Adaptive Quorum-inspired SLA-Aware Consistency for Distributed SDN Controllers

Fetia Bannour, Sami Souihi and Abdelhamid Mellouk
LISSI Laboratory, University of Paris-Est Créteil (UPEC), France

Abstract—**This paper addresses the knowledge dissemination problem in distributed SDN control by proposing an adaptive and continuous consistency model for the distributed SDN controllers in large-scale deployments. We put forward a scalable and intelligent replication strategy following Quorum-replicated consistency: It uses the read and write Quorum parameters as adjustable control knobs for a fine-grained consistency level tuning. The main purpose is to find, at runtime, appropriate partial Quorum configurations that achieve, under changing network and workload conditions, balanced trade-offs between the application's continuous performance and consistency requirements. Our approach was implemented for a CDN-like application that we designed on top of the ONOS controllers. When compared to ONOS's static consistency model, our model proved efficient in minimizing the application's inter-controller overhead while satisfying the SLA-style application requirements.**

I. Introduction

Existing SDN controller platforms have been architected according to different SDN control plane designs to meet specific requirements in terms of scalability, availability and performance. Consistency has also been regarded as an essential design principle for the physically-distributed, yet logically-centralized, SDN platforms [1]. The latter use conventional consistency models to manage the distributed state among the controllers in the cluster. The consistency models used in SDN can be categorized into *strong*, *eventual* and *weak* [2, 3, 4]. These static models have both advantages and drawbacks.

In large-scale SDNs, the *Strong Consistency* control model is extremely expensive and costly to maintain for certain applications. It requires important synchronization efforts among the controller replicas at the cost of causing serious scalability and performance issues. By contrast, the *Eventual Consistency* control model implies less inter-controller communication overhead as it sacrifices the strict consistency guarantees for higher availability and improved performance. In practice, many scalable applications in modern distributed storage systems like Apache's Cassandra [5] and Amazon's Dynamo [6] opt "by default" for eventual consistency to provide such requirements on a large scale. However, these applications might suffer from the associated relaxed consistency guarantees that may temporarily allow for too much inconsistency.

Recent research works in the area of distributed SDN control have explored the concept of *Adaptive Consistency* control for various applications [7, 8, 9, 10, 11]. Such categories of consistency models follow different adaptation strategies that mainly focus on dynamically adjusting the levels of consistency at run-time under varying network conditions in order to meet the application-defined consistency and performance needs. Unlike strong and eventual consistency options, adaptive consistency control models leverage the broad space of intermediate consistency degrees between these two extremes. They use time-varying consistency levels to support balanced real-time trade-offs between the desired consistency and performance requirements that can be specified in the application-defined Service-Level Agreements (SLAs) [12].

Main contribution: In this paper, we put forward an adaptive consistency model based on eventual consistency for the SDN controller applications that are deployed in large-scale networks. Notably, we target the class of applications that tolerate relaxed forms and degrees of eventual multi-consistency for the sake of scalability and performance, but yet can benefit from improved consistency features. More specifically, we propose a scalable and intelligent replication strategy following Quorum-replicated consistency models. The proposed model uses the Quorum replication parameters as the control knob, allowing for an adaptive fine-grained tuning and control over the consistency-performance trade-offs. These real-time trade-offs should provide minimal application inter-controller overhead while satisfying the application's continuous performance and consistency requirements specified in the given SLA. Our approach was implemented on the ONOS controller platform.

Outline: The rest of this paper is organized as follows: In Section II, we conduct a background review of eventual consistency models in modern distributed data-store systems. Inspired by the scalable consistency techniques used in these popular data-stores, we present, in Section III, our adaptive and continuous Quorum-inspired consistency model for the distributed SDN controllers. In Section IV, we describe the methodology for implementing our consistency strategy on a CDN-like application we designed on top of ONOS. Section V explains the test scenarios we developed to assess our proposal, and discusses the experimental results. Finally, Section VI gives some concluding remarks and points out possible directions of improvement of the proposed approach.

II. Background on eventual consistency in distributed data-stores

A. Consistency and performance metrics:

Guaranteeing the consistency of replicated data in distributed database systems has always been challenging. Today's fundamental consistency models (e.g. strong consistency, causal consistency, eventual consistency) ensure different discrete levels of consistency guarantees. For instance, the strong consistency model offers up-to-date data, but at the cost of high latency and low throughput. As a result, weaker forms of consistency -most notably the popular notion of eventual consistency- have been widely adopted in modern data-stores which need to be highly-available, fast and scalable [5, 6]. Despite being regularly desirable in practice for the latency and throughput benefits they offer, eventual consistency models provide no bounds on the inconsistency of the returned data. Another limitation of these models is that the trade-offs they make among consistency and performance are difficult to assess as stated by the CAP and PACELC theorems [13].

