Nomad: An Efficient Consensus Approach for Latency-Sensitive Edge-Cloud Applications

Zijiang Hao, Shanhe Yi, Qun Li
{hebo, syi, liqun}@cs.wm.edu
College of William & Mary, Williamsburg, VA, USA

***Abstract*—The rise of edge computing gives birth to a spectrum of delay-sensitive applications. Many of these applications build their services atop the functionality that the edge nodes quickly negotiate a unique order on the events received from a massive number of client devices, even under very high event rates. To this end, we propose a protocol, called Nomad, for achieving fast event ordering in edge computing environments. Nomad is designed as a consensus protocol that employs a lease-based approach to take advantage of the locality of the unbalanced workload across the system. It also dynamically adjusts the leadership distribution on the edge nodes based on the recent running history, and relies on a cloud-based arbitrator to resolve contentions. Experiments demonstrate that Nomad outperforms the existing solutions, such as Multi-Paxos, Mencius and E-Paxos, in achieving fast event ordering for large-scale, delay-sensitive edge-cloud applications.**

I. Introduction

Edge computing, also known as cloudlets [1], fog computing [2] and mobile-edge computing [3], is a new paradigm of distributed computing. The basic idea of edge computing is to provide elastic resources at the edge of network, serving the end devices with lower network latency than cloud computing. Since its birth, edge computing has drawn plenty of attention from the industry, because it is able to fulfill the requirements on network latency imposed by many distributed applications. More notably, edge computing is viewed as the enabler of a spectrum of emerging applications, such as IoT applications, big data analytics, and real-time mobile-edge applications.

Many of such applications are large-scale, geo-distributed ones. There could be dozens of separate edge networks in the system, backed by several interconnected cloud data centers. Massive end devices may simultaneously connect to the system via edge servers located at different places. Information is rapidly exchanged between the end devices and the edge servers, and some of the messages sent to the edge need to be spread across the entire system. In fact, many large-scale edge applications build their services atop the functionality that the system orders the messages received by the edge in a timely manner. However, implementing such a functionality is challenging. This is not only because the system may receive messages at a prohibitively high rate, but there may also be a good number of parties involved in, including both the cloud data centers and the edge servers in the edge networks. How to make a consistent decision in a distributed system is a classic problem in the distributed computing area, which is known as the consensus problem. This problem has been studied for decades and is still an attractive topic in the academia.

Among the existing consensus approaches in the literature, the Paxos-based ones ensure strong consistency, i.e., conflicting states will never occur in the system, and they complete the decision-making process as soon as all the conditions have been met, effectively reducing the user-perceived latency. Because we aim at achieving fast ordering on the messages for geo-distributed edge-cloud applications, and many of such applications cannot tolerate inconsistency on the message order, choosing a proper Paxos-based approach seems to be a plausible solution. However, existing Paxos-based approaches have a severe drawback that they fail to support large-scale distributed systems, because the message complexity grows dramatically with the increase of the number of system nodes. As mentioned previously, the applications we study may run on a great number of distributed parties, so the Paxos-based approaches cannot be directly utilized to solve this problem.

To this end, we propose Nomad, a consensus approach that achieves fast message ordering for geo-distributed edge-cloud applications. The main idea of Nomad is to divide the system into two levels, the cloud level and the edge level, and at each level, Nomad runs a consensus protocol that fits the network traits of that level. The two protocols also cooperate to adapt to and take advantage of the workload change in the system. To summarize, the contributions of this paper are fourfold.

- We formulate a problem of achieving fast message ordering for geo-distributed delay-sensitive edge-cloud applications, and propose two realistic application scenarios to show the significance of studying this problem.
- We design a novel Paxos-based consensus protocol for the edge level, which rapidly orders the messages in individual edge networks. It dynamically distributes the leadership of a sequence of Paxos instances among the edge servers, based on the recent running history, and introduces a cloud-based arbitrator to quickly resolve the contentions on the edge.
- We design a consensus protocol for the cloud level, which works with the edge-level protocol as a whole. It adopts a lease-based method to opportunistically transfer the control from the cloud data centers to the most heavily-loaded edge network, based on the recent running history.
- We implement a prototype of Nomad, and evaluate it on a testbed. The results show the high efficiency of Nomad. In particular, the edge-level protocol outperforms the existing Paxos-based solutions, such as Multi-Paxos [4], Mencius [5] and E-Paxos [6], under different experimental settings.

II. Background

Before elaborating the design of Nomad, we first introduce some preliminaries about edge computing and consensus.

A. Edge Computing

Edge computing has been proposed as an extension of cloud computing [2], [1], [7]. Its goal is to serve end users at the edge of network, providing better network conditions such as low network latency.

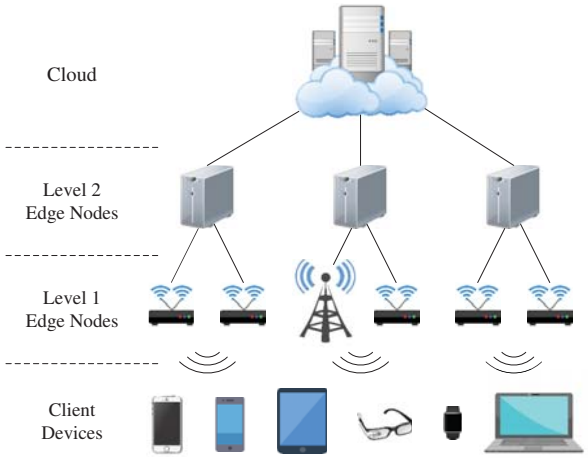


Fig. 1. A hierarchical edge computing architecture.

