Decentralized Kubernetes Federation Control Plane

Lars Larsson
Umeå University
Umeå, Sweden
larsson@cs.umu.se

Harald Gustafsson
Ericsson AB
Lund, Sweden
harald.gustafsson@ericsson.com

Cristian Klein
Umeå University
Umeå, Sweden
cklein@cs.umu.se

Erik Elmroth
Umeå University
Umeå, Sweden
elmroth@cs.umu.se

***Abstract*—This position paper presents our vision for a distributed decentralized Kubernetes federation control plane. The goal is to support federations consisting of thousands of Kubernetes clusters, in order to support next generation edge cloud use-cases. Our review of the literature and experience with the current centralized state of the art Kubernetes federation controllers shows that it is unable to scale to a sufficient size, and centralization constitutes an unacceptable single point of failure. Our proposed system maintains cluster autonomy, allows clusters to collaboratively handle error conditions, and scales to support edge cloud use-cases. Our approach is based on a shared database of conflict-free replicated data types (CRDTs), shared among all clusters in the federation, and algorithms that make use of the data.**

***Index Terms*—kubernetes, cloud computing, federated cloud computing, distributed algorithms, distributed systems, distributed databases**

I. Introduction

In this position paper, we present our vision of a decentralized control plane for federation of Kubernetes-based clusters. We posit that a decentralized control plane offers improvements in autonomy, resilience to failures, and scalability compared to current state-of-the-art centralized approaches.

Since its inception in the mid-2000s, cloud computing has made computational resources available on an on-demand basis. Profitability in operating cloud infrastructure depends on ability to leverage economies of scale with large data centers, and on acquiring cheap electricity and cooling for data centers [1], [2]. The majority of end-users, however, are located in densely populated urban areas. In addition to video streaming, many modern use cases such as connected vehicles, smart homes and cities, and surveillance depend on IoT. Cisco predicts 847 ZB of annual IoT traffic by 2021 [3], and many use cases also require low jitter and latency [4]–[6]. Alternatives to the centralized remote cloud model are required to support these use cases. Edge computing, i.e. processing data closer to the network edge, means less data needs to be transferred to centralized remote clouds [4].

With **edge locations predictively numbering in the thousands** for a large telco [7], driven both by a desire for supporting network function virtualization in telco clouds and to offer the computational infrastructure required to provide applications closer to end-users on the network edge, it is clear that managing the control plane for such large cloud and edge infrastructures needs to operate at previously unimagined scale. Feasible management of edge locations require them to be autonomous clusters [7].

This work has been funded by Ericsson AB, the Swedish Governments strategic effort eSSENCE, the Swedish Research Council (VR) under contract number C0590801 (Cloud Control), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

In the context of this work, we use the word **federation** to mean a *collaboration between a group of autonomous and interconnected clusters of computational resources*. These clusters are likely geographically dispersed, heterogeneous in hard- and software, and owned by a single organizational entity, such as a particular telco. Federations are formed to offer increased availability, robustness, and performance. These properties help the federation as a whole offer an increased level of service, even if parts of the federation experience temporary or, indeed, permanent failure.

In older literature, focus was mainly on bridging technological (heterogeneity in hard- or software) and administrative gaps between possibly competing cloud infrastructure [8], [9]. A key enabler for federations across disparate clusters in modern times comes in the shape of Kubernetes. Kubernetes provides an abstraction layer toward underlying differences in cloud infrastructure offered by different vendors and separates management of infrastructure from that of deployment of software. Infrastructure management is continuously carried out by cloud-specific plugins in Kubernetes, and users of Kubernetes can deploy services on the level of abstraction that Kubernetes provides. Therefore, Kubernetes is in a sense the ultimate embodiment of the compatibility API approaches of yesteryear. With Kubernetes installed on each cluster in a federation, underlying differences in technology should not matter from a user perspective.