Yu and Vahdat proposed TACT [14] which fills in the consistency spectrum by providing a continuous multi-dimensional consistency model. The latter can be leveraged by replicated Internet services to dynamically tune their fine-grained consistency-performance trade-offs based on client, service and network features. The authors quantify consistency by bounding the divergence of replicated data items in an application-specific manner using three metrics: *Numerical error*, *Order error* and *Staleness*. Bailis *et al.* [15, 16] developed probabilistic models to predict the expected consistency

978-3-903176-24-9 © 2019 IFIP

guarantees as measured by the *staleness* of reads observed by client applications in eventually-consistent Dynamo-style partial Quorums. The authors introduced PBS which provides bounds on the expected staleness in terms of versions (the *k-staleness* metric) and wall-clock time (the *t-visibility* metric). Another work [17] proposes a self-adaptive consistency approach called Harmony which embraces an intelligent estimation of the *stale read rate* metric in Cloud storage systems, allowing to adjust the consistency level at run-time according to application needs. That was achieved by elastically scaling up or down the number of replicas involved in read operations to preserve a low tolerable fraction of stale reads.

B. Adaptive consistency control

Modern distributed database systems supporting standard eventual consistency models suffer from the inevitable trade-offs between consistency and availability. To overcome this limitation, these systems introduced the concept of adaptive consistency to find appropriate consistency options depending on application needs and system conditions. In literature, adaptive consistency techniques have been broadly classified into two categories: *user-defined* and *system-defined* [18].

Modern storage systems like Cassandra fit into the category of user-defined adaptive consistency, as they offer multiple consistency options via built-in settings based on Quorum replication policies. Although they offer adaptive consistency on top of tunable consistency providing the applications with some control over the consistency-performance trade-offs, it is usually difficult for application developers to decide in advance about the required consistency options for a given request [18].

Unlike user-defined adaptive consistency where data should be mapped in advance to the desired consistency levels, system-defined adaptive consistency takes into account the fact that user and system behaviors may change dynamically over time making the consistency decision-making process challenging for application developers. That is why, system-defined techniques rely on system intelligence and adaptability to provide at run-time fine-grained control over the consistency guarantees. Many factors can be considered to dynamically predict the appropriate consistency like data access patterns, system load, and the application's consistency SLAs discussed in Section II-A. One famous form of system-defined adaptive consistency is the continuous consistency used in TACT [14]. Designing system-defined adaptive consistency requires careful considerations of the appropriate consistency adaptation strategy. In particular, existing adaptive mechanisms use different control knobs to be configured for consistency tuning such as the *consistency level*, the *artificial read delay*, the *replication factor* and the *read repair chance* [19].

III. The proposed adaptive Quorum-inspired consistency for SDN controllers

We propose a Quorum-based and *system-defined* adaptive consistency model for the distributed SDN controllers. Our approach is inspired by the Quorum consistency techniques used by modern data-stores [5, 6]. SDN controller platforms like ONOS rely on two consistency schemes with two levels of consistency: strong and eventual consistency [11]. While the first model is leveraged by the SDN applications that require strong correctness guarantees, the second model is intended for SDN applications that favor scalability and performance over strict consistency. Here, we target the second class of scalable applications that have relaxed consistency needs, but that can benefit from improved performance and automated SLA-aware consistency tuning at scale as offered by our proposed strategy.

A. A continuous consistency model for SDN

As discussed in [11], SDN applications can benefit from the continuous consistency model introduced by TACT [14], by continuously specifying their consistency needs using three metrics to capture the consistency spectrum and bound inconsistency: *Numerical Error*, *Order Error*, and *Staleness*. In this work, we focus on the type of applications whose consistency semantics can be expressed using data staleness as a metric to quantify consistency. With such SLA-style consistency metrics, such applications may avoid the challenges related to potentially *unbounded staleness* as in eventual consistency.

Generally speaking, the staleness metric measures data freshness in distributed data-stores: it describes how far a given replica lags behind in data operations in comparison to up-to-date replicas, either expressed in terms of time or versions. In the literature, the notion of data staleness falls into two common categories: staleness in time (*time-based staleness*) and staleness in versions (*version-based staleness*) [14, 15].