Figure 1 shows a hierarchical edge computing architecture. Client devices, including wearables, smartphones, tablets and laptops, are wirelessly connected to the level 1 edge nodes. Level 1 edge nodes are usually wireless access points and cellular base stations, which are further connected to the level 2 edge nodes via wired links. There could also be a backend block at the core of network. Note that there may be more than two levels of edge nodes in some architectures.

B. Consensus

Consensus studies the problem of how to achieve overall system reliability in the presence of a number of faulty nodes in a distributed system. It has been studied for decades but remains a hot research topic in the academia [8], [4], [5], [6], [9], [10]. Among the existing consensus solutions in the literature, the Paxos-based ones are widely adopted by the industry [11], because they guarantee strong consistency during execution, and can work efficiently in many settings.

There are essentially two phases in Paxos, i.e., Phase 1 and Phase 2. In Phase 1, the replica that has received a value from the client sends Prepare messages to the other replicas, and the other replicas will send back a Promise message if they haven't accepted any value. After the original replica has received the Promise messages from a quorum, i.e., a majority of the replicas, Phase 1 succeeds, and the replica will start Phase 2 by sending Accept messages to the others. Similarly, the other replicas will send back a Accepted message if they haven't accept any value. After the original replica has received the Accepted messages from a quorum, the value is acknowledged by the system, and the replica will commit it locally as well as informing the other to commit it. Notably, Phase 1 is for electing the leader, while Phase 2 is for making the system accept a value.

Some Paxos-based solutions in the literature, such as Multi-Paxos [4], Mencius [5] and E-Paxos [6], work in the way that they run a sequence of Paxos instances, and therefore they can be used to determine a unique order on the values received across the system. Multi-Paxos runs the sequence of Paxos instances with a fixed leader, skipping Phase 1 for all Paxos instances, which greatly reduces the communication costs for achieving consensus. However, the fixed leader will become the bottleneck of the system. Mencius eliminates this bottleneck by distributing the leadership evenly among the replicas in a round-robin way. Nevertheless, a slow replica in the system may greatly affect the performance of Mencius. E-Paxos determines the dependencies on the Paxos instances, working around this slow-replica problem. However, it imposes more communication costs, which may lead to worse performance than Mencius in many cases.

III. Motivational Scenarios

To better explain the design of Nomad, we first describe the following two application scenarios as examples.

A. Internet of Things (IoT) Payments

The Internet of Things (IoT) has grown rapidly in recent years [12]. The concept of IoT is to manage a massive number of smart devices with a global network infrastructure. Edge computing is usually considered as the best enabler for IoT systems, because IoT devices possesses limited hardware resources, and edge computing can serve them with low network latency [13], [14], [15]. As IoT devices become more and more ubiquitous in our daily life, making digital payments through IoT devices is considered an appealing application [16], [17]. With IoT payments, customers can make purchases anywhere at any time, as long as their client devices have been wirelessly connected to the nearby IoT devices.

Fig. 2. The edge computing infrastructure of the IoT payment application.

Figure 2 illustrates the edge computing infrastructure of the IoT payment application in a global range. Big black circles in the figure are cloud data centers, while small yellow circles

are edge nodes. Ellipses illustrate the "backing" range of the clouds; edge nodes falling into an ellipse are backed by the cloud at the center. IoT devices receive payment requests from customers, and forward them to the nearest edge node. By this means, payment requests are received by the edge computing infrastructure, and a transaction is triggered by each payment request. All payment transactions need to be globally ordered, in order to guarantee the validity of those transactions.

Notably, the payment workload can be quite unbalanced in such a scenario. The reason is that the payments are usually, if not always, made during daytime, or even only during several time periods, such as noon (lunch break) and evening (off work). Therefore, it is likely that in most of the time, only one cloud region receives very heavy workload, while the others receive very low or even no workload.

B. Location-based Massively Multiplayer Online Games

Consider the following scenario. A game company is planning to launch an online augmented reality (AR) game in the US. Players connect their AR devices, such as smartphones and AR glasses, to the gaming service. The game is location-based, meaning that the map in the game is generated from the real world, and any player's character is at the corresponding location of the player's real world location. The game is separated by cities, and most operations by the players are done inside cities they are residing in. Inter-city communications do exist, but are rare.

The game is designed to support a massive number of simultaneously online players in each city. As the number of such players can be very large, the game software is designed to run on the players' client devices, performing heavy computations such as video processing. The computations are deterministic, so the game software always yields the same output given that it starts from the same initial state and is fed the same input.

For each city, all the relevant operations from the players, regardless of inside the city (e.g., hunting monsters) or outside the city (e.g., teleporting to the city), should be ordered and fed into the relevant players' client devices. Only in this way, can the consistency on the game's logic for all the players be guaranteed. For this reason, the game company is supposed to build the gaming service for each city as an operation-ordering service. The service can be built completely on a cloud basis, but in order to provide satisfactory user experience under the potentially very high operation rate, relying on edge computing could be a good choice.

Figure 3 depicts the edge computing infrastructure of such a location-based multiplayer online game in the US. Similarly, big black circles are cloud data centers, and ellipses are the "backing" range of the clouds at the center. Suppose the three clouds shown in the figure cover all the cities in which the operation-ordering service is provided. As an example, one such city, namely the New York City, is enlarged and shown at the top right part of Figure 3. Five edge nodes, represented by small yellow circles in the figure, are deployed in the city. Clearly, the operation-ordering service for any city, such as the New York City, always receives highly unbalanced workload, because intra-city operations are overwhelmingly more than inter-city operations.

