International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 16, No.1 (Aug-24)

http://dx.doi.org/10.12785/ijcds/160141

Enhancing Data Consistency via a Context-Aware Dynamic Adaptive Model

Zohra Mahfoud[1] and Nadia Nouali-Taboudjemat[2]

[1]*Computer Science Department, USTHB, Algeria*
[2]*Research Centre on Scientific and Technical Information (CERIST), Algeria*
[1]*LIM, Bouira University, Algeria*

Received 29 May. 2023, Revised 17 Apr. 2024, Accepted 26 Apr. 2024, Published 1 Aug. 2024

Abstract: This paper introduces a new model for managing data consistency in large-scale, geo-distributed storage systems, pivoting on the concept of dynamic adaptive consistency. Recognizing the challenges in balancing consistency and availability in such systems, we propose an innovative, context-aware model that categorizes operations into "consistent blocs." This categorization allows for a more granular and efficient management of consistency levels, representing a notable improvement over adaptive approaches that employ a uniform consistency model across all operations or apply a single consistency level to each operation independently. Our model dynamically adapts these blocs' consistency levels in response to real-time changes in the operational context; this can include, for example, variations in network latency, data access patterns, and workload intensity, ensuring optimal data consistency tailored to current conditions.

Our approach extends traditional adaptive consistency models by introducing more flexibility. A middleware architecture achieves this goal by introducing an Adaptation Manager that dynamically adjusts consistency levels. We implement this model and evaluate its performance using the YCSB benchmark on a Cassandra cluster. Our results reveal significant flexibility in expressing users' requirements and prompt responsiveness in the dynamic adaptation of the policy. Our proposition holds significant benefits for applications where rapid adaptation to context is crucial.

Keywords: Consistency, Replication, Distributed databases, Transactions, NoSQL, Big Data.

1. Introduction

Nowadays, data processing is in a new age; data reaches huge volumes, is generated, must be treated with high velocity, and uses various formats. This kind of data, often qualified as big data, originates from various sources, including cloud computing and big science. In such large-scale systems, data is generally replicated across geo-distributed datacenters to guarantee high availability.

Managing replicas involves carefully considering data consistency. Consistency measures the freshness of data returned by the database; the strong consistency level always returns the most recent value of data, contrary to the weak level that allows stale values. Strong consistency is ideally preferred; however, many applications tolerate degraded levels of consistency to enhance availability. This trade-off is often necessary because maintaining both strong consistency and high availability simultaneously is challenging, as stated by the CAP theorem [1] and the PACELC theorem [2].

To figure out the consistency level, two categories of evaluative criteria are pivotal: one focuses on server-side, while the other examines aspects related to the client-side [3], [4], [5], [6]:

Regarding the server-side, the classification depends on three factors: the total number of replicas (N), the number of replicas needed to commit a write (W), and those required for a read operation (R). Obviously, engaging all the replicas (N=W=R) for both reading and writing operations ensures strong consistency. Nevertheless, a configuration where the sum of replicas for read and write operations exceeds the total number (W+R>N) can also maintain strong consistency, though with fewer replicas. This approach reduces conflicts between operations, thereby enhancing performance. This principle forms the basis of quorum systems, which operate on the premise of a majority of replicas.

On the client-side, the interest is given to the manners of propagating write operations among replicas and choosing replicas from which to read data. The following examples illustrate three models, each characterized by a distinct level of consistency. The strict model synchronously propagates

E-mail address: z.mahfoud@univ-bouira.dz, nnouali@cerist.dz

updates across replicas, ensuring that read operations always return the most recently written values. In the eventual consistency model, all replicas are guaranteed to eventually converge to the same state, although stale values may be read before this state is reached. The read-your-writes model ensures that a user consistently sees their own updates or a more recent version. The strict model provides strong consistency, distinguishing it from the others. The read-your-writes model, while offering a more consistent view to a user compared to eventual consistency, still falls under weak consistency since it permits the reading of outdated values.

Configurable consistency offers the flexibility of selecting the model to use from a set of proposed models. The configuration is generally supported at the read/write operation level.

Adaptive consistency proposes to automate the switch between consistency models according to an adaptation criterion. Adaptive consistency ensures a flexible compromise between consistency and availability since it forces strong consistency in critical situations and tolerates degraded levels otherwise to offer more availability. For instance, a Webshop is a classic transactional system that needs classically strong consistency. However, weak consistency is safe until a threshold is reached. Also, in the case of auction systems, it is beneficial to use weak consistency until approaching deadlines. In these examples, adaptive consistency allows for higher availability without harming database consistency by switching weak and strong consistency according to some adaptation criteria like threshold and time.

This paper aims to enhance adaptive consistency by incorporating a greater degree of flexibility and dynamism through a context-aware approach that more accurately aligns with application-specific requirements regarding consistency. We introduce a model that organizes operations into 'consistent blocs' based on their individual consistency needs. These blocs, characterized by their adaptive nature, employ criteria that automatically adjust consistency levels in response to the context. Such categorization leads to a more granular control of consistency levels, representing a notable improvement over conventional approaches that employ a uniform consistency model across all operations or apply a single consistency level to each operation independently. Moreover, the model facilitates the dynamic adaptation of the adaptation criteria themselves, allowing ensuring optimal data consistency tailored to fluctuating conditions such as changes in network latency, data access patterns, and varying workload intensities. The implementation of the proposed policy-based architecture as a middleware makes its simple integration with database systems that support configurable consistency. For the performance evaluation, we use the YCSB benchmark with a Cassandra cluster. The experimental results prove the flexibility and the dynamicity of the proposed solution.

The rest of the paper is organized as follows: Section 2 presents related work, and Section 3 presents the proposed solution. Section 4 discusses the evaluation, and section 5 summarizes the work.

2. Related Work

Research on consistency progresses to meet emerging requirements of distributed systems. Initially, a large set of consistency models is proposed to support the basic requirements of applications, such as the strict model, Sequential consistency, Eventual consistency, Read-your-writes, Causal consistency, and Session consistency. From time to time, new models are proposed to deal with specific requirements, such as the per-record timeline consistency model proposed by Yahoo! to manage the mail service [7].

Configurable consistency is very suitable for modern storage systems like Amazon SimpleDB [8], Amazon DynamoDB [9], and Cassandra [10].

