Support of Strong Consistency on Fog Applications

Diogo Lima[1,2]
[1] Escola Superior de Hotelaria e Turismo do Estoril
Lisbon, Portugal
dlima@lasige.di.fc.ul.pt

Hugo Miranda[2]
[2]LASIGE, Faculdade de Ciências, Universidade de Lisboa
Lisbon, Portugal
hamiranda@ciencias.ulisboa.pt

François Taïani[3]
[3]Univ Rennes, CNRS, Inria, IRISA - UMR 6074
Rennes, France
francois.taiani@irisa.fr

Abstract

Providing scalability for mobile distributed applications often leads the applications to compromise their consistency requirements. However, some of these applications depend on strong consistency to guarantee that one consistent state is observed by all participants over (some of) the shared objects of the application. While deploying the application in the Cloud facilitates state management and trivially solves the strong consistency requirements, this also creates a geographical barrier between mobile devices and the application servers that penalises performance with an unavoidable latency and jitter. Fog Computing architectures can mitigate this impact by deploying surrogate servers at the network edge but its performance depends of a middleware service able to monitor the application and deploy each fragment of the state at the most convenient location. Such alternative leads to the emergence of distributed transactions that require increasing efforts to maintain strong consistency. This paper addresses how virtual synchrony, in particular a technique named Light-Weight Group (LWG), can help supporting strong consistency for mobile applications in the Fog Computing environment.

Keywords Mobile Distributed Applications, Fog Computing, Strong Consistency, Light-Weight Groups

ACM Reference Format:
Diogo Lima[1,2], Hugo Miranda[2], and François Taïani[3]. 2019. Support of Strong Consistency on Fog Applications. In *Proceedings of 6th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The evolution of wireless technologies has lead to a significant change in mobile applications. Mobile applications have evolved from standalone applications into large scale distributed applications that rely on the network to mediate the interactions among a large number of users, managing shared objects (the application state) that each user can read and write. Examples of such applications include collaborative applications for smart cities and traffic management [5], vehicle sharing applications (e.g. DriveNow[1]), multi-user games (e.g. Ingress, Pokemon Go) and social networks (e.g. Foursquare, Twitter). A challenge for these applications is to guarantee that all participants in the system observe the same state over (some of) the shared objects of the applications. The current trend is to maintain all application state and mediate all interactions among users in a centralized infrastructure, hosted in Cloud datacenters. However, it creates a geographical barrier between the end users and the application state. In particular, the limited bandwidth between user devices and the Cloud infrastructure, as well as the latency and jitter that unavoidably results from this distance will negatively impact application performance. Accurately assessing the impact of such long distance access between mobile devices and the Cloud may be difficult as it depends of various conditions and may even vary from region to region. As an alternative, we provide an approximation using the Amazon EC2 instances located in different datacenters around the globe to represent the importance of such long distance accesses based on wired and more stable connections. In Table 1 we present the round trip times between the different EC2 instances, where we can observe that the measurements can have an order of magnitude as high as the hundreds of milliseconds. Projecting these results to the communication between mobile users and the Cloud, these values are only expected to be worse, exacerbating the negative impact between the geographical distance between the end users and the application state.

Round trip time (in ms)	N. America (Virginia)		
Asia (Mumbai)	182 ms	Asia (Mumbai)	
Europe (Frankfurt)	89 ms	111 ms	Europe (Frankfurt)
S. America São Paulo	140 ms	321 ms	232 ms

Table 1. Round trip time (in ms) between Amazon EC2 datacenters located in different continents.

Fog Computing [2] proposes the deployment of *surrogate servers* at the edge of the network, at common places such as bus terminals, shopping malls, public services, etc. Relying

[1]https://www.drive-now.com

PaPoC'19, March 2019, Dresden, Germany
2019. ACM ISBN 978-1-4503-6276-4…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

on the existence of such surrogates servers, Fog Computing's objective is to bring the application state, previously located in the Cloud, in proximity to the end users. This approach has the potential to mitigate jitter, end-to-end communication latency, as well as traffic load in the backbone networks [16] to support the computational demand of latency-sensitive largely geographically-distributed applications.

However, its performance strongly depends on the ability of a middleware service to deploy each of the application state components at their most convenient locations, a problem hereafter referred as *geo-aware state deployment service*. On the other hand, while deploying application state closer to the end users, this also leads to the emergence of distributed transactions that require application state that might be stored in different locations. As a consequence, increasing efforts are needed to ensure that consistency is maintained for the applications.