In this work, we adopt the staleness metric from a strictly time-based perspective. In our SDN application, we characterize staleness by an "Age of Information (AoI)" timeliness metric [6] that describes the difference between the query time of a data item and the last update time on that item. If the last successfully received update was generated at time $u(t)$ then its age at time t is $\Delta(t) = t - u(t)$. Besides, SDN applications can benefit from SLA-style performance metrics: We consider the read request latency as our continuous performance metric, and we assess the application inter-controller overhead.

B. Our Quorum adaptation consistency strategy for SDN

1) Quorum-replicated consistency

The Quorum size for reads (R) or writes (W) is the number of replicas that must acknowledge a read or write operation before considering it as successful. Different choices of Quorum configurations ensure different consistency guarantees:

- Strong consistency can be guaranteed with *strict quorums* that satisfy the condition that sets of replicas written to and read from need to overlap: $R + W > N$, given N replicas and the read and write quorum sizes R and W.
- Eventual consistency occurs with *partial quorums* that fulfill the condition that sets of replicas written to and read from need not overlap: $R + W \leq N$, given N replicas and the read and write quorum sizes R and W.

Traditionally, partial Quorum systems ensure eventually-consistent guarantees with no limit to the inconsistency of the returned data, which is not acceptable for certain applications. With PBS [15], it was possible for applications to quantify the consistency level (using staleness) and assess the staleness-latency trade-offs for partial Quorums. Building on these concepts, we suggest an adaptive consistency model for SDN applications using partial Quorums, given their latency and scalability benefits. To measure the applications' consistency semantics (e.g staleness) and meet their consistency needs (e.g bounded staleness), we leverage the continuous consistency model (Section III-A). Using eventually-consistent Quorums, it is possible to configure the size of read and write quorums such that $R + W \leq N$ to ensure different consistency levels (e.g degrees of staleness). These multiple configurations allow the applications to achieve various staleness-latency trade-offs.

2) Adaptive architecture

In the following, we describe the main architecture components of our adaptive consistency model (see Figure 1):

- `Application SLA Module`
  This module allows the SDN applications to express

their high-level SLA-style consistency (staleness) and performance (latency) needs. Our consistency model continuously measures the real-time metrics that quantify the consistency-latency trade-offs for our SDN applications.

- `Workload Identifier Module`
This module identifies the application's workload characteristics. It considers 3 workloads that are representative of different application scenarios [20]. The first workload has a balanced ratio between read and write operations. The second one represents a write-dominated scenario. The third one describes a read-intensive scenario.
- `Monitoring Module`
This module is responsible for periodically gathering the application traffic information in a non-intrusive manner. It measures the system KPIs for different read/write Quorum configurations and according to different application workload scenarios. These KPIs include the performance (response latency) and consistency (staleness) metrics related to client requests for specific application contents, as well as the generated read and write application overhead.
- `Automatic Module`
The choice of the size of read and write Quorums used for read and write operations is a fundamental factor that affects the application's consistency guarantees but also the network performance. However, selecting the right Quorum configuration is a non-trivial task. This module attempts to find the appropriate time-varying partial Quorum configurations while taking into account the current application workload conditions. The main objective is to minimize the overhead generated by the application (the scalability challenge) and potentially other network and application metrics, while satisfying the consistency and performance SLAs specified by the Application SLA Module. Our automatic module is fed with a set of application workload characteristics that are gathered by the Workload Identifier Module. In our case, it relies on a Machine Learning Module to predict the appropriate partial Quorum configuration for the determined workload, and then feed them to the Reconfiguration Module.
- `Machine Learning Module`
This module uses a Q-Learning (QL) Reinforcement Learning (RL) technique. The main idea is to train an *agent* that interacts with the *environment* by performing *actions* that change the environment, going from one *state* to another. These actions result in a *reward* received by the agent as an evaluation of its actions. That way, the agent learns some rules and develops a *policy* for choosing actions that maximize its reward. The QL update rule uses the *Q-function* representing the quality of an action in given state. This function is used for updating the *Q-table* at each episode. The agent should also achieve a strategy for balancing the exploration/exploitation trade-off inherent to RL. That dilemma consists in choosing a certain action at each episode: either to *exploit* the environment by selecting the best action at that specific time step given the knowledge provided by the Q-table, or to *explore* the environment by choosing random actions. After each action, the agent updates the Q-table.
In our case, the agent attempts to learn online the best combination of R and W in an environment built using our Monitoring Module. An action is defined as an update (increasing/decreasing) of R and W to certain possible values, thereby transforming the environment to a state defined by a new estimation of the network (overhead) and application (latency and staleness) metrics. The reward received by the agent for updating the Quorum parameter values is a function of the read and write overheads to be minimized and potentially other metrics. The agent learns also how to respect some constraints to satisfy the application needs specified in the given SLA.
- `Reconfiguration Module`
This module adjusts dynamically the values of R and W. It relies on the Automatic Module to optimize the quorum configuration. The reconfiguration process is a non-blocking process that is able re-configure at run-time the Quorum settings selected by the Automatic Module.
- `Quorum-based Replication Module`
Given the quorum replication settings, we adopt the following strategy when reviewing the techniques used by eventual consistency models in SDN controllers [11]:
  - Replication strategy: In existing SDN controller platforms like ONOS, eventually-consistent stores employ an optimistic replication technique that consists in replicating local updates across all controllers in the cluster, hence causing control plane overhead. Instead, we put forward a partial quorum replication strategy where an eventually-consistent data-store writes a data item on the local replica first and then sends it potentially to another set of replicas, obeying the given write quorum W. To serve read requests, we propose that the eventually-consistent data-store fetches the data from the local replica first and then potentially from another set of replicas, depending on the given read quorum R. This is in contrast to ONOS's strategy where the read requests are always processed by the local replica.
  - Reconciliation mechanism: In SDN controllers like ONOS, the optimistic replication strategy is complemented by a background reconciliation mechanism (e.g. Anti-Entropy). That periodic process ensures that the system state across all replicas eventually converges to the consistent state. This is particularly useful in repairing out-of-date replicas and fixing inconsistencies potentially resulting from controller failures. In this work, we assume that the system is reliable as we experiment with well-functioning emulated topologies in the absence of failures. Thus, we propose to deactivate the reconciliation protocol and focus on the replication strategy. However, we note that using additional Anti-Entropy (*expanding partial quorums* [15]) might be useful in particular cases where inconsistencies become high and can no longer be tolerated by the applications.