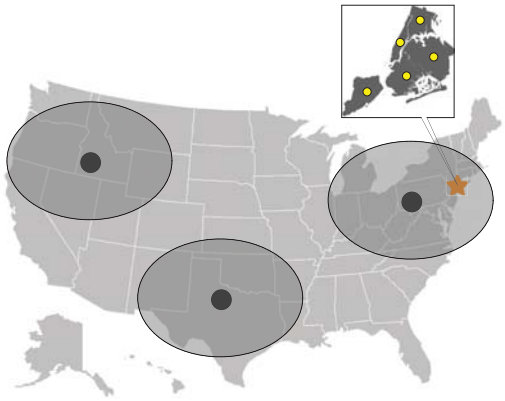


Fig. 3. The edge computing infrastructure for a location-based multiplayer online game in the US. The New York City is shown at the top right part.

IV. The Design of Nomad

In this section, we first formulate the problem using the two application scenarios given in Section III, and summarize the assumptions made by our solution. After that, we describe the design of Nomad in detail.

A. Problem Formulation

For both application scenarios described in Section III, we observe that the edge computing infrastructure, including both the cloud data centers and the edge nodes, is used to order the data received across the system. More specifically, all the data is initially received by the edge nodes, and then spread into the system for ordering. The workload of the applications possesses the following features.

- Every piece of data being ordered is of a small data amount. It is reasonable to consider that the payment transactions and the operations are always less than 1 KB, and in most cases, only tens of bytes.
- The workload can be very heavy. Consider the IoT payment application, for example. It should be designed to support high transaction rates, as it aims to serve many customers in the global range. Similarly, the location-based game should be able to support a large number of simultaneously online players, who generate operations frequently and rapidly.
- The workload can be highly unbalanced. For example, the IoT payment application serves customers at different time zones, but it is likely that most payments are made during several fixed time periods. In the location-based online game case, intra-city operations are far more frequent than inter-city operations.
- The workload may change dramatically from time to time. Notably, the workload of the IoT payment application may be sometimes concentrated in one region and later switches to another, leading to changing workload unevenness. The workload of the location-based online game does not change so dramatically, but the workload distribution on the edge

nodes may still change from time to time, e.g., during the daytime, many players appear in the office area, while after evening, most of they appear in the residential area.

For both applications, it can also be observed that the user-perceived latency is the most significant indicator of achieving satisfactory user experience. For the IoT payment application, the customers usually make payments while they are walking or driving. If the payments cannot be acknowledged in a short time, the benefits of using the application will be undermined. Similarly, for the location-based game, if a player performs an operation, such as attacking a monster, she would expect to see the outcome as quickly as possible. If the outcome is generated after a long time, the gaming experience will be greatly damaged. If this frequently happens, the game will be soon deserted by the players. In fact, edge computing has been proposed as an extension of cloud computing mainly because it provides lower latency. For this reason, the goal of employing edge computing in most cases, from our understanding, is to achieve good user-perceived latency for the application.

To summarize, the problem we study is how to achieve fast event ordering in an edge-cloud computing environment, under the conditions that 1) the events being ordered are all of small data amounts, and 2) the workload can be very heavy, highly unbalanced, and ever-changing at both the cloud level and the edge level. Note that we employ the term "events" to indicate the data being ordered, rather than using any other word such as "transactions" or "operations".

B. Assumptions

To design an effective solution to the problem, we make the following assumptions on the edge-cloud computing system.

- At any time, at most a minority of the cloud data centers may become unreachable, i.e., experiencing failures or network-partitions.
- A cloud data center may back many edge networks that are geographically separated.
- At any time, at most a minority of the edge nodes in the same edge network may become unreachable, i.e., experiencing failures or network-partitions.
- At any time, for any edge network, at least one living cloud can access all the living edge nodes in that edge network.
- The RTT in an edge network may be uneven, i.e., some edge nodes in the edge network may have a longer network delay when communicating with the other.

These assumptions are reasonable from our point of view. The first and the third assumption are similar to those made by other distributed computing solutions [8], [4], [5]. The second one is derived from the application scenarios described in Section III. The fifth one is derived from the fact that the edge nodes in an edge network may be geographically scattered and have different network distances to the others. The forth one, however, needs more inspections. We make this assumption for the failover purpose. It is possible in the real world that a network partition makes all the edge nodes in an edge network unreachable to the outside. However, as failover is important in many cases, service providers, such as those described in Section III, have the motivation to fulfill this assumption. They may employ some engineering methods, such as setting a backup satellite network in the system, to work around the full network partitions of the edge networks.

C. Consensus on the Edge

As discussed above, the workload across the system can be highly unbalanced and sometimes concentrated in a particular edge network. Therefore, it could be beneficial to opportunistically order the events at the heavily-loaded edge networks and only involve the remote clouds when necessary. For this reason, we first suppose that the system only consists of one edge network and one backend cloud, and discuss how to achieve fast event ordering in such a situation. A general-case discussion will also be given in the sequel.

When the system contains only one edge network and one backend cloud, it falls back to the typical form of distributed system. There are several Paxos-based consensus solutions in the literature that can effectively order the data received by a distributed system, such as Multi-Paxos, Mencius and E-Paxos. Their goals, however, are to achieve low latency when the workload is low and high throughput when the workload is heavy. The goal of our solution, in contrast, is achieve as low user-perceived latency as possible in any cases, especially when the workload is heavy.