Based on the example of a well known multi-user mobile game, this paper focuses on how the virtual synchrony [1] technology can provide the support needed to guarantee strong consistency for such distributed edge applications. In particular, known for its scalabilty problems, we explore how virtual synchrony can be optimized by benefiting from the existence of partial groups defined at the application level by the geo-aware state deployment service to create multiple fine grained virtual synchronous groups, in a technique named Light-Weight Group (LWG) [6].

2 Problem Statement

We consider a wide scale multi-user and augmented-reality mobile game such as Ingress[2]. The game consists in having two teams disputing *portals*, located in public landmarks. The most common game action (hereafter all actions will be referred as *transactions*) consists in a user interacting with a portal located in proximity to either attack the portal if it belongs to the other team or reinforce its defenses, possibly earning items as a reward. One important item that a player can obtain is the *portal key*. This key allows interactions with the portal even when the player is located outside the portal's range. This is an important aspect of the game as it allows the players to link two portals where one is located within the player's range and a portal key is already owned for another remote portal. When three portals are linked together, an *influence area* is created, and that is when a team is rewarded with points.

To benefit from the Fog Computing paradigm, in our system model, applications such as Ingress are supported by servers deployed at the edge of the network, hereafter named *surrogates*. These surrogates are responsible for managing the application state of a set of portals located in proximity and serve as application end points to the mobile applications of the players. This deployment mitigates the latency and jitter that result from the distance between end users and the application state hosted on a centralized infrastructure and alleviates traffic in the Internet backbone. The surrogates are also connected to the other surrogates and to cloud datacenter(s) by a high speed wired network. Still, expectations are that, to avoid latency and increase Quality of Experience (QoE), most of the traffic produced by clients is handled at surrogates.

We assume that the application state is composed of data that can be either unique to each user (personal data), collaborative data relevant to a specific geographical location (geo-aware data), and general application logic data (global data). In Ingress, personal data correspond to the users' items: experience level, team affiliation, etc. Geo-aware data correspond to the information related to each portal in an area that is managed by a surrogate. Information such as the team owning the portal and its current defense status. Finally we consider as global data all information related to each team such as total points, number of days in the lead, etc., and the chat communication provided by the game to all players and team only members. Personal and geo-aware data are stored in a specific surrogate, while global data is stored in the Cloud.

Depending on the number of sites involved, transactions can be either *local*, *regional* or *global*. Local transactions are those that access data stored at a single location. These improve system performance by reducing transaction latency. Regional transactions are those that require the participation of at least two surrogates, but not the whole system. This occurs, for example, when a player wants to link a local portal with a remote one.

Ideally, being a location-based mobile game, it is expected that most of the transactions will still be local. However, since player mobility must also be considered during game play, the system should continuously attempt to maximize the amount of local transactions by relying on a geographically-aware state deployment service acting as an oracle running in the background (hereafter named *GeoDS*). The GeoDS becomes eventually aware of the source of the accesses to state items and decides the most suitable location of each state item and coordinates its transfer between locations.

Figure 1 graphically depicts the implementation of our model. GeoDS is managed by a service that conceptually acts as an oracle (1), becoming eventually aware of the source of the accesses to state items. The oracle is the component with full knowledge of the system dataset and deployment. Periodically, the oracle re-evaluates the current state deployment, decides the most suitable location of each state item and coordinates its transfer between surrogates. To perform its tasks, the oracle collects transaction logs, delivered in the background by the surrogates (2). State deployment evaluation cycles are triggered after some predefined number of transaction logs is received, which can be tuned to fit

[2]https://www.ingress.com/game/

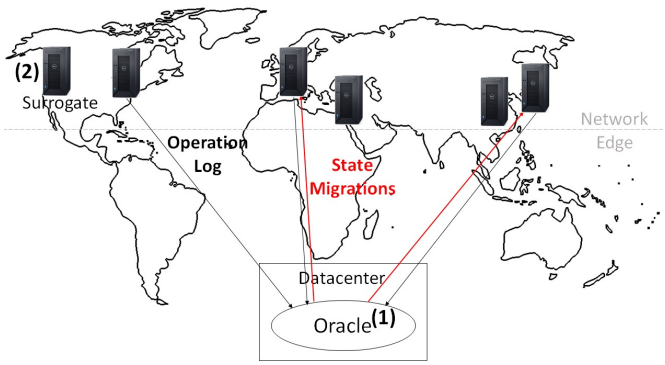

Figure 1. System model.

operational constraints and system scalability. Detailing geo-aware date placement strategies is out of the scope of this paper, but additional information can be found in Sec. 3. However, notice that this GeoDS does not represent a point of contention for the system. As it runs in the background, even if it crashes or becomes overloaded, the only consequence is that no state item migrations will be performed. More distributed transactions that could have been avoided may occur from the fact the state deployment is not being re-evaluated, but the surrogates will still be able to respond to client's requests independently from the GeoDS.