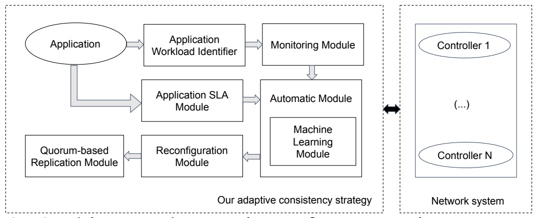


Fig. 1: Architectural overview of our consistency strategy

IV. Implementation approach on ONOS

In this section, we show the details for implementing our proposed adaptive consistency strategy on the ONOS controllers.

A. Design of a CDN-like application

To validate our proposal, we develop a new distributed Content Delivery Network (CDN) application running on top of a cluster of ONOS controllers in an emulated SDN network. Our application replicates contents from providers to hosting cache servers that are located in multiple geographical locations (ONOS domains) close to users. In our tests, we consider a single origin server in each ONOS domain. The main idea is to serve client hosts with the most up-to-date copies of the requested content and within a reasonable time (low latency).

More specifically, our application consists of two main components: An `ApplicationManager` and a `DistributedApplicationStore`. The Application Manager is in charge of creating a virtual network of cache servers and providing mesh connectivity between these server hosts. The Distributed Application Store persists and synchronizes the information received by the manager. It is backed by a distributed eventually-consistent map for storing the service's application state, namely the list of origin servers in the network and their respective set of generated contents: `EventuallyConsistentMap<OriginServerID, Set<Content>>`. Each content that is created on the origin server and then eventually propagated to cache servers has five properties; a `ContentName`, an `ID`, a real-time `CreationTime`, a `LogicalTimestamp` and a `Version`.

Each controller replica that is responsible for an ONOS domain operates on a local view of the eventually consistent map. That view consists of the local origin server from the same ONOS domain with its generated set of contents and other potential origin servers located in different ONOS domains in the network with their respective set of contents, as seen by the local replica after application state synchronization.

We also design a cached map that is local to each controller application instance and that represents the contents cached in the local CDN server within the same ONOS domain. The local cached map is closely linked to the local view of the eventually consistent map and it reflects the contents stored in the local CDN server. The latter performs the functions of an origin server and a cache server: It contains the contents created locally (origin server) and potentially other contents that are replicated from other origin servers (cache server). More specifically, on a local controller, updates to the eventually consistent state map might trigger specific actions to feed the local CDN server and then update the local cached map. If the update to the content is associated in the map with the local origin server, that means that the updated content has already been generated on that server. If the update to the content is associated in the map with another origin server from another ONOS domain, our application checks the relevance of that content. If the content is important to our application, then the update to the content gets automatically pulled from the origin server to the local CDN (cache) server and gets cached in the local `CachedMap <ContentName, Set<Content>>`.

