and equipment that can be idle for a long period, companies are paying for the resources they temporarily use.

1.3 Emerging Computing Paradigms

With the Internet of Things (IoT) becoming part of our daily life and our environment, a rapid growth in the number of connected devices is expected. IoT is expected to connect billions of devices and humans to bring promising advantages. According to [55] the number of connected objects in the world will increase from 11.7 billion in 2020 to 30.9 billion in 2025. Data generated by IoT devices is tremendously huge which obligates companies to use the cloud to process and store data. As the data velocity and volume increases, moving the big data from IoT devices to the cloud might not be efficient, or might be even infeasible in some cases due to bandwidth constraints. On the other hand, as time-sensitive and location-aware applications emerge (such as patient monitoring, real-time manufacturing, or self-driving cars), the distant cloud will not be able to satisfy the ultra-low latency requirements of these applications, provide location-aware services, or scale to the magnitude of the data that these applications produce [56] . Moreover, in some applications, sending the data to the cloud may not be a feasible solution due to privacy concerns. In order to address these issues, there is a quintessential need for a computing paradigm that takes place closer to connected devices [57]. Therefore, Edge Computing [58] and Fog computing [59] paradigms have been proposed to address such challenges. Figure 1.3 illustrates the difference

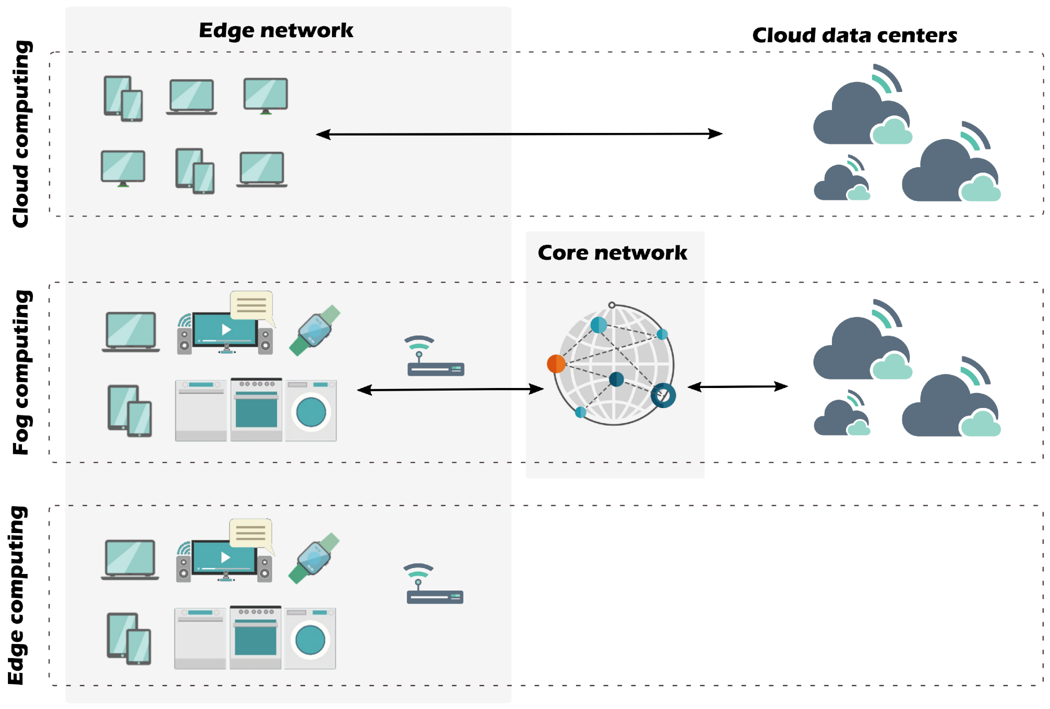

Figure 1.3: The difference between Cloud, Edge, and Fog Computing.

between Cloud, Edge, and Fog Computing.

1.3.1 Edge Computing

Edge computing is defined as a model of distributed computing that employs technologies allowing to perform computation at the Edge of the network [58]. Edge computing augments the cloud computing paradigm, where all processing is performed in data centers, in order to achieve better scalability, lower latencies, better data privacy, and a more efficient usage of resources (including a more effective energy consumption). In Edge computing, processing is performed cooperatively by Edge devices and cloud servers. Which computations are performed on which of these components depend on a number of factors, including both the capacity of the nodes and the latency requirements. The term "Edge" refers to any computing and network resources along the path between data sources (devices) and cloud data centers. For example, a mobile phone is considered as an Edge between body things and cloud as well as a gateway in a smart home is the

Edge between home things and cloud. The same thing with a micro data center and a Cloudlet [60] that form the Edge between mobile devices and cloud. The rationale of Edge computing is that computing should happen as near as possible to data sources. Edge computing is interchangeable with Fog computing [59], but Edge computing focus more toward the thing's side. Three possible approaches are: Cloudlets [22], MEC [23], and Fog computing [24]. In [37], the main differences are: Cloudlets are computers or clusters of computers with above average performance and well-connected to the Internet; MEC is an architecture for deploying a wide variety of applications on top of some host, and it is the key to the growth of the Internet of Things (IoT); Fog computing needs both cloud and Edge nodes (the others can exist without the cloud) and it runs applications for satisfying a specific use case.

1.3.2 Fog Computing

Fog computing enables computation, storage, networking, and data management on network nodes in close proximity to IoT devices, bridging the gap between the cloud and end devices. As a result, computing, storage, networking, decision making, and data management occur not only on the cloud, but also as data travels the IoT-to-Cloud path (preferably close to the IoT devices). In Intelligent Transportation Systems, for example, GPS data can be compressed at the Edge before being transmitted to the cloud [61]. The OpenFog Consortium defines Fog computing as [62] "a horizontal system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum. " In Fog computing, a "horizontal" platform permits computer operations to be disseminated across platforms and industries, whereas a "vertical" platform encourages segregated applications [63]. A vertical platform may be well-suited to a siloed application, but it ignores platform-to-platform interaction in other vertically focused platforms. Fog computing provides a flexible platform for operators and customers to meet their data-driven needs while also allowing for a horizontal design. Fog computing is intended to provide a solid foundation for the Internet of Things.
Even though both Fog computing and Edge computing shift processing and storage to the network's Edge and closer to IoT devices, the two paradigms are not interchangeable. In reality, the OpenFog Consortium claims that Edge computing is frequently mislabeled as Fog computing. Fog computing is hierarchical and provides computing, networking, storage, control, and acceleration everywhere from the cloud to the Edge, whereas Edge computing is limited to processing at the Edge [62] (Refer to Figure 1.3). Moreover, in a tutorial article [64] about Fog and Edge, the authors explain that "Fog is inclusive of cloud, core, metro, Edge, clients, and things, "

and “Fog seeks to realize a seamless continuum of computing services from the cloud to the things rather than treating the network Edges as isolated computing platforms, ” and “Fog envisions a horizontal platform that will support the common Fog computing functions for multiple industries and application domains, including but not limited to traditional telco services. ” [64]

1.3.3 Cloudlet computing

The Cloudlet is proposed by Carnegie Mellon University as a new element that extends the mobile device-cloud architecture, to meet the latency challenge of mobile cloud computing. As defined in [60], "a Cloudlet is a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices". Cloudlets are small data centers (miniature clouds, as the name suggests) that are typically one hop away from mobile devices A Cloudlet’s logical closeness to the mobile device is required to reduce end-to-end latency between the mobile device and the cloud. The network position is an appropriate place for the Cloudlet. Which could, for example, be placed in a cellular base station or a Wi-Fi access point (AP). The goal is to offload compute from mobile devices to network Edge VM-based Cloudlets. Although contemporary Cloudlet computing research is primarily driven by academia, it offers significant potential in areas such as wearable cognitive assistance and web applications companies.

1.4 Challenges and Issues of Big data applications in different computing paradigms

1.4.1 Data Durability

For Big Data applications, data Durability is a major challenge. Data loss can have a variety of implications, ranging from catastrophic failures of mission-critical programs to minor glitches. Storage architects have developed creative approaches to ensure data durability over the years, relying on a variety of strategies like as replication and log-based systems. At the level of disk storage, RAID (Redundant Array of Independent Disks) is a very popular solution [65]. To ensure data Durability, RAID systems integrate several disk drives with diverse data distribution algorithms. However, such technology falls short of real-world cloud scales [66]. To ensure data durability, cloud storage systems rely (at a higher level) on hinted Writes (based on logs) and geo-graphical replication. Glacier [67] is an Amazon service that allows you to store data for a long time. The data durability of Amazon Glacier is guaranteed to be 99.999999999 percentile. Its architecture is based on a massive number of geographically replicated tape libraries. Amazon Glacier, on the other hand, is not an online system because retrieving data takes hours.

1.4.2 Fast Access

The increase in data volume, variety, and velocity is accompanied by an increase in demand for high performance and quick access. Given the massive volumes of data, cloud providers and Big Data companies are employing a variety of ways to handle this problem. When compared to traditional hard drives, technological advancements in Flash Memory and Solid-State Drives (SSD) cut access time by around 8 times [68]. Novel storage system architectures rely on replication to give faster access to local copies of data at the software level. In the case of geographical distribution, Optimistic Replication [69] allows substantially faster access because data can be obtained immediately from a geographically close replica while being asynchronously propagated to further copies. Allowing in-memory processing within distributed storage systems is another way for enabling faster access. This allows storage users to access the majority of their data from RAM rather than drives. Memtables are in-memory data structures used by systems like Google BigTable [15] and Cassandra [19] to give quick access to a huge amount of data directly from memory.

1.4.3 Scalability

Cloud computing is a viable choice for Big Data applications if the services can grow to meet the needs of the computation. To support the expanding amount or veracity of Big Data sets, storage infrastructure must be able to efficiently utilize a large number of resources (e.g., virtual disks) and combine them elastically and constantly. At the same time, in order to minimize performance bottlenecks and idle resources, the burden must be spread proportionally with the resources (i.e. load balancing) [70]. This is true for handling data access requests as well as for scheduling compute activities. To achieve system scalability and load balancing, it's crucial to know how traditional techniques designed for smaller scales (e.g., in the field of distributed shared memory, cluster scheduling) can be leveraged, extended, and adapted to the unprecedented scale brought to reality by cloud infrastructures.

1.4.4 Low-latency and high-throughput access to data under heavy concurrency

For data analysis, Big Data processing necessitates a high degree of parallelism (i.e., many compute nodes concurrently access, process and share subsets of the input data). As a result, many application instances use the cloud storage system to receive input data, write results, report their state for monitoring and fault tolerance, and write their computation's log files. For many scientific applications, such data access have a significant impact on overall execution time. To provide low-latency and high-throughput access to data in such settings, existing cloud storage systems must be enhanced with means to enable extensive concurrent access to shared data while avoiding the often high expense of traditional synchronization mechanisms.

1.4.5 Stream processing

Stream processing is the management and analysis of continuous sequences of relatively small data items [71], often known as events, in real time. Given large amounts of data received in real-time, at rising rates in diverse circumstances, stream processing is quickly becoming one of the most important classes in the area of Big Data. Because of the continuous nature of the stream and often small amounts of the events that make it up, managing such data is a different problem than managing static or stored data. In reality, as reported in [72], an extensive examination of thousands of commercial projects and millions of machine hours of computing demonstrated that the performance of the stream process is strongly dependent on the management and transfer of events, as well as their latencies. As a result, there is an increasing demand for a high-performance system for cloud-based event streaming.

1.4.6 Security and Privacy

The current state of big data development is beset by a slew of issues that have been identified by people in recent years. Privacy and Security concerns have long persisted among them [73]. There is a huge amount of data nowadays that conventional hardware and software can't keep up with the massive number of different sorts of data being generated at breakneck speed. Traditional data technologies can no longer process, store, analyze, or manage big data because it has grown too complicated and dynamic. Then there's the issue of data privacy, which is worsening. Traditional security techniques, which are designed to protect modest amounts of static data, are insufficient. Anomaly detection analytics, for example, would yield much too many outliers. Similarly, utilizing existing cloud computing technologies to solve them is difficult. Because huge data poses security threats in terms of storage, processing, and transmission, among other things. It is necessary to protect data security and privacy. Big data security and privacy preservation, however, are more complex than other security challenges (such as data security in cloud computing). Big data adds to the variety, velocity, and volume of security management challenges that are already present in traditional security administration. Information from multiple sources across the company will most likely be stored in big data repositories. Managing secure access to such a diverse set of data is difficult.

1.4.7 Data Availability

Data is said to be available if all (or most) access requests result in a successful response [74, 75]. For many Big Data applications, availability is a must-have. Customers must constantly receive a response to their requests in order to avoid financial losses, hence web services such as Web shops require a high level of availability. For example, the average cost of IT downtime has been estimated to 5,600 $ per minute [76]. In this context, other architectures that guarantee high levels of availability at a large scale appeared. Amazon Dynamo storage [18] is used by the Amazon.com e-commerce platform [77], as well as many other Amazon Web Services, to achieve extraordinarily high levels of availability. For a peak client load of 500 requests per second, Amazon Dynamo sacrifices a few of the ACID strong semantics in order to provide a Service Level Agreement (SLA) guaranteeing that a response will be supplied within 300 milliseconds for 99 percent of the time.

1.4.8 Failure Tolerance and Disaster Recovery

Commodity hardware is often used in cloud storage infrastructures. As a result, failures of components are the rule rather than the exception. One of the most difficult tasks for Big Data platforms and infrastructures is to keep them functioning when one or more components fail.Fault tolerance is commonly achieved through redundancy and replication. RAID (Redundant Array of Independent Disks) is a frequently used solution. The scalability of these techniques, on the other hand, is a serious concern. Optimistic asynchronous Replication [69] is a popular technique that scales to large areas. This approach provides a fault-tolerant alternative to synchronous replication that allows for scalable performance. Furthermore, disaster recovery capabilities are provided by replicating data across multiple geographical locations and ensuring its durability. Within major cloud providers, it is now quite simple and inexpensive to replicate data across various continents. In this case, systems can recover even if most or all of the data centers in a geographical area have been compromised by a malicious actor.

1.5 Our Focus: Replication and Consistency

1.5.1 Replication

Replication consists in storing multiple copies of the same data on multiple storage devices or storage nodes. For Big Data, replication is a critical aspect because it is required to handle the Big Data issues in the era of planet-scale services and applications. It provides the necessary tools for ensuring data durability, enabling fast access via local replicas, increasing data availability, and ensuring fault tolerance. Data propagation from one replica to another has traditionally been done asynchronously. Fundamentally, data is immediately propagated to other replicas for each update or write before the operation is declared successful. However, with today's expanding sizes, this technique has shown its limitations, as it fails to deliver scalable performance, particularly at large scale replication. Because read and write operations must wait for answers from all replicas, regardless of their location, before returning a success, the need for synchronization, along with high network latencies, has a negative influence on operation latencies. To solve this problem, Optimistic (or lazy) Replication [69, 78, 79] was introduced. The propagation of data to other replicas (or a subset of replicas) is done lazily and at a later time. Update and write operations are allowed to succeed in this circumstance since the data has been committed locally but has not yet been fully propagated. Although, a transient replica divergence may occur as a result of this condition, Optimistic Replication enhances performance. The network latency has no effect on the latency of read and write operations. Furthermore, the storage system's throughput is not slowed down by the additional traffic generated by immediate data propagation to other replicas. However, this strategy raises a significant and difficult issue: data consistency.

1.5.2 Consistency

Data consistency is difficult to be guaranteed between different data copies. Traditional databases (RDBMS) and file systems are constrained by strict semantics (ACID for RDBMS and POSIX for file systems) where a strong data consistency is one of the primary requirements. Strong consistency is frequently achieved through synchronous replication, which ensures that consumers perceive replicas of the same data in the same state. This was not a problem at the early systems scales. Strong consistency however may be excessively costly in Big Data systems and applications, incurring a significant performance overhead and limiting data availability. To get around this issue, eventual consistency [80, 81] has gained a lot of attraction in recent years. It allows for some temporal inconsistency at times, but guarantees that all replicas will eventually converge to a consistent state. This is suitable for many modern Big Data applications, including a wide range of Web applications where update conflicts are few and easy to resolve. Furthermore, using Optimistic Replication improves performance and availability. In this context, various trade-offs regarding consistency emerge. They include the Consistency-Performance trade-off and the Consistency-Availability trade-off among others. Therefore, managing consistency within storage systems management is crucial especially how can they deal with Big Data challenges.

Conclusion

n recent years, the growth of data volumes has changed the way IT businesses manage their hardware and software resources. New computing paradigms and data processing platforms were introduced as a result. A few infrastructure types, such as Cluster, Grid, and Cloud Computing, provide the required tools to efficiently house, transmit, and analyze data to meet the demands of the developing Big Data phenomena. The Cloud Computing paradigm, in particular, provides the on-demand flexibility and elasticity of resource leasing that Big Data platforms require. To deal with the wide range of Big Data applications, it presents several models and service levels. Furthermore, replication is a crucial feature in the cloud to deal with data availability, data durability, fast access, and fault tolerance issues. Replication, on the other hand, raises the issue of data consistency between replicas. In this context, We highlighted the issue of consistency management as well as its significant impact and repercussions in dealing with Big Data challenges. The next chapter will study the adaptive consistency model as a solution to the trade-off consistency-performance. Moreover, it will highlight the existing adaptive consistency approaches, make a comparative study according to a set of proposed comparative criteria, and investigate the future research directions.

Chapter 2

First Contribution: A comparative analysis of adaptive consistency approaches in cloud storage

Introduction

NoSQL storage systems are used extensively by web applications and provide an attractive alternative to conventional databases due to their high security and availability at a low cost. High data availability is achieved by replicating data in different servers in order to reduce access time lag, network bandwidth consumption, and system unreliability. Hence, data consistency is a major challenge in distributed systems. In this context, strong consistency guarantees data freshness but affects directly the performance and availability of the system. In contrast, weaker consistency enhances availability and performance but increases data staleness. Therefore, an adaptive consistency strategy is needed to tune, during run-time, the consistency level depending on the criticality of the requests or data items. Although there is a rich literature on adaptive consistency approaches in cloud storage, there is a need to classify as well as regroup the approaches based on their strategies. This Chapter will establish a comparative analysis of existing adaptive consistency approaches. To achieve our goal, we first give the main concepts of consistency in distributed systems including adaptive consistency. Then, we define a set of criteria that describe the design or the behavior of such an approach. Thereafter, we review 19 proposed works by describing their main contributions and then summarizing in a table their behaviors towards the defined criteria. This comparative analysis aims to clarify the suitability of these approaches for candidate cloud systems. Furthermore, it provides researchers with challenges and open issues in this research area. This work is published in "Journal of Parallel and Distributed Computing" [21].

2.1 Context and Problem statement

With the fast development of processing and storage technologies and the success of the Internet, computing resources have become cheaper, more powerful and more available than ever before. This technological trend has enabled the realization of a new computing paradigm called cloud computing [3]. A client of the cloud can lease just the resources he needs in a Pay-as-You-Go manner [4] with very little knowledge of the physical resources. Nowadays, cloud computing is the best alternative to grid and cluster computing because it performs well with data-intensive applications [5, 6], and companies like Google, Amazon, and Facebook deal with peta- and terabytes of data every day. In this context, storage management and performance within clouds is extremely important.

In distributed systems, replica technology [7] is a key technology in enhancing performance of the system. With the assistance of replica and redundancy of data in several domains [8, 9, 10, 11, 12, 13], the distributed file system is able to reduce access time lag, network bandwidth consumption and system unreliability. However, this also leads to the problem of replica consistency management. Namely, when data is accessed and read by multiple users, inconsistency occurs in replicas, and as a result, the system's consistency and accuracy are directly influenced. The study on replica consistency aims to achieve synchronism among multiple replicas. The most popular strategies provided by storage systems are strong consistency and eventual consistency [14]. Strong consistency ensures all of the replicas to be updated immediately. There is no difference between replicas. Therefore, every access to replicas will get fresh data. But the cost of maintaining the freshness of replicas increases significantly, which cuts down the availability of replicas and the system's performance. For instance, Google BigTable [15], Microsoft Azure Storage [16] and Apache HBase [17] provide strong consistency. Eventual consistency doesn't update all the replicas immediately, so it tolerates replica divergence. But it promises all the replicas to be consistent at a specific time. For instance, Amazon Dynamo [18], Cassandra [19] and MongoDB [20] provide eventual consistency.