The Kubernetes special interest group for federation has provided two iterations of federation methodology and supporting tools. The most recent is presented in detail in Seciton II. It targets forming federations on the scale of dozens of Kubernetes clusters[1]. The number of cloud regions of the major public cloud providers is about a dozen per cloud provider in 2020. Such a target for these tools is likely sufficient for users of centralized clouds, even if one would deploy a low number of multiple environments (development, staging, production) across all regions. However, future generation edge clouds numbering in the thousands require scalability

[1]As per discussion with project leads in official `#sig-multicluster` Slack channel on February 27–28, 2019.

several orders of magnitude larger.

In this work, we explore the question: "how can we design a federation control plane able to handle federations of thousands of Kubernetes clusters?" To that end, we present our vision of a federation control plane for Kubernetes that:

- scales to thousands of federated clusters, suitable for edge and telco cloud use cases, where the entire federation belongs to a single administrative entity (e.g., telco provider);
- is decentralized to avoid having a single point of failure, and allows each constituent cluster equal right to modify global state;
- depends on conflict-free replicated data types to ensure safe concurrency and autonomy; and
- is resilient to failures, should a cluster in the federation become unavailable.

II. Kubernetes federation state of the art

Conceptually, a federated Kubernetes control plane must deal with two separate tasks: *configuration* of federated resources and *propagation* of decisions and related data. In configuration, we include continuous operations to define, configure, and remove federated resources. The output from operations in the form of decisions and data related to these decisions, must then be propagated to at least the affected clusters in the federation.

In the current iteration of the Kubernetes Multicluster special interest group's KubeFed controller, both of these tasks are centralized to components deployed in a single host cluster. Member clusters register with the host cluster, and the KubeFed controller is then solely responsible for continuously propagating data to the member clusters. Member clusters are not "aware" that they are part of a federation, since the KubeFed controller acts as an external privileged admin and clusters do not inter-connect. The loss of this host cluster would therefore render the entire federation without control. Member clusters would still manage their allotted share of federated Kubernetes resources, but because they neither directly communicate nor have awareness of each other, no new decisions or error corrections can be made until the host cluster comes back.

KubeFed defines federated Kubernetes resources, to consist of templates, placement directives, and cluster-specific overrides. The KubeFed controller continuously takes in the templates, determines which clusters should be used for placement, and applies any cluster-specific overrides (i.e. Cluster A should deploy an external load balancer with particular settings, as it is located in a particular public cloud) before making standard Kubernetes API calls to the targeted clusters. KubeFed controllers monitor and collect status from these resources. Should a discrepancy occur, the KubeFed controller will dynamically decide on updates. Due to centralization, users of KubeFed interact solely with the host cluster. No local changes to federated resources can be made directly to member clusters since the KubeFed controller will revert any such changes.

The software company Box developed "kube-applier" for their operational needs. It propagates information by having each cluster continuously applying changes found in a cluster-specific Git repository. Similar Git-centric workflows can trigger CI/CD pipelines. However, this targets only the *propagation* aspect, and thus lacks the ability to quickly and automatically detect and circumvent errors in, e.g., current scheduling or application deployment across the federation. Since sites do not collaborate at all, outages cannot be detected or compensated for.

A. Scheduling across Kubernetes federations

Scheduling, in the context of this work, is the mapping of Kubernetes resources onto clusters. It includes selecting, e.g., which clusters are suitable for a particular federated Deployment of Pods, and how many Pods each cluster should deploy. This process is continuous, as changes in resource availability occur dynamically.

Placement directives allow users control over how scheduling across the federation is allowed. Total number of replicas (Pods) a Deployment consists of must always be specified in the ReplicaSchedulingPreference. Optional placement directives defined by KubeFed allow users of federated Deployments to specify: (a) which clusters are candidates for placement, (b) a relative scheduling weight per cluster (e.g., a cluster with weight 2 should deploy twice as many as one with weight 1), and (c) minimum and maximum number of replicas per cluster. For instance, if a cluster has insufficient capacity to deploy a requested number of Pods, a different cluster can be selected to deploy them instead.

Scheduling is a continuous process, because resource availability and allocation is subject to change. In KubeFed, this process is iteratively carried out by the KubeFed controller in the host cluster. Because there is no way to reserve resources ahead of time in Kubernetes, KubeFed schedules a portion of Pods onto each cluster, verifies how many Pods could be started, and adjusts accordingly.