As such, we design a new Paxos-based consensus protocol for ordering the events on the system containing only one edge network and one backend cloud. Similar to the existing Paxos-based ordering protocols, our protocol also executes a sequence of Paxos instances among the system nodes. More specifically, the Paxos instances are pre-assigned to the edge nodes, i.e., when they are executed, they start from Phase 2 and Phase 1 is considered already finished. As mentioned in Section II-B, Phase 1 is for electing the leader while Phase 2 is for proposing a value. Therefore, the leader nodes of the Paxos instances are artificially determined in advance in our consensus protocol, which reduces the communication cost and thus improves the ordering performance. This does not violate the correctness of consensus, as proved by Mencius. Unlike Mencius, which distributes the leadership of the Paxos instances among the system nodes in a fixed round-robin way, however, our protocol distributes the leadership dynamically according to its running history, for achieving as low ordering latency as possible. The design philosophy of our consensus protocol is summarized as follows.

- To adapt to the workload change on the edge, our protocol dynamically distributes the leadership of the Paxos instances on the edge nodes according to its running history, assigning more leadership to more heavily-loaded edge nodes.
  The intuition of doing so is that the workload has temporal and spatial locality when it is changing, which can be used to predict the workload condition in the near future.
- Any edge node can proactively skip a Paxos instance that belongs to another if no event has been committed in this Paxos instance.

The intuition of doing so is to guarantee low latency in the presence of differences between the predicted workload and the real workload.

- If an edge intends to commit an event but fails for many times because its Paxos instance has been skipped by the others, it delegates the event to the backend cloud.
  The intuition of doing so is to reduce the side effects caused by the skipping scheme.

We call this consensus protocol the "adaptive edge consensus protocol", because it is designed for the edge network and can adapt to the workload change on the edge. The protocol divides the sequence of Paxos instances into epochs, i.e., subsequences of Paxos instances with a fixed length N_{epoch}. At the beginning of each epoch, the protocol examines the past running history and decides how to distribute the leadership among the edge nodes for this epoch. When an edge node receives an event from the client, it tries to commit the event in its next Paxos instance in the sequence. If an edge node has already committed an event e in its Paxos instance ins, but some Paxos instances belonging to the others and ahead of ins in the sequence have no event committed yet, the order of e is still undetermined, and ins is blocked by the Paxos instances. If ins has been blocked by the Paxos instances for a long enough time T_{skip}, the edge node will try to skip those Paxos instances. If an edge node intends to commit an event, but fails for N_{fail} times because its Paxos instances have been skipped by the others, it sends the event to the cloud. The cloud collects all such events and orders them by their arriving time. By the end of each epoch, the cloud sends the sequence of all such events it has collected in this epoch to the edge nodes. When receiving this event sequence, the edge nodes append it to their local event sequence, and close the current epoch. This ending interaction between the cloud and the edge nodes is similar but different to the other Paxos instances in the sequence, and is hence called a "quasi-Paxos instance" in the sequel.

Several details of the protocol should be highlighted. First, the leadership distribution scheme is deterministic, so at the beginning of each epoch, the edge nodes will generate the same leadership distribution without communicating with each other. This reduces the communication cost and improves the protocol performance. Second, unlike the Paxos instances, the quasi-Paxos instances cannot be skipped. This essentially sets an upper bound on the latency for the events under contentions, which is comparable to that of directly using the cloud for event ordering. Third, the edge nodes cannot start the next epoch until the quasi-Paxos instance of the current epoch is closed, meaning that all the edge nodes have to wait for the cloud at the end of each epoch. This is for the failure recovery purpose, which will be discussed later. By carefully choosing N_{epoch}, the quasi-Paxos instances will not block the protocol at all, or at least will not block the protocol for a significantly long time. Last, as the cloud is used to resolve the contentions among the edge nodes, it is generally called the arbitrator of the protocol.

Notably, all actions, including proposing an event, skipping a Paxos instance, and closing an epoch, require the initial party to collect the acceptance from a quorum, i.e., a majority of the edge nodes, before succeeding. Because at any time, at most a minority of the edge nodes may fail, there will always be some living edge nodes that can tell the outcomes of the actions that have ended. Therefore, the correctness of the consensus protocol is guaranteed. Due to the space limit, we omit the algorithms of the actions and the proof of their correctness.

When the current epoch has been closed, there are several ways to determines the leadership distribution for the upcoming epoch. We have designed three schemes for doing so.

- Previous Epoch Only. When using this scheme, the protocol determines the leadership distribution completely based on the running history of the previous epoch that has just been closed. To be more specific, suppose there are N_{eff} effective (i.e., non-skipped) Paxos instances existing in the previous epoch, while $N_{e,i}$ of them belong to Edge Node i. Moreover, suppose the cloud commits N_{del} events at the end of the epoch, and $N_{d,i}$ of them are delegated by Edge Node i. In such a case, Edge Node i will be the leader of $(N_{e,i} + N_{d,i}) * N_{epoch}/(N_{eff} + N_{del})$ Paxos instances. These Paxos instances will be arranged from the beginning of the upcoming epoch, interleaving with those assigned to the other edge nodes as much as possible.
- Previous N Epochs. This scheme determines the leadership distribution by looking at the running history of the previous N epochs, denoted as Epoch N (the latest), Epoch $N-1$, ..., and Epoch 1 (the earliest), respectively, and each Epoch X ($X = 1, ..., N$) is assigned a weight $w_X = X/\sum_{I=1}^{N} I$. The protocol first calculates the committed event ratio for each Epoch X and each Edge Node i, i.e., $ratio_{X,i} = (N_{e,i} + N_{d,i})/(N_{eff} + N_{del})|_X$. Then in the upcoming epoch, Edge Node i will be the leader of $(\sum_{I=1}^{N} w_X * ratio_{X,i}) * N_{epoch}$ Paxos instances. Clearly, the Previous Epoch Only scheme is a special case of the Previous N Epoch scheme, i.e., $N = 1$.
- Previous N Epochs with Random Weights. This scheme is similar to the previous one, expect that the weights assigned to the epochs are calculated in a (pseudo-)randomized way. More specifically, for Epoch X, the protocol picks a number r_X from $\{1, 2, ..., f * N\}$ with the same probability, where $f = 1000$ is the broadening factor, and the weight assigned to Epoch X is $w_X = r_X/\sum_{I=1}^{N} r_I$. Note that this calculation is actually pseudo-randomized and hence deterministic, not violating the rules of distributing the leadership.