Another key aspect of Ingress is its need for strong consistency. Consider the most common transaction in gameplay: interacting with a portal; and that these portals are usually located in crowded public landmarks, it is likely that multiple players may be interacting with a portal (either to defend it or attack it) concurrently. In that case, it is fundamental that the results of the players' actions are seen by all participants in a consistent state. Otherwise, game actions performed by concurrent players could result in state inconsistencies where one player might observe the portal as belonging to one team, while another concurrent player would observe the same portal as belonging to the other team. This would inevitably violate the application logic, resulting in triggering additional mechanisms such as rollbacks to bring the state back to a consistent state and would penalise the application's QoE. This challenge is trivially solved when concentrating all application state in the Cloud. However, by leveraging the Fog Computing architecture, additional mechanisms are needed to guarantee that strong consistency is equally maintained in our system model.

3 Related Work

At a first glance, some similarities can be found between our Fog Computing system model with the GeoDS and caching, specially with Content Delivery Networks (CDNs) such as Akamai.[3] However, significant differences exist between both worlds. While caching is focused on deciding which data must be stored locally to avoid slower data accesses, its main focus is to design eviction policies to cope with limited memory. Extended to global wide distributed systems, CDNs add challenges such as bandwidth and possible limitations between the replicas and the original location to decide which data to cache and where. However, even if the goal is to deploy data closer to the clients [10, 12], CDNs and caching in general only consider read-only replicas of data. Any write access must be performed in the replica's primary replica and cached copies must be invalidated. In opposition, our problem aims at partial replication with state consistency, identifying the ideal location for each single copy of the data and knowing that data can be edited and new content created by clients. Thus the goal of the GeoDS is to minimize latency in data access, regardless of the type of operation performed over the data (read or write). Additionally, while in CDNs each server can take its caching decisions individually, in GeoDS the system needs to be considered as a whole to improve its overall performance, and thus the deployment decision needs to be centralized since a full knowledge over the system's data access patterns is needed.

The problem addressed in this paper is essentially related to geo-replicated cloud storages, which may pursue different objectives and thus different consistency criteria. The GeoDS is leveraged by previous research results on data partitioning.

3.1 GeoDS

Data partitioning was originally proposed to address scalability and performance requirements of database management systems. Horizontal and vertical partitioning were some of the earliest strategies to emerge [7].

Graph partitioning algorithms are a particular approach to data partitioning as they consider the implicit association created between state items by transactions. The objective of the graph-based approach reported in [4] is to partition database items across storage nodes so that most transactions only need to access one node thus centralizing locking and consistency, while improving load balancing. Based on this approach, [14, 15] proposed a set of graph-based GeoDS approaches that leverage on data correlation to improve performance of Fog Computing based systems by trying to maximize local transactions. Theses approaches assume that partition imbalances are expected to reduce latency, as long as they represent the effective utilization patterns of the data.

3.2 Eventually Consistent Cloud Storages

The main goals of large scale geo-replicated cloud storages are to provide availability and fault-tolerance via data replication. Ideally, geo-replicated cloud storages should also provide strong consistency so that all users see the shared objects in a consistent state. However, the latter consistency criterion is too expensive in terms of latency to be used on commercial large scale geo-replicated cloud storages. In

[3] https://www.akamai.com/

opposition, modern systems such as MongoDB, CouchDB, Amazon Elastic Compute Cloud (EC2) choose to weaken the consistency criterion and opt for eventual consistency [19]. In eventual consistency, the shared objects eventually converge to the same state once replicas stop updating. In the meantime, clients can observe different states for the same shared object. However, applications such as Ingress depend on strong consistency and cannot rely on such solutions without penalizing user's QoE.