B. State synchronization and content distribution

The custom eventually consistent map we use for the synchronization of our CDN application state is based on our own implementation of the `EventuallyConsistentMap <K,V>` distributed primitive. Indeed, the new implementation we propose for the eventual consistency map abstraction models the Quorum-inspired consistency discussed in III-B1.

In particular, it takes into account the size of the write Quorum W when replicating the updates related to our application's eventually consistent map among the controllers (see Figure 2). On each local replica, updates to the local map are queued in time in different `EventAccumulators` that are allocated for different controller peers. The latter are selected randomly, and their number depends on W. Whenever an event accumulator is triggered to process the previously

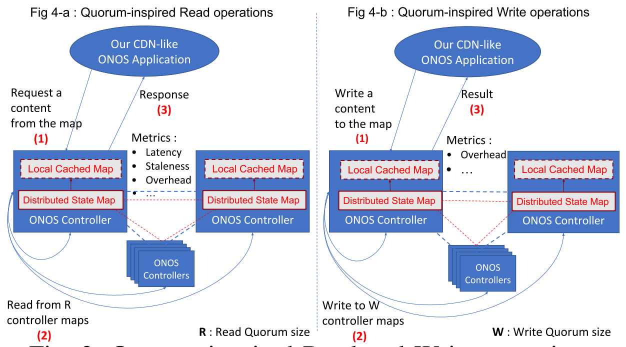


Fig. 2: Quorum-inspired Read and Write operations

accumulated events and propagate them to the associated peer, that peer is removed from the list of quorum peers. New updates will immediately trigger the creation of a new accumulator associated with a new randomly selected peer that is is added to the list of quorum peers. That accumulator will collect the updates along with the other event accumulators associated with the rest of the quorum peers. That way, we guarantee that updates to the eventually consistent map on a local replica are replicated at run-time to exactly W replicas including the local replica.

C. Content delivery to customers

During a client read operation, our controller application instance running on the local controller within the same ONOS domain as that client, receives the read request to be fulfilled following Quorum-inspired read consistency protocols (see Figure 2). If the read consistency level is higher than ONE ($R > 1$), then the local controller which serves in our case as the coordinator node sends the read request to the remaining randomly-selected controller replicas forming the read Quorum. The size R of the read Quorum including the local replica is set in advance by the read consistency level.

We use ONOS's `ClusterCommunicationService` to assist communications between the local controller and the rest of the replicas in the read Quorum. The local controller sends the read request message with a certain subject to each of the concerned controllers using that service's `sendAndReceive` method. It expects a future reply from each of these controllers which had already subscribed to that message subject.

That said, to serve the client read request for a certain content, each controller that has subscribed to the specified message subject receives the request and uses the application's handler function for processing the incoming message. The application instance on each controller replica of the read Quorum (including the local replica) consults the local cached map. Using that map, each application instance compares the cached versions of the requested content (`ContentName`) based on their `LogicalTimestamps` to determine the freshest version of the content. Then, it produces a reply containing the selected `Content` with its five properties discussed in IV-A and the IP address of the local cache server delivering that content.

The local controller replica playing the role of the coordinator waits for the read Quorum of replicas to respond. Then, it merges the R responses (including that from the local replica) to determine the location of the freshest version of the requested content among the concerned CDN servers (equal to R in our scenario). Finally, it sends the final response to the client and makes sure a host-to-host connectivity intent is added between the client and the determined cache server using the `ONOS Intent Framework`. Based on that response, the client which has issued a HTTP request specifying the URL of the requested content, is redirected, using our CDN-like strategy and a DNS resolution service, to the selected cache server to retrieve the specified version of the content. After each client request, our application collects the continuous consistency and performance metrics related to that request:

- Performance metrics:
  - Network-related metrics: We consider the application inter-controller overhead as a performance metric. We first capture all inter-controller traffic using TCP port 9876. Then, we filter the captured traffic based on some conditions to assess the application's inter-controller overhead due to write and read operations. Our goal is to minimize that overhead based on the application SLA and workload, as well as the network context.

$$AppOverhead = WriteOverhead + ReadOverhead \quad (1)$$

  - Client-centric metrics: We use the response time to a client request as a performance metric. It consists of the delay to fetch the proper version of the requested content from the local cached maps of the application instances running on the R controllers of the read Quorum (`Latency1`) and the delay to retrieve that version from the selected cache server host (`Latency2`).

$$ResponseTime = Latency1 + Latency2 \quad (2)$$

- Consistency metrics:
  As explained in Section III-A, we consider staleness from a strictly time-based perspective: It describes the Age of Information in terms of wall-clock time. Accordingly, the staleness of the application content `C` being returned by a read operation at a given time is measured as follows:

$$Staleness(C) = QueryTime - CreationTime(C) \quad (3)$$

  We also set the staleness ranges used in the consistency SLA based on the application content refresh rate.