Cases where one cluster (Cluster A) lacks available resources and another (Cluster B) needs to temporarily take on additional load can occur. Afterwards, KubeFed will try to converge back to the originally intended load division. This requires optimistically attempting to schedule more resources onto Cluster A. Should this new allocation succeed, the surplus scheduled on Cluster B can be reduced. Thus, at times, there may be a larger total number of Pods deployed than the user specified, but the system eventually convergences to the correct deployment size.

B. KubeFed deficiencies

We find that KubeFed is not suitable for the edge computing use-cases we consider in this work. Relying on a single point of failure inherently poses risks in large-scale distributed systems. KubeFed also, in its current implementation, targets too few clusters for our use-case. Also, edge clusters may have long latencies towards a central controller.

In the future, we would also like to extend the KubeFed API such that it is possible to specify placement constraints such as affinity and anti-affinity on the level of clusters in a federation [10]. Placement constraints are available within a single cluster on the host level, but not on a cluster level.

III. Overview and architecture

Our proposed solution aims to replace the central KubeFed controller with distributed federation controllers in each cluster. The API defined by KubeFed and the community should stay the same however, as we think that compatibility with current tooling is a key enabler for industry adoption. Each controller applies any templating, placement applicability, and value substitutions on the federated resources. This preserves cluster autonomy, provides failure resilience, and scales to orders of magnitude required for edge computing deployments.

The federation resources are propagated by a distributed federated database that makes use of conflict-free replicated data types (CRDTs). As CRDTs are central to our proposed solution, we briefly devote some paragraphs to introducing readers to the problem that they solve and how they work. Distributed databases are distributed systems wherein *nodes* manage shared *objects* that have associated *values*. As multiple nodes are able to update a shared object's value concurrently, problems can occur. A *conflict* is a disagreement between nodes about the actual current value of a given object. Handling conflicts is known as *concurrency control*.

CRDTs, introduced by Shapiro et al. [11], are a class of distributed data types with mathematically provable conflict-free properties, due to restricting the operations that can be carried out against them. A semi-lattice operation $\cdot$ that exhibits associativity ($x \cdot (y \cdot z) = (x \cdot y) \cdot z$), commutativity ($x \cdot y = y \cdot x$), and idempotency ($x \cdot x = x$) can be shown to be safe from conflicts [11]. Importantly, order of when operations are applied does not matter, and are safe to repeat. Thus, they can be distributed among the nodes in a distributed database safely via, e.g., a Gossip protocol [12].

CRDTs do not constitute a free lunch, however. Mathematically proven guaranteed freeness from value conflict comes at a network traffic cost, as object representations often grow either with number of nodes or by carrying object history. More efficient ways of transmitting this information exists, e.g. by sending only the deltas [13]. When CRDTs were introduced, the known set of data types that could be implemented as CRDTs was limited [11], [14]. However, more recent work has provided support for CRDT-powered linked lists [15] and nested maps [16], which means that fully generalized data storage is possible. The latter is implemented in the Riak KV database.

In practice, large deployments of CRDT-based systems show great scalability [17], [18]. Famously, the video game "League of Legends" chat feature handles tens of thousands of chat messages per second by millions of concurrent players [19]. The bet365 online gambling company has used Riak KV for years to handle business-critical data that updates with high frequency [20]. Thus, there should be no obvious problem preventing CRDTs to be used on the scale of edge computing.

Kubernetes itself relies on a distributed key-value store type of database called etcd. Etcd uses the Raft algorithm [21] to achieve consensus among nodes about object values. As such, it uses pessimistic concurrency control in that a sufficiently large subset of master nodes ($N/2+1$, where N is the number of master nodes in the distributed system) must vote whether to allow an object update. Due to Raft reliance upon timing for determining membership in the system, etcd is notoriously sensitive to network and disk I/O performance [22], [23] and requires special tuning for use in wide-area networks. Thus, performance on dependable local networks is high and stable, but as distance and latency increases, etcd performance decreases rapidly. Yet the biggest problem is the size of the distributed system: etcd clusters should not exceed 7 master nodes in size, with 5 being suggested for production-ready use [24]. Because we envision a federation of clusters two orders of magnitude larger than that, scattered over wide geographical areas, reliance upon a distributed etcd is not possible.