The data consistency can be encountered in cloud systems more than in non-cloud systems [82]. V.Cheng and G.Wills define in [82] a model to compare cloud and non-cloud storage of big data. One of their goal was to show the consistency between the actual and expected execution time on cloud and non-cloud systems. The paper confirms that there is no comparable consistency on a non-cloud system. Therefore, in this comparative study we will only focus on data consistency in cloud storage systems.

Providing one consistency strategy is only suitable for particular scenes since the clients of cloud storage are multifarious and not all the applications need the same level of consistency. In addition, a required consistency strategy of an application is variable at run-time. Take an online conference as an example [83], in which the data should be modified under strong consistency when an important speech is taking place. Otherwise, users accept eventual consistency for better performance. In this context, a self-adaptive approach is needed to dynamically adjust the consistency strategy according to the cloud system's dynamicity and the application's demands. Therefore, adaptive replica consistency can satisfy the application requirements and minimize the transaction cost at the same time.

Many works in the literature addressed the adaptive consistency and the trade-off between consistency and availability. The proposed approaches have used different protocols to provide adaptive consistency. Most of them defined a metric such as: read/write frequency [84], file heat [85], stale reads [86], etc. and used statistical or probabilistic models to calculate it. Hereafter, the system changes the level of consistency when the calculated value exceeds the defined threshold. Some approaches took monetary cost into consideration when they defined their metrics. However, others used version vectors [87] (information about update history on the data item) for every copy of data to choose the suitable consistency for every object.

To our knowledge, there is no work in this research area that established a comparative analysis of the existing adaptive consistency approaches. Since the works that we have reviewed described only a few proposed approaches as related works. Establishing a comparative study of the proposed approaches is necessary to propose a robust adaptive consistency approach. In the present work, we review the most popular adaptive techniques in cloud systems using a set of criteria proposed for this purpose.

2.2 Related work

This section discusses the main contributions of this work compared with the most relevant existing comparative studies by noting all considered elements around consistency in the cloud. Almost all of the comparative contributions that we have found on this topic describe just some proposed approaches as related work. On the other hand, we note that, in the literature, there is a lack of deep comparative analysis that established an exhaustive analysis of the principle exist adaptive consistency approaches. Hence, this section looks to evaluate the best known comparative adaptive techniques in cloud systems by using a set of criteria and measures for this purpose. In each comparative study between the proposed approaches we will focus on these main issues: granularity of consistency strategy, conflict detection and resolution, monetary cost consideration, threshold definition and adaptive policy.

Prior surveys [88, 89, 90] have covered different aspects of the adaptive consistency approaches in cloud storage. In particular, surveys published by *H-E Chihoub* et al. over the last few years elaborated on various topics within the consistency scope such as the concept, benefits and models [91, 92], the architecture elements and the design of the security challenges in adaptive consistency [88]. During the reviewing of some comparative analysis topics in some specific sections, we have noted that none of these surveys has particularly focused on the exhaustive analysis of the existing adaptive consistency approaches. While the data duplication may be implemented using the existing distributed adaptive consistency approaches, their great number along with their particular pros and cons made the choice extremely difficult for those who attempted to adopt a distributed data duplication architecture in a large-scale context. In order to assist and promote recent initiatives to put into practice the adaptive consistency paradigm, this survey proposes original classifications that make comparisons between the broad range of proposed platform solutions with respect to various availability, reliability and performance criteria.

In this context and in line with the vision of Shin et al. [88], three main key reasons for the problems studied in the management can be summarized as follows:

1. For the goal of high availability of the stored data, data replication across storage entities in an efficient way is used. However, these schemes may mitigate data lock-in mainly in case of the appearance of new data stores. Hence, replication is a solution that offers availability. In fact, this solution prevents other functionalities such as data lock-in, data protection and data erasure coding.

2. In case that data-intensive applications are deployed within a single data store, the problems of data overflow and network bottlenecks have appeared. The survey of Shin et al. shows some proposed solutions to solve this problem.

3. Data security is one of the strongest factor in cloud storage and data duplication. It concerns the implemented hardware, users' access and the designed architecture. Many parameters may be changed when the data security plays its role such as data availability, data privacy, data location and data storage size. To obtain secured distributed data in the cloud storage the authors have referred to many solutions.

In the survey [90], data consistency was tackled, but not as a primordial factor. The authors deal with the data management and they tackle the data consistency only in one section by defining the three main aspects, i.e., level, metric, and model. In the data consistency section, the authors show the taxonomy of data consistency by focusing on these three main aspects in order to determine how much a consistency is stronger than another. The authors explain in detail the I/O-level consistency which requires conservative assumptions like concurrent write-write and read-write upon the same data. This survey focuses mainly on this level of consistency. Furthermore, the authors of [90] have drawn a weak consistency taxonomy together with the influenced factors to encounter the data conflicts which happen as operations to the same data in multi-master systems. In fact, this survey studies the data storage management in cloud environments. The consistency was discussed as a secondary issue. Hence, this work is a global survey which requires the design of a common data model and standard APIs for different cloud databases to help application providers. To reach this end, the authors assume that the ideal solutions for this challenge are to complete data transfer within a budget and deadline especially for OLTP applications that demand high response time.

As another work, Viotti and Marko [89] defined a structured and comprehensive overview of different consistency notions that appeared in distributed systems research. Their work scope was restricted to non-transactional semantics that apply to single storage object operations. The paper aims to complete the existing surveys done in the context of transactional database consistency semantics. The authors define their own non-transactional consistency model of a distributed system for reasoning about different consistency semantics. In fact, this work gives a deep knowledge about consistency for non-transactional systems, called linearizability. The designed authors's hierarchy of non-transactional consistency models shows six main component models: Fork based, session, causal, per-object, Synchronized and Staleness-based models. In all component models, the authors focus on the explanation of the models use mode and discussion of the principal existing work for these component models. We observe that this survey maps the consistency semantics to different practical systems and research prototypes. Therefore, the contribution of Viotti and Marko is to classify the existing work of consistency in non-transaction distributed systems depending on the proposed hierarchy. However, it does not include a comparative study of the existing work.

2.2.1 The main contributions of this comparative analysis

The present work aims to fill this void by drawing a synthesized of comparative study of 19 proposed consistency approaches for cloud storage. We investigate the consistency in the cloud and take a strong overview of each analyzed consistency approach. Hereafter, the comparison of the different approaches proposed for adaptive consistency in the cloud is shown and discussed. We may cite the two comparative studies in [93] and [91]. In [93], the authors presented a literature review that summarized the consistency management in distributed systems, grids and the cloud. For each proposed adaptive consistency approach in the cloud, they gave a short description and criticized it. In [91], the authors defined three comparison criteria: the level at which the consistency is specified, the cloud storage system in which the approach is implemented and the test-bed which is used to evaluate the solution. Then, they compared the three approaches according to mentioned criteria.

2.3 Consistency in the cloud

Consistency concepts and its relationship with different storage system features, such as performance and scalability, has been widely addressed. We first review the definition consistency in traditional database systems. Then, we recall its definition in distributed systems including cloud systems. Finally, we review the consistency models and classifications.

2.3.1 Consistency in database systems

In traditional database systems, consistency is defined as a property of transactions [94][95]. It builds with Atomicity, Isolation and Durability the well-known acronym: ACID properties. Consistency refers to the fact that the transaction takes the system from one consistent state to another. Note that a transaction may violate some of the integrity constraints during its execution. However, once it terminates, it must restore the system to a consistent state. When transactions are executed concurrently, the transaction processing system must ensure that the execution of a set of concurrent and correct transactions also maintains the consistency of the data. Atomicity requires all the operations of a transaction to be treated as a single unit; hence, everything in a transaction succeeds or the entire transaction is rolled back. Isolation refers to the fact that transactions cannot interfere with each other, i.e., transactions cannot read the intermediate results of other transactions. Finally, durability requires the results of a committed transaction to be made permanent in the system.

2.3.2 Consistency in distributed systems

In distributed systems, consistency is defined in a trade-off with availability and partition tolerance in the CAP theorem [74, 75]. The theorem states that only two of the three properties can be achieved simultaneously within a distributed system. In this context, consistency refers to the requirement that the clients should have the feeling of working on a single node regardless of the number of replicas. This is equivalent to requiring a total order on all operations and operations act as they are executing on a single node. Availability means that every request sent by a client to a non-failing node should obtain a successful response. Partition tolerance means that the system should continue delivering its services even if some part of the system loses many messages arbitrarily and only the total network failure is allowed to cause the system to respond incorrectly.

2.3.3 The consistency's use areas and benefits

Big data storage and processing are considered as one of the main applications for cloud distributed systems. Additionally, the concept of the Internet of Things (IoT) paradigm has advanced the research on Node to Node communications and enabled novel tele-monitoring architectures for several applications such as E-Health, Forest monitoring and others. However, there is a need for safe data storing in decentralized cloud systems and IoT systems. The purpose of this subsection is to show the gathered benefits of consistency by citing some existing work and some methods of big data processing within cloud distributed systems.

Nowadays, cloud computing, fog computing, big data and the Internet of Things are popular paradigms and their features can be combined for shaping the next era of data processing, storing and forwarding. Many applications need a data replication model, where the data consistency is a challenge [96, 97, 98]. We may find many techniques which are proposed for systems restoring processes. In this kind of systems, a restoring process fires a transaction consistency across consistency groups, e.g., disaster recovery systems. Several consistency groups are defined for replication. For instance, the author in [96] demonstrates a multi-purpose approach for disaster recovery. The consistency is involved to check whether any data has been lost, damaged or corrupted in the disaster recovery process.

We may cite several works in the literature where data consistency plays an important role. In order to show its benefits in different fields, the data consistency discussion will be divided into three main classes.

1. With the aim to provide an efficient disaster recovery system, the storage manager must generate multiple recovery copies of the client file and oversee the transmission of this recovery copies to remote sites. What is needed in this situation, is a way to ensure the data consistency of the generated multiple recovery copies. The author of [96] has proposed a big data system disaster recovery which focuses on multi-site and multi-technique approaches to ensure that if one method does not succeed, there are other methods that can retrieve and restore the data. The author considers the the data consistency as an essential parameter in the experimentation and result. To reach this aim, the author has checked the data consistency of the proposed system during its experimentation with the aim to identify whether any data has been lost, damaged or corrupted in the disaster recovery process. The

authors in [99] studied techniques that maintain stored data consistency when migrating virtual machines from a first site to a second site. The main objective aims to relocate the virtual machines when any migration is planned from the source site to the destination site during maintaining a consistent data. The data consistency challenge can be particularly observed while maintaining a consistent set of data with a pair of consistent data images between the protected source site and the destination recovery site. The authors uses the disaster recovery as an exemplary scenario with the aim to avoid a disaster that is predicted to occur in the vicinity of the source site. Another work can be cited [100], in which the data consistency plays an important role. The authors of [100] have tackled the problem of backing up an entire micro-service architecture in their work, where a running application was decomposed into multiple micro-services in such a way that it can be recovered after a disaster strike. The main objective of this work aims to ensure, in case of disaster, the impossibility to achieve a holistic recovery that brings back all micro-services in a globally consistent state.

2. It is also necessary to know the data consistency advantages and uses in terms of security and privacy challenges. For example, in applications where different central entities collaborate with each other, they must be able to exchange trust information in order to fix inconsistencies in the reputation values [101]. For instance, any time instantiation of edge devices in fog computing can be used by multiple smart applications with set of users which raises the issue of edge device security and makes appear the consistency challenges [102]. The authors of [103] consider the consistency of data to be a primordial factor in several security aspects. The data integrity needs the action of maintaining consistency, where ethical values are important for cloud service providers to protect integrity of cloud user's data with honesty, truthfulness and accuracy at all time. The work of [104] has studied the eventual consistency as a relaxed trade-off model in order to link the strong consistency and availability. The authors of this work introduce *Byzec* which is a protocol that makes eventual consistency as secure as possible. The proposed protocol allows the service to run in an eventually consistent manner.

3. Software Defined Networking (SDN) is a promising network paradigm that aims to solve such problems and accelerate innovation. Its principal is to decouple the control plane and the data plane to achieve a logically centralized control architecture providing programmability to configure the network [105]. We may cite several works focussing on the design choice of distributed SDN controller with the aim to show that to build an efficient distributed controller

it should consider the following aspects: scalability, robustness, consistency and security [106, 107, 108, 109]. The papers of [110, 111] define the use of adaptive consistency models in the context of distributed SDN controllers. The authors of [110] aim to show how the typical SDN controller architecture can be extended in order to support adaptive consistency. The obtained results showed that adaptive controllers were more resilient to sudden changes in the network conditions than the non-adaptive consistency controllers. However, in [111], an adaptive consistency model for SDN controllers that employs concepts of eventual consistency models has been introduced by defining the concept of runtime adaptation of consistency levels in state synchronization for a distributed SDN control plane.

2.3.4 Consistency models

In this subsection, we introduce the main consistency models adopted in earlier single-site storage systems and in current Geo-replicated systems. Many works addressed this consistency models such as [91], [80] and [112]. They call the highest level strong consistency, and the lowest level weak consistency. Between strong and weak consistency, they define other models that provide better performance than strong consistency and fewer conflicts than weak consistency. In the following, we adopt the classification of Werner Vogels [80] due to its powerful discrimination and characterization of models. He states that there are two ways to view consistency. The first is from the client point of view: how the client observes writing operations. The second point of view is from the server side: how the system manages updates and which guarantees are provided with respect to updates.

1. *Client-Side Consistency*: this kind of consistency investigates in how the client observes the changes. For instance let us assume that there are three independent processes: A, B and C which need to communicate to share information as shown in Figure 2.1. When process A updates a data item, there are three possible cases:

   - *Strong consistency*: An access by any process, after completeness of the update, will return the recent data (Figure 2.2). In other words, strong consistency guarantees that all replicas are in a consistent state immediately after an update.
   - *Weak consistency*: The system does not guarantee that subsequent accesses will return the recent data (Figure 2.3). The period between the write operation and the moment when it is guaranteed that any process can see the recent data is called the inconsistency window.

- *Eventual consistency*: After a window time, the storage system guarantees that all processes will see the recent data if no new updates are made to the object (Figure 2.4). This is a specific form of weak consistency and the size of the inconsistency window can be determined based on factors such as the load on the system, communication delays and the number of replicas in the system.

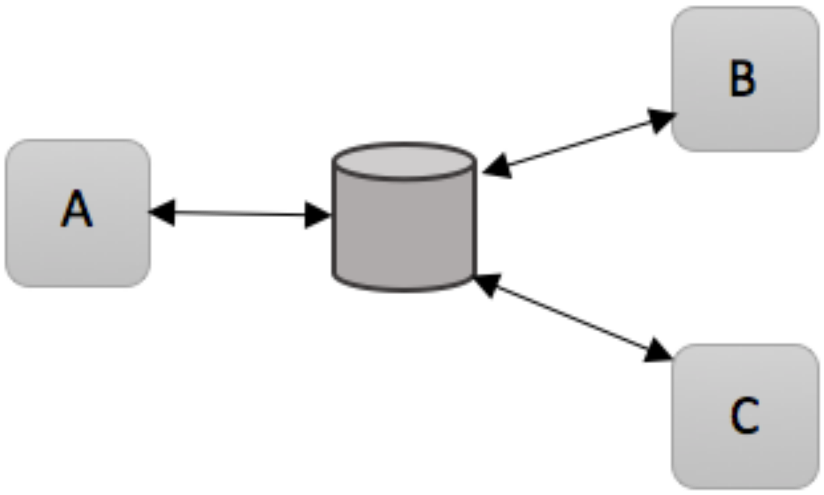

Figure 2.1: Model.

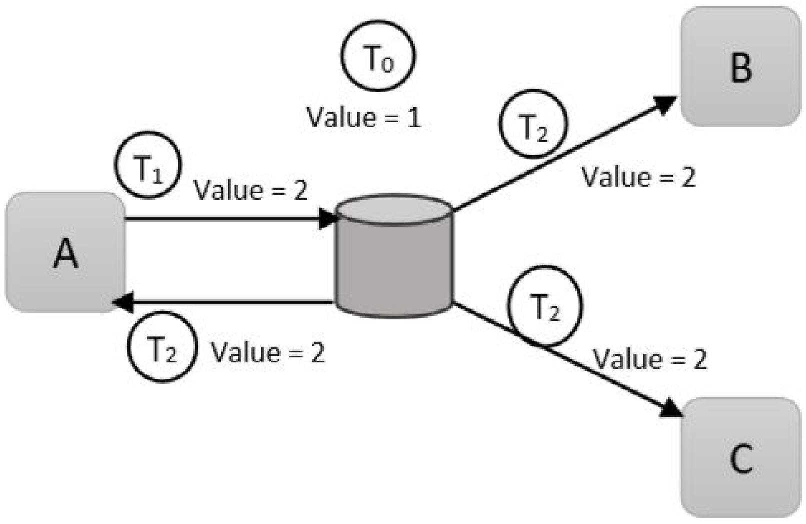

Figure 2.2: Strong consistency.

2. *Server-Side consistency*: On the server-side, the consistency deals with how the updates flow through the system to differentiate the modes that can be experienced by application developers. For instance, let us suppose that N is the number of replicas in the system, W the number of replicas involved in a

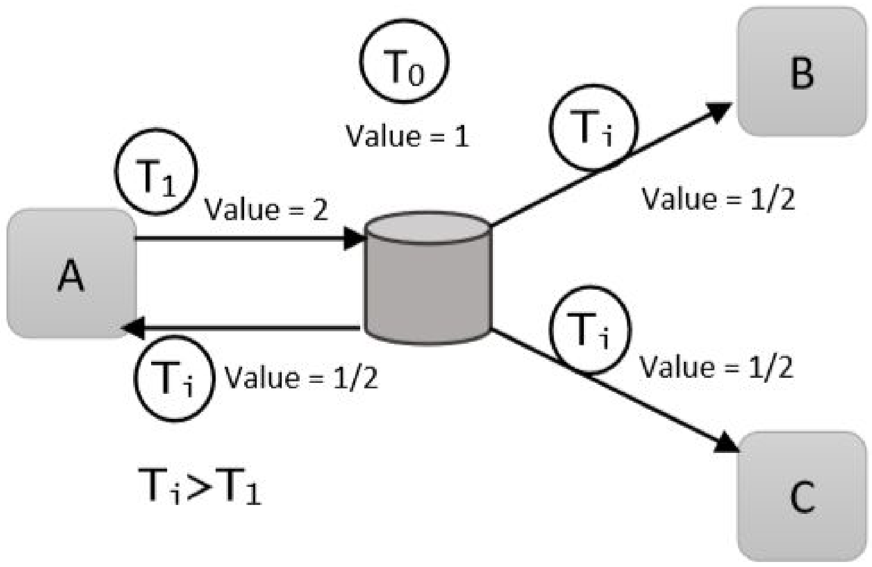

Figure 2.3: Weak consistency.

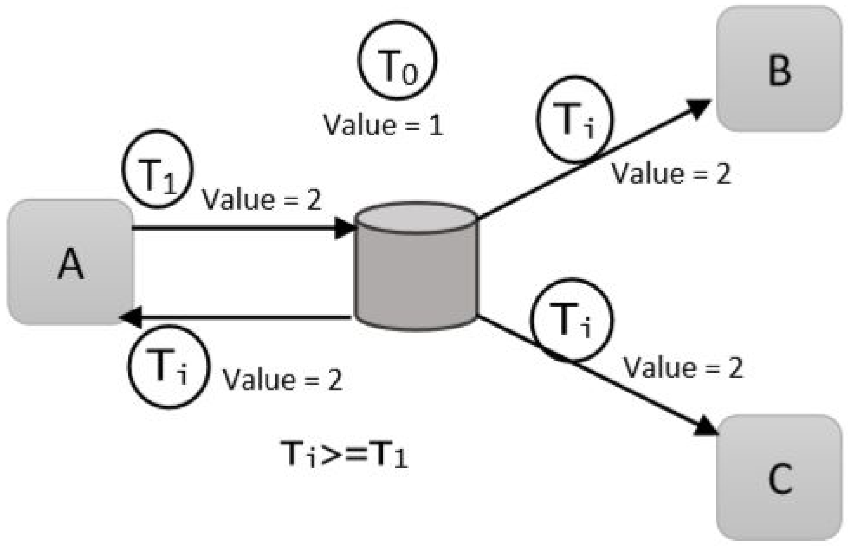

Figure 2.4: Eventual consistency.

write/update operation and R the number of replicas that are contacted when a data object is accessed through a read operation.