V. PERFORMANCE EVALUATION

A. Experimental setup

Our experiments are conducted on an Ubuntu 18.04 LTS. We use ONOS 1.13, Mininet 2.2.1, and the provided *onos.py* script to start an emulated ONOS network on a single machine. Wireshark is used as a sniffer to capture the inter-controller traffic (TCP port 9876). In this section, we test our consistency approach which we will refer to as ONOS-WAQIC (ONOS-With Adaptive Quorum-Inspired Consistency) for brevity.

1) TCL-Expect scripts

We write two `Expect` scripts where we specify the required steps to automate the tasks for our test scenarios on ONOS-WAQIC. The first script (`main.exp`) is mainly used to connect to the Mininet CLI, launch the ONOS network topology, issue the client requests for contents, and collect the associated metrics. The second script (`onos.exp`) (called by the first script) is mainly used to run spawned processes that interact with the running ONOS instances through ONOS's CLI, and launch the application's CLI commands that we developed to perform many actions. The latter include setting R and W, and adding/updating the CDN contents to the cache server hosts (and to our application's eventually consistent map).

2) OpenAI Gym simulator

To implement our ML Module (Section III-B2) for our CDN-like application on ONOS-WAQIC, we build a simulator using the Python-based OpenAI Gym [21] RL toolkit. We build a new environment to simulate knowledge exchange in an ONOS cluster: We start by preparing an offline dataset using our TCL scripts. The dataset stores the information collected by our Monitoring Module about the clients' content requests. For a given client request (see Section IV-C), the returned information contains the current R and W values, the expected returned version of the content (content update step), the actual returned version of the content, *the staleness* of the returned content, the delay incurred in searching for the freshest version of the content from the R controller replicas (latency1), the read overhead, the write overhead and the application scenario determined by the Workload Identifier Module.

Our ML Module feeds the dataset to our simulator so that the agent learns online the R and W parameters. It learns indeed the Kernel Density Estimation (KDE) for each metric using the data of some clients. That data is selected with respect to the current configuration of R and W which was set following an action performed by the agent (see Section III-B2). Using KDE, our ML Module estimates the expected metrics for each selected Quorum configuration (used for updating the Q-table).

3) Various learning agent policies

We implemented three learning agents that adopt different policies. The latter are evaluated through five scenarios. Each scenario reflects a specific use case (e.g. a consistency-favoring application). To minimize the application's inter-controller overhead, our agents use the estimated overhead as a negative "reward" when performing actions (setting R and W) that change the environment state. The controlled and constrained agents are proposed to improve the simple greedy agent.

i) *A simple ϵ-greedy agent* [22]: It follows a simple ϵ-greedy policy with a fixed ϵ (ϵ is the exploration rate and (1-ϵ) is the exploitation rate). We test three ϵ-agents: ϵ-greedy5 (ϵ=0.5), ϵ-greedy10 (ϵ=0.10) and ϵ-greedy15 (ϵ=0.15).

ii) *A controlled ϵ-greedy agent*: This agent follows a dynamic ϵ-greedy strategy where ϵ decays as the episode count increases. The aim is to account for the fact that the agent learns more about the environment in time and becomes more confident and "greedy" for exploitation. To satisfy the latency and staleness thresholds, the first two agents reject at each episode any action-state violating these constraints by removing its value from the Q-table.

iii) *A constrained ϵ-greedy agent*: To help the agent learn to satisfy the SLA, we create a Q-constraint list that we update over the episodes. Its size is the number of potential R and W combinations such that $R + W \leq N$. The list represents the number of constraint violations by each Quorum configuration. The constraints are the latency and staleness SLA thresholds. During exploitation, we update that list and use it to generate a new Q-list containing the Quorum configurations that give less constraint violations. These configurations are exploited: They are compared using their values in the Q-table to select the best Quorum configuration (action-state) at that episode.

B. Results

To assess ONOS-WAQIC for our CDN-like application, we run our TCL scripts (Section V-A1) with a 5-node ONOS cluster according to different scenarios. In the latter, we use different partial Quorum configurations (R,W) and we follow various application workloads with respect to different ratios between read and write operations. We use this collected data as an input to our QL simulator (Section V-A2). In the simulator, we set α to 0.7 and the episode number to 1000. We also consider 3 scenarios that reflect different application needs in terms of performance and consistency (see Table I).