The first scheme is the simplest one and can rapidly adapt to the change on the workload. The drawback of this scheme is that if an edge node has experienced some bad conditions, such as network jitters, but later recovers, it may take a long time for the edge node to regain its leadership share in the epochs. The second scheme involves more previous epochs to deal with this situation, but it may suffer when the workload changes rapidly. The third scheme adopts a randomized approach, and is thus more resilient to the transient changes on the workload and on the network conditions than the second one, but suffers from the same problem as the second scheme.

Figure 4 illustrates how the adaptive edge consensus pro-

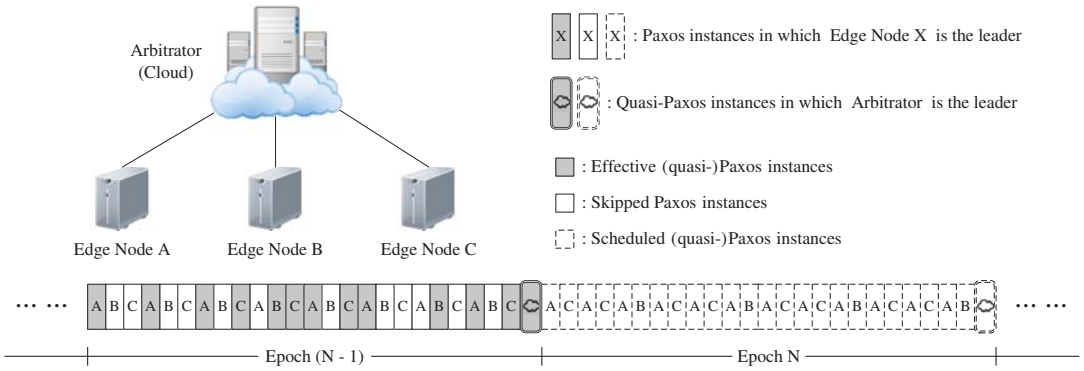


Fig. 4. The adaptive edge consensus protocol. The leadership distribution scheme shown in the figure is the Previous Epoch Only scheme.

tocol works. Three edge nodes, i.e., Edge Node A, B, and C, compose the edge network, and a cloud at the backend acts as the arbitrator. Two epochs, Epoch (N – 1) and Epoch N, are shown in the figure. Suppose the cloud does not commit any event but merely closes Epoch (N – 1) at the end of it. Because the protocol is using the Previous Epoch Only scheme to distribute the leadership, in Epoch N, Edge Node A is assigned $6 * 24/12 = 12$ Paxos instances, Edge Node B is assigned $2 * 24/12 = 4$ Paxos instances, and Edge Node C is assigned $4 * 24/12 = 8$ Paxos instances. Note that after this calculation, the Paxos instances are scattered as evenly as possible throughout Epoch N.

D. Working with the Clouds

As mentioned in Section IV-C, the adaptive edge consensus protocol is designed for systems with only one edge network and one backend cloud. When the system contains multiple edge networks and multiple backend clouds, such as those described in Section III, the adaptive edge consensus protocol cannot work effectively without combining with another cloud-level protocol. For this reason, we have designed a cloud-level protocol to work around this problem.

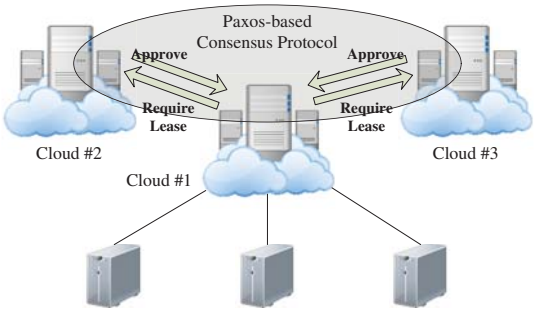
We design the cloud-level protocol as follows. The system initially works in the equality mode, i.e., all the clouds work together using a Paxos-based consensus protocol, ordering all the events across the system. Any Paxos-based consensus protocol can be used here, but the leadership-sharing ones are preferable, such as Mencius. The edge nodes are only used to forward the events they have received to their backend cloud. Because all the cloud can see all the events that have been ordered, they can learn the workload condition of the system. If a cloud notices that in the past N_{ob} events that have been ordered, more than $ratio_{thresh} * N_{ob}$ events come from the same edge network it backs, the cloud will try to make the system work in the master-slave mode. It does so by asking for a lease from the other clouds using the same consensus protocol for ordering the events. After the cloud successfully receives the lease approvals from a quorum, it informs the edge network, and the edge network will work with the cloud using the adaptive edge consensus protocol. Now the edge network and the cloud become the master, and all the other parties become the slaves, i.e., the system is working in the master-slave mode. The protocol is thus called the lease-based master-slave protocol in our solution.