Pileus [11] proposes a service level agreements (SLAs) based model that invites users to describe their consistency requirements as a set of sub-SLAs listed in order of priority. Sub-SLAs define how to switch consistency models according to latency level. The latter is influenced by replica configuration and network conditions.

Several studies on adaptive consistency have discussed the criteria for characterizing different consistency policies. Consistency Rationing [12] rations data into three distinct categories based on their consistency needs: weak, strong, and adaptive. This framework delineates five adaptive consistency policies, determined by factors such as staleness probability, temporal points, and specific thresholds. Chameleon [13] introduces an offline modeling approach that analyses the application behavior along the timeline to select the most appropriate consistency policy for each period. The policies proposed include Strong or Eventual consistency, Static or Dynamic consistency, and Local or Geographical consistency. Dynamic Consistency, in particular, adapts consistency levels to improve performance, cost, or energy efficiency. The adaptation protocols of this approach are further detailed by Harmony [14] and Bismar [15] in the paragraphs below.

Another branch of research in adaptive consistency involves the development of adaptive protocols. These protocols periodically monitor incoming operations (both read and write) to predict the behavior of the storage system. Such monitoring permits to act dynamically to preserve a defined tradeoff between consistency and availability. Harmony [14] employs a model that estimates, in real-time, the minimum number of replicas needed to maintain an acceptable level of data staleness, based on network latency and average write size. Here, 'staleness' denotes the proportion of stale read operations. The work detailed in [16] utilizes a threshold time gap for reading values at an acceptable correctness level and for delaying conflicting operations. The time gap [17] is estimated using a consistency

index, which measures the possibility of the correctness of the next incoming read.

CC-Paxos [18] proposes a configurable protocol for data replication based on the well-known Paxos protocol [19]. Users specify their preferences in terms of a tradeoff between consistency and availability. User preferences are satisfied by scaling up and down the number of requested replicas at runtime.

Calibre [20] describes a consistency protocol that forwards read requests to the replicas containing the most recent version of the item. The requested replicas are identified by consulting a registry that records information about the last updates. Similarly, Bismar [15] proposes a tradeoff between consistency and monetary cost. It defines consistency-cost efficiency as a metric for evaluating the price of consistency models. This metric is related to the cost of instances, storage, network, and staleness rates. Bismar adapts the consistency level at runtime to reduce the monetary cost and preserve low rates of staleness. The consistency-aware model [21] proposes incrementing NoSQL database capabilities by integrating transactional mechanisms; the paper focuses on providing different levels of atomicity while respecting a required level of consistency.

MinidoteACE [22] proposes an adaptive consistency system for edge computing environment based on Minidote [23]. The idea consists of incrementing the Minidote model based on casual consistency by adding the ability to handle selected operations with strong consistency.

[24] proposes visible adaptive consistency, a new advanced model based on Tree-Based Consistency and Rose Tree Algorithm. This model allows for dynamic adjustments in the consistency level, ensuring that the system remains efficient and responsive to changes without compromising the integrity and reliability of the data stored in the Cloud Database Management System. This approach is especially crucial in heterogeneous environments where different system parts may have varying consistency requirements.

Stabilizer [25] offers a flexible and customizable approach to defining consistency models in geo-replicated cloud applications catering to various needs and scenarios. This flexibility is achieved through a domain-specific language that allows clients to specify their consistency requirements and adjust the system's behavior to their specific application's needs.

The paper [26] introduces the CAL theorem, which extends the CAP theorem to distributed cyber-physical systems (CPS). CAL introduces a new language for expressing tradeoffs between availability and consistency in CPS and guides system design choices between end devices, edge computers, and the cloud. [27] demonstrates the significant impact of consistency on performance, availability, and energy consumption in NoSQL-based storage systems. The study focuses on basic models of consistency.

In the following section, we introduce our solution that aligns with prior research in supporting for adaptive methods to manage consistency. Our approach enhances the adaptation criteria based on the specific context and increases the granularity at which consistency is managed, shifting the focus from the operation level to the bloc level. These concepts present the potential for integration into emerging languages designed to express consistency requirements. Furthermore, our solution allows for dynamic adjustments of the adaptation criteria, thereby enhancing flexibility and adaptability.

3. Dynamic Adaptive Consistency

This section outlines different aspects of our dynamic adaptive consistency model and describes its implementation using a policy-based architecture.

A. A Model for Dynamic Adaptive Consistency

Dynamic Adaptive Consistency Model models queries into a set of consistent blocs that regroup operations characterized by the same consistency requirements. A static consistent bloc refers to a set of operations following a unique model of consistency, while adaptive consistent blocs follow consistency policies. A policy defines a set of rules that defines the consistency model to use for each context relevant to the application. Rules can be adapted at runtime.

The proposed model (Figure 1) describes queries as a set of Consistent Blocs (CB) characterized by Consistency Requirements (CR). $Q = (CB_i, CR_i), i >= 1$

- A Consistency Bloc is composed of a set of operations (Op) characterized by the same consistency requirements $CB_i = Op_{ij}, i, j.$

- An operation is a "read" or "write": $Op_{ij} = r/w$

- A Static consistent bloc requires a Consistency Model (CM), while an adaptive consistent bloc requires a Consistency Policy (CP): $CR_i = CM_i/CP_i$

- A Consistency Model (CM) refers to a consistency model provided by the system.

- A Consistency Policy (CP) is defined by a set of rules(R): $CP_i = (R_{ij}), i, j >= 1$

- A Rule (R) associates a Consistency Model (CM) with a Context Descriptor (CD): $R_{ij} = CM_{ij}, CD_{ij}$

- A Context Descriptor (CD_{ij}) describes the different information about the context that influences the choice of a consistency model.