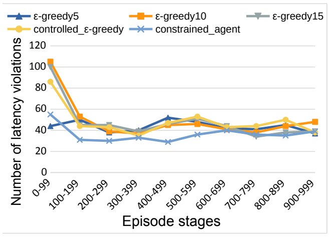
Test scenarios	Latency threshold (ms)	t_Staleness threshold (ms)	k_Staleness Version old
n°1	10	300000	3
n°2	50	180000	2
n°3	100	100000	1

TABLE I: Application SLA scenarios

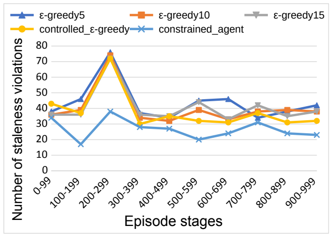
Using our dataset and knowing the refresh rate of our CDN-like application, we learn the t_staleness ranges, namely the relationship between the t_staleness value of a certain content being returned and by how many versions that returned content is old. As a result, estimating these ranges allowed us to set the time-based staleness thresholds in the SLA while having an idea about the associated version-based staleness thresholds.

In each scenario, our application expresses the performance and consistency SLAs using the latency and staleness thresholds (in ms). For example, in scenario n°3, our application which is consistency-favoring enforces this SLA: It expects that a read operation gets a reply in under 100ms and returns a content value no older than 100seconds (no older than 1 version stale). Accordingly, our consistency approach attempts to find the best R and W combination that minimizes the overhead while ensuring the desired performance-consistency trade-off. In Figures 3, 4 and 5, we show the results of our experiments for the three considered application scenarios.

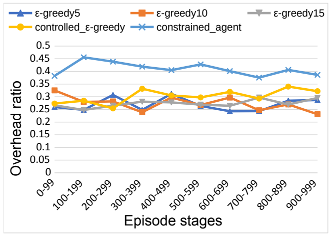
In particular, Figure 3 shows that, in a latency-sensitive application scenario, the constrained agent policy is the most appropriate. The number of constraint violations decreases with episode stages (Figures 3(a) and 3(b)) at the cost of adding little overhead as compared to the first two agent policies, but it remains 60% lower when compared to the standard ONOS implementation. We also notice that the three agents converge towards Quorum configurations where $R = 1$ (i.e. $(R = 1, W = 2)$, $(R = 1, W = 3)$ and $(R = 1, W = 4)$). This is due to the given strong constraint on latency.



(a) Latency violations



(b) Staleness violations



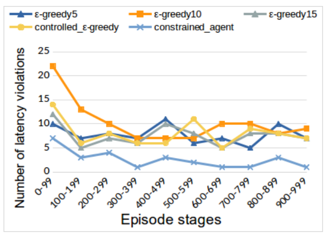
(c) Overhead ratio

Fig. 3: Scenario 1: Latency-sensitive application

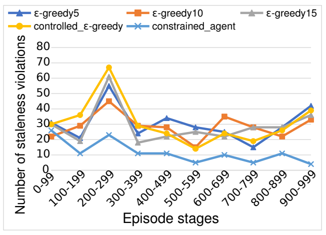
Figure 4 shows that, in a balanced application scenario, the constrained agent policy provides the best trade-offs at runtime between the application's latency and staleness needs (Figures 4(a) and 4(b)). It converges towards balanced Quorum configurations (i.e. $(R = 2, W = 2)$ and $(R = 2, W = 3)$).

As can be seen from Figure 5, in a consistency-favoring application scenario, both the constrained and ϵ-greedy5 agents perform well at reducing the staleness violations (Figure 5(a)). Besides, all agents comply well with the relaxed latency constraint. They converge towards a common Quorum configuration $(R = 3, W = 2)$. In this scenario, the gain in overhead is significant; almost 70% as compared to the standard ONOS.

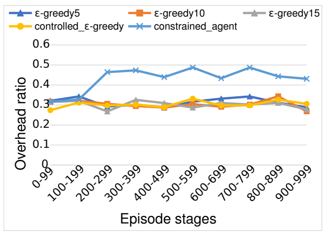
Other scenarios were tested like an application scenario where latency is favored and consistency is completely relaxed (”any”). Our results showed that, in such scenarios, the agents converge to common Quorum configurations $(R = 1, W = 1)$.