Notably, when the system is working in the master-slave mode, all slaves, including both the other clouds and the other edge networks backed by the master cloud, will forward the events they have received to the master cloud. The master cloud will order all such events locally, together with those received from the master edge network for contention resolving, based on the event arriving time. It will then commit all the locally-ordered events in the upcoming quasi-Paxos instance, thus determining the global order of them. After that, the order of all the events will be broadcast by the master cloud to the slave clouds.

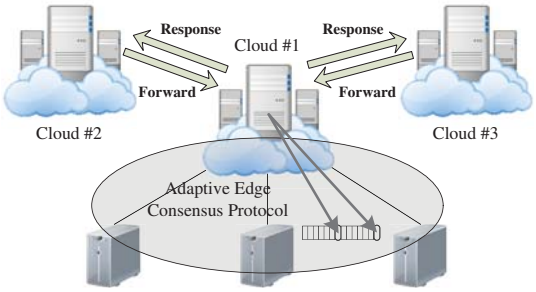
Figure 5 shows how the leased-based master-slave protocol works. It is worth mentioning that the lease can only be used by the master for N_{lease} epochs. After the N_{lease} epochs, the other parities consider that the lease is expired, and the system will return to the equality mode unless the master cloud has successfully received an extension for the lease from the other clouds. The extension will be another N_{lease} epochs, and the master can extend the lease for potentially many times. When receiving a lease request or a lease-extension request, a cloud will check if it has more than N_{out} events that are not ordered yet. If the answer is yes, it will reject the request. Otherwise, it will accept the request. Clearly, the lease-based master-slave protocol can adapt to and take advantage of the highly unbalanced workload, achieving low user-perceived latency in the scenarios similar to those introduced in Section III. The solution proposed by this paper, i.e., Nomad, is essentially the combination of the adaptive edge consensus protocol and the lease-based master-slave protocol.

E. Dealing with the Failures

As summarized in Section IV-B, at any time, a minority of the clouds may fail or become network-partitioned simultaneously. Similarly, at any time, a minority of the edge nodes in an edge network may fail or become network-partitioned. It is essential to guarantee the correctness of Nomad, i.e., when



(a) Cloud #1 observes that most workload of the system has been received by an edge network it backs. It then asks for a lease to handle the workload at that edge network, through the Paxos-based consensus protocol.



(b) The edge network receives the lease, and works with Cloud #1 using the adaptive edge consensus protocol. Cloud #2 and Cloud #3 forward the events they have received to Cloud #1, which in turn commits them into the quasi-Paxos instances.

Fig. 5. The lease-based master-slave protocol.

an event has been committed, any party in the system will not observe a contradicting state of the event at any time in the future. Otherwise, inconsistency occurs in the system, violating the design rules of Nomad.

When the system is working in the equality mode, all the clouds cooperate with each other using an existing Paxos-based consensus protocol in the literature. In such a case, the correctness of Nomad can be guaranteed by the Paxos-based consensus protocol.

When the system is working in the master-slave mode, however, two failure cases need to be taken into consideration. First, when some master edge nodes fail or become network-partitioned, the correctness of Nomad can be guaranteed. As mentioned in Section IV-C, all actions, including proposing an event, skipping a Paxos instance and closing an epoch, require the initial party to collect the acceptance from a majority of the master edge nodes before succeeding. Therefore, at any time, at least one living master edge node can tell the outcomes of the actions that have ended, so no contradicting state will occur. Second, when the master cloud has failed, the correctness of Nomad is still guaranteed, because the master cloud can only commit events with the acceptance of a majority of the master edge nodes.

Notably, a severe consequence resulted from the failure of the master cloud is that the process of the Nomad protocol will be completely blocked, because the master edge nodes have to synchronize with the master cloud at the end of each epoch. In contrast, other kinds of failures will not block the process of Nomad. To work around this problem, Nomad treats the slave clouds as the backups of the master cloud. When a slave node suspects that the master cloud has failed, it will directly contact the master edge nodes, asking them to accept it as the new arbitrator. After the slave node has collected the acceptance from a majority of the master edge nodes, it becomes the new arbitrator, and works with the master edge nodes until the lease expires. The correctness of this process is guaranteed by requiring the acceptance from a majority of the master edge nodes.

It should be mentioned that the aforementioned method for arbitrator failover is possible because at least one living cloud can access all the living master edge nodes at any time, as assumed in Section IV-B. However, a network partition in the real world may make all the master edge nodes unreachable to the outside, and if no ad-hoc solution is employed for handling this problem, such as setting a satellite network for backup, arbitrator failover cannot be accomplished. In such a case, the protocol cannot make any progress until the arbitrator recovers. Nevertheless, the correctness of consensus will not be violated, and the protocol will continue working as soon as the arbitrator has recovered.