- *Strong consistency*: if $W + R > N$, the system will provide strong consistency and two cases are possible in this formula:
  - *Case 1*: If the number of replicas is 5 (N = 5) and the number of responses required to complete write queries is 5 too (W = 5), it is sufficient to read from one replica (R=1) to get the most recent data $W + R > N$.
  - *Case 2*: If the number of replicas is 5 (N = 5) and the client requires just two replicas to response to his write queries (W=2), as shown in Figure 2.5,and if he reads from more than three replicas during a read operation ($R > 3$), then there will be at least one replica that will give the most recent data and the system will still provide strong consistency $W + R > N$.
- *Weak consistency*: If $W + R \leqslant N$, the system provides Weak/eventual consistency. This means that there is small number of replicas that are guaranteed to have the latest write, and during a read operation, it is less likely to read from a replica that has the latest write (Figure 2.6).

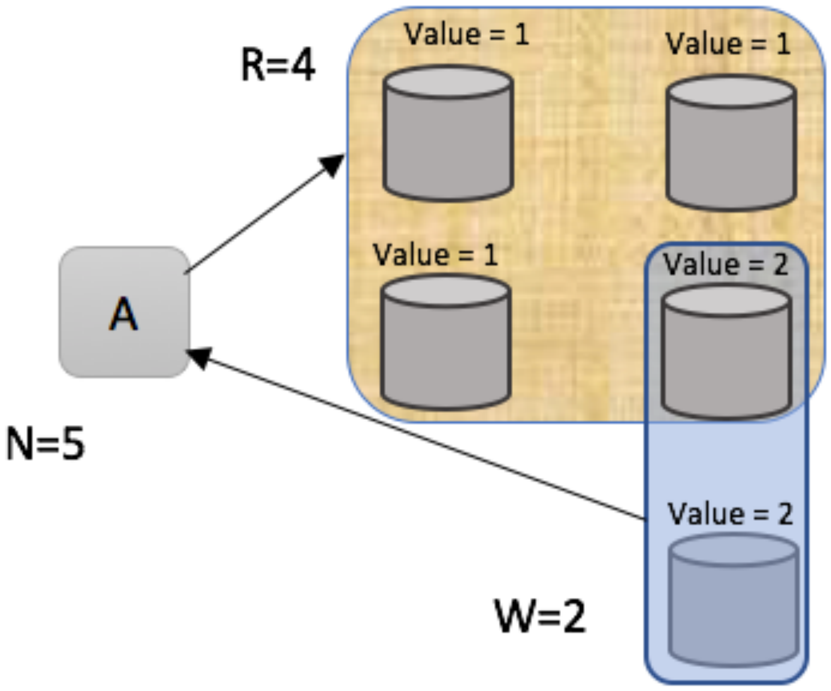

Figure 2.5: Strong consistency.

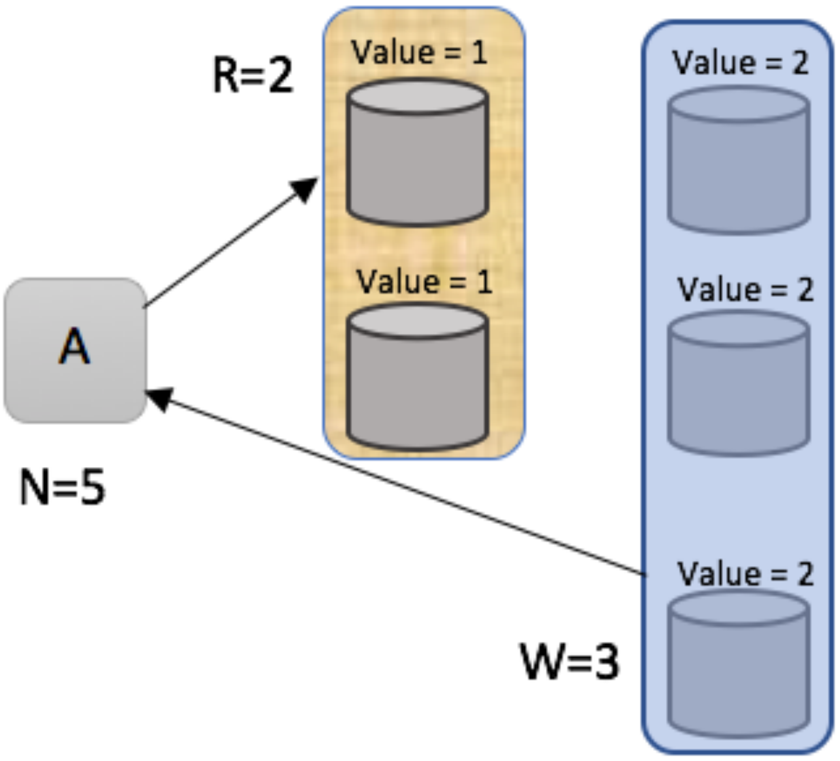

Figure 2.6: Weak consistency.

2.3.5 Adaptive consistency

In modern database systems, strong consistency generates always an additional cost on request latency, availability and scalability of the system. In order to find a trade-off between consistency and both performance and availability of the system, most of the modern database systems guarantee eventual consistency by default and allow increasing the level of consistency according to client needs. Increasing the level of consistency ensures better data consistency but deceases the system performance and availability. Therefore, instead of relying on a single consistency level, tuning the consistency level with other supplementary consistency options makes the system more proficient. This phenomenon of using the appropriate consistency option depending on the criticality of the requests or data items is known as adaptive consistency [112].
Several kinds of applications need adaptive consistency for example Web shop applications store different kinds of data that need different consistency levels. The customer's credit card information and the price of the items must be handled under strong consistency. However, clients buying preferences and logging information could be handled under weak consistency. An online auction system is another example that needs adapative consistency but the selection of a consistency level in this case is based on time. At the beginning, the data is not so important for the

final deal, so the requirement of consistency is low. With the deadline approaching, auctioned items become very popular and the customers rely more and more on the latest data to make the next bid. Eventually, the data should be always up-to-date and modified under strong consistency.

2.4 Comparison criteria of adaptive consistency approaches

When talking about adaptive consistency we distinguish numerous criteria that are to be described. Among these issues we are particularly interested in the following ones: architectural model, operation level, granularity, adaptive policy, prediction or statistic based, conflicts consideration, threshold, monetary costs, security and privacy, provided consistency and implementation tools. These criteria have a direct influence on the development of adaptive consistency approaches.

2.4.1 Architectural model

Consistency management is a functionality of the storage system that is queried by cloud clients. Thus, each proposed consistency approach should take a position in the storage system architecture or his interaction with client applications. According to their architectures, the proposed adaptive consistency approaches come in two forms: either an additional module between the storage system and the application or an entire storage system. In the first case, the additional module can be on the top of the storage system, in bottom of client application or as a middleware.

2.4.2 Operation level

This criterion defines the side on which the approach is applied and divides the approaches into two categories: query side approaches and object side approaches. Approaches in the first category focus on the query or the transaction level which means that the consistency level in these approaches is tuned according to the application needs. The object side approaches, however, define the consistency guarantees on the data by dividing them into categories and treat each category differently depending on the provided consistency level.

2.4.3 Granularity

Granularity means that the consistency level is applied to the hole database or just a part. In fact, among the proposed approaches some work on the global database, others fragment the database onto categories and there exist who works on file.

Applying strong consistency to part of the system or critical objects only can give higher performance and optimize the cost of implementation.

2.4.4 Adaptive policy

An adaptive policy is a set of algorithms, models or techniques that are used to provide adaptive consistency. It captures the system's need and gives the best trade-off between consistency and both performance and availability by tuning the consistency to the suitable level during the execution time. These policies are based on probabilistic models, artificial intelligence techniques or other predictive algorithms. Choosing a good adaptive policy leads to a pertinent adaptive consistency approach.

2.4.5 Prediction or statistic based

The adaptive policy may be based on prediction or statistic. Predictive policy uses probabilistic models, regression models or other intelligent techniques to predict the system state. This kind of models is very powerful in performance but it does not give exact results. However, the statistic model calculates the rate of a given metric during runtime and compares it to a defined threshold to change the consistency level when it is achieved. The statistic model can give exact results but it produces additional and complicate calculations.

2.4.6 Conflicts management

We chose this criterion to investigate wether the proposed approach has considered or not conflicts between transactions . We define which mechanism was used to detect and resolve the conflicts. Normally, conflicts between replica occur highly when the storage system runs transactions optimistically i.e., provides a low level of consistency. Contrarily, raising the level of consistency decreases the percentage of conflicts.

2.4.7 Threshold

When an adaptive consistency policy uses a metric, it defines a minimum or maximum value for this metric. Reaching the threshold triggers the system to change his behavior. Defining a threshold in such an approach means that the system does not calculate the application needs and that the tuning parameters are put manually.

2.4.8 Monetary costs

The cost of a consistency level in the cloud describes the different resources necessary to obtain this level in geo-replicated storage systems. Different consistency levels result in different costs; high consistency implies high cost per transaction and reduced availability but avoids penalty costs. Low consistency leads to lower costs per operation but might result in higher penalty costs. The monetary cost is one of the most interesting advantages of cloud storage. Hence, any consistency approach should take this criterion into consideration.

2.4.9 Security and privacy

Security and privacy are always key concerns in information technology to which more challenges are brought about by cloud computing and big data. In fact, moving data and applications to a third party service provider and replicating objects makes the challenge more critical. Furthermore, the availability of cloud services is one of the most important security issues which directly affects the business of cloud service providers and also its customers. Thereby, any consistency approach should take into consideration these two factors.

2.4.10 Provided consistency

Data consistency is already provided by storage systems which can ensure many levels of consistency. Thereby, the application can choose the suitable level for its needs from the beginning. Adaptive consistency approaches were proposed to optimize consistency management by tuning the consistency to the suitable level during the run-time. To achieve this goal, existing adaptive approaches behave in three manners: switch between existing consistency levels to assign the nearest one to the application needs and the system state, moderate the weak level by an artificial delay to enhance consistency when it is needed, provide a new level according to the application needs.

2.4.11 Implementation tools

Evaluating a proposed approach in cloud and big data environments is quiet difficult due to the high monetary charges and the large amount of data needed to complete the implementation. Nevertheless, using real implementation tools gives more credibility to the evaluation results. Since they are free of charge and easy to use, simulation tools are widely used to evaluate contributions and to emulate real environments. However, the obtained results are less credible.

2.5 Proposed approaches for adaptive consistency

In the literature, many approaches were proposed to adjust the consistency level in the cloud. In this section, we discuss the different approaches and their characteristics according to different criteria defined in the previous section.

2.5.1 IDEA

The infrastructure IDEA [113] (an Infrastructure for Detection-based Adaptive consistency guarantees) was presented to guarantee the adaptive consistency of replicated services. IDEA enables many functions including quick inconsistency detection and resolution, consistency adaptation and quantified consistency level guarantees. For each object of the system, IDEA divides the nodes into two layers where the top layer includes those nodes that frequently update this object and the bottom layer consists of the remaining nodes. The inconsistency detection and resolution policies are based on a new extended version vector where a triple of information is attached for every object (numerical error, order error, staleness).

2.5.2 Consistency rationing

Consistency rationing [114] divides the data into three consistency categories: A, B, and C. The A category ensures strong consistency guarantees and shows high cost per transaction. The C category ensures session consistency, shows low cost, but will result in inconsistencies. Data in the B category are handled with either strong or session consistency depending on the specified policy. In this paper, the authors present and compare many policies to switch consistency guarantees (conflict probability, time policy, fixed threshold, demarcation policy and dynamic policy). The optimization in this paper is based on allowing the database to exhibit inconsistencies if it helps to reduce the cost of a transaction but does not cause higher penalty costs. The aim is to find a trade-off between transactions cost in the case of strong consistency, and penalty costs (financial) in the case of weak consistency.

2.5.3 An application-based adaptive replica consistency

This paper [84] proposed an adaptive mechanism for consistency management that allows the system to automatically switch between consistency strategies according to update frequency and read frequency at runtime. The authors also proposed also a model structure that can manage the different consistency levels. The nodes in the proposed structure are put in the following order: one master node, three deputy nodes and many child nodes. The adaptive consistency mechanism divides the consistency levels into four categories according to statistical read frequency P_r and update frequency P_w(a combination of high and low values for every one). For each checkpoint interval τ, the system evaluates the consistency category that the application needs according to P_r and P_w in the interval. If the category is not the same as the current one, it will be changed in the next interval. They demonstrated, via simulations, that the adaptive approach reduces significantly the global throughput generated when applying strong consistency as well as it provides better performance than weak consistency.

2.5.4 RedBlue

RedBlue consistency [115] was introduced to provide an adaptive consistency approach by decoupling operations into two types: red operations that require a strong consistency and blue operations that are executed under weaker consistency. To color an operation red or blue, the authors define the conditions that ensure the non-violation of application constraints and the convergence of all replicas to the same final state. Intuitively, commutative operations may be blue if they do not violate the application constraints. An extension of the proposed approach consists in dividing original application operations into two categories: a generator operation that has no side-effect and which is executed only at the primary site, and a shadow operation, which is replicated to all sites. Only shadow operations are colored red or blue.

2.5.5 Harmony

Harmony [86] is based on the estimation model of stale reads that will be adjusted to the application needs. In this approach, the application provides the appropriate stale read rate (*app_stale_read*) and the modules added by Harmony compute the stale read rate of the system (θ) using a probabilistic model and compare it with *app_stale_read*. If θ is the greater value, the algorithm calculates the number of replicas (N) needed to attenuate the stale read rate and modifies the consistency level according to the obtained N.

2.5.6 Bismar

Bismar [92] takes the monetary cost into consideration, both for the evaluation and selection of consistency levels in the cloud. Accordingly, the authors define a new metric called consistency-cost efficiency. Based on this metric, they propose an economic consistency model, called Bismar, that adaptively tunes the consistency level at run-time in order to reduce the monetary cost while simultaneously maintaining a low fraction of stale reads. The proposed approach uses a probabilistic consistency model to estimate the stale reads and the relative costs of the application according to the current read/write rate and the network latency. Then the algorithm selects the consistency level that offers the most equitable consistency or cost trade-off (given by the maximum consistency-cost efficiency value: max[consistency(cl)/cost(cl)]).

2.5.7 Pileus

Pileus [116] is a key-value storage system that supports consistency-based service level agreements (SLAs). With SLAs, applications can declare their consistency and latency priorities. With the offered consistency choices, applications can indicate a decreasing series of desired consistency/latency trade-offs (SubSLAs). Furthermore, applications that share the same data are allowed to obtain different consistency guarantees. Pileus tries to satisfy the best desired service due to the configuration of replicas and the network conditions. It may not always succeed in satisfying the first subSLA. Therefore, Pileus tries out lower subSLAs that are acceptable but less desirable. Finally, the system returns an error code and no data if none of the subSLAs can be satisfied.

2.5.8 DepSky

DepSky [117] is a virtual storage cloud, a cloud of clouds, which is accessed by its users to manage data items stored in a group of individual clouds. It targets four limitations of individual clouds: availability, consistency, privacy and monetary costs. The DepSky system enhances data availability and consistency by exploiting data replication on several clouds and by introducing proportional consistency. Thus, the system allows access to the data whereas a subset of them is reachable and provides the same level of consistency of the subordinate clouds. To minimize the monetary cost and improve the privacy, DepSky uses erasure codes [118] to store only a fraction (typically half) of the total amount of data in each cloud and to avoid storing clear data. Furthermore, stored data are encrypted and the encryption key is shared by the underlying clouds so that individual faulty clouds cannot reconstruct it to disclose the data.

2.5.9 File heat-based self-adaptive replica consistency

This work proposed the algorithm MRFU [85] (Most Recent and Frequently Used) to calculate the file heat during an interval of time I. The file heat in this algorithm is a combination of access time and access frequency. The self-adaptive consistency strategy proposed in this paper selects the consistency level of a file between strong and eventual consistency according to the file heat. The value of file heat is calculated during a time interval I and compared with a threshold. The strong consistency strategy is adopted when file heat exceeds the predefined threshold, and the eventual consistency strategy is adopted when file heat is under the threshold value to cut network bandwidth consumption. The authors also proposed a replica management model that divides the system into three levels: one storage control node that controls many main replicas, each of which is connected to many other subreplicas.

2.5.10 Consistency tuner

Consistency tuner is a protocol based on the consistency index [119] (number of correct reads/total number of reads). To adjust the consistency index to a desired value the protocol predicts the correctness of an incoming read request using a logistic regression classifier and a neural network classifier. These statistical predictive models use the number of replicas R_p and the time gap G_p between the read and the last update as input parameters. The authors also implemented the CICT (consistency index based consistency tuner) in an architecture of a web based database application and demonstrate the relationship between the number of replicas and the threshold of a time gap (minimum value of time gap between an update and a succeeding read request) using a statistical linear regression analysis.

2.5.11 Fine-tuning the consistency-latency trade-off

The trade-off consistency-latency is addressed by proposing and evaluating two techniques [120]. The first is a novel technique called continuous partial quorums (CPQ) that assigns the consistency level on a per- operation basis by choosing randomly between multiple discrete consistency levels with a tunable probability. The second technique, called artificial delays (AD) uses a weak client-side consistency level and injects an artificially tunable delay into each storage operation. This technique boosts consistency by allowing more time for updates to propagate through the system, which decreases the likelihood of consistency anomalies at the cost of increasing latency.

2.5.12 A self-adaptive conflict resolution with flexible consistency guarantee

This paper [93] presents an adaptive and hierarchical model using version vectors of replicas to ensure consistency management. The proposed approach divides the consistency management into two levels: global management between data centers in the cloud and local management in the data center. For the local management, the multi-agent technology is used to modulate the different parts of the system which tunes the consistency between three levels (optimistic, hybrid and pessimistic) according to the rate of write operations. The authors also propose the integration of a conflict detection and resolution mechanism between the replicas based on version vectors. This mechanism resolves the conflict in the local data center and then the conflicts between different data centers in the cloud.

2.5.13 Incremental consistency guarantees

ICG [121] targeted the consistency/performance trade-off by developing an abstraction interface, called *Correctables*, between applications and replicated objects. Therefore, rather then struggling with the complexity of distributed storage protocols, developers will just focus on tuning the consistency level of the ongoing operation. After invoking an operation on a replicated object, the abstraction interface provides a view on the operation result for each specific consistency level. Primary obtained views reflect operation results under weak consistency while stronger consistency will be guaranteed within latest views.

2.5.14 Safe serializable secure scheduling

The paper [122] addresses the trade-off between strong consistency guarantees and strong security properties in decentralized systems. The authors state that distributed transaction scheduling mechanisms cannot prevent unauthorized access to confidential information. Security risks are potential due to the aborting messages of failed transaction. Thus, they proposed the staged commit protocol that can secure the transactions scheduling using relaxed monotonicity. The defined protocol divides a transaction into stages, each of which can be securely committed using a traditional protocol.

2.5.15 OptCon

OptCon [123] is a machine learning-based framework that can automatically predict a matching consistency level that satisfies the latency and staleness thresholds specified in a given service sevel agreement (SLA). For this reason, OptCon provides the following dynamic parameters as input variables to the learning algorithms: the read proportion in the operation, the number of user threads spawned by the client, and the number of network packets transmitted during the operation in addition to the client-centric consistency level. Many machine learning techniques were implemented in OptCon: to visualize the significance of the model parameters and the dependency relations among these parameters, it used Logistic regression and Bayesian learning. Furthermore, for more accurate predictions computed directly from the data, OptCon implemented Decision Tree, Random Forest, and Artificial

Neural Networks (ANN). The framework provides to users and developers the choice of a suitable learning technique that best suits the respective application domain and use case.