B. Consistent Bloc

The concept of consistent blocs is proposed to classify operations into groups according to consistency requirements. Operations in a static consistent bloc use the same

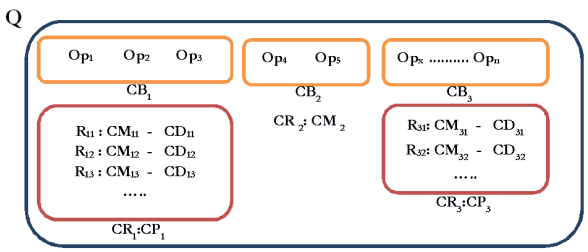

Figure 1. Dynamic Adaptive Consistency Model

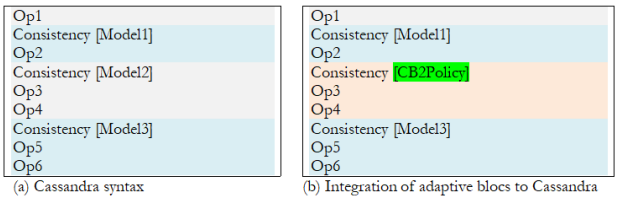

Figure 2. Consistent blocs with Cassandra

model of consistency, regardless of the context. In contrast, operations in an adaptive consistent bloc follow a policy of consistency that adapts the used consistency model according to context.

To clarify this vision, we mention that the idea behind static consistent blocs is implemented by several configurable consistency models. The syntax of Cassandra (Figure 2 (a)), for example, allows defining series of operations that use different models of consistency. A default model is used if none is set. This specification allows the defining of several static consistent blocs. To integrate the concept of adaptive blocs into Cassandra, a policy of consistency should be referenced, where needed, instead of a simple model (Figure 2 (b)).

Consistent blocs define a level of granularity for specifying users' requirements using various levels of consistency in addition to consistency policies that ensure the adaptation of consistency. Static blocs are designed to improve system performance because they make consistency customized to users' needs as they permit for avoiding strong consistency for operations that tolerate weak level. Adaptive blocs go deeper as they allow avoiding strong consistency for an operation if the context tolerates weak consistency.

For example, in the case of inventory management applications where data is traditionally manipulated with strong consistency, it is possible to classify operations into several consistent blocs with different consistency levels. Besides static blocs that impose strong consistency, we can define a static bloc with weak consistency to handle data that is rarely updated and can tolerate some inconsistency, like product designation. Also, we can define an adaptive bloc to switch consistency according to a defined threshold for operations related to the quantity of products in stock.

C. Context Descriptor

The context [28] refers to "any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Context-aware systems can adapt their behaviors according to the current contexts in order to provide the appropriate response with or without the user's intervention [29]. Early context-aware applications used the user's location to forward calls to the closest phone. The utilization of this kind of application was increasingly extended to cover diverse domains like mobile applications [30], Internet of Things [31], and industrial applications [32].

Our context-aware consistency model adapts consistency to context. The information of context that influences the choice of consistency depends on the nature of the applications and their needs. For example [12], [13], [33]:

- The threshold is related to E-commerce and stock management; the quantity of an item in stock can be considered as a criterion to decide on the consistency level; above the threshold, a weak level is authorized to increase availability; elsewhere, a strong level is imposed to avoid overselling product items.
- Time is important for a large category of applications. For instance, bids in an auction system can be managed with weak consistency for several days before the end of the auction. As the deadline approaches, the exact view of submissions is required, which imposes strong consistency.
- Disconnections are frequent in mobile contexts and disaster situations; weak consistency can be acceptable to get any version of data in case of difficulty in reaching other nodes. When the connection improves, the strong level can be imposed to manipulate data more consistently.
- The staleness presents the percentage of the stale reads per the total number of reads. Some applications prefer to tolerate weak consistency if the staleness is negligible and to impose strong consistency otherwise.
- Users may not have the same needs in terms of consistency, so it is possible to switch between the different models according to users' profiles in order to provide a better compromise between consistency and availability.

The list above is not exhaustive; every type of application has specific requirements, and new needs can arise due to emerging utilizations. Therefore, users are responsible

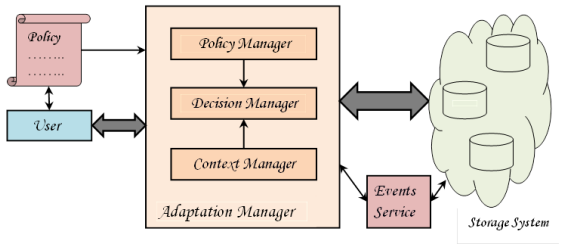

Figure 3. A policy based architecture for dynamic adaptive consistency

```
<policy>
    <pol name="name">
        < ConsistencyModel> ConsistencyModel </ ConsistencyModel >
        < ContextDescriptor >
            <Criterion1>ValCriterion</Criterion1>
            <Criterion2>ValCriterion2</Criterion2>
            ...
        </ ContextDescriptor >
    </pol>
    <pol name="name">
            ...
    </pol>
</policy>
```

Figure 4. The Policy specification

for identifying the different information of context.

In our solution, all information about the context is grouped in the context descriptor that consists of a composite adaptation criterion used to switch between the different consistency models according to context, which permits the definition of a policy that accurately meets users' requirements.

D. A Policy-Based Architecture for Dynamic Adaptive Consistency

Policy-based architectures are commonly used to implement adaptive systems in different domains, such as the Internet of Things for vehicular Cloud [34] and disaster situations [35] . This kind of architecture is characterized by the particularity of separating the policy from other mechanisms of adaptation, which allows supporting dynamic adapting of the policy itself at runtime [30], [33] .

Our Policy-based Architecture (Figure 3) is implemented as a middleware between the user and the storage system. The core of this architecture is the Adaptation Manager, which is responsible for the adaptation process. The components of the adaptation manager have different roles:

- The Decision Manager decides the consistency model to adopt for every consistent bloc.
- The Context Manager manages variables of context based on information provided by the events service.
- The Events Service is charged with inspecting the environment and collecting context information.
- The Policy Manager is responsible for analyzing and interpreting the policy. The Policy is a set of rules that define the behavior to be adopted by the systems according to some criteria. The specification of the policy can be updated at any time to support the dynamic changes of the users' needs and/or the execution context.

The adaptation manager is placed above the storage system. The component inspects the context in order to select the consistency model to use from a list of models proposed by the storage system, which makes the solution easy to integrate with systems supporting configurable consistency since it does not affect the existing architectures or the internal operations. However, the solution as it is does not fit precisely systems that offer a unique consistency model like relational databases.