Our architecture, making use of a distributed federated database storing CRDTs, is shown in Fig. 1. As shown, the federation control plane exists in addition to standard Kubernetes components. We envision extending the Kubernetes API server such that it separates operations upon federated objects from operations upon standard, cluster-local, objects. Federated operations are passed to our federation control plane components, which depend on the distributed federated database for conflict-free management of shared objects. When our federated control plane components shall create Kubernetes resources on the cluster upon which they are running, they use the Kubernetes API to do so. Section IV shows the flow of traffic in greater detail, and Section V outlines how CRDTs are used for resource scheduling, i.e., collaboratively assigning Kubernetes resources to clusters.

IV. Federation control plane traffic flow

In our proposed system, the federation control plane communicates indirectly via a distributed database. The contents of this database is sufficient to make all clusters in the federation aware of the intended goal and the current state of the system. Distributed federation controllers use this information to modify their local Kubernetes clusters via Kubernetes' API.

All federation-related user-facing APIs are invoked via the Kubernetes API server (see Fig. 1). Thus, they benefit from security and access control provided by the API server. The declarative and eventual consistency properties of all Kubernetes API interactions are also preserved. Thus, our proposed system behaves similar to current-generation KubeFed, except that there is no (single) host cluster limitation.

API calls concerning federated resources are handled by the distributed federated controllers. They create, update, or delete information in the federated database, which makes use of CRDTs. Thus, the federation member clusters have all

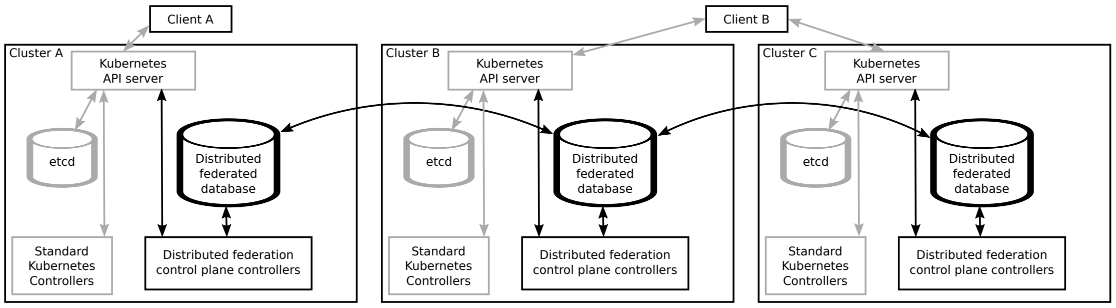
Client A
Client B
Cluster A
Cluster B
Cluster C
Kubernetes API server
etcd
Distributed federated database
Standard Kubernetes Controllers
Distributed federation control plane controllers

Fig. 1. Architectural overview of distributed federation control plane where new components and communication directions are in black, standard Kubernetes ones are in gray.

information required to autonomously perform scheduling as a distributed decision.

Riak KV, the database currently under evaluation for our implementation, has shown great scalability properties [20], [25], as long as care is taken to not make the distributed objects themselves too large (reports indicate that objects larger than 500kB should be avoided [16], [26]). Riak KV gives up consistency in favor of availability in the case of network partitions and to achieve low latency otherwise [27], which are acceptable trade-offs for use with Kubernetes, which itself operates with only an eventually consistent guarantee.

V. Resource scheduling

We propose that resource scheduling in a federation of Kubernetes clusters be carried out via a distributed algorithm, rather than in a central host cluster.

Neither centralized nor distributed federation controllers can ahead of time know how many Pods can be deployed onto each cluster. Nor can any guarantees be made regarding future resource availability. Thus, the distributed federation controllers continuously and collaboratively update a shared data structure with information about how many Pods of a particular federated Deployment they have deployed locally.

Placement directives dictate what the ideal scheduling decision looks like, by listing (possibly weighted) candidate clusters and any lower or upper limits on Pods that can be deployed onto any particular candidate cluster. When, due to resource unavailability, ideal scheduling cannot be realized, clusters can iteratively collaborate to redistribute resources among them (e.g. using cluster ID order or other suitable agreed up on information as tie-breaker). By storing current scheduling status in a CRDT, updates can be seen by federation controllers in the other federation member clusters.