2.5.16 SPECSHIFT

SPECSHIFT [124] is a tuning framework that uses artificial delays to adjust the consistency level according to network conditions and workload characteristics. The delay is calculated from a combination of empirical measurement and probabilistic analysis and injected at the beginning of each read and at the end of each write. Developers of this framework also presented a probabilistic model of consistency under latency optimized settings that precisely captures the relationship between consistency, system workload and network latency.

2.5.17 Selective data consistency

The paper [125] proposes a selective model for transactional application data consistency. The model is built on the replicated data store MongoDB. To boost application performance, strong consistency is only applied to selected critical data objects. However, less critical objects can have lower consistency levels. The second contribution of this paper is the mathematical function that analyses data criticality using the number of reads and the number of writes of data items. This function is designed for a shopping application and considers three indices for determining total criticality: popularity index, stock sales index and stock purchase quantity index.

2.5.18 Adaptive consistency policy for kafka

Kafka [126] is a distributed streaming platform that allows to build real-time data pipelines and streaming applications. The work [127] proposes a replica adaptive synchronization strategy for Kafka based on the message heat and replica update frequency. For every period of time T_i, the heat and the update frequency of a partition (unit of data replication) are calculated following particular formulas. If the partition heat is greater than the update frequency, synchronous replication (high consistency level) should be granted for this partition. Otherwise, the partition will receive updates asynchronously.

2.5.19 FogStore

FogStore [128] is an extension designed on top of existing distributed storage systems to allow their seamless integration into a Fog Computing environment.

To provide best consistency management in Fog nodes, Fogstore ensures two additional functionalities: Fog-aware replica placement and context-aware differential consistency. A replica placement strategy should optimize the placement to achieve minimal response time between the copy of a data record, the data sources (devices) and the clients. Context-aware differential consistency matches the consistency level of the client's query to the client's context. In fact, clients querying the Fog data store often have an individual context, the position in particular, which can influence their requirements on consistency.

2.6 A comparison of the different approaches proposed for adaptive consistency in the cloud

Table 2.1 summarizes the evaluation of adaptive consistency approaches according to the criteria.

	The comparison criteria									
adaptive consistency approaches	Architecture	Operation level	Granularity	Adaptive policies	Prediction	Conflicts management	Metric in which the threshold is defined	Monetary cost	Security and privacy	Provided consistency	Implementation tools
IDEA [113]	Middleware	Object	Object	Statistical model(Version vector)	×	Uses version vector to detect conflicts	Implicit on calculated consistency value	×	×	New level	Planet-Lab
Consistency rationing [114]	Middleware	Object	Category	Probability, Time policy, Fixed threshold, Demarcation, Dynamic policy	×	Uses conflict probability as a metric	Conflict probability, fixed threshold, dynamic policy	✓	×	Switch between strong and session consistency	Amazon EC2, S3, TPC-W
ABARC [84]	Entire CMS	Query	Global	Statistical model(Read/write frequency)	×	×	Time between two writes	×	×	Moderated level	OptorSim
RedBlue [115]	CMS + MySQL as Backend	Query	Global	Coexistence of multiple consistency levels	×	×	×	×	×	Switch between many consistency levels	Gemini, MySQL
Harmony [86]	Middleware + ACSM	Query	Global	Probability	✓	×	×	×	×	New level	EC2/Grid'5000, Cassandra, Ycsb
Bismar [92]	Middleware	Query	Global	Probability, Consistency-cost efficiency	✓	×	×	✓	×	New level	EC2/Grid'5000, Cassandra, Ycsb
Pileus [116]	Entire CMS	Query	Global	Coexistence of multiple consistency levels	×	×	×	×	×	Switch between many consistency levels	Pileus, Ycsb
DepSky [117]	Middleware	Query	Global	Proportional consistency	×	×	×	✓	✓	Proportional consistency	DepSky, Amazon S3, Windows Azure, RackSpace, Nirvanix
FHBC [85]	Entire CMS	Object	File	Statistical model(file heat)	×	×	File heat	×	×	Switch between strong and eventual consistency	OptorSim
Consistency Tuner [119]	Middleware	Query	Global	RBFNN, Logistic regression, Linear regression	✓	×	Time gap	×	×	New level	Amazon SimpleDB, TPC-C
CPQ [120]	ACSM	Query	Global	CPQ + Probability	✓	×	×	×	×	Choosing randomly between weak and strong consistency	Amazon EC2, Cassandra, Ycsb
Artifical delay [120]	ACSM	Query	Global	Weak consistency level + Tunable Artifical delay	×	Uses Artifical delay to decrease conflicts rate	×	×	×	Moderated level	Amazon EC2, Cassandra, Ycsb
SACRFCG [93]	Entire CMS	Query	Global	Statistical model(write rate)	×	Uses version vector to detect conflicts	Write rate	×	×	Switch between weak and strong consistency	CloudSim (v3)

ICG [121]	Middleware	Query	Global	Coexistence of multiple consistency levels	×	×	×	×
4S [122]	Entire CMS	Query	global	Staged commit protocol	×	√	×	×
OptCon [123]	Middleware + ACSM	Query	Global	Logistic regression, Bayesian learning, Decision Tree, Random Forest and ANN	√	×	Latency, staleness	×
SPECSHIFT [124]	Middleware	Query	Global	Probability	√	Uses artificial delay to decrease conflicts rate	×	×
SCM [125]	Middleware	Object	Critical data	Criticality analyser	×	×	×	×
ACPK [127]	Middleware	Object	Partition	Statistical model(topic heat and update frequency)	√	×	Topic heat and update frequency	×
FogStore [128]	Middleware	Query	Regions	Client context-awareness	√	×	×	×

ABARC:An Application-Based Adaptive Replica Consistency.
FHBC: File heat-based self-adaptive replica consistency.
SACRFCG:Self-adaptive conflict resolution with flexible consistency guarantee.
ICG:Incremental Consistency Guarantees.
SCM: Selective Consistency Model.
ACPK: Adaptive consistency policy for kafka.
4S:Safe Serializable Secure Scheduling.
√ : the approach used the criterion.
× : the approach didn't use the criterion.
CMS : Consistency Management System.
ACSM : Additional Client-Side Module.

Tableau 2.1: Summary table of evaluation criteria.

2.7 Discussion

We have given a review of the works that propose adaptive approaches to managing consistency in the cloud. The main goal of these approaches is to optimize the performance of cloud systems and to provide a suitable consistency level to client applications.

According to the architectural model of the proposed adaptive consistency approaches, we state that most of the approaches provide adaptive consistency as an additional functionality to the existing consistency policy. Since data consistency is already managed by storage systems, this functionality is injected in the middle between a server and a client application in the form of middleware. Some approaches add a module to the client application to communicate with the middleware like in *Harmony* and *OptCon*. However, authorś approaches implement necessary adaptive functionalities in the client side module like in *DepSky, CPQ*, and *Artificial Delay*. On the other hand, few propositions have designed an entire consistency management system such as *ABARC, Pileu, FHBC, SACRFCG, and 4S*. Approaches that add adaptive consistency functionality to the existing storage system are more effective, either as a middleware or on the client side. In fact, rather than facing all challenges of designing an entire consistency management system, these approaches focus only on the adaptive consistency challenge and avail of the existing consistency management. Moreover, the additional module can be tested in the real environment contrary to the consistency management system which is usually tested by simulation tools.

Most of the studied approaches apply the chosen consistency level to the overall system with the exception of six approaches: *Consistency Rationing* divides the system into categories, *SCM* applies strong consistency on critical data, *ACPK* divides the system into partitions, *ForStore* divides the system into regions and *FHBC* and *IDEA* focus on the object. These works tune the consistency level according to data importance unlike the other works that focus on application needs. Only the *FogStore* approach considers client and data contexts at the same time. Categorizing data objects according to their criticality allows the system to apply lower consistency levels to lower critical categories, and therefore, enhance performance mainly in the case of high-volume data with a small percentage of critical data. On the other hand, data categorization may adversely affect system performance if it relies on a statistical model or other complex functions.

To optimize the consistency/performance trade-off, the proposed approaches use intelligent techniques that predict the suitable consistency level like *Harmony, Bismar, Consistency Rationing, CPQ, OptCon, SPACHIFT, and ACPK*. However, other approaches use statistics like *Consistency Tuner, ABARC and SACRFCG.*

Unlike these approaches, *FHBC, IDEA, 4S, FogStore and ICG* use particular protocols to enhance consistency management. Due to the large volume and variety of data, applying statistical models obligates the system to collect statistics for each operation or data object which will generate huge additional charges in processing and storage. Therefore, approaches that use machine learning techniques overcome statistics inconvenience by treating only a subset of data (training data) to generate the predictive model. Thus, intelligent techniques enhance performance but provide less accurate prediction.

When talking about conflicts, two approaches use version vectors to detect conflicts between different replicas of the same object: *SACRFCG and IDEA*. *AD and SPECSHIFT* use artificial delays to allow more time for update propagation through the system, which decreases the likelihood of conflicts. *Consistency rationing* uses the probability of conflicts as a metric that is used to tune the consistency level when it is compared with defined thresholds. *Staged protocol* proposed in *4S* aims to reduce aborting messages and thereby it reduces the conflict rate. Traditional conflict management mechanisms become more and more complex when moving to the cloud and big data environments as they save all historical events for each object. Artificial delay and probability represent an alternative solution of traditional mechanisms as they have an optimistic behavior to predict and as they reduce most conflicts without affecting performance.

The threshold is used by several approaches as a trigger to change the consistency level when it is achieved by the value of the defined metric. Among these approaches we find: *consistency rationing, consistency tuner, ABARC, FHBC, SACRFCG, OptCon and ACPK*.

Only three works take into consideration the monetary costs of different consistency levels. *Consistency rationing* optimizes the runtime cost and makes a trade-off between consistency cost and penalty cost of inconsistencies. *Bismar* proposes a method to calculate the cost of consistency and combines it with the rate of stale reads to choose the best level of consistency. *DepSky* uses erasure codes to store only a part of the total amount of data in each cloud, and therefore minimizes the monetary cost. We state that the best way of considering monetary cost in an adaptive consistency approach is to combine it with other input parameters. In this case, the monetary cost can have a direct effect on the adopted level of consistency.

Despite its importance in cloud storage, security and privacy was not considered by most of the proposed approaches. Only two works raised security issues with adaptive consistency. *DepSky* used an erasure code and an encryption with a shared key. *4S* proposed the staged commit protocol which can prevent unauthorized access to confidential information.

The proposed approaches adapt the consistency level during runtime by giving

a suitable level according to application needs and the system state. Most approaches switch between existing levels and give the nearest one. However, a second type of approaches give a new calculated level from their defined policy like *IDEA, Harmony, Bismar, Consistency Tuner, 4S and OptCon.* A third type of approaches moderate the existing level to enhance consistency like *ABARC, AD an SPECSHIFT.* We state that delivering a new calculated level should be the best solution especially when the existing storage system provides few and divergent consistency levels. Thus, calculating the desired level gives the most appropriate consistency guarantees.
Only three works used simulation in their implementation: *ABARC, FHBC and SACRFCG.* The remaining works used real implementation which adds more credibility to their results.

2.8 Challenges and future research directions

2.8.1 Consistency in emerging computing architectures

The fast evolution of Internet of Things (IoT) applications has brought about more challenges to classical centralized cloud computing such as network failure and high response time. Fog and mobile edge computing targets these challenges by bringing the cloud closer to IoT devices which provides low latency and secure services. An adaptive consistency approach in this environment should consider more criteria specifically replica and edge servers placement [129, 130], client context-awareness [128], security and privacy [102]:

- *Replica and Edge servers placement*: Bringing computation power to the edge of network reduces latency but obstructs many services that need to access centrally stored data. Thus, a replica placement strategy is recommended to place data originating from a central server towards the end devices in the cloud network in order to decrease data access response time and enhance the benefits of edge computing. Moreover, the locations of edge servers are critical to the access delays of mobile users and the resource utilization of edge servers, which makes edge servers placement highly crucial. Inefficient edge server placement will result in long access delays and heavily unbalanced workload among edge servers.

- *Client context-awareness*: Contrary to classical cloud computing, different clients accessing the distributed Fog computing data store often have an individual context. This context can influence their requirements on consistency. For instance, a situation aware application [131] should consider conditions and things that happen at a particular time and place.

- *Security and privacy*: A hacker, for example, could deploy malicious applications on an edge node by exploiting vulnerability. If a fog node is hacked, it can get false input and formulate bad results which can affect the performance of the whole application. Therefore, the development of methods to characterize and detect malwares at large scale is recommended. Moreover, since user specific data need to be temporarily stored on multiple edge locations, privacy issues will need to be targeted along with security challenges.

Consistency is also needed in another new paradigm: Software-Defined Networking [132]. SDN is an emerging network architecture that separates the control plane from the data plane in order to provide programming network configuration. To avoid single point of failure and reduce response latency, multiple controllers are needed. Therefore, consistency requires that every controller has an identical global view of the network state. The challenging task in distributed SDN controllers is to maintain a consistent and up-to-date global network view for SDN applications while preserving good performance.

2.8.2 Data clustering to categories

Applying the same level of consistency for the overall system is not always practical when data in the cloud have not the same importance or the same access frequency. However, one difficulty that arises is the categorization of data objects due to large volumes of data and various data types. The related challenge in this situation is to provide an automatic mechanism that allows splitting data into different consistency categories by applying clustering techniques [133]. In this way, according to its enclosing data, every category should be given the most appropriate consistency level.

2.8.3 Intelligent techniques for adaptation and other input parameters

A statistic policy is very expensive to implement in big data due to the large amount of heterogeneous data. So, using an intelligent technique is better in terms of performance and costs. Traditional machine learning algorithms were restricted to execution on large clusters given the large computational requirements. Thus, recent researches in applying machine learning classification schemes are directed to the cloud [134, 135] to exploit its availability and cheaper data storage. These researches are branded under the name of 'Deep Learning'. Deep learning frameworks, like

Google TensorFlow[1] and Nervana Cloud[2], provide APIs and software libraries to perform complex learning tasks in a reasonable time and to give more accurate results.
Adaptive consistency approaches studied in this work have considered many parameters in the input set of learning algorithms such as: number of replicas, time gaps, network packet count, average throughput, etc. We propose to consider other parameters to tune the consistency level, as for example, access permissions. A role based access control is an access control policy that classifies clients according to their roles in the system. Hence, clients that have the same role should get the same consistency level. The client context in fog computing environments can be also used to predict the adequate consistency level. Thus, client who changes his place or access period should not maintain the same consistency guarantees.

2.8.4 Minimizing monetary cost

Taking monetary costs into consideration is very important in such approaches because it is among the principle goals of using the cloud. Stronger consistency causes higher monetary cost by means of synchronous replications that introduce high latencies due to cross-sites communication. On the other hand, lower levels of consistency may reduce the monetary cost by favoring performance to latency, but increase the rate of stale reads which raises costs if the application imposes penalties to incorrect operations. Partial replication [136] can give better performance with low costs for some applications. In fact, placing replicas at only a subset of data centers significantly reduces the number of messages sent with each write operation. Thus, partial replication reduces both storage and communication costs.

Conclusion

In this chapter, we have presented a comparative study between adaptive consistency approaches. A taxonomy of the adaptive consistency approaches has been introduced by defining several comparison criteria such as architectural model, conflicts, granularity, adaptive policy, operation level, threshold, monetary costs, security and implementation tools. Furthermore, this work reviews an interesting number of studies on adaptive consistency in cloud computing which allowed us to build a table-based analysis that summarizes our comparative study. A deep discussion of the behavior of each approach is also given for each criterion. According to the results of the studied approaches, we have deduced for each criterion the suitable choice for each situation. Based on the analysis, we have identified

[1]https://www.tensorflow.org/.
[2]https://www.nervanasys.com/cloud/.

several research challenges that need further investigation. These challenges include consistency in emerging computing architectures, intelligent learning techniques, monetary cost and data clustering. The next chapter will introduce what we propose in this research area: A new adaptive consistency approach in Edge Computing Environments.

Chapter 3

Second Contribution: A new adaptive causal consistency approach in Edge Computing Environment

Introduction

Edge computing is a new computing paradigm that has emerged to offload computation and storage to edge devices in order to get a shorter response time and more efficient processing. Nevertheless, adopting a consistency scheme that can conserve multiple replicas while guaranteeing a good level of consistency is an open issue. Moreover, a data store with only one consistency level is not suitable for applications that have different consistency requirements. In this chapter, we propose MinidoteACE, a new adaptive consistency system that is an improved version of Minidote a causally consistent system for edge applications. Unlike Minidote which supports only causal consistency, our model allows applications to run also queries with stronger consistency guarantees. Hence, in addition to causal consistency, clients can also perform updates with a stronger consistency level which is weaker than the typical strong consistency as it just ensures the arrival of queries to the system nodes, not their execution. Providing adaptive consistency allows application designers to choose the convenient consistency level for each operation according to application needs and the system's context. The application can specify which operations should be run under causal consistency and which others should have stronger consistency guarantees. Experimental evaluations show that throughput decreases only by 3.5% to 10% when replacing a causal operation with a strong operation. However, update latency increases significantly for strong operations up to three times for 25% update workload. This work is published in "International Journal of Computing and Digital Systems" [22].

3.1 Context and Problem statement

Choosing a consistency model in a distributed data store is a critical decision mainly when the latter is geo-replicated. The growing requirement for short response time and wide availability has led many geo-replicated data stores to prefer weak consistency models over strong consistency models. However, using weak consistency models, such as eventual consistency [80, 112] or adaptive consistency [21], renders the programming model more complicated to application developers, who are obligated to fix explicitly data inconsistency issues introduced by these models. This has led to the emergence of the causal consistency model [137] that has been demonstrated to be the strongest consistency model in an always-available system. Causal consistency captures the potential causal relationships between events within one client and also the reads-from relation between different clients. The causal consistency model ensures that a write operation will not be applied until all the operations that precede it are applied. Causal consistency is a good choice for geo-replicated data stores, since it can reduce the anomalies issued by eventual consistency, yet it tolerates partitions and avoids long latencies associated with strong consistency. However, causal consistency is a relaxed consistency model that optimistically allows replicas to confirm updates concurrently to achieve higher availability. This sometimes leads to conflicts across replicas. A Conflict-free Replicated Data Types (CRDTs) [138] are proposed as a proven solution to handle conflicts. A CRDT is an abstract data type that ensures that replicas can be modified without coordinating with each other and if they have received the same set of updates, they reach the same state, deterministically.

In this context, AntidoteDB [139, 140] is a highly available geo-replicated key-value data store. Thanks to causal consistency and CRDTs, AntidoteDB allows developers to build solid applications without reducing performance or horizontal scalability provided by AP/NoSQL storage systems (systems that sacrifice consistency to ensure availability and Partition tolerance). Minidote [23] is a lightweight version of AntidoteDB designed for Edge Computing Environment. Like AntidoteDB, Minidote keeps providing causal consistency as the only consistency level in the system. In fact, Minidote supports two types of operations: read and write (update). Read operations are performed locally. After receiving a client read query, the corresponding node uses only the local replica to answer the query. Similarly, write operations are applied and committed locally before broadcasting their effects and dependencies to other nodes. A node that receives effects of a remote update, can apply them to the local replica after verifying the update dependencies. Minidote then uses CRDTs to resolve inconsistencies generated by concurrent updates.

However, causally consistent systems including Minidote face two issues: 1) They provide only a single consistency level which is not sufficient for particular kinds of applications. Taking an online auction as an example of the systems that change the recommended consistency level during the run-time. For the early bids of the auction, the system does not require strong guarantees because the data are not so important for the final deal. However, When the deadline draws near, the participants should see the latest bids to make the next bid. Therefore, the consistency requirements become higher and the data should be always modified under strong consistency. 2) They do not distinguish between system replicas and clients, in other words, each node of the cluster is considered as a replica of the service and a client of the service at the same time. The latter assumption means implicitly that every client is always associated with one of the nodes. In such a case, the client sticks to the same replica, and the system provides *Sticky availability* [141] . However, clients are independent entities, which can switch from one replica to another, and which, in some situations, may lose communication to a system replica. Namely, the client needs high availability. But if a client moves from one node to an alternative one, it could mean giving up causal consistency: this can happen if the alternative node does not receive all the client's updates, and therefore it leads to breaking the read-your-writes guarantee implied by causal consistency.