The inputs for the adaptation manager consist of (i) the policy specification and (ii) the context information. Firstly, the policy is specified and can be updated by the user at any time. Within the adaptation manager, the Policy manager is tasked with interpreting the policy and identifying necessary adaptations. Secondly, context information is gathered by the event service, which operates over the storage system. This information is then managed by the context manager, which collaborates with the event service to handle variables related to the context.

When receiving an operation that requires dynamic consistency, the Decision manager collaborates with both the context and policy managers to select the appropriate consistency model. This is achieved by matching context variables with the policy rules that are pertinent to the operation's consistent bloc.

When receiving an operation that requires dynamic consistency, the Decision manager collaborates with the context and policy managers to select the appropriate consistency model. For this, it matches variables of context to rules of policy related to the operation consistent bloc.

E. Specification of the Adaptation Policy

An XML (Figure 4) specification is adopted to describe the different policies of consistency that are defined by consistency models and associated context descriptors. This specification provides the flexibility to describe consistency requirements and to ensure dynamic adaptation of the adaptation criteria.

4. Performance Evaluation

To evaluate the proposed solution, we used YCSB (Yahoo! Cloud Serving Systems Benchmark) [36], which is used mainly for benchmarking database systems for cloud computing and big data. YCSB implements many APIs to connect to relational and a large number of NoSQL databases and allows configuring various parameters such as the percentage of read and write operations, the number of

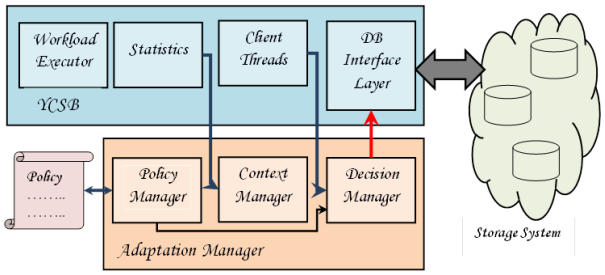

Figure 5. Implementation for consistency adaptation with YCSB

```
<policy>

<pol_db1 name="lowLatency">
<cmr>QUORUM</cmr><cd><minWL>0</minWL><maxWL>1000000</maxWL></cd> </pol_db1>

<pol_db1 name="highLatency">
<cmr>ONE</cmr><cd><minWL>1000000</minWL><maxWL>500000000</maxWL>
</cd></pol_db1>

<pol_db2 name="low_th_Lat">
<cmr>QUORUM</cmr><cd><minRL>0</minRL><maxRL>1000000</maxRL><minTh>3000</min
Th> <maxTh>100000</maxTh></cd> </pol_db2>

<pol_db2 name="meduim_th_Lat">
<cmr>TWO</cmr><cd><minRL>1000000</minRL><maxRL>100000000</maxRL><minTh>3000<
/minTh><maxTh>100000</maxTh></cd> </pol_db2>

<pol_db2 name="highLatency">
<cmr>ONE</cmr><cd><minRL>0</minRL><maxRL>100000000</maxRL><minTh>0</minTh>
<maxTh>3000</maxTh></cd> </pol_db2>

</policy>
```

Figure 6. Policy used by experiment_1.

threads and the consistency model. The output of YCSB measures systems performances in terms of throughput and latency. For the storage system, we chose Cassandra as it offers a configurable consistency with a rich set of consistency models [10].

A. Tests Methodology

Our experiments uses a cluster composed of five virtual nodes deployed on a virtualized platform composed of two servers. Every node is equipped with a CPU of 2.40 GHz and a hard disk of 100 GB.

Each node in the cluster is a separate machine running Apache Cassandra, collaboratively working to store and manage data. Among these nodes, one is configured to run YCSB. This node acts as a client that generates database operations in the Cassandra cluster. The YCSB tool node executes predefined workloads, simulates different types of database operations, and collects performance metrics such as throughput and latency.

Throughput refers to the number of operations that the database system being tested can handle per unit of time. Throughput is typically measured in operations per second (ops/sec). When running a benchmark test using YCSB, it executes a defined set of database operations, such as reads, writes, updates, and deletes, based on the specified workload. After the test is completed, YCSB calculates and reports the throughput, which indicates how many of these operations were successfully completed in one second. High throughput means that the database can handle a large number of operations quickly, which is often desirable in scenarios where performance and responsiveness are critical.

Latency represents the delay between the initiation and completion of an operation, such as a read, write, update, or delete. It is typically measured in milliseconds (ms) or microseconds (us). This metric is crucial for understanding how long a database takes to respond to individual requests under various conditions. Read Latency refers to the time taken to complete a read operation. Meanwhile, write latency measures the time taken to complete a write operation.

The implementation presented in (Figure 5) serves as a proof of concept for our tests. In this setup, YCSB initiates the workload and collaborates with the adaptation manager to meet consistency requirements. The adaptation manager receives two primary inputs: the policy file and the output from YCSB. The Decision Manager, at regular intervals (every 30 seconds), compares the context information against the policy in order to decide the appropriate consistency model for every bloc. The context information is sourced by the context manager from YCSB's output.

For our experiments, we employed Workload A in YCSB, known for its balanced distribution of 50% read and 50% write operations. Each test was conducted over a period of 30 minutes. The key steps of the testing process, included:

- Clearing the database to ensure a clean state for each test.
- Loading data for 10 minutes to adequately populate the database.
- Running the benchmark test for a duration of 30 minutes.
- Collecting and analyzing the output from YCSB. In cases where an anomaly was detected, such as a node shutting down, we addressed the issue, repeated the test. Otherwise, we proceeded to record the obtained values.

B. Experiment_1

This experiment aims to demonstrate how to express user needs in a policy and to measure the performances of an adaptive dynamic consistency model. The model used here equitably divided operations into four consistent blocs: the two static blocs "sb1" and "sb2" follow strong and weak consistency, respectively, while the dynamic blocs "db1" and "db2" use the policy defined by the specification of the policy in (Figure 6). "db1" uses a simple policy to switch between two policies according to "write_latency". "db2" defines three rules of policy based on an adaptation criterion composed of "throughput" and "read_latency".

The proposed schema for policy demonstrates the ease

of expressing consistency requirements and the flexibility in supporting different aspects of the adaptive consistency model.