A. Example

Say that Clusters A, B, and C are all candidate clusters for a given federated Deployment. The total number of Pods is set to 80. Cluster A has relative weight 2, whereas Clusters B and C each have relative weight 1. A placement directive limits Cluster C to maximum number of replicas at 10. Ideal distribution is therefore that Cluster A takes $2/3$ of the 10 Pods Cluster C is forbidden to deploy, and Cluster B $1/3$ of them, due to their weights.

Ideal deployment sizes at each cluster can be autonomously and independently derived from these placement directives. Cluster A should deploy 47 Pods and Clusters B and C 23 and 10 Pods, respectively. This is the initial state in Figure 2.

However, some event may limit the number of Pods Cluster A can actually deploy. In Figure 2, this is shown as a 7-pointed star, and could be that a Kubernetes node in the cluster stopped being Ready. The federation controller in Cluster A notices and updates its value of the CRDT to 33.

Cluster B allocated its allotted number of Pods and deployed them correctly. As it continuously monitors the distributed database, it eventually observes that Cluster A has changed its allocation to 33, which brings the total to below 80. In response, Cluster B alters its own allocation to iteratively deploy Pods that Cluster A failed to allocate, bringing the total of deployed Pods to the desired 80 (for space reasons shown in Figure 2 as single step). Cluster C also notices the discrepancy, but is unable to deploy more Pods, due to the placement directives. The decisions to deploy more or stay at the same number of Pods is shown as $+$ and $=$, respectively.

Federation controllers at Clusters A and B are aware that the current distribution of Pods is not the ideal, because the placement directives are not perfectly adhered to. Therefore, they will, when resources become available at Cluster A, iteratively allocate resources at Cluster A and deallocate at Cluster B. As with KubeFed, temporary deployment of too many Pods can occur before convergence to the desired distribution is attained.

VI. Related Work

Dividing functionality into a higher-level control plane separate from a lower-level "data plane" is not a new concept — indeed, Kubernetes does so on the master/worker node level within clusters. Extending this separation across clusters leads to yet a higher level of control plane abstraction: a meta-level where the cluster-specific control plane in a cluster A cannot directly affect that of cluster B (i.e., it cannot dictate upon which worker nodes a given workload should be deployed [10]), but the overall meta-level system can make it the responsibility of B to handle a subset of the

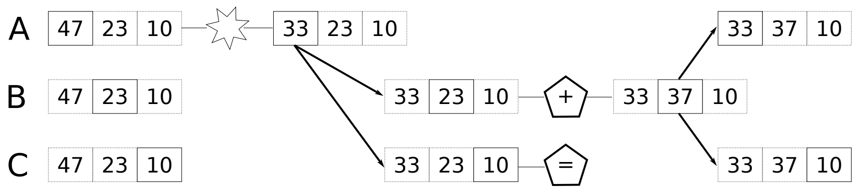

Fig. 2. Distributed autonomous federation controllers scheduling a Deployment by interacting indirectly via a distributed counter of deployed Pods to each cluster.

workload. In particular as federations grow larger, decisions about how to use them must become automated. Adding to manually specified rules limiting deployment to a particular geographical region for legal purposes, one can imagine that other selection criteria such as trust [28] will also be very important.

Control planes can be found in the related field of software-defined networking (SDN). In SDN, the physical infrastructure (networking equipment) constitutes the "data plane", which handles traffic flows in a way that is configured by the control plane [29]. The latter often has a bigger-picture view of the overall system, and can be meta-level managed on a higher level by distributed global control planes such as HyperFlow [30] or Onix [31].

A recent decentralized approach to orchestrating containerized applications was presented by Jiménez and Schelén [32]. Their decentralized orchestrator for containerized microservice applications (DOCMA) system targets scheduling container runtimes on an edge computing scale of thousands of nodes, and uses a decentralized control plane to do so. However, crucially, their goals are to make DOCMA itself an orchestrator, making it a distributed alternative to Kubernetes federations. DOCMA uses the Kademlia distributed hash table [33] to create an overlay network among federation members. Applications are mapped onto a subset of nodes via an ID mapping function that takes Kademlia routing keys into account. The nodes communicate via a DOCMA protocol, and deploy the application after a node suitability search (the paper does not explain how this is done).