To overcome these issues, we propose a new adaptive consistency strategy that we call: MinidoteACE. MinidoteACE offers clients two levels of consistency: causal and strong. MinidoteACE behaves as Minidote system for Causal operations. However, it uses the implementation of *Causal Stability* [24] to carry out strong operations.

3.2 Related works

Many prior work efforts have studied data management, mainly the trade-off between consistency and availability that has been involved in building distributed data stores. Therefore, there are many systems providing different semantics to application developers. For instance, Google BigTable [15], Microsoft Azure Storage [16] and Apache HBase [17] provide strong consistency. These systems provide simple semantics, but suffer from long latencies and partition-intolerance. Other systems like Amazon Dynamo [18], Cassandra [19] and MongoDB [20] provide eventual consistency which provides excellent performance and tolerates partitions, but renders the programming model more complicated due to inconsistency handling difficulties. MinidoteACE takes an intermediate position in this trade-off by embracing causal consistency semantics.

Many previous works have recognized the convenience and applicability of causal consistency. COPS [137] was the first system that defined the concept of *Causal+ consistency*, a causally consistent model that ensures the convergence offered by eventual consistency. Causally consistent systems need to track causal dependencies between operations in the form of a piggybacked metadata which is used to tag operations and to capture the causal past of clients with each operation. Multiple forms have been used to represent metadata, COPS [137] for instance uses explicit causal histories that, for each operation, enumerate all the operations that are in its causal path. Logical clocks are also used to track causality in ChainReaction [142], SwiftCloud [143], and Orbe [144]. The simple logical clock has an entry for each replica that will be incremented upon a new update is applied on the replica. Other approaches like GentleRain [145],Saturn [146], Cure [147], and Legion [148] use physical clocks that tag operations with the physical local time. However, Okapi [149] and Eunomia [150] choose to make a combination between logical and physical clocks: hybrid clocks. However, tracking causality accurately requires maintaining an incremental size of metadata that may affect the system performance. Our model uses smaller size timestamps than vector clocks to encode the causality between messages, a causal graph to store the dependencies between messages, and an efficient algorithm for causal delivery and stability. Several recent works have developed storage systems for the edge environment. DPaxos [151] and EdgeCons [152] rely on Paxos-based protocols that provide only strong consistency and suffer from long latencies. FogStore [128] however, implements two functionalities that allow it to manage consistency between Fog nodes in an optimal manner. The first one aims to minimize the response time between the clients, the devices, and the copy of a data record by relying on a replica placement strategy. The second functionality is a context-aware mechanism that chooses the consistency level of the client's query according to the client's context. PathStore [153] also supports several consistency levels according to the client needs mainly, eventual, session, and strong consistency. Although Fogstore and Pathstore strive to optimize latency, they do not preserve causality. Gesto [154] is a hierarchical architecture that allows cloud data stores to cover edge networks while providing causal consistency. Gesto splits replicas into geo-replicated groups where each group of edge replicas has one datacenter. The native replication protocol of the cloud data store is used between data centers. However, a novel causality tracking mechanism is integrated into each group. Moreover, Gesto uses a multipart timestamp enabling scalability and fast migration. Minidote (detailed in section 4) is also a causally consistent system for the edge. Unlike our model, Gesto and Minidote provide only causal consistency which is not sufficient for all kinds of applications that require stronger consistency guarantees.

3.3 Background and definitions

In this section, we aim to understand the main concepts around our system. In fact, our contribution is based on the Minidote system which is a lightweight key-value storage system that is designed for Edge Computing Environment. Minidote is a causally consistent system that allows it to provide fairly high scalability and performance while keeping a good consistency level. We introduce firstly, causal consistency by giving its definition, implementation, and causal stability concept. Later, we define CRDT data types as a way of convergent conflict handling. Finally, we give an overview of the computing environment of our system: Edge computing.

3.3.1 Causal Consistency

Causal consistency captures the notion that different data store nodes should see causally-related operations in the same order. Causality [155, 156] is a happens-before relationship between two events. For two operations, a and b, we say that a happens before b or, alternatively, b causally depends on a, and we write $a \rightsquigarrow b$, if and only if at least one of the taking after conditions hold:

- Thread-of-execution: a and b belong to a single thread of execution, and a precedes b.
- Reads-from: b reads the value written by a when b is a read operation and a is a write operation.
- Transitivity: There is some other operation c such that $a \rightsquigarrow c$ and $c \rightsquigarrow b$.

Consequently, a storage system provides causal consistency, if it does not commit any write operation before executing all its causally-related operations. For example, in a social network, Mark updates an old post on his wall. Alan sees the update and writes a comment about it. Then, Alan's comment shouldn't appear to their friend Paul before he sees Mark's update on the post as the comment of Alan causally depends on the update of Mark.

3.3.1.1 Causal consistency implementation

The implementation of Causal consistency can be performed using two simple steps [137]:

- Assign a set with all the preceding operations to each operation; this set we'll be called the dependencies of o and refer to it as *dep[o]*.
- Perform o once all its preceding operations (operations in *dep[o]*) have been executed.

Also, let's mention the set of executed operations by *Executed.*
If the process sends an operation o, then the dependencies of o will be the set of operations that have been performed at that time, i.e. *dep(o)=Executed.* If the operation p follows the operation o, then the dependencies set of p contains o :

$$o \in dep(p) \tag{3.1}$$

It may however occur that p does not cause o and o does not cause p. If so, o and p will be competing with each other :

$$o \notin dep(p) \wedge p \notin dep(o) \tag{3.2}$$

3.3.1.2 Causal stability

An operation o is causally stable (denoted by stable(o)) if it belongs to the dependencies set of each operation p that will be executed in the system [157, 24]:

$$stable(o) \equiv \forall p.p \notin Executed \Rightarrow o \in dep(p) \tag{3.3}$$

In other words, stable(o) means that o has been executed by all the nodes of the system. Hence, every new operation will be in the future of o.
Let's have a cluster with three nodes: a, b and c. o, p, q and r are operations that will be Executed in the cluster.

- a performed o and then p.
- b noticed o and p, and then performed q.
- c noticed o and performed r.

So, the Executed sets for each node are the following *Executed*:

- *Executed_a = {o,p}.*
- *Executed_b = {o,p,q}.*
- *Executed_c = {o,r}.*

Remember that: the set of operation dependencies is the set of operations performed at the moment of operation submission. We know therefore at this point that any submitted operation (at any node) will include o as a dependence. This means that o is stable. We can't say the same about any of the remaining operations.

3.3.2 Conflict-free Replicated Data Type (CRDTs)

In a causally consistent system, there is no need to order concurrent operations; two operations can be replicated in any order if they are not causally related, to avoid synchronization cost. However, If two concurrent update operations target the same object, they are in conflict and produce an inconsistency in the system. Avoiding conflicts usually requires ad-hoc handling [158, 159, 18] in the application logic by employing an automatic strategy to handle conflicts deterministically, in the same manner, at every replica. For example, the last-writer-wins rule has been adopted by most of the existing causally consistent systems [137, 160, 145], where the last update overwrites the other updates. Antidote and Minidote rely on CRDTs [138, 139] which are an abstract data type built to be replicated at multiple replicas. CRDTs have a clearly designed interface and exhibit two attractive properties: (1) there is no need for coordination between replicas associated with update operations; (2) two replicas can reach the same state after receiving the same set of updates since they guarantee state convergence by adopting mathematically sound rules. CRDTs include sets, counters, maps, LWW registers, lists, graphs, among others. As an example, a counter data type can handle the following operations: increment(C) and decrement(C). The implementation of a CRDT counter will guarantee that the state of the counter will reach the same value at different replicas whatever the order of increment and decrement operations.

3.3.3 Edge Computing

Edge computing [58] is defined as a model of distributed computing that employs technologies allowing to perform computation at the edge of the network. In contrast to the Cloud computing paradigm where data centers handle all storage and processing services, Edge computing aims to improve system scalability and data privacy, reduce latencies and ensure an effective usage of resources (mainly reducing energy consumption). In edge computing, cloud servers and edge devices work together to perform processing. Which computations are conducted on which of these components is determined by a variety of factors, including node capacity and latency requirements. We call an “edge“ any processing, storage, or networking resource located along the path between cloud data centers and edge devices. A cell phone, for example, is regarded as an edge between body things and the cloud, whereas a gateway in a smart home is considered as an edge between house things and the cloud.

3.4 An overview of Minidote system

Minidote [23] is a replicated key-CRDT store that provides causal consistency with atomic batch-reads and batch-updates while the data is automatically replicated on each node and concurrent updates are resolved using CRDTs. Compared to Antidote, Minidote doesn’t support interactive transactions and replica sharding which makes it more lightweight and therefore allow it to run well on less powerful devices. Minidote only has to keep the latest version available, whereas Antidote must be able to serve all snapshots used by currently running transactions.

The inter-dc replication service of Antidote is replaced with a different causal broadcast service provided by Camus [161]. Camus is a CAusal MUlticast Service, that provides different back-ends to guarantee a reliable dissemination and delivery respecting causal order at all replicas using the service. The back-end used in Minidote is an implementation of tagged casual broadcast (TCB) protocol [24] which guarantees that messages/operations will be delivered respecting the end-to-end happen-before relation as seen by the application.

3.4.1 Concepts and data structures

Before we describe the different components of Minidote architecture, we will present some concepts and data structure that will be used frequently in the next paragraphs and sections.

- State: Each minidote_server instance has a state that describes the current status. The state contains a vector clock, a list of dots that causally precede the next update i.e, dependencies, and the dot: the tag of the current operation of this node.
- CRDT object: like AntidoteDB, Minidote is a key-value data store so it identifies each object with its key, type, and bucket. So we write *Object = {key, type, bucket}*. When updating an object, the following parameters should be introduced to the query: the object identifier, the update operation, and the given value. For example, to increment a counter by 1, we write *Update = {Bobj, increment, 1}*.
- Vector clock: A vector clock of a system of N nodes is an array of N logical clocks used to manage a partial ordering of events in a distributed system and to keep causality relationships [162, 163]. As in Lamport timestamps, The causality relationship (called happened-before) captured is defined based on passing of information, rather than passing of time. The vector clock is used to track causal information between operations. Operations piggyback this information to allow their delivery in an order respecting the causality of events.
- Dot: a pair of *(node_Id, counter)*, which serves as a unique identifier for an operation/message of Minidote node to be broadcast.
- Dpgraph: The dependency graph is used to store messages that are not ready to be delivered yet. Graphs better characterize the partial order rather than

queues which are better for totally ordered events. The graph uses dots to identify its vertices where each vertex contains the operation's effects, the status of the message (missed, received, delivered, or stable), successors_set, and predecessors_set where predecessors_set/successors_set is a set of dots that precede/succeed this dot.

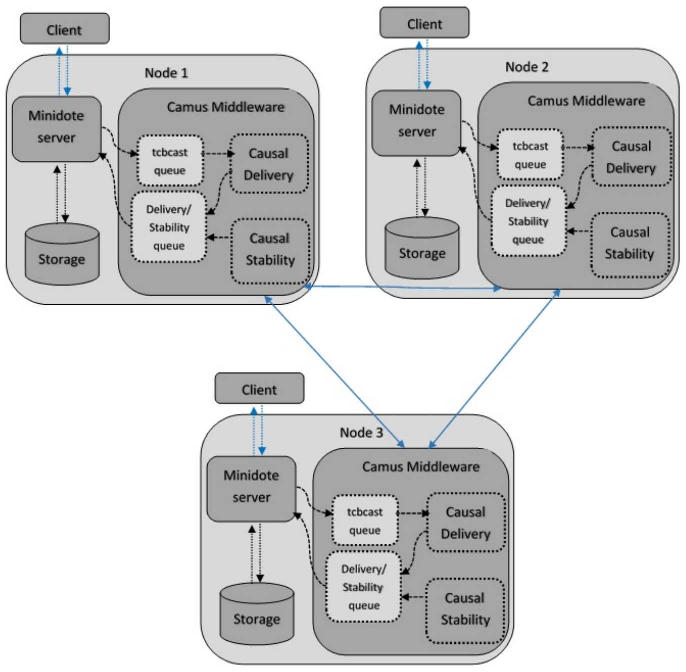

Figure 3.1: A cluster of three Minidote nodes.

3.4.2 Minidote architecture

Figure 3.1 shows the architecture of Minidote node and its interactions with clients and other nodes. Each Minidote node contains three principal modules: Minidote_server, Camus_middleware, and CRDT storage.

3.4.2.1 Minidote_server

Minidote_server is the brain of Minidote node which interacts with clients and other modules. A client can perform atomic batch-reads and batch-updates using Minidote APIs: Read_objects (Keys, clock) and Update_objects (updates, clock). For read operations, the client specifies Crdt keys of requested objects and a vector clock. After execution, Minidote_server returns values of the input keys and a new vector clock. However, for update operations, the client should specify a vector clock and for each update: an object's key, an update operation, and input parameters. Minidote_server returns, after execution, a new vector clock. Since Minidote_server has a state, vector clock, dot, and dependencies, for each read or update operation, it should wait if the operation's vector clock is later than the current state's vector clock. Then, it asks for locks of the object's keys, if they are not free the query should wait. Hereafter is a brief description of the main procedures of Minidote_server:

- Read_objects: Minidote_server performs read operations from local storage and then sends answers to the client.
- Update_objects: After acquiring locks, Minidote_server applies updates on the local data store, and then it increments the value that corresponds to the local node on the vector clock. After that and in a parallel way, it releases locks, answers the client by the new vector clock, and calls the broadcast function of the middleware which broadcasts the update effects to the other nodes.
- Deliver_remote_Update: this order is received from the middleware to apply a remote update. Minidote_server sends the received update effects to its local storage, updates the vector clock of the state, and adds the received dot to its dependencies.

3.4.2.2 Camus_middleware

The middleware is a low-level layer that ensures causality between messages using the following functions.

- Broadcasting updates (Tcbcast): When Minidote_server calls this function, the middleware creates a new vertex for it in the Dpgraph and labels it as 'received'. Then, it broadcasts a message containing the update effects, its dependencies, the dot, and the vector clock.
- Receiving a remote message: Upon receiving a remote message, the middleware creates a new vertex in *Dpgraph* for the received dot with status 'received'. Then it checks its dependencies, if they all have been received, it calls the function *Deliver*. Otherwise, it creates a new vertex in *Dpgraph* for each dot in the dependencies that have not yet been received with status 'missed'.
- Message delivery (*Deliver*): when this function is called, the middleware sends the corresponding message to Minidote_server and labels the corresponding vertex as 'delivered' in the *Dpgraph*. Then, it checks each dot in the *successors_set*. If all its predecessors have been delivered, it calls the function *Deliver* for the corresponding message. The middleware notes also that each dot in the *predecessors_set* has been received in the sender node. If a dot is labeled as 'received' in all the nodes, it will be labeled as 'stable' and the function *Stable* will be called for this dot.
- Stable: when a dot is labeled as 'stable', it will be removed from each *predecessors_set* in *Dpgraph* and then, its vertex will be removed from *Dpgraph*.

3.4.2.3 Storage

Minidote uses the CRDT libraries developed in AntidoteDB. CDRTs support high-level replicated data types such as counters, sets, maps, and sequences which are designed to work correctly in the presence of concurrent updates and partial failures.

3.4.3 Consistency Protocol

Hereafter, we will summarize the behavior of Minidote towards client requests:

- Each node has two main components: Minidote_server and Camus_middleware.
- For *read* operations, Minidote performs the query from the local instance and forwards the answer to the client.
- The client (Application) performs an update u via client api at Minidote_server of a node i.
- Minidote_server applies the update u locally and tags it by a new dot. Then, it calls the broadcast function of Camus that broadcasts the effects of the update u to the other nodes. At the same time, it replies to the client request by the new vector clock.
- When a node j receives the effects of u through its middleware, it delivers the effects to Minidote_server to be applied if all its dependencies have been already delivered. The update u will be then inserted into the dependencies of new updates generated by the node j.
- When the node i receives updates from all nodes containing u in their dependencies, it labels u as local stable.
- u becomes stable in the node j when it is included in the received update dependencies of all other nodes except i.

3.5 MinidoteACE: Proposed improvements and Adaptive consistency approach

Minidote provides only causal consistency and uses Crdts to handle conflicts. In MinidoteACE, we keep the same architecture as Minidote and the same behavior for read operations as well. However, MinidoteACE exposes two types of updates: causal and strong. The system keeps the same behavior as Minidote for causal

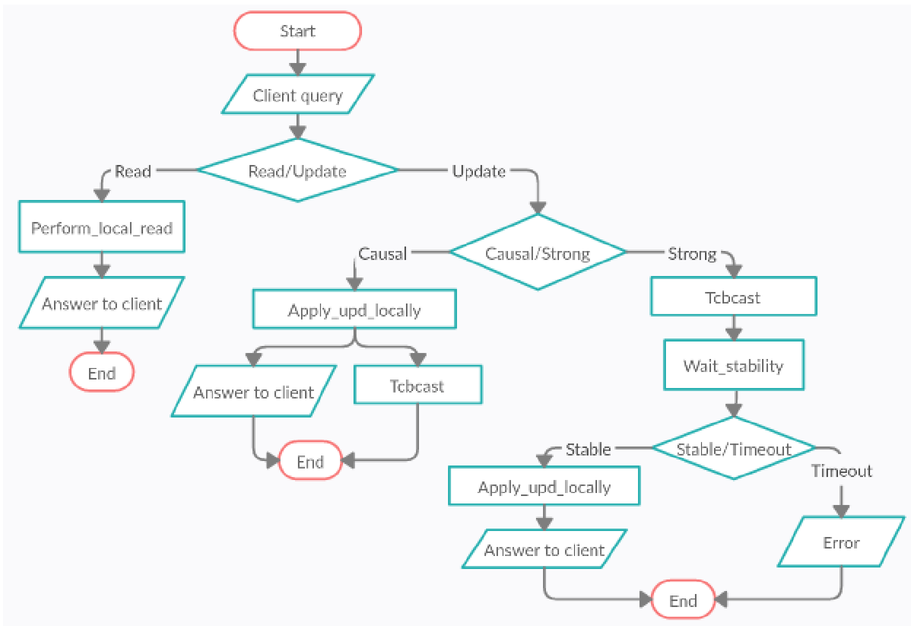

Figure 3.2: Flowchart for consistency proposal.

consistency operations. But in the case of strong consistency operations, it follows these steps.

- The client performs an update u at a node i via its Minidote_server.
- Minidote_server tags the update u and broadcasts it through the middleware to the other nodes without applying it locally.
- When a node j receives u, it inserts it to the strong received updates set $SRUS$. The latter will be piggybacked to all updates and commits sent by the node j.
- When the node i receives updates from all other nodes containing u in their $SRUS$, it considers u as stable. Then, it delivers u locally and broadcasts a commit message to the other nodes.
- When the node j receives a commit message of u, it labels u as stable and delivers it to Minidote_server to be applied.

Hence, *Read_objects* API keeps the same behavior. However, *Update_objects* API needs a third parameter that specifies the consistency level of update operations. Therefore, Update_objects API takes the following form: Update_objects (updates, clock, consistency). Figure 3.2 summarizes the behavior of MinidoteACE for read/update operations.

3.5.1 Detailed functions and Algorithms

- Read_objects: We keep the read operation behavior without modification. When the client calls the Read_objects API, Minidote_server uses the function described in Algorithm 3.1 to perform read operations. Minidote_server adds the current server state to the parameters introduced by the client to form the input parameters of read operation. Before performing read operations, Minidote_server ensures that the keys are free and compares the operation clock with the state clock and locks the keys. After reading objects, Minidote_server releases the keys and returns outputs to the client.