The graphs displayed below are generated by launching strong, weak and our adaptive consistency models. The number of threads starts at 4 and is then incrementally increased in steps of 250, beginning from 250 and continuing up to 5000. This allows us to illustrate the system's performance in terms of throughput (see Table I and Fig. 7), Read latency (see Table II and Fig. 8), and Write latency (see Table III and Fig. 9).

TABLE I. Throughput Evaluation - Experiment 1

Thrs nbr	Strong(Ops/s)	Adapt.(Ops/s)	Weak.(Ops/s)
4	1309	2053	2283
250	2053	2442	2920
500	2442	2814	3805
750	2601	2849	4106
1000	2424	3150	3893
1250	2194	3008	3716
1500	2318	2761	3946
1750	2247	3646	4513
2000	2761	3522	4283
2250	2796	3203	4141
2500	2849	4000	4548
2750	2938	3769	4654
3000	2796	3858	5699
3250	2619	3327	5504
3500	2477	3929	5557
3750	2460	3469	4637
4000	2460	4000	5663
4250	2584	4336	5292
4500	2265	4424	5575
4750	2318	3398	4389
5000	2300	3911	4601

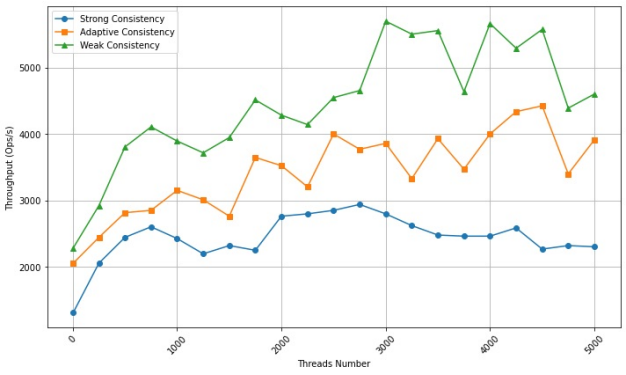

Figure 7. Throughput evaluation- Experiment_1

Figure 7 clearly demonstrates the variations in throughput across different consistency models. For instance, at 4 threads, weak consistency yields a throughput of 2283 Ops/s, significantly higher than the 1309 Ops/s achieved by strong consistency. Similarly, the adaptive consistency model shows a throughput of 2053 Ops/s, placing it between the strong and weak models. This trend is consistent as the number of threads increases. At 1000 threads, weak consistency reaches a throughput of 3893 Ops/s, whereas strong consistency is at 2424 Ops/s, and adaptive consistency manages 3150 Ops/s. Notably, at 5000 threads, the throughput for weak, strong, and adaptive consistencies are 4601 Ops/s, 2300 Ops/s, and 3911 Ops/s, respectively. These figures demonstrate the capability of the adaptive model to achieve intermediate throughput levels. While the adaptive process incurs some latency due to policy management, its impact on throughput is marginal, as evidenced by the adaptive model consistently maintaining intermediate performance levels between the strong and weak consistency models throughout various thread counts.

TABLE II. Read Latency Evaluation - Experiment_1

Threads nbr	Strong (us)	Adaptive (us)	Weak (us)
4	2796.35	5598.67	3548.36
250	265957.55	129179.44	75987.72
500	265957.55	197568.50	121580.43
750	319148.92	296352.57	189969.48
1000	471124.71	372340.29	265957.55
1250	653495.53	471124.71	410334.33
1500	638297.84	623100.16	387537.98
1750	820668.65	509118.40	410334.33
2000	896656.38	600303.81	493921.07
2250	896656.38	737081.92	607902.82
2500	919452.73	661094.19	577507.46
2750	1003039.47	759878.27	653495.53
3000	1185410.28	919452.73	516717.42
3250	1618540.96	1056230.84	592705.14
3500	1474164.18	911854.06	668692.86
3750	1588145.94	1139817.58	645896.51
4000	1694528.68	1033434.49	661094.19
4250	1679331.35	965045.43	797872.30
4500	2165653.39	1033434.49	782674.62
4750	2127659.36	1466565.17	1109422.56
5000	2325227.86	1284194.35	1071428.52

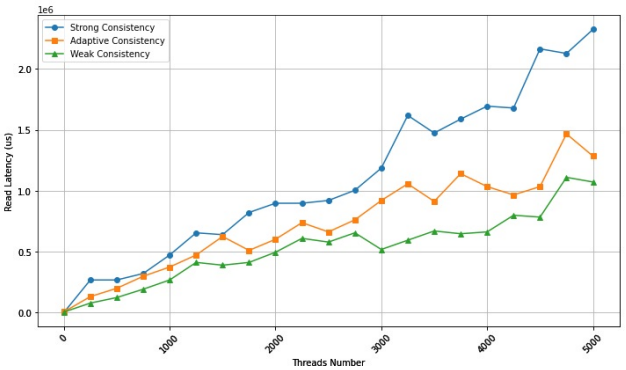

Figure 8. Read Latency evaluation- Experiment_1

Figure 8 presents a detailed comparison of read latency across the strong, weak, and adaptive consistency models as the number of threads varies. For instance, at just 4 threads, the read latency for strong consistency is recorded at 2796.35 microseconds (us), compared to 3548.36 us for weak consistency and 5598.67 us for adaptive consistency. As the number of threads increases, these latencies evolve distinctively. At 1000 threads, the read latencies are significantly different: 471124.71 us for strong consistency, 265957.55 us for weak consistency, and 372340.29 us for adaptive consistency. This trend is maintained at higher thread counts; for example, at 5000 threads, the read latencies are 2325227.86 us for strong consistency, 1071428.52 us for weak consistency, and 1284194.35 us for adaptive consistency. These values highlight the trade-offs inherent in each model. While weak consistency generally offers lower latency, suggesting faster access times, it does so potentially at the expense of data freshness or accuracy. On the other hand, strong consistency, with its higher latency, ensures data integrity and consistency but at the cost of speed. The adaptive model seeks to balance these aspects, achieving intermediate latency levels that suggest a compromise between data integrity and access speed. This balance is particularly evident at higher thread counts, where the adaptive model manages to maintain latencies that are significantly lower than strong consistency but slightly higher than weak consistency, showcasing its ability to adapt based on the context and workload demands.