3.3 Strong Consistent Solutions

State machine replication (SMR) [17] is a common model to implement strong consistent services. In SMR, replicas agree on the order in which they execute commands, relying on a consensus algorithm such as Paxos [13], and every replica reaches the same state. However, it suffers from inherent scalability problems. Since every replica executes every command, adding servers does not increase system throughput. Scalable state machine replication tries to mitigate this problem by partitioning application state so that commands are executed in only a subset of replicas. However, this solution is limited if many multi-partition commands are executed, leading the partitions to frequently communicate. DS-SMR [9] tries to solve the scalability problems of state machine replication by re-partitioning the state dynamically, based on the workload. State reconfiguration follows a simple approach that transfers all required data items to a single partition prior to initiating the transaction. However, this solution introduces a non-negligible latency with state item migrations during transaction execution since it follows a reactive approach. On the other hand, we address state reconfiguration in a proactive manner and exploit the existence of partial groups identified by a GeoDS at the application level to create multiple fine grained virtual synchronous groups to guarantee consistency.

The ZooFence [8] is another service that favors strong consistency. It uses the Apache ZooKeeper coordination service with a partitioning algorithm that builds dependable and consistent partitioned services. The algorithm is based on a leader election and an atomic queue for each partition and, once the service is partitioned, clients can either (i) issue a command directly to a single ZooKeeper instance if that command can be treated by a single instance or (ii) insert a command on the appropriate partition queue. In the latter case, an *executor* component is responsible to forward the commands to the associated ZooKeeper instances, merge their results and reply to the client. However, in opposition to our work, ZooFence considers partitions to be immobile: once the partitions are defined, state cannot migrate among partitions. Instead, we assume that state is mobile and should migrate among server instances during the system's life.

4 The Case for Strong Consistency in Large Scale

Regardless of the GeoDS strategy followed, it is still unrealistic to achieve a perfect usage scenario and subsequent state deployment where all state items are used in a single location for a large scale distributed application. As a result, regional transactions still need to be taken in consideration as an important aspect of the system. Firstly because, as being distributed, they introduce an inevitable negative impact on system performance. Secondly, the system must still maintain the application's strong consistency guarantees so that users' actions are seen by all participants in a consistent state when executing such transactions. Otherwise, this might violate application logic and result in triggering additional mechanisms such as rollbacks to bring the state back to a consistent state, penalizing in further application's QoE.

To achieve such goal, we rely on virtual synchrony [1]. Virtual synchrony organizes processes into groups. A process can communicate with all members of a group, can join or leave a group, and can also participate in various groups. A membership service is responsible for maintaining the list of the correct processes of a group, called the *view* [3]. If a process wants to join, leave or has been detected to have crashed within a group, virtual synchrony starts a view change. It relies on reliable broadcast to ensure that a message is either delivered to all processes in that group view or to none. This all-or-nothing property is central to keeping all members of the group synchronized and, when combined with total order ordering, is fundamental to maintain a consistent state of the group. Within a view, messaging delivery can follow other orderings such as an unordered reliable multicast, FIFO, or causal ordering.

However, this model presents some scalability problems. Firstly because it is difficult to scale up the message delivery ordering and the group membership management. Secondly, because message delivery latency increases with the size of the group. To mitigate such problems some solutions have been proposed such as *Gossip* multicast [11] that loosens message delivery guarantees and optimistic virtual synchrony [18] which tries to improve the blocking penalty experienced by applications when a view change is performed by estimating the set of members of the next view and optimistically send messages to be provisionally delivered in the next view.

On the other hand, we intend to provide the strong consistency guarantees, relying on Virtual Synchrony, for mobile distributed applications while maintaining their scalability requirements by using the Light-Weight Group (LWG) [6] abstraction. LWGs consist in creating multiple fine grained virtual synchronous groups that are mapped to a single core group in a dynamic and transparent manner. This allows the

LWGs to share common resources, such as the failure detection protocol, in order to amortize the overhead of maintaining the membership information of a large number of groups. More importantly, LWGs provide the scalability needed for large scale mobile distributed applications as the member affiliation of each LWG is much smaller than a coarser virtual synchronous group. That is, the number of members participating in a single LWG is expected to be a small subset of all processes present in the system.

In the context of the Fog Computing environment and considering the example of Ingress, we intend to explore this abstraction based on the data usage patterns collected from the application. As mentioned earlier, application state is composed of personal, geo-aware and global data. While the GeoDS will try to deploy local data at the most convenient location to maximize local transactions, the LWG technique will be used to address regional transactions that manipulate both local and geo-aware data at the same time. For example, regional transactions arise when players are interacting with a portal in a specific area but their state is stored in a different surrogate since it is where they most frequently play the game. In that case, the portal's surrogate and the surrogate storing the player's personal data need to communicate in order to guarantee that player actions are valid. Moreover, since the player is currently located in that specific area, she will be only able to interact with the portals co-located in the same area. Hence the surrogate responsible for that portal area and the surrogate storing the information of that player can create a fine grained light-weight group with both as the only members to validate their respective transactions. On the other hand, as the player may move and start interacting with portals in an area belonging to another surrogate, the former surrogates can leave the LWG previously created and the surrogate storing the state of the moving player can form a new LWG with the surrogate storing the data of the portals that player is now manipulating with. This is done transparently as the core synchronous group still in the same state.