(a) Latency violations

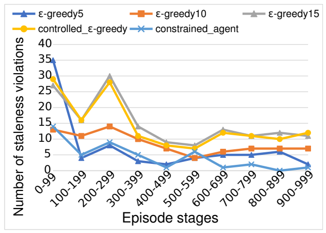


(b) Staleness violations

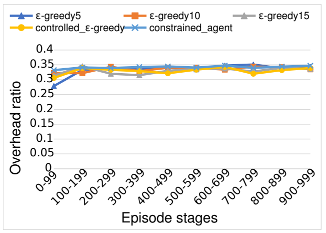


(c) Overhead ratio

Fig. 4: Scenario 2: Consistency/Latency-balancing application



(a) Staleness violations



(b) Overhead ratio

Fig. 5: Scenario 3: Consistency-favoring application

VI. Conclusion

In this paper, we studied the use of an adaptive and continuous consistency model for the distributed ONOS controllers following partial Quorum consistency. Our approach was implemented for a CDN-like application we designed on ONOS. It uses an intelligent Quorum-based replication strategy based on various QL approaches. Our experiments showed that the constrained ϵ-greedy approach we tested on a 5-node ONOS cluster proved efficient in helping our CDN-like application find at-runtime the appropriate read an write Quorum replication parameters. In fact, the determined time-varying partial Quorum configurations achieved, at runtime, balanced trade-offs between the application's continuous performance (latency) and consistency (staleness) requirements. These real-time trade-offs also ensured a substantial reduction in the application's inter-controller overhead (especially in a large-scale network) while satisfying the application SLAs.

In our experiments, we consider a fixed application workload scenario (a read-intensive scenario). Moving from one workload scenario to another (e.g a write-intensive scenario) is being addressed as part of our ongoing work. It would be also interesting to separate the read and write overheads when minimizing the application inter-controller overhead to investigate the impact on the generated R and W parameters. Finally, we note that our Quorum consistency model can be enhanced by leveraging the compulsory Anti-Entropy reconciliation mechanisms (*expanding partial Quorums*) [11]. Such mechanisms are useful in special cases (e.g. controller crashes) where the consistency observed by the applications is at high risk and cannot be fixed by only adjusting the Quorum sizes.

References

[1] F. Bannour, S. Souihi, and A. Mellouk, “Distributed sdn control: Survey, taxonomy, and challenges,” *IEEE Communications Surveys and Tutorials*, vol. 20, pp. 333–354, 2018.

[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: A distributed control platform for large-scale production networks,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010.

[3] ONOS. [Online]. Available: https://onosproject.org/

[4] ODL. [Online]. Available: http://opendaylight.org/

[5] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[6] J. Zhong, R. D. Yates, and E. Soljanin, “Minimizing content staleness in dynamo-style replicated storage systems,” *CoRR*, vol. abs/1804.00742, 2018.

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

[8] 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)*, May 2017, pp. 1–7.

[9] M. Aslan and A. Matrawy, “A clustering-based consistency adaptation strategy for distributed SDN controllers,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018.

[10] E. Sakic and W. Kellerer, “Impact of adaptive consistency on distributed sdn applications: An empirical study,” *IEEE Journal on Selected Areas in Communications*, p. 13, 2018.

[11] F. Bannour, S. Souihi, and A. Mellouk, “Adaptive state consistency for distributed onos controllers,” in *2018 IEEE Global Communications Conference(Globecom)*, 2018, pp. 1–7.

[12] 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*, ser. SOSP ’13, 2013, pp. 309–324.

[13] 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, Feb. 2012.

[14] 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*, ser. OSDI’00, Berkeley, CA, USA, 2000.

[15] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Probabilistically bounded staleness for practical partial quorums,” *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 776–787, Apr. 2012.

[16] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Quantifying eventual consistency with PBS,” *Commun. ACM*, vol. 57, no. 8, pp. 93–102, Aug. 2014.

[17] H. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Prez, “Harmony: Towards automated self-adaptive consistency in cloud storage,” in *2012 IEEE International Conference on Cluster Computing*, Sept 2012, pp. 293–301.

[18] S. P. Kumar, “Adaptive Consistency Protocols for Replicated Data in Modern Storage Systems with a High Degree of Elasticity,” Theses, CNAM, Mar. 2016.

[19] C. S. Nguyen Ba, “Adaptive control for availability and consistency in distributed key-values stores,” Theses, University of Illinois, 2015.

[20] M. Couceiro, G. Chandrasekara, M. Bravo, M. Hiltunen, P. Romano, and L. Rodrigues, “Q-opt: Self-tuning quorum system for strongly consistent software defined storage,” in *Proceedings of the 16th Annual Middleware Conference*, ser. Middleware ’15, 2015, pp. 88–99.

[21] “Openai gym project,” https://gym.openai.com/.

[22] H.-A. Tran, S. Souihi, D. A. Tran, and A. Mellouk, “Mabrese: A new server selection method for smart SDN-based CDN architecture,” *IEEE Communications Letters*, vol. 23, pp. 1012–1015, 2019.