TABLE III. Write Latency Evaluation - Experiment_1

Threads nbr	Strong(us)	Adaptive(us)	Weak(us)
4	750.74	698.62	529.13
250	11261.26	10510.52	8283.14
500	21771.78	17267.28	12048.22
750	29279.28	24024.04	15060.25
1000	42792.79	31531.53	25602.43
1250	54054.05	41291.28	24096.40
1500	59309.30	54054.05	37650.61
1750	73573.59	48048.04	36144.58
2000	84084.07	57807.82	44427.72
2250	76576.58	69069.08	47439.76
2500	86336.32	63813.80	52710.86
2750	91591.60	74324.32	54216.86
3000	118618.63	87087.10	49698.79
3250	155405.41	97597.59	55722.90
3500	141891.89	87087.10	60993.97
3750	154654.67	108108.11	54969.90
4000	166666.67	100600.61	64006.04
4250	162162.16	97597.59	78313.26
4500	187687.68	102852.86	78313.26
4750	202702.70	138888.90	101656.62
5000	215465.48	125375.39	103162.66

Figure 9 shows write latency comparisons across strong, weak, and adaptive consistency models. At 4 threads, strong consistency has a latency of 750.74 us, compared to adaptive consistency at 698.62 us and weak consistency at 529.13 us. This pattern continues at higher thread counts; for instance, at 5000 threads, the latencies are 215465.48 us for strong, 125375.39 us for adaptive, and 103162.66 us for weak consistency.

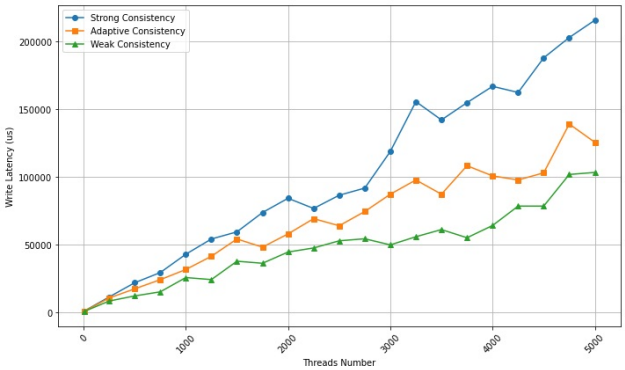

Figure 9. Write Latency evaluation- Experiment_1

These values indicate that, similar to read latency, weak consistency offers the lowest write latency, strong consistency the highest, and adaptive consistency a balanced intermediate. This demonstrates the adaptive model's ability to modulate write latency between the speed of weak consistency and the data integrity of strong consistency

Experiment 1 provides a comparative analysis of strong, weak, and adaptive consistency models in terms of throughput, read latency, and write latency. For throughput, weak consistency consistently outperforms strong consistency, with adaptive consistency achieving intermediate levels. This is evidenced by higher throughput values for weak consistency across various thread counts, indicating its efficiency in managing data operations. In terms of latency, weak consistency generally offers lower latency, suggesting faster access, while strong consistency, prioritizing data freshness, shows higher latency. Adaptive consistency successfully balances between these two, maintaining intermediate latency levels.

We note that the adaptation process incurs a slight increase in latency, stemming from the periodic review of the policy file and the management of context variables. However, this marginal increase does not notably impact the overall system performance, as the results demonstrate that the adaptive dynamic consistency achieves intermediate levels of performance.

C. Experiment_2

This experiment is designed to demonstrate the efficiency of dynamically adapting consistency levels. Initially, the system is set to operate under weak consistency. At the 500-second mark, we adapt the policy to switch to strong consistency. Then, at 800 seconds, the policy is adjusted again to revert to weak consistency. These adaptations are efficiently executed by simply editing the policy file.

The results demonstrate the effect of dynamic adaptation on system performance. Before the policy change, the throughput, as illustrated in (Table IV and Figure 10), maintains a relatively high level, averaging around 5619 ops/s between 300 to 400 seconds. However, after the adaptation at 500 seconds (point 'a' on the graph), the throughput decreases significantly, dropping to 4738 ops/s by 550 seconds. This trend continues until the second adaptation at 800 seconds (point 'b'), where the throughput begins to recover, evidenced by an increase to 5571 ops/s at 850 seconds, and further to 6023 ops/s at 1000 seconds.

TABLE IV. Throughput evaluation - Experiment_2

Timeline (s)	Throughput (ops/s)
100	4595
150	5595
200	5714
250	5357
300	5619
350	5619
400	5642
450	5833
500	5714
550	4738
600	4952
650	4547
700	4690
750	4714
800	4619
850	5571
900	5738
950	5095
1000	6023

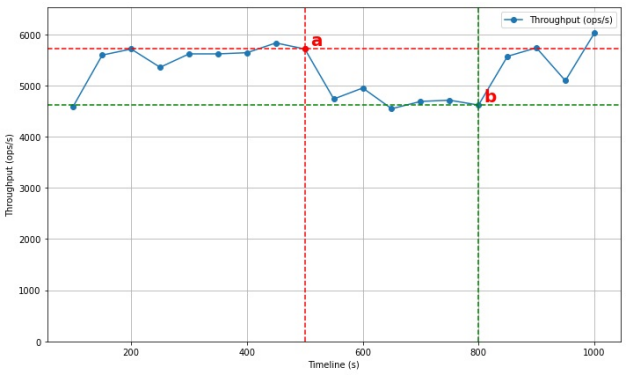

Figure 10. Throughput evaluation - Experiment_2

The results illustrated in(Table V and Figure 11) shed light on how dynamic adaptation affects read latency. As indicated in Figure 11, prior to the policy change at 500 seconds, read latency fluctuates, with a notable peak of 603846.10 microseconds (us) at 250 seconds, but generally stays around the range of 538461.42 us to 549999.88 us between 300 and 450 seconds. Following the adaptation at 500 seconds (point 'a' on the graph), there's a noticeable increase in latency, peaking at 676922.96 us by 700 seconds. However, after the second adaptation at 800 seconds (point 'b'), the latency begins to decrease, eventually reaching significantly lower levels such as 515384.50 us at 900 seconds and 211538.40 us at 1000 seconds. This demonstrates the system's ability to revert to more efficient latency levels following the adaptive adjustments.