Algorithm 3.1: Read operation

```
Input: Keys, Clock, State
Output: Values, NewClock
1 wait_release_locks(Keys);
2 check_clock(Clock, State.Vc);
3 lock(Keys);
4 $Values = read\_crdtObjects(Keys, Clock)$;
5 $NewClock = State.Vc$;
6 unlock(Keys);
```

- Update_objects: We improve this function by adding the possibility to handle strong consistency operations. Our improvements are illustrated in Algorithm 3.2. The client who sends the update operation should specify the vector clock, the consistency level, and the update details (object identification, operation type, and the value). Minidote_server uses these elements in addition to the current state as input parameters to the update function that executes the following steps.
  - Firstly, Minidote_server checks the keys of the corresponding objects, and if they are acquired by another operation, it should wait. Then, it compares the operation's vector clock with that of the current state. If the stump of the query is greater, this means that the node misses some updates so it should wait until missed updates arrive (lines 1-3). After that, it acquires the locks of targeted keys and then increments its own logical clock in the vector by one (line 4).
  - Secondly, Minidote_server checks the consistency parameter (lines 5-12), if the consistency is 'causal', it applies the updates locally by calling CRDT APIs. Then, it calls the Tcbcast function of the middleware to broadcast the updates to other nodes. However, if the consistency is 'strong', it broadcasts the updates directly and waits for its stability to be able to apply them locally.
  - Finally, Minidote_server updates its state's vector clock by the current vector clock. The latter is returned as an answer to the client query and the Keys should be released(lines 13-15).

Algorithm 3.2: Update operation.

```
Input: Updates, Clock, Cons, State
Output: NewClock
1  wait_release_locks(Updates.keys);
2  check_clock(Clock, State.Vc);
3  lock(Updates.Keys);
4  $NewVc := increment(State.Vc)$;
5  if $Cons$ = $'causal'$ then
6      apply_updates_locally(Updates);
7      $State.Dot := tcbcast(Updates, NewVc, State.Deps, State.Dot, Cons)$;
8  else
9      $State.Dot := tcbcast(Updates, NewVc, State.Deps, State.Dot, Cons)$;
10     wait_stability(Updates);
11     apply_updates_locally(Updates);
12 end
13 $State.Vc := NewVc$;
14 $NewClock := NewVc$;
15 unlock(Updates.Keys);
```

- Broadcasting updates (Tcbcast): Tcbcast is a function of camus_middleware which is called by Minidote_server to broadcast updates to other nodes (Algorithm 3.3). When it is called, the middleware increments the dot by one to identify the message holding the new update (line 1). Then, the middleware adds a new vertex to the dependency graph for the new update (line 2). After that, it checks the consistency. If the latter is strong, the dot is added to the set of strong dots and the new vertex of the graph is labeled as 'received'. However, in the case of causal consistency, the new vertex is labeled as 'delivered' (lines 3-8). Finally, a message will be sent to the other nodes of the cluster. This message contains the updates, its identification (Dot), the dependencies, the consistency type, and the set of strong dots (lines 9 and 10).

Algorithm 3.3: Broadcasting updates(Tcbcast)

```
Input: Updates, Vc, Deps, Dot, Cons, Dpgraph
Output: NewDot, Dpgraph
1  $NewDot := increment\_dot(Dot)$;
2  Add NewVertex to Dpgragh for NewDot;
3  if $Consistency$ = $\acute{s}trong\acute{}$ then
4      Add NewDot to StrongDots;
5      Mark NewVertex as 'received';
6  else
7      Mark NewVertex as 'delivered';
8  end
9  $Msg := \{Updates, NewDot, Deps, Cons, StrongDots\}$;
10 broadcast_to_other_nodes(Msg);
```

- Receiving Remote updates: This function will be called when the middleware receives a broadcasting message of remote updates from another node

(Algorithm 3.4). Upon receiving a remote message, the receiver adds the corresponding dot to the set of strong dots if the consistency is strong. Then the function *check_strong_dot* is called to verify the status of strong dots (lines 2-5). After that, a new vertex in the dependency graph should be added, if it has not been added before, for the received dot (line 6). Then, the middleware creates a new vertex for each dot in the dependencies of the received dot if it has not been created before and in the latter case the vertex is labeled as 'missed' (lines 9-12). The received dot is inserted into the successors_set of all the vertices of its dependencies (line 13). Finally, for a causal consistency dot, it will be delivered if all its Dependencies have been delivered by calling the function *Deliver*(lines 15 and 17).

Algorithm 3.4: Receiving Remote updates

```
   Input: Msg, LSDots, Dpgraph
   Output: LSDots, Dpgraph
1  $\{Updates, Dot, Deps, Cons, RSDots\} := Msg$;
2  if Consistency = 'strong' then
3      Add Dot to LocalStrongDots;
4  end
5  $\{LSDots, Dpgraph\} := check\_strong\_dot(LSDots, RSDots, Dpgraph)$;
6  Add NewEntry to Dpgragh for Dot, if it is not exist;
7  Mark Dot as 'received';
8  foreach Dotd in Deps do
9      if not exist a vertex of Dotd in Dpgraph then
10         Add NewVertex to Dpgragh for Dotd, if it is not exist;
11         Mark NewVertex as 'missed';
12     end
13     Add Dot to successors_set of Dpgraph(Dotd) ;
14 end
15 if all dots in Deps have been delivered and Cons = 'causal' then
16     Deliver(Updates, Dot, Dpgraph);
17 end
```

- Delivering message (*Deliver*): Delivery means send the remote update from the middleware component to Minidote_server of the same node. This function has three steps (Algorithm 3.5):
  - label the corresponding vertex in the dependency graph as 'delivered' and send the updates to Minidote_server (lines 1 and 2).
  - Check the stability of the predecessors_set dots by noticing their vertices in the dependency graph as arrived at the sender node. A dot becomes stable if it is noticed as arrived at all the nodes (lines 3-8).
  - Check the dots in the successors_set if they can be delivered (lines 9-13).

Algorithm 3.5: Deliver Msg

```
   Input: Updates, Dot, Dpgraph
   Output: Dpgraph
1  Mark Dpgraph(Dot) as 'delivered';
2  Send Updates to Minidote_server;
3  foreach Dotd in Predecessors_set of Dpgraph(Dot) do
4      Find Dotd in Dpgraph and notice it as arrived at SenderNode;
5      if Dotd is noticed as arrived at all the nodes of the cluster then
6          $Dpgraph := Stable(Dotd, Dbgraph)$;
7      end
8  end
9  foreach Dotd in Successors_set of Dpgraph(Dot) do
10     if all Predecessors of Dotd have been delivered then
11         Deliver Dotd ;
12     end
13 end
```

- Check strong dots: It is a new function that we have added to manage strong consistency operations (Algorithm 3.6). It is called when the middleware receives any message from the other nodes by exploring the piggybacked set of strong dots. So the local node can determine which dots among its broadcast dots have arrived at the sender node to notice them. If a dot has arrived at all the cluster nodes, the middleware labels its vertex in the Dpgraph as 'stable' and then notify Minidote_server to apply the corresponding updates locally. Finally, a commit message is broadcast to the other nodes. When a node receives the commit message, its middleware calls the functions Deliver and Stable for the corresponding dot.

Algorithm 3.6: Check strong dots

```
   Input: LocalStrongDots, RemoteStrongDots, Dpgraph, SenderNode
   Output: LocalStrongDots, Dpgraph
1  foreach Dotd in LocalStrongDots do
2      if Dotd exist in RemoteStrongDots then
3          Notice Dotd as arrived at SenderNode;
4          if Dotd has arrived at all the nodes then
5              Notify Minidote_server to Apply the corresponding updates locally;
6              $Dpgraph := stable(Dotd, Dbgraph)$;
7              Broadcast a commit message to the other nodes for Dotd;
8          end
9      end
10 end
```

- Stable dot: When a dot becomes 'stable', it should be removed from the predecessors_set in the overall graph and its vertex should be removed from the dependency graph as well (Algorithm 3.7).

Algorithm 3.7: Stable dot

```
   Input: Dotd, Dpgraph
   Output: Dpgraph
1  foreach Vertex of Dpgraph do
2      Remove Dotd from Predecessors_set if it is exist;
3  end
4  Remove Vertex of Dotd from Dpgraph;
```

3.5.2 Clarification example

Take a cluster of two nodes MinidoteACE as shown in Figure 3.3. We will show how MinidoteACE handles client reads and updates.

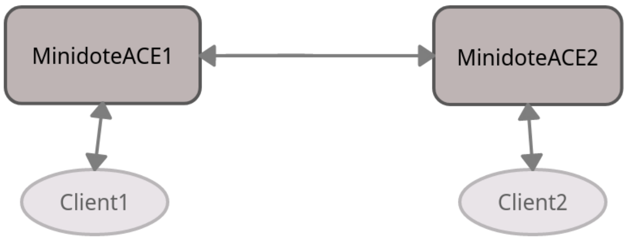

Figure 3.3: An example of cluster of two MinidoteACE nodes.

Let Ctr1= {"g1", crdt_counter, "bucket"} be an object of MinidoteACE where his key is “g1”, his type is a counter, and his bucket is “bucket”.

Update_object: When Client 1 wants to update the object Ctr1, he sends the following instruction to MinidoteACE 1:

Vc1 = update_objects([{Ctr1,increment,3}], {0,0}, causal).

This instruction increments Ctr1 three times (Ctr1 = Ctr1+3) under causal consistency. {0,0} is a vector clock that means that this operation has no dependencies. If the instruction is executed in the first node, the resulting vector clock Vc1 will take the value {1,0}.

Read_object: To read this object, the Client 2 send the following query:

{Value1,Vc2} = read_objects([Ctr1],{0,0}).

He should then obtain {Value1,Vc2} = {3,{1,0}}. In fact, Client 2 can read the value that has been updated by Client 1 and replicated by the system. Moreover, the obtained vector clock indicates that Client 1 has performed one update.

3.5.3 Implementation

We have implemented MinidoteACE by working on Minidote-tcb: a branch of Minidote GitHub repository which is written in Erlang and uses Camus. We have used the branch *with stability* of Camus that contains the implementation of stability. The main modules that we have replaced are: Minidote

API, Minidote_server, Camus and Camus_middleware. The source code of MinidoteACE is available in our GitHub repository [1]. We have made the necessary amendments in the other modules of Minidote and Camus to enable MinidoteACE to work conveniently.

3.6 Evaluation

We have used the benchmark Basho_bench which is developed within the project AntidoteDB[164, 165]; the latter is built on the original benchmark for Riak core. Basho_Bench[166] is a popular Erlang application that has a pluggable driver interface. Basho_Bench can be extended to serve as a benchmarking tool for data stores and generate performance graphs. The benchmarking tool is useful for repeatable and accurate performance. Basho_bench utilizes two particular indicators of performance: throughput and latency.

- Throughput: the number of operations performed over the defined period of time, including all possible types of operations.
- Latency: the time between sending a query and the completion of the reply.

The experiments have been performed on a cluster of MinidoteACE nodes incorporated with a traffic generator node. A traffic generator runs one copy of Basho_Bench that generates and sends commands to MinidoteACE nodes which are identified by their IP addresses and port numbers. Figure 3.4 shows how traffic generators and MinidoteACE nodes are organized in a cluster.

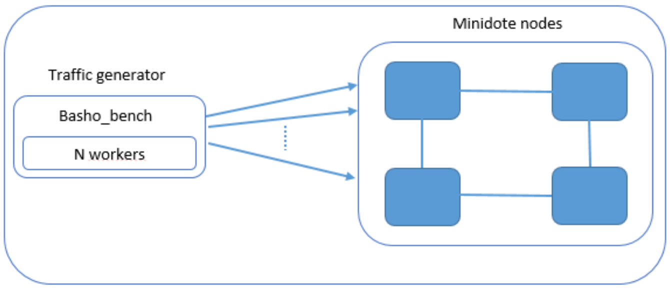

Figure 3.4: A cluster of MinidoteACE nodes with a traffic generator.

3.6.1 Experimental structure

Figure 3.5 illustrates our experimental setup. Hereafter, we describe the main components of the experimental architecture.

[1]https://github.com/nacer-git/MinidoteACE

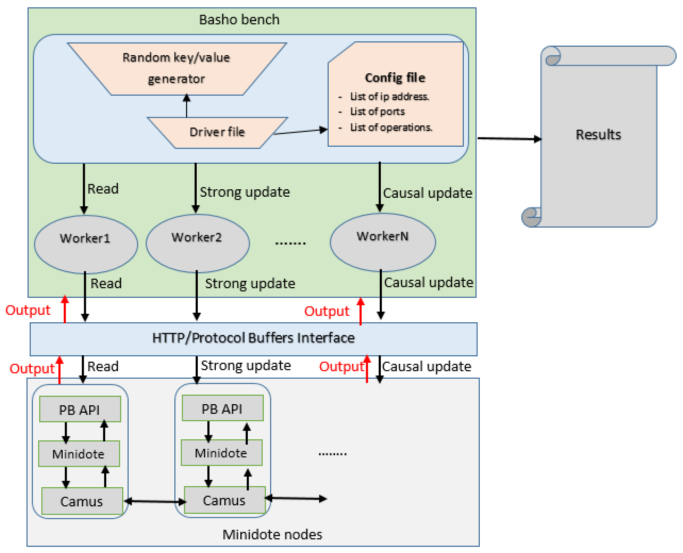

Figure 3.5: Experimental structure.

3.6.1.1 Traffic generator

The benchmark of AntidoteDB has two additional files, that make the original Basho_bench compatible with Antidote:

- The driver (*basho_bench_driver_antidote_pb.erl*): This file defines the initialization of a benchmarking thread and how it executes transactions. We have adapted the driver file to work properly with our system. Therefore, it can generate three kinds of operations: *read*, *causal update*, and *strong update*. Each generated operation is sent within a transaction of a single operation to the target MinidoteACE node through the protocol buffer interface.
- The configuration file (antidote_pb.config): This file is given as a starting parameter to the Basho_bench, it contains the benchmark parameters that can be adjusted according to the test's purpose. The configuration file contains the following parameters:
  - Target: defines the ip addresses and the ports of MinidoteACE nodes.
  - Driver: defines the file name of the driver which makes the workloads that were configured to be benchmarked for the specified targets.
  - Operations: This parameter configures which operations the driver should run and the weight of each operation. We put the following operations in the benchmark: read, causal update, and strong update.
  - Duration: An integer that defines the duration of the benchmark in minutes.
  - Threads: The number of parallel threads that will be run during one benchmark.

- Data type: defines data types for generated data as well as operations to be applied for each defined type in the benchmark.
- Keys generators: takes the form {key_distribution, Max_key} which defines the key generator distribution (Pareto or Uniform) in which the keys will be generated as well as the maximum value.
- Values generators: takes the same form as keys generator.

3.6.1.2 Protocol Buffer Interface and data flow inside the system

We have adapted the Protocol Buffer of AntidoteDB to work conveniently with MinidoteACE. A worker process randomly selects one of the defined operations in the configuration file. Then, the selected operation is encapsulated in a transaction before encoding it using the protocol buffer. Each encoded command is sent to a target MinidoteACE node which is selected randomly too. Upon receiving a command, the Protocol Buffer API of MinidoteACE decodes the command and sends it to MinidoteACE API. The API then encodes the response of MinidoteACE and transfers it to the traffic generator. In the traffic generator, there is a module that catches results and collects it in a result file.

3.6.2 Experiment Setup

The experiments were performed in an environment using a laptop with 8 GB DDR3 RAM, AMD Quad-Core Processor A8-7410 (up to 2.5 GHz), and 1 TB of Hard Disk Drive. Ubuntu 18.0.4 LTS (64-bit) was installed.

3.6.3 Performance configuration

Based on measuring performance and workload, Basho-bench is a benchmarking tool that performs reads and updates. After adapting the driver according to the input format of MinidoteACE, The operations that a driver might run are in the format of (causal update,x,strong update,y,read,z) which means that in each generated (x+y+z) operations, Basho_bench will generate x causal update operations, y strong update operation and z read operations. When y=0, This means that the benchmark does not contain any strong operation. Therefore, MinidoteACE will act like a Minidote system which allows us to compare our consistency approach with the existing approach. To evaluate the performance of MinidoteACE system and compare it with Minidote, we launched the experiments illustrated in Table 3.1 on a cluster of 3 to 6 nodes during one minute. The grey rows contain experiments that evaluate our adaptive consistency approach (Experiments: 3, 4, 6, and 7). The white rows, however, will evaluate Minidote behavior (Experiments: 1, 2, and 5).

For simplicity, we fixed the number of concurrent threads to 10, we chose the Pareto distribution for key generation and the Uniform distribution for value generation.

Tableau 3.1: Experiments details and abbreviations.

	Causal updates	Strong updates	Reads	Notation
Experiment 1	0	0	1	Read only
Experiment 2	1	0	9	1C_9R (10% updates)
Experiment 3	0	1	9	1S_9R (10% updates)
Experiment 4	1	1	18	1C_1S_18R (10% updates)
Experiment 5	5	0	15	5C_15R (25% updates)
Experiment 6	4	1	15	4C_1S_15R (25% updates)
Experiment 7	3	2	15	3C_2S_15R (25% updates)

3.6.4 Results and discussion

In the following sections, we address the performance indicators and describe the results according to different experimental scenarios between reads, causal updates and strong updates.

3.6.4.1 Throughput results

The experimental results about throughput performance for each experiment are illustrated in Figure 3.6. As it is shown in the results, the throughput performance in the read only experiment is almost three times bigger than the other cases (2440 operations) due to the absence of update operations that have a higher latency which decreases the performance. For the other experiments, we notice that the performance decreases when we increase the proportion of update operations. Moreover, the throughput performance is inversely proportional to the number of nodes i.e, a cluster that has a small number of nodes has a higher performance. Despite strong updates having higher latency, the difference with causal updates in performance is not huge between workloads when executing them. For experiments 2 and 3, the performance will be reduced by 3.5% when replacing causal update operations by strong operations. However, it will be less than 10% when replacing two out of five causal update operations by strong operations (Experiments 5 and 7).

3.6.4.2 Read latency results

Figure 3.7 illustrates the 95th percentile read latencies that we have measured in each experiment. The results show that read operations have the smallest latency for the read only workload (about 12 milliseconds for whatever the number of nodes in the cluster). However, this latency increases when the proportion of updates or the number of nodes in the cluster increases. We note also that sticking strong updates to the workload gives a remarkable decrease of read latency (up to 30% for 10% updates between experiment 2 and 3). This happened because strong updates have higher latency so the workload should have lower throughput performance, and hence, lower latency for read operations.

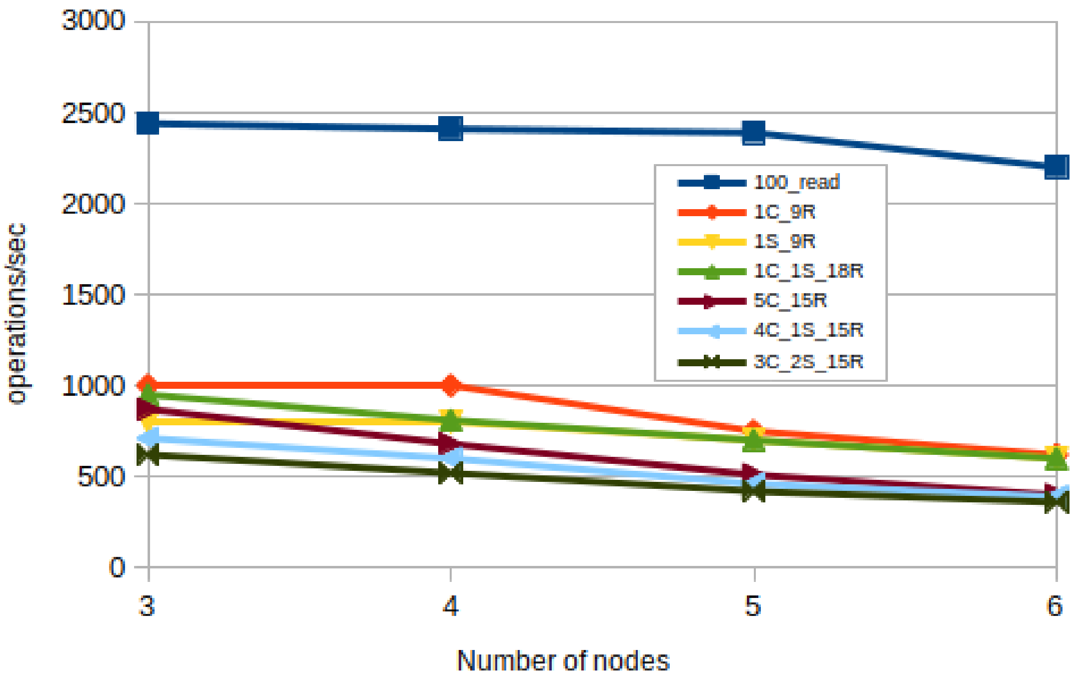

Figure 3.6: Throughput performance.