Our work differs from DOCMA not only in underlying technology choices, but also in goal and scope. We believe that Kubernetes compatibility to be key for industry adoption, rather than trying to replace it with yet another application orchestrator.

VII. Summary

The decentralized control plane problem consists of two parts: *configuration* of federated resources and *propagation* of decisions and related data. Our work focuses on addressing the former by eventually consistent algorithms for reconciling global state, while leveraging a distributed CRDT-powered distributed database for the latter in a scalable and conflict-free manner.

Thus, in this position paper, we have proposed a distributed federation control plane for Kubernetes federations. Similar to the current state of the art, KubeFed, it leverages the standard Kubernetes API server to offer support for federated resource management. In contrast to KubeFed, the clusters in the federation are aware of each other and collaborate using a distributed database and distributed algorithms. Thus, we avoid having a centralized system with a single point of failure, preserve cluster autonomy, and allow clusters to collaboratively respond to error conditions. The use of a distributed approach scales to thousands of clusters in a federation, a scale that we, and other researchers in the field [32], [34]–[36], envision is required to support future generation edge cloud infrastructures.

References

[1] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, Dec. 2014. [Online]. Available: https://doi.org/10.1145/2656204

[2] V. D. Reddy, B. Setz, G. S. V. R. K. Rao, G. R. Gangadharan, and M. Aiello, "Metrics for sustainable data centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 3, pp. 290–303, July 2017.

[3] Cisco Systems Inc., "Cisco global cloud index: Forecast and methodology, 2016–2021 white paper," 2016, visited 14 February 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

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

[5] F. Cicirelli, A. Guerrieri, G. Spezzano, and A. Vinci, "An edge-based platform for dynamic smart city applications," *Future Generation Computer Systems*, vol. 76, pp. 106 – 118, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16308342

[6] P.-H. Tsai, H.-J. Hong, A.-C. Cheng, and C.-H. Hsu, "Distributed analytics in fog computing platforms using tensorflow and kubernetes," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sep. 2017, pp. 145–150.

[7] R.-A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, and J. M. Soares, "Edge computing resource management system: a critical building block! initiating the debate via openstack," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/cherrueau

[8] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1–4:11, July 2009.

[9] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.

[10] L. Larsson, D. Henriksson, and E. Elmroth, "Scheduling and monitoring of internally structured services in cloud federations," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, pp. 173–178, ISSN: 1530-1346.

[11] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free replicated data types,” in *Stabilization, Safety, and Security of Distributed Systems*, X. Défago, F. Petit, and V. Villain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 386–400.

[12] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, “Epidemic algorithms for replicated database maintenance,” in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC 87. New York, NY, USA: Association for Computing Machinery, 1987, p. 112. [Online]. Available: https://doi.org/10.1145/41840.41841

[13] P. S. Almeida, A. Shoker, and C. Baquero, “Efficient state-based crdts by delta-mutation,” in *Networked Systems*, A. Bouajjani and H. Fauconnier, Eds. Cham: Springer International Publishing, 2015, pp. 62–76.

[14] A. Bieniusa, M. Zawirski, N. Preguiça, M. Shapiro, C. Baquero, V. Balegas, and S. Duarte, “An optimized conflict-free replicated set.” [Online]. Available: http://arxiv.org/abs/1210.3368

[15] P. Nicolaescu, K. Jahns, M. Derntl, and R. Klamma, “Near real-time peer-to-peer shared editing on extensible data types,” in *Proceedings of the 19th International Conference on Supporting Group Work*, ser. GROUP 16. New York, NY, USA: Association for Computing Machinery, 2016, p. 3949. [Online]. Available: https://doi.org/10.1145/2957276.2957310

[16] C. Meiklejohn, “On the composability of the Riak DT map: expanding from embedded to multi-key structures,” in *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*, ser. PaPEC ’14. Amsterdam, The Netherlands: Association for Computing Machinery, Apr. 2014, pp. 1–2. [Online]. Available: https://doi.org/10.1145/2596631.2596635