TABLE V. Read Latency Evaluation - Experiment_2

Timeline (s)	Read Latency (us)
100	638461.48
150	573076.81
200	523076.86
250	603846.10
300	538461.42
350	542307.69
400	549999.88
450	538461.42
500	538461.42
550	653846.04
600	642307.57
650	665384.50
700	676922.96
750	634615.38
800	665384.50
850	538461.42
900	515384.50
950	603846.10
1000	211538.40

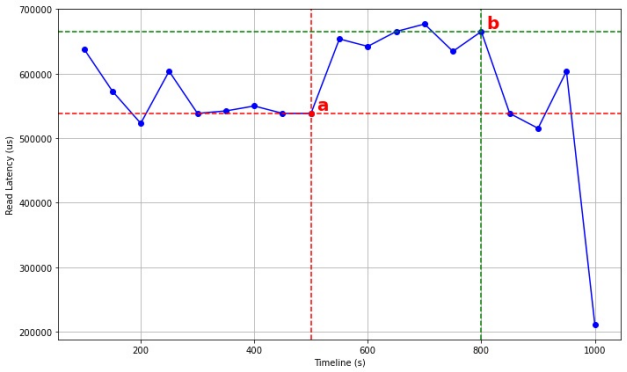

Figure 11. Read Latency evaluation - Experiment_2

Similar to the read latency pattern, the write latency, as illustrated in Table VI and Figure 12, also reflects the impact of dynamic adaptation. Initially, it shows some variation, like peaking at 584615.37 microseconds (us) at 250 seconds. Following the policy change at 500 seconds, there's an increase, reaching up to 639999.98 us at 700 seconds. After the second adaptation at 800 seconds, write latency decreases, realigning with earlier levels, indicative of the system's efficient response to policy adjustments. This trend mirrors the adjustments observed in read latency,

underlining the adaptability of the system under different consistency policies

TABLE VI. Write Latency Evaluation - Experiment_2

Timeline (s)	Read Latency (us)
100	638461.48
150	573076.81
200	523076.86
250	603846.10
300	538461.42
350	542307.69
400	549999.88
450	538461.42
500	538461.42
550	653846.04
600	642307.57
650	665384.50
700	676922.96
750	634615.38
800	665384.50
850	538461.42
900	515384.50
950	603846.10
1000	211538.40

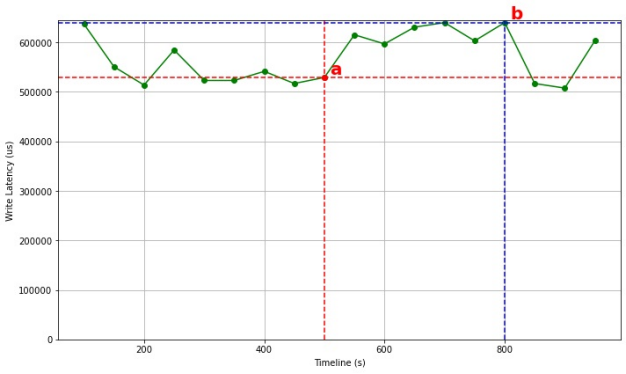

Figure 12. Write Latency evaluation - Experiment_2

The results from Experiment-2 clearly illustrate the immediate impact of policy adaptation on system performance, especially in terms of throughput and latency. Notably, the performance changes align precisely with the moments of adaptation at 500 seconds and 800 seconds, matching the policy checking interval of 30 seconds. This indicates a prompt response to policy adaptations. However, for longer periods for checking the policy, it would be beneficial to implement mechanisms that can detect policy changes and report them to the decision manager. Extending our tests to a large-scale level promises to provide interesting results. Indeed, communications delays and disconnection rates in the case of geo-distributed datacentres vastly exceed the characteristics of our virtualization platform. This context is known to lead to increase considerably latency and also to decrease throughputs. The proposed solution would then make the adaptation cost more negligible and allow us to take more advantage of its mechanism.

5. Conclusion and Future Work

Adaptive consistency models provide a flexible and efficient tradeoff between consistency and availability by automatically switching between different consistency models according to adaptation criteria. Through this research, we aim to bridge the gap in existing data management strategies for adaptive consistency by offering a solution that is flexible and immediately responsive to changing requirements about consistency in distributed storage environments.

Our dynamic adaptive consistency model proposes to offer more flexibility and dynamicity. The model classifies operations into consistent blocs according to consistency requirements. This classification allows for a more granular and efficient management of consistency levels. A static consistent bloc uses a unique model of consistency. In contrast, adaptive consistent blocs follow a policy of consistency that adapts the used consistency model according to the context. A policy defines alternative consistency models to switch according to the context. The context descriptor regroups all the criteria about the context that influence the choice of consistency, which permits the user to express their needs more accurately.

The proposed policy-based architecture supports the different aspects that characterize our dynamic adaptive model. The architecture is implemented as a middleware above the storage system. The adaptation process requires introducing the policy specification and the context information to interact with the storage system and execute operations according to the specified policy of consistency. The solution is easily integrated with systems supporting configurable consistency.

The experimentation part demonstrates the high flexibility of the proposed solution in expressing users' needs through consistent blocs and context descriptors. It also proves that the dynamic adaption of the policy is taken into account immediately by the system. The performance evaluation shows interesting and promising results.

Our experimental results, though promising, were limited to a virtualization platform and may not fully encapsulate the challenges of large-scale, geo-distributed data centers. Future work should involve testing the solution in these environments, where issues like higher latency and lower throughput are significant, to provide a more comprehensive evaluation of its adaptability and performance. Additionally, we plan to explore two other directions in our future work. Firstly, we aim to improve the specification of policies and the construction of consistent blocs using machine learning and semantic tools. Secondly, we intend to extend the solution to support more transactional properties and to conduct an experimental study on a larger-scale platform.

References

[1] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502.

[2] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, 2012.

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

[4] D. Mosberger, "Memory consistency models," *ACM SIGOPS Operating Systems Review*, vol. 27, no. 1, pp. 18–26, 1993.

[5] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *computer*, vol. 29, no. 12, pp. 66–76, 1996.