3.6.4.3 10% update results

Figure 3.8 and figure 3.9 illustrate the latencies of causal and strong updates for experiments 2, 3 and 4 where update operations account for 10% of the workload. In these figures, the abbreviation *1C_9R_causal_upd* represents the 95th percentile/average latency of a causal update operation in experiment 2. Similarly, *1S_9R_strong_upd* represents the 95th percentile/average latency of a strong update operation in experiment 3. The same notation is used by *1C_1S_9R_causal_upd* and *1C_1S_9R_strong_upd* in experiment 4. The results show that replacing causal updates by strong updates increases update latency between 23% and 100% according to the number of nodes in the cluster for average latency results (experiments 2 and 3). However, for 95th latency results, latencies of the two types are almost the same. We notice also that when dividing the proportion of updates between the two types, strong update latency is greater almost four times for a cluster of three nodes and two times for a cluster of six nodes.

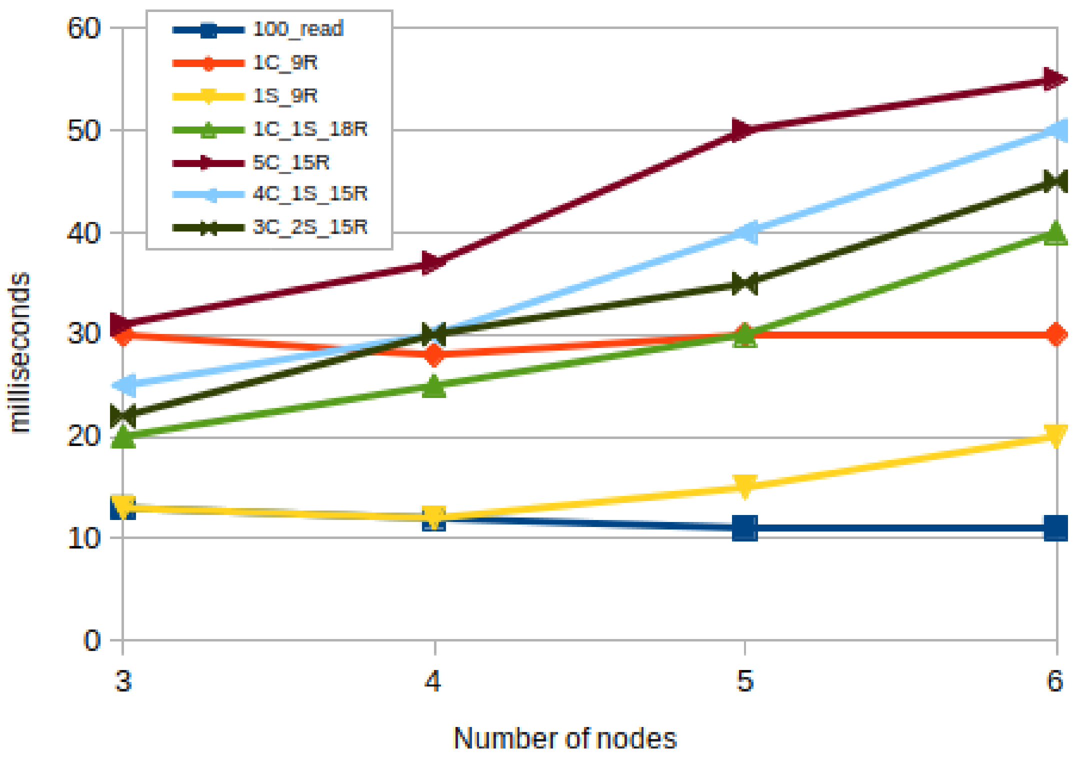

Figure 3.7: 95th percentile read latency.

3.6.4.4 25% update results

Figure 3.10 and figure 3.11 illustrate the latencies of causal and strong updates for experiments 5, 6 and 7 where update operations account for 25% of the workload. These figures use the same notation used in figures 3.8 and 3.9. Mean latency results show that when replacing one out of five causal updates by a strong update, its latency becomes three to two times greater according to the number of nodes in the cluster. Moreover, strong update latency becomes five to three times greater when replacing two out of five causal updates by strong operations. However, for 95th percentile results, strong operation latency is almost double of causal operation's latency when replacing one or two out of five causal operations for three or four nodes in the cluster. But when the number of nodes increases the gap between the two cases widens and becomes 70 milliseconds of difference for six nodes.

3.6.5 Limitations of the proposed approach

Although MinidoteACE allows to overcome causal consistency issues, It suffers from several limitations that we highlight in the following points:

- The strong consistency level introduced in MinidoteACE enables it to execute operations under stronger consistency guarantees. However, strong

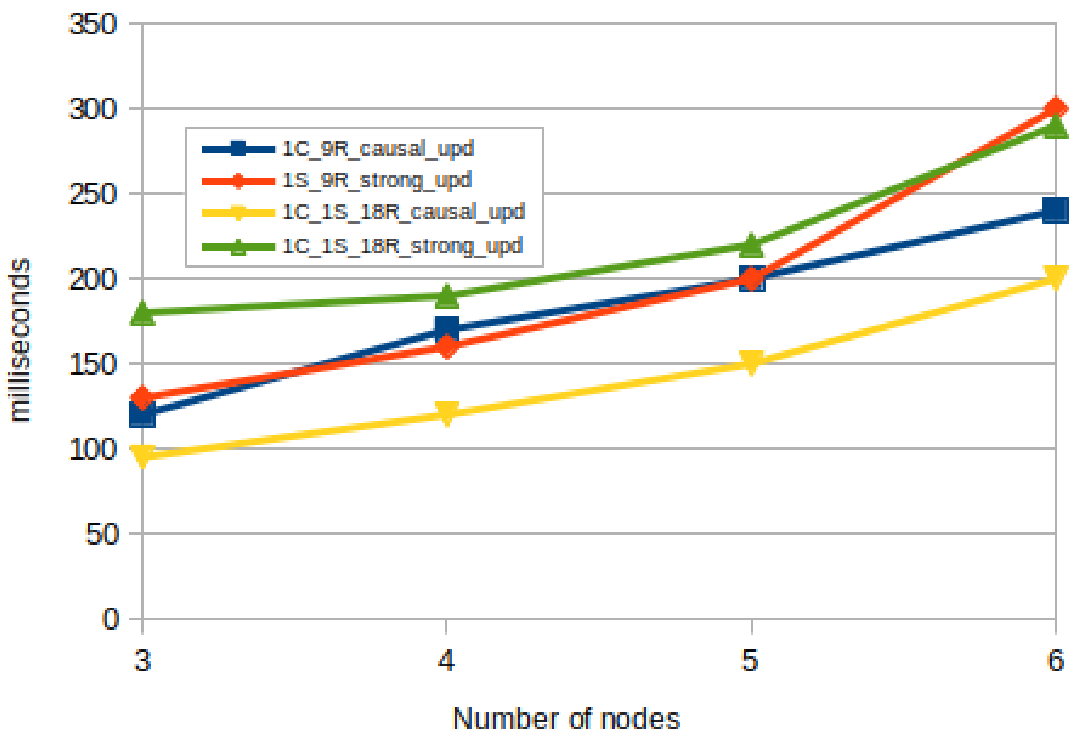

Figure 3.8: 10% updates: 95th percentile update latency.

consistency decreases the system performance (it increases Latency and decreases throughput). Hence. Clients should find a suitable trade-off between consistency and performance.

- Evaluations show that the latency of strong updates becomes quite high when the rate of update operations exceeds 25% of overall operations.
- "Strong updates" risque to be aborted due to a possibly long waiting time. The protocol used to enforce "total" requires stability information to be collected from all nodes. If this information is not collected after some timeout an "error" state is reached.

Conclusion

n this chapter, we have presented MinidoteACE, a new adaptive consistency approach in the edge computing environment. Our model enables applications to run queries with causal or strong consistency. To achieve this aim, we have improved the Minidote system by adding the ability to handle strong consistency operations. The new consistency level is stronger than causal consistency, but it does not emulate the typical strong consistency since it only checks that the update has arrived at the nodes of the cluster. We have experimentally evaluated

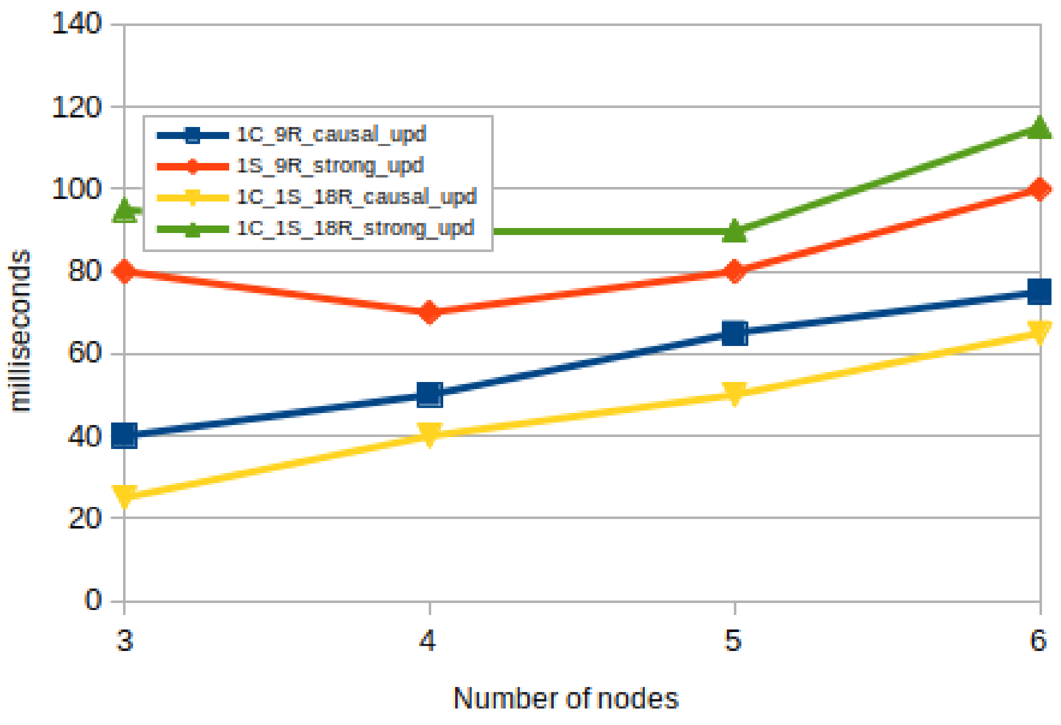

Figure 3.9: 10% updates: Average update latency.

MinidoteACE using Bash_bech: a benchmarking tool that has been modified to evaluate AntidoteDB. To do that, We performed the necessary amendment on the Basho benchmark to fit MinidoteACE. Our evaluation proves that MinidoteACE can support certain proportions of strong operations without significantly affecting latency or throughput. To the best of our knowledge, MinidoteACE is the only causally consistent system that provides more than consistency level in edge computing environment.

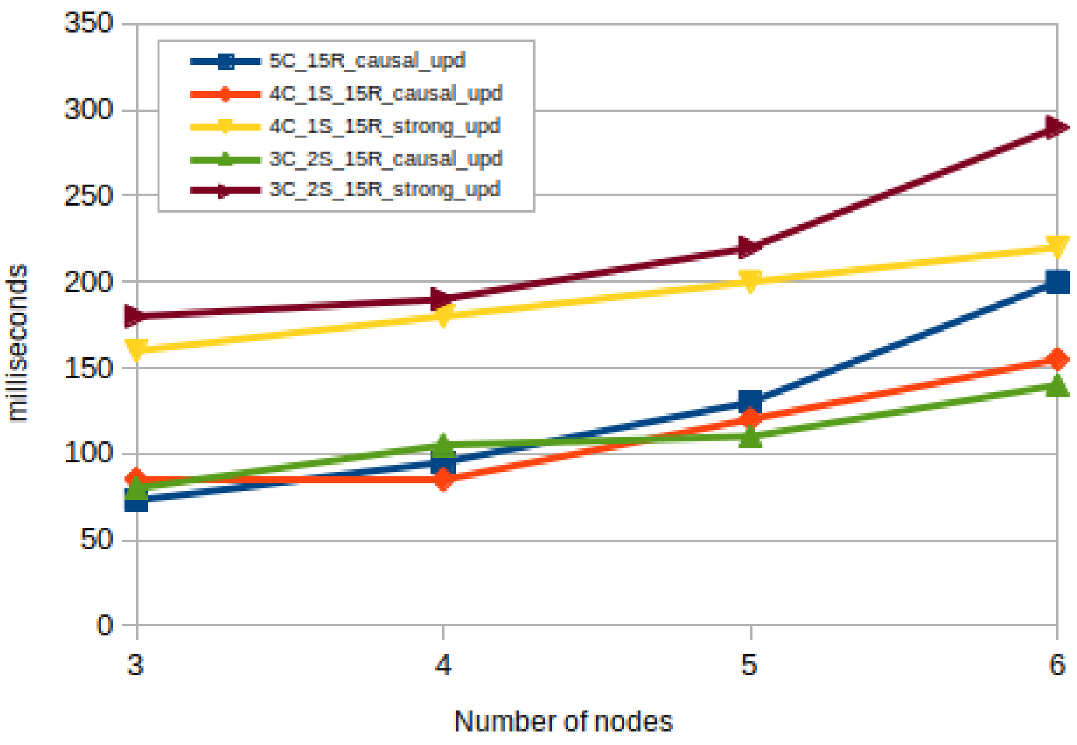

Figure 3.10: 25% updates: 95th percentile update latency.

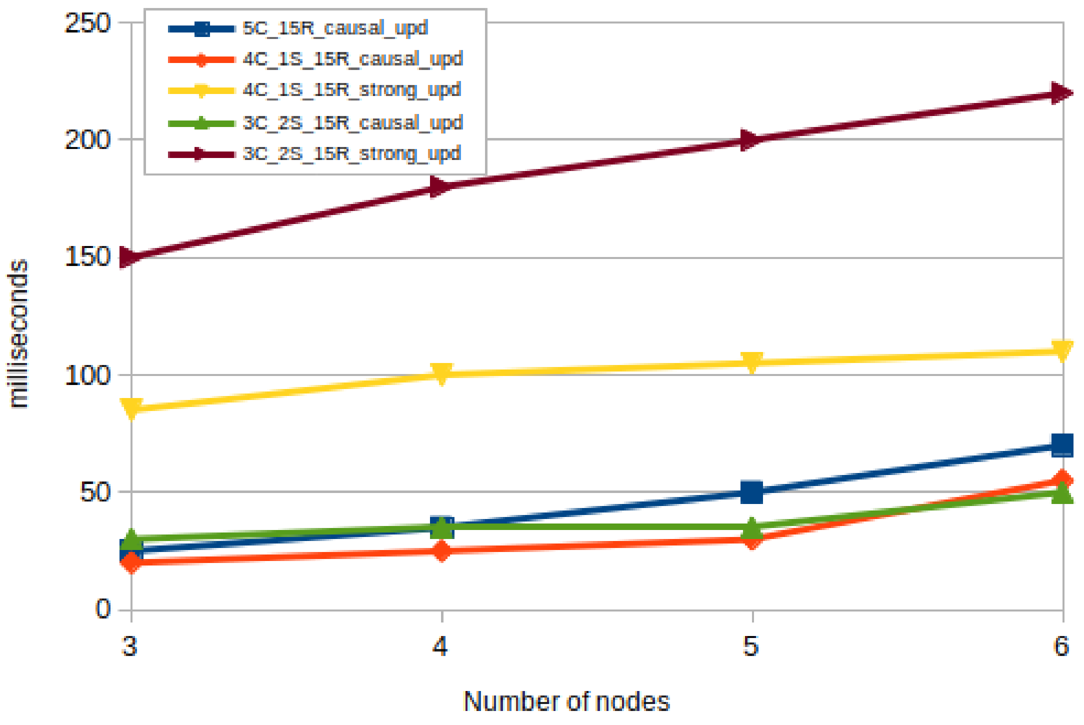

Figure 3.11: 25% updates: Average update latency.

Conclusion and perspectives

Consistency management in the cloud is an open issue that has been tackled in many works in the literature. The main issue in consistency is the trade-off between consistency in one hand and availability and performance on the other hand. Adaptive consistency is a consistency approach that tune the consistency level according to application needs as well as the system state. This thesis investigates adaptive consistency management of Big data in the cloud and offers two major contributions.

First Contribution

We established in the first contribution a comparative study of the proposed adaptive consistency models considering all aspects of data consistency in the Cloud as well as in emerging computing paradigms. To perform this study, we have first defined a set of comparative criteria that describe the design or the behavior of such an approach. Then, we analyze the most popular adaptive techniques in the literature. For each work, we highlighted its main contribution and summarized its behavior towards the defined criteria. We summarized then our analysis in a table in order to view globally the study and be able to discuss different aspects of adaptive consistency. Finally, We gave challenges and future research directions. We can outline the main results of this contribution in the following points:

- A middleware is the suitable architectural model to add the adaptive consistency functionality to an existing storage system and enhance its consistency management strategies.
- A statistical policy is very expensive to implement in big data due to large amount of heterogeneous data. So, using an intelligent technique is better in terms of performance and costs.
- Taking monetary costs into consideration is very important in such approaches because it is one the principal goals of using the cloud.

- Applying the same level of consistency for the overall system is not always practical when data in the cloud have not the same importance or the same access frequency.
- It is necessary that the adaptive policy makes a combination between transaction level and object statistics (operation side and data side) to define the consistency guarantees.

Second Contribution

In the second contribution, we proposed an adaptive consistency approach in Edge Computing environments. Our system offers two levels of consistency to clients: causal and strong for each update they send. To achieve our goal, we relied on Minidote, a causally consistent system for Edge Computing applications, where we used its behavior to perform causal operations. However, for strong operations, we used the existing implementation of causal stability which ensures that a strong operation will not be executed until it will arrive at all other nodes. The new consistency behavior that our system provides does not wait for update operations to be committed, it ensures just their arrival to all the system nodes to commit the operation. Hence, the new consistency level is weaker than the typical strong consistency. Our system allows application designers to adaptively choose the convenient consistency level for each operation according to application needs and the system's context. The application can specify which operations should be run under causal consistency and which others should have stronger consistency guarantees.

Perspectives

To overcome the thesis's drawbacks and improve its results, the future aims and our suggestions for future works are:

- Applying the same level of consistency for the overall system is not always practical when data in the cloud have not the same importance or the same access frequency. The related Challenge to this situation is to provide an automatic mechanism that allow splitting data into different consistency categories by applying clustering techniques. And so, according to its enclosing data, Every category should be given the most appropriate consistency level.
- Considering other parameters to tune consistency level. For example, Role Based Access control which is an access control policy that classify clients according to their roles in the system. Hence, clients that have the same role

should get the same consistency level. Client context in the fog computing environments can be also used to predict the adequate consistency level. thus, client change his place or access period should not maintain his in consistency guarantees.

- For MinidoteACE, we aim to improve its performance by enabling it to support a higher rate of strong consistency updates. Moreover, we aim to avoid waiting for a long timeout by investigating the possibility of implementing ”strong” operations while collecting only a majority of confirmations. which gives quorum consistency: another consistency level between causal and strong.

Bibliography

[1] "5 v's of big data [online]." https://bigdatapath.wordpress.com/2021/01/27/5-vs-of-big-data/. (Accessed on April 2021).

[2] "Infrastructure layers [online]." https://www.oreilly.com/library/view/cloud-native-infrastructure/9781491984291/ch01.html. (Accessed on April 2021).

[3] P. Mell and T. Grance, "The nist definition of cloud computing," *Communications of the ACM*, vol. 53, no. 6, p. 50, 2010.

[4] H. Jin, S. Ibrahim, T. Bell, L. Qi, H. Cao, S. Wu, and X. Shi, "Tools and technologies for building clouds," in *Cloud Computing*, pp. 3–20, Springer, 2010.