Moreover, in order to cope with scalability and the fact that maintaining these various LWGs bring a non negligible cost to the system, we intend to create LWGs considering two aspects. Firstly, in terms of group granularity, LWGs must be created between pairs of surrogates. This represents the smallest member affiliation set possible for each LWG, guaranteeing that the cost of group membership information is the lowest possible and thus scalable. On the other hand, not all possible combinations of surrogates pairs will result in the a creation a LWG. Instead of having N^2 LWGs, where N is the number of surrogates present in the system, LWGs will only be created if considered *relevant* to the system. I.e. when the analysis of the application state usage identifies that recurrent regional transactions are being requested between a pair of surrogates that justify the creation o LWG to forward and validate such transactions. Figure 2 graphically depicts

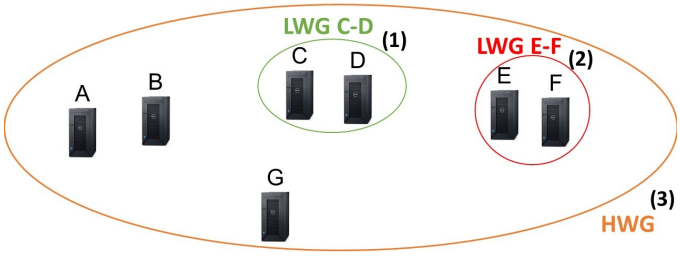

Figure 2. A LWG-based system to support distributed applications in Fog Computing.

our system model based on this LWG abstraction. LWGs (1)-(2) are created between surrogates that recurrently share application state in regional transactions. In this example of a system composed with 7 surrogates, only two LWGs exist: one between surrogates C and D, as well as one between surrogates E and F. On the other hand, as the application state usage does not identify relevant regional transactions for surrogates A, B and G, they do not participate in any LWG.

Upon receiving a new transaction request from a client, it is up to the receiving surrogate to identify the location of the state items involved in the transaction and identify whether it is local, regional or global. Local transaction are trivially executed and validated solely by that surrogate. In case of a regional transaction, the surrogate receiving the transaction request is responsible for forwarding the latter to the respective LWG suited to execute that transaction. Finally, if a transaction involves data stored in 3+ surrogates we rely on the underlying virtually synchronous support group, i.e. the Heavy-Weight Group (HWG) (3) that have all the surrogates in the system as participants to validate that transaction. As these kind of *global* transactions are expected to be rare, the HWG is not expected to be a bottleneck to the system. This model is thus based on a two-tier group membership architecture, where full and partial membership groups coexist to provide support of strong consistency for edge applications that rely on the Fog Computing paradigm.

5 Conclusion and Future Work

Providing scalability for mobile distributed applications often leads the applications to compromise their consistency requirements. However, some of these applications depend on strong consistency to guarantee that one consistent state is observed by all participants over (some of) the shared objects of the application. While deploying the application in the Cloud facilitates state management and trivially solves the strong consistency requirements, this also creates a geographical barrier between mobile devices and the application servers that penalises performance with an unavoidable latency and jitter. Fog Computing architectures can mitigate this impact by deploying surrogate servers at the network

edge and providing the application state closer to the end users, but its performance depends of a middleware service (GeoDS) able to monitor the application and deploy each fragment of the state at the most convenient location. However, such alternative leads to the emergence of distributed transactions that require increasing efforts to maintain strong consistency. In this paper we have addressed how virtual synchrony, in particular the Light-Weight Group abstraction, can be leveraged by the decisions observed from GeoDS to create fine grained virtual synchronous groups to cope with distributed transactions. The advantage of this solution is that is provides scalability for the system as the number of members in each LWG is much a smaller subset of all participants, without relaxing the strong consistency requirements of the application. These multiple LWGs are mapped onto an underlying virtually synchronous support group, the Heavy-Weight Groups (HWG) that have all the surrogates in the system as participants to support global transactions. This model is thus based on a two-tier group membership architecture, where full and partial membership groups coexist to provide support of strong consistency for edge applications that rely on the Fog Computing paradigm.