V. Evaluation

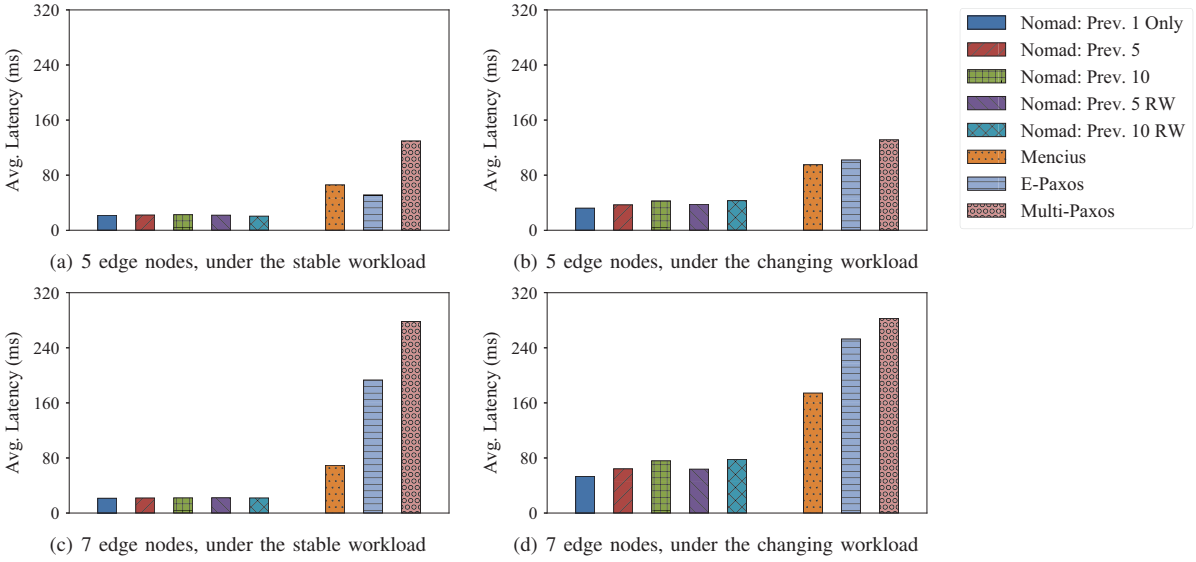
To evaluate Nomad, we have implemented a prototype, and deployed it on a testbed. Experiments on the prototype shows the performance of Nomad under different situations.

A. Test Setup

We first build an edge-cloud testbed. Three PC servers are used as three clouds, and several laptops are used as the edge nodes that form an edge network. The edge network is backed by one of the clouds. The RTT between the edge nodes is set to 10 ms, expect a slow one, which has a 40 ms RTT to the others. The RTT between the edge nodes and their backend cloud is 60 ms. As the workload for testing is simulated, the RTT between the client and the edge is assumed to be 10 ms. The RTT between the clouds is set to 100 ms. The bandwidth between any two parties is set to 100 Mbps, and the message size is set to 1 KB across the system.

B. Performance of the Adaptive Edge Consensus Protocol

After building the testbed, we implement a prototype of the adaptive edge consensus protocol, with the three leadership distribution schemes described in Section IV-C, and deploy it on the testbed. N_{epoch} is set to 100. Two types of workloads are simulated. The first one is a stable one; every edge node stably receives 1,000 events per second. The second one is a changing one. Every edge node stably receives 500 events per second, and another workload, which is the sum of all these individual workloads, moves from one edge node to another in a round robin manner, with a rate of 500 events per second. For example, suppose there are five edge nodes in the system,



(a) 5 edge nodes, under the stable workload

(b) 5 edge nodes, under the changing workload

(c) 7 edge nodes, under the stable workload

(d) 7 edge nodes, under the changing workload

Fig. 6. The average user-perceived latency of different consensus protocols under different settings.

denoted as Edge Node $1, 2, ..., 5$. In the first second, Edge Node 1 receives $500 + 500 * 5 = 3000$ events, and each of the other edge nodes receives 500 events. In the second second, Edge Node 1 receives $3000 - 500 = 2500$ events, Edge Node 2 receives $500 + 500 = 1000$ events, and each of the other edge nodes receives 500 events. In the sixth second, Edge Node 1 receives 500 events, Edge Node 2 receives 3000 events, and each of the other edge nodes receives 500 events. Then in the seventh second, the workload starts to move from Edge Node 2 to Edge Node 3, and so on. Using this workload, we simulate the typical situation of the IoT payment application.

We feed the two workload to our prototype. Five leadership distribution schemes are tested, i.e., Previous Epoch Only, Previous 5 Epochs, Previous 10 Epochs, Previous 5 Epochs with Random Weights, and Previous 10 Epoch with Random Weights. For comparison purposes, we also implement Multi-Paxos, Mencius and E-Paxos, and feed the workloads to them. Note that we assume the edge network can utilize non-FIFO network links, so for Mencius, piggybacking messages is not allowed. With these settings, two groups of experiments are conducted. In the first group, the edge network contains 5 edge nodes. In the second group, it contains 7 edge nodes.

Figure 6 depicts the results of those experiments. Clearly, the Nomad protocol outperforms the other three protocols in all settings, especially when using the Previous Epoch Only scheme. When using other leadership distribution schemes, the average user perceived-latency is slightly larger than that of using the Previous Epoch Only scheme. As the Previous N Epoch and Previous N Epoch with Random Weights schemes are designed for fast leadership recovery, this means that they work with acceptable overhead.

C. Performance of the Leadership Distribution Schemes

To determine the effectiveness of the three leadership distribution schemes, i.e., the Previous Epoch Only scheme, the Previous N Epochs scheme, and the Previous N Epochs with Random Weights scheme, we conduct the following experiment on the prototype. The edge network is configured to have five edge nodes. The workload is that in the first 50 epochs, each edge node stably receives 1,000 events per second. Then in the following 10 epochs, one edge node other than the slow one receives no event at all, while the workload on the others keeps unchanged. After that, the workload on the unloaded edge node returns to 1,000 events per second. This simulates the situation that an edge node experiences a transient problem but soon recovers. Figure 7 shows the changes on the leadership share of the temporally unloaded edge node caused by the changing workload.