[5] S. Zhang, X. Chen, S. Zhang, and X. Huo, "The comparison between cloud computing and grid computing," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, vol. 11, pp. V11–72, IEEE, 2010.

[6] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li, "Comparison of several cloud computing platforms," in *Information Science and Engineering (ISISE), 2009 Second International Symposium on*, pp. 23–27, IEEE, 2009.

[7] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, pp. 464–474, IEEE, 2000.

[8] V. Serbanescu, F. Pop, V. Cristea, and G. Antoniu, "Architecture of distributed data aggregation service," in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 727–734, IEEE, 2014.

[9] V. Serbanescu, F. Pop, V. Cristea, and G. Antoniu, "A formal method for rule analysis and validation in distributed data aggregation service," *World Wide Web*, vol. 18, no. 6, pp. 1717–1736, 2015.

[10] J. W. Bates and M. Aldred, "System and method for secure and reliable multi-cloud data replication," June 24 2014. US Patent 8,762,642.

[11] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, "Energy-efficient data replication in cloud computing datacenters," *Cluster computing*, vol. 18, no. 1, pp. 385–402, 2015.

[12] S. V. Karve, J. S. Kulkarni, S. S. Patel, A. R. Pathak, and S. R. Patil, "Cloud based data migration and replication," July 7 2015. US Patent 9,075,529.

[13] I. Kertiou, S. Benharzallah, L. Kahloul, M. Beggas, R. Euler, A. Laouid, and A. Bounceur, "A dynamic skyline technique for a context-aware selection of the best sensors in an iot architecture," *Ad Hoc Networks*, vol. 81, pp. 183–196, 2018.

[14] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney, "File and object replication in data grids," *Cluster Computing*, vol. 5, no. 3, pp. 305–314, 2002.

[15] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[16] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, *et al.*, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 143–157, ACM, 2011.

[17] "Apache hbase." http://hbase.apache.org/, February 2018.

[18] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.

[19] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[20] "mongodb." http://www.mongodb.org/, February 2018.

[21] A. Khelaifa, S. Benharzallah, L. Kahloul, R. Euler, A. Laouid, and A. Bounceur, "A comparative analysis of adaptive consistency approaches in cloud storage," *Journal of Parallel and Distributed Computing*, vol. 129, pp. 36–49, 2019.

[22] A. Khelaifa, S. Benharzallah, and L. Kahloul, "A new adaptive causal consistency approach in edge computing environment," *International Journal of Computing and Digital Systems*, vol. 12, pp. 945–960, 2022.

[23] "Minidote github repository." https://github.com/LightKone/Minidote. (Accessed on April 2021).

[24] C. Baquero, P. S. Almeida, and A. Shoker, "Making operation-based crdts operation-based," in *IFIP International Conference on Distributed Applications and Interoperable Systems*, pp. 126–140, Springer, 2014.

[25] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The digitization of the world from edge to core. idc white paper, november 2018," *URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf*.

[26] E. Dumbill, "Defining big data." https://www.forbes.com/sites/edddumbill/2014/05/07/defining-big-data/?sh=350fdec55667, May 2014.

[27] K. Davis, *Ethics of Big Data: Balancing risk and innovation.* " O'Reilly Media, Inc.", 2012.

[28] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition," tech. rep., and productivity. Technical report, McKinsey Global Institute, 2011.

[29] I. O'Reilly Media, *Big Data Now: 2014 Edition.* " O'Reilly Media, Inc.", 2015.

[30] M. Stonebraker, "What does 'big data'mean," *Communications of the ACM, BLOG@ ACM*, 2012.

[31] "Big data – what is it? [online]." http://www.sas.com/big-data/. (Accessed on April 2021).

[32] T. O. G. B. S. Issue., "Posix.1-2008, ieee 1003.1-2008," *URL: http://pubs.opengroup.org/onlinepubs/9699919799/*.

[33] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters.," in *FAST*, vol. 2, 2002.

[34] S. Donovan, G. Huizenga, A. J. Hutton, C. C. Ross, M. K. Petersen, and P. Schwan, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the Linux Symposium*, vol. 2003, 2003.

[35] R. B. Ross, R. Thakur, *et al.*, "Pvfs: A parallel file system for linux clusters," in *Proceedings of the 4th annual Linux showcase and conference*, pp. 391–430, 2000.

[36] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, 2006.

[37] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 29–43, 2003.

[38] “Hadoop distributed file system (hdfs).” http://hadoop.apache.org/, 2021.

[39] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[40] F. Engineering, “Under the hood: Scheduling mapreduce jobs more efficiently with corona.” https://engineering.fb.com/2012/11/08/core-data/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/, November 2012.

[41] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[42] K. Stanoevska-Slabeva and T. Wozniak, “Cloud basics–an introduction to cloud computing,” in *Grid and cloud computing*, pp. 47–61, Springer, 2010.

[43] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” 2008.

[44] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[45] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds: A berkeley view of cloud computing,” Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[46] “Google docs.” https://docs.google.com, 2021.

[47] “Microsoft office live.” https://www.officelive.com, 2021.

[48] D. Sanderson, *Programming google app engine: build and run scalable web apps on google’s infrastructure*. " O’Reilly Media, Inc.", 2009.

[49] D. Chappell *et al.*, “Introducing the windows azure platform,” *David Chappell & Associates White Paper*, 2010.

[50] G. Pierre and C. Stratan, “Conpaas: a platform for hosting elastic cloud applications,” *IEEE Internet Computing*, vol. 16, no. 5, pp. 88–92, 2012.

[51] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system,” in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 124–131, IEEE, 2009.

[52] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, “Elastic management of cluster-based services in the cloud,” in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pp. 19–24, 2009.

[53] D. Robinson, *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster*. Emereo, 2008.

[54] “What are public, private, and hybrid clouds?.” https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/#benefits, 2021.

[55] V. Lionel Sujay, “Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025.” https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/, 2021.

[56] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, “The cloud is not enough: Saving iot from the cloud,” in *7th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[57] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[58] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[59] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[60] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[61] J. Acharya and S. Gaur, "Edge compression of gps data for mobile iot," in *2017 IEEE Fog World Congress (FWC)*, pp. 1–6, IEEE, 2017.

[62] "Openfog reference architecture for fog computing." https://www.openfogconsortium.org/ra/, 2021.

[63] "Fog computing brings new business opportunities and disruptions." https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Fog-computing-brings-new-business-opportunities-and-disruptions, 2021.

[64] M. Chiang, S. Ha, F. Risso, T. Zhang, and I. Chih-Lin, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 18–20, 2017.

[65] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pp. 109–116, 1988.

[66] "Revolutionary methods to handle data durability challenges for big data." http://www.intel.com/content/www/us/en/big-data/big-data-amplidata-storage-paper.html, 2021.

[67] "Amazon s3 glacier & s3 glacier deep archive." https://aws.amazon.com/s3/glacier/, 2021.

[68] "Hdd vs. ssd." https://www.diffen.com/difference/HDD_vs_SSD, 2021.

[69] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 42–81, 2005.

[70] W. H. C. Almeida, L. de Aguiar Monteiro, A. C. de Lima, R. R. Hazin, and F. Escobar, "Survey on trends in big data: Data management, integration and cloud computing environment," 2019.

[71] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.

[72] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou, "Comet: batched stream processing for data intensive distributed computing," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 63–74, 2010.

[73] W. Fang, X. Z. Wen, Y. Zheng, and M. Zhou, "A survey of big data security and privacy preserving," *IETE Technical Review*, vol. 34, no. 5, pp. 544–560, 2017.

[74] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, 2000.

[75] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[76] "The cost of downtime." https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/, 2021.

[77] "Amazon.com." https://www.amazon.com/, 2021.

[78] R. Ladin, B. Liskov, and L. Shrira, "Lazy replication: Exploiting the semantics of distributed services," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pp. 43–57, 1990.

[79] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, "Providing high availability using lazy replication," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 360–391, 1992.

[80] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[81] M. Shapiro and B. Kemme, "Eventual consistency," 2017.

[82] V. Chang and G. Wills, "A model to compare cloud and non-cloud storage of big data," *Future Generation Computer Systems*, vol. 57, pp. 56–76, 2016.

[83] T. Chang, G. Popescu, and C. Codella, "Scalable and efficient update dissemination for distributed interactive applications," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 143–150, IEEE, 2002.

[84] X. Wang, S. Yang, S. Wang, X. Niu, and J. Xu, "An application-based adaptive replica consistency for cloud storage," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pp. 13–17, IEEE, 2010.

[85] Z. Zhou, S. Chen, T. Ren, and T. Wu, "File heat-based self-adaptive replica consistency strategy for cloud storage'," *J Computers*, vol. 9, no. 8, pp. 1928–1933, 2014.

[86] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez, "Harmony: Towards automated self-adaptive consistency in cloud storage," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pp. 293–301, IEEE, 2012.

[87] H. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation-Volume 4*, p. 21, USENIX Association, 2000.

[88] Y. Shin, D. Koo, and J. Hur, "A survey of secure data deduplication schemes for cloud storage systems," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 74, 2017.

[89] P. Viotti and M. Vukolić, "Consistency in non-transactional distributed storage systems," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, p. 19, 2016.

[90] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 91, 2017.

[91] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez, "Consistency management in cloud storage systems.," in *Advances in data processing techniques in the era of Big Data*, CRC PRESS, 2014.

[92] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez, "Consistency in the cloud: When money does matter!," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 352–359, IEEE, 2013.

[93] S. Limam and G. Belalem, "A self-adaptive conflict resolution with flexible consistency guarantee in the cloud computing," *Multiagent and Grid Systems*, vol. 12, no. 3, pp. 217–238, 2016.

[94] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *CONCURRENCY CONTROL AND RECOVERY IN DATABASE SYSTEMS*. Addison- Wesley, 1987.

[95] A. K. Elmagarmid, *Database transaction models for advanced applications*. Morgan Kaufmann Publishers Inc., 1992.

[96] V. Chang, "Towards a big data system disaster recovery in a private cloud," *Ad Hoc Networks*, vol. 35, pp. 65–82, 2015.

[97] G. Tarasuk-levin, P. W. P. Dirks, I. Langouev, and C. Kolovson, "Maintaining consistency using reverse replication during live migration," June 6 2017. US Patent 9,672,120.

[98] M. Bharde, S. Bhattacharya, and D. D. Shree, "Store edge ripplestream: Versatile infrastructure for iot data transfer," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[99] E. N. Bournival, D. L. Black, S. Cohen, A. Natanzon, and M. J. Halstead, "Maintaining stored data consistency of a plurality of related virtual machines across a plurality of sites during migration," May 16 2017. US Patent 9,652,333.

[100] G. Pardon and C. Pautasso, "Consistent disaster recovery for microservices: the cab theorem," *IEEE Cloud Computing*, 2017.

[101] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[102] A. S. Sohal, R. Sandhu, S. K. Sood, and V. Chang, "A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments," *Computers & Security*, vol. 74, pp. 340–354, 2018.

[103] V. Chang and M. Ramachandran, "Towards achieving data security with the cloud computing adoption framework.," *IEEE Trans. Services Computing*, vol. 9, no. 1, pp. 138–151, 2016.

[104] A. Shoker, H. Yactine, and C. Baquero, "As secure as possible eventual consistency," in *Proceedings of PaPoC'17*, ACM, 2017.

[105] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "A software engineering perspective on sdn programmability," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1255–1272, 2016.

[106] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed sdn controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, 2017.

[107] I. Farris, T. Taleb, Y. Khettab, and J. S. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2018.

[108] V.-G. Nguyen, T.-X. Do, and Y. Kim, "Sdn and virtualization-based lte mobile network architectures: A comprehensive survey," *Wireless Personal Communications*, vol. 86, no. 3, pp. 1401–1438, 2016.

[109] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. Ramos, and A. Bessani, "Design and implementation of a consistent data store for a distributed sdn control plane," in *2016 12th European Dependable Computing Conference (EDCC)*, pp. 169–180, IEEE, 2016.

[110] M. Aslan and A. Matrawy, "Adaptive consistency for distributed sdn controllers," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, pp. 150–157, IEEE, 2016.

[111] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed sdn control plane," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2017.

[112] S. P. Kumar, *Adaptive Consistency Protocols for Replicated Data in Modern Storage Systems with a High Degree of Elasticity.* PhD thesis, Conservatoire national des arts et metiers-CNAM, 2016.

[113] Y. Lu, Y. Lu, and H. Jiang, "Adaptive consistency guarantees for large-scale replicated services," in *Networking, Architecture, and Storage, 2008. NAS'08. International Conference on*, pp. 89–96, IEEE, 2008.

[114] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: pay only when it matters," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 253–264, 2009.

[115] C. Li, D. Porto, A. Clement, J. Gehrke, N. M. Preguica, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary.," in *OSDI*, vol. 12, pp. 265–278, 2012.

[116] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 309–324, ACM, 2013.

[117] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 12, 2013.

[118] A. G. Dimakis and K. Ramchandran, "Network coding for distributed storage in wireless networks," in *Networked Sensing Information and Control*, pp. 115–134, Springer, 2008.

[119] S. P. Phansalkar and A. R. Dani, "Tunable consistency guarantees of selective data consistency model," *Journal of Cloud Computing*, vol. 4, no. 1, p. 13, 2015.

[120] M. McKenzie, H. Fan, and W. Golab, "Fine-tuning the consistency-latency trade-off in quorum-replicated distributed storage systems," in *Big Data (Big Data), 2015 IEEE International Conference on*, pp. 1708–1717, IEEE, 2015.

[121] R. Guerraoui, M. Pavlovic, and D.-A. Seredinschi, "Incremental consistency guarantees for replicated objects.," in *OSDI*, pp. 169–184, 2016.

[122] I. Sheff, T. Magrino, J. Liu, A. C. Myers, and R. van Renesse, "Safe serializable secure scheduling: Transactions and the trade-off between security and consistency," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 229–241, ACM, 2016.

[123] S. Sidhanta, W. Golab, S. Mukhopadhyay, and S. Basu, "Adaptable sla-aware consistency tuning for quorum-replicated datastores," *IEEE Transactions on Big Data*, vol. 3, no. 3, pp. 248–261, 2017.

[124] S. Chatterjee and W. Golab, "Self-tuning eventually-consistent data stores," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pp. 78–92, Springer, 2017.

[125] V. GHANEKAR and D. S. PHANSALKAR, "Selective consistency model on mongodb big data store," *Journal of Theoretical and Applied Information Technology*, vol. 96, no. 5, 2018.

[126] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, pp. 1–7, 2011.

[127] Z. Guo and S. Ding, "Adaptive replica consistency policy for kafka," in *MATEC Web of Conferences*, vol. 173, p. 01019, EDP Sciences, 2018.

[128] H. Gupta and U. Ramachandran, "Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, pp. 148–159, ACM, 2018.

[129] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, 2018.

[130] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.

[131] S. A. Hossain, M. A. Rahman, and M. A. Hossain, "Edge computing framework for enabling situation awareness in iot based smart city," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 226–237, 2018.

[132] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101–118, 2017.

[133] C.-W. Tsai, S.-J. Liu, and Y.-C. Wang, "A parallel metaheuristic data clustering framework for cloud," *Journal of Parallel and Distributed Computing*, vol. 116, pp. 39–49, 2018.

[134] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, IEEE, 2017.

[135] S. Sridharan, K. Vaidyanathan, D. Kalamkar, D. Das, M. E. Smorkalov, M. Shiryaev, D. Mudigere, N. Mellempudi, S. Avancha, B. Kaul, *et al.*, "On scale-out deep learning training for cloud and hpc," *arXiv preprint arXiv:1801.08030*, 2018.

[136] T.-Y. Hsu, A. D. Kshemkalyani, and M. Shen, "Causal consistency algorithms for partially replicated and fully replicated systems," *Future Generation Computer Systems*, vol. 86, pp. 1118–1133, 2018.

[137] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't settle for eventual: scalable causal consistency for wide-area storage with cops," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 401–416, 2011.

[138] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*, pp. 386–400, Springer, 2011.

[139] "Antidote: the highly-available geo-replicated database with strongest guarantees [online]." `https://pages.lip6.fr/syncfree/index.php/2-uncategorised/52-antidote.html`. (Accessed on April 2021).

[140] D. D. Akkoorath and A. Bieniusa, "Antidote: the highly-available geo-replicated database with strongest guarantees," tech. rep., Tech. U. Kaiserslautern. https://syncfree.lip6.fr . . . , 2016.

[141] R. Guerraoui, M. Pavlovic, and D.-A. Seredinschi, "Trade-offs in replicated systems," *IEEE Data Engineering Bulletin*, vol. 39.

[142] S. Almeida, J. Leitão, and L. Rodrigues, "Chainreaction: a causal+ consistent datastore based on chain replication," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 85–98, 2013.

[143] M. Zawirski, N. Preguiça, S. Duarte, A. Bieniusa, V. Balegas, and M. Shapiro, "Write fast, read in the past: Causal consistency for client-side applications," in *Proceedings of the 16th Annual Middleware Conference*, pp. 75–87, 2015.

[144] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel, "Orbe: Scalable causal consistency using dependency matrices and physical clocks," in *Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1–14, 2013.

[145] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel, "Gentlerain: Cheap and scalable causal consistency with physical clocks," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–13, 2014.

[146] M. Bravo, L. Rodrigues, and P. Van Roy, "Saturn: A distributed metadata service for causal consistency," in *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 111–126, 2017.

[147] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro, "Cure: Strong semantics meets high availability and low latency," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pp. 405–414, IEEE, 2016.

[148] A. van der Linde, P. Fouto, J. Leitão, N. Preguiça, S. Castiñeira, and A. Bieniusa, "Legion: Enriching internet services with peer-to-peer interactions," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 283–292, 2017.

[149] D. Didona, K. Spirovska, and W. Zwaenepoel, "Okapi: Causally consistent geo-replication made faster, cheaper and more available," *CoRR*, vol. abs/1702.04263, pp. 1–12.

[150] C. Gunawardhana, M. Bravo, and L. Rodrigues, "Unobtrusive deferred update stabilization for efficient geo-replication," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 83–95, 2017.

[151] F. Nawab, D. Agrawal, and A. El Abbadi, "Dpaxos: Managing data closer to users for low-latency and mobile applications," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 1221–1236, 2018.

[152] Z. Hao, S. Yi, and Q. Li, "Edgecons: Achieving efficient consensus in edge computing networks," in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[153] S. H. Mortazavi, B. Balasubramanian, E. de Lara, and S. P. Narayanan, "Pathstore, a data storage layer for the edge," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 519–519, 2018.

[154] N. Afonso, M. Bravo, and L. Rodrigues, "Combining high throughput and low migration latency for consistent data storage on the edge," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–11, IEEE, 2020.

[155] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[156] M. Ahamad, G. Neiger, J. E. Burns, P. Kohli, and P. W. Hutto, "Causal memory: Definitions, implementation, and programming," *Distributed Computing*, vol. 9, no. 1, pp. 37–49, 1995.

[157] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of Convergent and Commutative Replicated Data Types," Research Report RR-7506, Inria – Centre Paris-Rocquencourt ; INRIA, Jan. 2011.

[158] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer, "Bayou: replicated database services for world-wide applications," in *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*, pp. 275–280, 1996.

[159] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session guarantees for weakly consistent replicated data," in *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, pp. 140–149, IEEE, 1994.

[160] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pp. 313–328, 2013.

[161] "Camus github repository." `https://github.com/lightkone/camus`. (Accessed on April 2021).

[162] C. J. Fidge, "Timestamps in message-passing systems," in *Proc. of the 11th Australian Computer Science Conference (ACSC 1988)*, pp. 56–66, 1988.

[163] F. Mattern, "Virtual time and global states of distributed systems," in *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pp. 215–226, Elsevier Science Publishers B.V, 1988.

[164] "Antidotedb basho bench [online]." `https://antidotedb.gitbook.io/document-ation/benchmarking/basho_bench`. (Accessed on August 2021).

[165] "Antidotedb benchmarks github repository [online]." `https://github.com/AntidoteDB/Benchmarks`. (Accessed on June 2021).

[166] "Basho_bench github repository." `https://github.com/basho/basho_bench/`. (Accessed on September 2021).