[17] X. Lv, F. He, W. Cai, and Y. Cheng, “A string-wise CRDT algorithm for smart and large-scale collaborative editing systems,” vol. 33, pp. 397–409. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474034616301811

[18] W. Yu, “A string-wise CRDT for group editing,” in *Proceedings of the 17th ACM international conference on Supporting group work*, ser. GROUP ’12. Association for Computing Machinery, pp. 141–144. [Online]. Available: https://doi.org/10.1145/2389176.2389198

[19] T. Hoff, “How league of legends scaled chat to 70 million players – it takes lots of minions,” Online http://highscalability.com/blog/2014/10/13/how-league-of-legends-scaled-chat-to-70-million-players-it-t.html (visited 2020-03-04), Oct 2014.

[20] M. Owen, “Using Erlang, Riak and the ORSWOT CRDT at bet365 for Scalability and Performance,” Stockholm, Sweden, Jun. 2015.

[21] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[22] etcd Development and Communities, “Performance,” Online https://github.com/etcd-io/etcd/blob/master/Documentation/op-guide/performance.md (visited 2020-03-04), Apr 2019.

[23] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, “Impact of etcd deployment on kubernetes, istio, and application performance,” vol. n/a, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2885. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2885

[24] etcd Development and Communities, “Frequently asked questions (FAQ),” Online https://github.com/etcd-io/etcd/blob/master/Documentation/faq.md (visited 2020-03-04), Aug 2019.

[27] 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, conference Name: Computer.

[25] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, “Performance Evaluation of NoSQL Databases: A Case Study,” in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, ser. PABS ’15. Austin, Texas, USA: Association for Computing Machinery, Feb. 2015, pp. 5–10. [Online]. Available: https://doi.org/10.1145/2694730.2694731

[26] R. Brown, Z. Lakhani, and P. Place, “Big(ger) sets: decomposed delta CRDT sets in Riak,” in *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data*, ser. PaPoC ’16. London, United Kingdom: Association for Computing Machinery, Apr. 2016, pp. 1–5. [Online]. Available: https://doi.org/10.1145/2911151.2911156

[28] U. Ahmed, I. Petri, O. Rana, I. Raza, and S. A. Hussain, “Risk-based service selection in federated clouds,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 77–82.

[29] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized? state distribution trade-offs in software defined networks,” in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN ’12. Helsinki, Finland: Association for Computing Machinery, Aug. 2012, pp. 1–6. [Online]. Available: https://doi.org/10.1145/2342441.2342443

[30] A. Tootoonchian and Y. Ganjali, “HyperFlow: a distributed control plane for OpenFlow,” in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN’10. San Jose, CA: USENIX Association, Apr. 2010, p. 3.

[31] 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*, ser. OSDI’10. Vancouver, BC, Canada: USENIX Association, Oct. 2010, pp. 351–364.

[32] L. L. Jiménez and O. Schelén, “DOCMA: A decentralized orchestrator for containerized microservice applications,” in *The 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications (IEEE Cloud Summit 2019)*. IEEE, 2019.

[33] P. Maymounkov and D. Mazieres, “Kademlia: A peer-to-peer information system based on the XOR metric,” in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[34] F. Quesnel and A. Lèbre, “Cooperative dynamic scheduling of virtual machines in distributed systems,” in *Euro-Par 2011: Parallel Processing Workshops*, M. Alexander, P. D’Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, and J. Weidendorfer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 457–466.

[35] D. Balouek, A. Lebre, and F. Quesnel, “Flauncher and DVMS — deploying and scheduling thousands of virtual machines on hundreds of nodes distributed geographically,” in *IEEE International Scalable Computing Challenge (SCALE 2013), held in conjunction with the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid computing (CCGrid’2013)*. IEEE, 2013.

[36] Y. Xia, X. Etchevers, L. Letondeur, A. Lebre, T. Coupaye, and F. Desprez, “Combining heuristics to optimize and scale the placement of iot applications in the fog,” in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, Dec 2018, pp. 153–163.