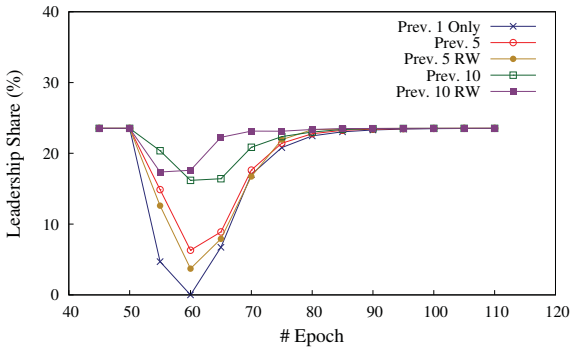


Fig. 7. The leadership share of a temporarily unloaded edge node.

Clearly, when the Previous Epoch Only scheme is being utilized, the leadership share changes sharply with the workload change. On the other hand, when utilizing the other two schemes, the leadership share changes in a moderate manner. Moreover, the Previous 10 Epochs schemes produce the most stable outcomes, while the results of the Previous 5 Epoch schemes are in-between those of the Previous Epoch Only

scheme and those of the Previous 10 Epoch schemes. This suggests that when the protocol takes more previous epochs into consideration, it can resist the transient bad conditions to a larger extent. On the other hand, this also suggests that the protocol will adapt to the workload change more rapidly when considering only one previous epoch.

D. Performance of the Leased-based Master-Slave Protocol

We also conduct an experiment to examine how the lease-based master-slave protocol works. Three PC servers are used as three clouds, denoted as Cloud #1, #2 and #3. Cloud #1 is set to be the backend cloud of the edge network. The edge network is configured to have five edge nodes. The workload is that in the first 5 seconds, every cloud receives 500 events per second. For Cloud #1, the workload is completely from the backed edge network, i.e., each edge node in the edge network receives 100 events per second. For Cloud #2 and #3, we merely simulate the condition that the events are received by a virtual edge network backed by the cloud. Then in the following 5 seconds, the five edge nodes backed by Cloud #1 receives 1,000 events per second, so Cloud #1 receives 5,000 events per second, while the workload on the other two clouds keeps unchanged. After that, the workload on Cloud #1 returns to 500 events per second. This simulates a situation similar to that of the IoT payment application. Furthermore, $ratio_{thresh}$ is set to 0.5, N_{ob} is set to 2,000, and N_{lease} is set to 10. Mencius is implemented as the consensus protocol that connects the three clouds. Figure 8 shows the changes on the average user-perceived latency in this process.

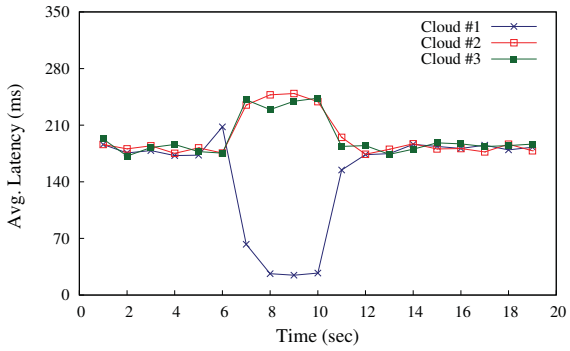


Fig. 8. The average user-perceived latency of the three clouds.

From Figure 8, it can be figured out that the lease-based master-slave protocol can adapt to the change on the cloud-level workload very quickly, in about only one second for both entering and exiting the high-load phase. This proves that the lease-based master-slave protocol can work quite efficiently. Note that the lease granted to Cloud #1 is extended for several times during the high-load phase before being revoked by the system. Choosing a smaller $ratio_{thresh}$ and a smaller N_{ob} can help the protocol adapt to the workload change more quickly, but may introduce undesirable overhead if transient changes on the cloud-level workload frequently occur in the system.

VI. Conclusion

In this paper, we present Nomad, a consensus protocol achieving fast event ordering for large-scale edge-cloud applications. Nomad consists of an edge-level adaptive consensus protocol and a cloud-level master-slave protocol, which can work together to efficiently order the events received across the system. We have implemented a prototype of Nomad and deployed it on a real-world testbed. Evaluation on the prototype reveals that Nomad outperforms the existing consensus solutions in edge computing environments.

Acknowledgment

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grant CNS-1816399.

References

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

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012.

[3] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-Edge Computing (MEC) Industry Initiative*, 2014.

[4] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[5] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for wans," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '08, 2008.

[6] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13, 2013.

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[8] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[9] H. Cui, R. Gu, C. Liu, T. Chen, and J. Yang, "Paxos made transparent," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15, 2015.

[10] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports, "Building consistent transactions with inconsistent replication," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15, 2015.

[11] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06, 2006.

[12] C.-L. Hsu and J. C.-C. Lin, "An empirical examination of consumer adoption of internet of things services," *Comput. Hum. Behav.*, vol. 62, no. C, pp. 516–527, 2016.

[13] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. MoBiData '15, 2015, pp. 37–42.

[14] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *Proceedings of the 10th International Conference on Wireless Algorithms, Systems, and Applications*, ser. WASA '15, 2015, pp. 685–695.

[15] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proceedings of 2015 the Third IEEE Workshop on Hot Topics in Web Systems and Technologies*, ser. HotWeb '15, 2015, pp. 73–78.

[16] Secure Technology Alliance, "Iot and payments: Current market landscape," https://www.securetechalliance.org/wp-content/uploads/IoT-Payments-WP-Final-Nov-2017.pdf, 2017.

[17] Visa, "Visa brings secure payment solutions to the internet of things," https://usa.visa.com/visa-everywhere/innovation/visa-brings-secure-payments-to-internet-of-things.html, 2018.