Work on the problem of supporting for strong consistency for Fog applications continues. The next step consists in materializing the solution presented in this paper and validate its benefits by evaluating it with a concrete large scale distributed mobile application.

Acknowledgment

This work was supported by FCT through funding of *doit* project, ref PTDC/EEI–ESS/5863/2014, LASIGE Research Unit, ref. UID/CEC/00408/2019, and the Individual Doctoral Grant, ref SFRH/BD/120631/2016.

This work was partially funded by the PAMELA project of the French National Research Agency (ANR-16-CE23-0016).

References

[1] Kenneth P. Birman. 1993. The Process Group Approach to Reliable Distributed Computing. *Commun. ACM* 36, 12 (Dec. 1993), 37–53. https://doi.org/10.1145/163298.163303

[2] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu. 2014. *Fog Computing: A Platform for Internet of Things and Analytics.* 169–186.

[3] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming* (2nd ed.). Springer Publishing Company, Incorporated.

[4] C. Curino, E. Jones, Y. Zhang, and S. Madden. 2010. Schism: A Workload-driven Approach to Database Replication and Partitioning. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 48–57. https://doi.org/10.14778/1920841.1920853

[5] M. Gerla. 2012. Vehicular Cloud Computing. In *2012 The 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net).* 152–155. https://doi.org/10.1109/MedHocNet.2012.6257116

[6] B B Glade, K P Birman, R C B Cooper, and R van Renesse. 1993. Lightweight process groups in the Isis system. *Distributed Systems Engineering* 1, 1 (1993), 29. http://stacks.iop.org/0967-1846/1/i=1/a=004

[7] L. Gruenwald and M. H. Eich. 1990. Choosing the best storage technique for a main memory database system. In *Procs. of the 5th Jerusalem Conf. on Next Decade in Information Technology.* 1–10. https://doi.org/10.1109/JCIT.1990.128263

[8] R. Halalai, P. Sutra, É. Rivière, and P. Felber. 2014. ZooFence: Principled Service Partitioning and Application to the ZooKeeper Coordination Service. In *2014 IEEE 33rd Int'l Symp. on Reliable Distributed Systems.* 67–78. https://doi.org/10.1109/SRDS.2014.41

[9] L. L. Hoang, C. E. Bezerra, and F. Pedone. 2016. Dynamic Scalable State Machine Replication. In *46th IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN).*

[10] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. 2001. Constrained mirror placement on the Internet. In *Proceedings IEEE INFOCOM 2001. Conf. on Computer Communications. 20th Annual Joint Conf. of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, Vol. 1. 31–40 vol.1. https://doi.org/10.1109/INFCOM.2001.916684

[11] K. Jenkins, K. Hopkinson, and K. Birman. 2001. A gossip protocol for subgroup multicast. In *Proceedings 21st International Conference on Distributed Computing Systems Workshops.* 25–30. https://doi.org/10.1109/CDCS.2001.918682

[12] M. Karlsson and M. Mahalingam. 2002. Do We Need Replica Placement Algorithms in Content Delivery Networks. In *In Proceedings of the Int'l Workshop on Web Content Caching and Distribution (WCW.* 117–128.

[13] Leslie Lamport. 1998. The Part-time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (May 1998), 133–169. https://doi.org/10.1145/279227.279229

[14] Diogo Lima, Hugo Miranda, and François Taïani. 2017. Can Graphs Solve the Geo-aware State Deployment Problem?. In *INFORUM 2017 - Atas do 9 Simpósio de Informática*, João Paulo Barraca, Helena Rodrigues, António Teixeira, and José Maria Fernandes (Eds.). 221–232.

[15] D. Lima, H. Miranda, and F. Taiani. 2018. Weighting Past on the Geo-Aware State Deployment Problem. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM).* 01–03. https://doi.org/10.1109/WoWMoM.2018.8449808

[16] P. Mach and Z. Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys Tutorials* 19, 3 (thirdquarter 2017), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318

[17] Fred B. Schneider. 1990. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (Dec. 1990), 299–319. https://doi.org/10.1145/98163.98167

[18] J. Sussman, I. Keidar, and K. Marzullo. 2000. Optimistic Virtual Synchrony. In *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000.* 42–51. https://doi.org/10.1109/RELDI.2000.885391

[19] Werner Vogels. 2009. Eventually Consistent. *Commun. ACM* 52, 1 (Jan. 2009), 40–44. https://doi.org/10.1145/1435417.1435432