[6] S. Sakr and A. Y. Zomaya, *Encyclopedia of big data technologies*. Springer International Publishing, 2019.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[8] A. W. Services, "Consistency summary - amazon simpledb," accessed: March 2023. [Online]. Available: https://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/ConsistencySummary.html

[9] AmazonWebServices, "Amazon dynamodb: Read consistency," accessed: March 2023. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html

[10] A. Cassandra, "Apache cassandra," accessed: March 2023. [Online]. Available: http://cassandra.apache.org/

[11] 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*, 2013, pp. 309–324.

[12] 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.

[13] H.-E. Chihoub, M. Pérez, G. Antoniu, and L. Bougé, "Chameleon: customized application-specific consistency by means of behavior modeling," Ph.D. dissertation, INRIA Rennes-Bretagne Atlantique, 2013.

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

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

[16] S. P. Phansalkar and A. R. Dani, "Tunable consistency guarantees of selective data consistency model," *Journal of Cloud Computing*, vol. 4, no. 1, pp. 1–12, 2015.

[17] S. Phansalkar and A. Dani, "Predictive models for consistency index of a data object in a replicated distributed database system'," *WSEAS Transactions on Computers*, vol. 14, pp. 395–401, 2015.

[18] H. Sun, B. Xiao, X. Wang, and X. Liu, "Adaptive trade-off between consistency and performance in data replication," *Software: Practice and Experience*, vol. 47, no. 6, pp. 891–906, 2017.

[19] L. Lamport, "Paxos made simple, fast, and byzantine." in *OPODIS*, vol. 3, 2002, pp. 7–9.

[20] S. P. Kumar, S. Lefebvre, R. Chiky, and E. Gressier-Soudan, "Calibre: A better consistency-latency tradeoff for quorum based replication systems," in *Database and Expert Systems Applications: 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part II 26*. Springer, 2015, pp. 491–503.

[21] M. T. González-Aparicio, A. Ogunyadeka, M. Younas, J. Tuya, and R. Casado, "Transaction processing in consistency-aware user's applications deployed on nosql databases," *Human-centric Computing and Information Sciences*, vol. 7, pp. 1–18, 2017.

[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, no. 1, pp. 945–960, 2022.

[23] LightKone, "Minidote github repository," https://github.com/LightKone/Minidote, accessed: March 2023.

[24] J. Dizdarevic, Z. Avdagic, F. Orucevic, and S. Omanovic, "Advanced consistency management of highly-distributed transactional database in a hybrid cloud environment using novel r-tbc/rta approach," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–31, 2021.

[25] P. Li, L. Pan, X. Yang, W. Song, Z. Xiao, and K. Birman, "Stabilizer: Geo-replication with user-defined consistency," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 359–369.

[26] E. A. Lee, R. Akella, S. Bateni, S. Lin, M. Lohstroh, and C. Menard, "Consistency vs. availability in distributed cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–24, 2023.

[27] C. Gomes, M. N. de O. Junior, B. Nogueira, P. Maciel, and E. Tavares, "Nosql-based storage systems: influence of consistency on performance, availability and energy consumption," *The Journal of Supercomputing*, pp. 1–25, 2023.

[28] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, Feb 2001.

[29] P. Temdee, R. Prasad, P. Temdee, and R. Prasad, "Context and its awareness," *Context-Aware Communication and Computing: Applications for Smart Environment*, pp. 15–31, 2018.

[30] N. Nouali-Taboudjemat, O. Nouali, and H. Drias, "La validation dynamiquement adaptable des transactions mobiles. une approche sensible au contexte utilisant des politiques d'adaptation." *TSI-Technique et Science Informatiques*, vol. 30, no. 3, p. 243, 2011.

[31] A. Hassani, A. Medvedev, P. D. Haghighi, S. Ling, M. Indrawan-Santiago, A. Zaslavsky, and P. P. Jayaraman, "Context-as-a-service platform: exchange and share context in an iot ecosystem," in *2018 IEEE International Conference on Pervasive Computing and*

Communications Workshops (PerCom Workshops). IEEE, 2018, pp. 385–390.

[32] P. Rosenberger, D. Gerhard, and P. Rosenberger, "Context-aware system analysis: Introduction of a process model for industrial applications." in *ICEIS (2)*, 2018, pp. 368–375.

[33] Z. Mahfoud and N. Nouali-Taboudjemat, "A policy-based architecture for adaptability in web services transaction," in *Proceedings of the International Conference on Web Informations and Technologies (ICWIT 2013)*, Tunisia, 2013.

[34] T. Karthick, P. Gouthaman, L. Anand, and K. Meenakshi, "Policy based architecture for vehicular cloud," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. IEEE, 2017, pp. 118–124.

[35] A. V. Ventrella, F. Esposito, and L. A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 80–87.

[36] "Yahoo cloud serving benchmark," https://github.com/brianfrankcooper/YCSB/wiki, accessed: March 2023.

Zohra MAHFOUD Is a PhD student at Houari Boumediene University of Science and Technology (USTHB) in Algiers, Algeria. Teacher of Computer Science and member of LIM laboratory at Bouira University, Algeria, The author's research journey has spanned web services, cloud computing, and big data, all centered around the influence of emerging technologies on transaction management. Their primary research contribution focuses on proposing adaptive and dynamic mechanisms to enhance system performance.

Nadia NOUALI-Taboudjemat obtained her Doctorate (PhD) Degree in computer science from Houari Boumediene University of Science and Technology USTHB Algiers, Algeria in 2007. Before that. She is currently working as a Director of Research, permanent full-time senior researcher at CERIST (Research Centre in Scientific and Technical Information), Her research interests include pervasive/ubiquitous computing, context-aware computing, distributed/parallel and mobile computing, Transaction processing, wireless networks, ICTs, Web mapping, Cloud computing, security. Her recent researches include Big Data management, Big Graphs processing, High Performance Computing and Internet of things (IOT). She is particularly interested by the application domain of ICT-based disaster management and smart cities and green applications. She worked on several projects as the principal investigator/leader and she published many papers in international peer-reviewed journals and conference proceedings. She also served as TPC member of many international conferences, guest editor of special issues. She chairs the Steering Committee of the International ICT-DM Conference (ICT for Disaster Management) of which she is the main founding member who initiated the first edition ICT-DM'2014 in Algiers, Algeria. Since then, ICT-DM is held every year in a different country throughout the world.