Decentralized reconfiguration management in distributed clouds

Miloš Simić[1[0000-0001-8646-1569]] , Boris Lauš, Tamara Ranković[1[0009-0009-4973-2716]]

Faculty of Technical Sciences, University of Novi Sad, Serbia
milos.simic@uns.ac.rs

Abstract. Managing the reconfiguration of large-scale, geo-distributed cloud systems presents significant challenges, particularly when ensuring minimal downtime and scalability. Traditional approaches rely on centralized control mechanisms, which can become bottlenecks in modern distributed environments. As new cloud models, such as the distributed cloud (DC), emerge to meet the demands for lower latency and improved privacy, scalable and reliable reconfiguration strategies are essential. This work focuses on enhancing the existing reconfiguration architecture by reducing dependence on centralized components. We propose a hybrid approach for reconfiguring DC applications and system components that integrates a centralized method for smaller deployments with a decentralized, gossip-based, method, for large-scale DCs. Our solution enables efficient propagation of configuration updates across clusters while allowing selective application of changes based on node labels. Beyond reconfiguration, our approach can facilitate other cluster-wide information-sharing tasks. Our solution improves scalability, fault tolerance, and flexibility, making distributed cloud reconfiguration more efficient and practical for diverse deployment scenarios.

Keywords: distributed systems, peer-to-peer systems, cloud, distributed cloud, reconfiguration, gossip protocols, peer to peer systems

1. Introduction

Efficient operation and reconfiguration of large scale geo-distributed systems are complex tasks [1]. We must ensure that an existing system can be reconfigured for different purposes, usually without introducing any downtime. An additional problem is reconfiguration of applications running in such environments. Traditional techniques, relying on centralized points where users submit their new configuration, that is then delivered to physical nodes, are not scalable and can be too time-consuming [2]. As new cloud models are being developed to meet the needs of modern applications, it is necessary to provide novel solutions for reconfiguring both applications and system components in such complicated environments, with improved scalability. Reliable and straightforward configuration process is vital to the wider applicability and adoption of these models.

In this work, we explore a cloud computing model called the distributed

cloud (DC) [2]. What sets the DC apart from traditional centralized cloud systems is its ability to form on-demand, geographically DCs that complement the existing cloud infrastructure. This decentralized approach ensures that modern application requirements, such as low latency and enhanced privacy, are effectively met. Our goal is to expand upon the current reconfiguration architecture [3] by developing strategies that minimize the need for centralized control. Specifically, this paper focuses on enabling efficient and reliable reconfiguration of DC systems and applications, and the propagation of information across clusters using gossip protocols.

Current solutions only partially address the specific challenges of this problem. Therefore, we design and implement a more scalable and resilient peer-to-peer configuration dissemination strategy. By combining the centralized solution [3] with a newly developed decentralized approach for large-scale deployments, we can accommodate a wider range of potential requirements.

In addressing the problem, we review the existing body of work and discuss the enhancements made to the naive centralized information dissemination, with the focus on the use of gossip protocols and their advantages. Afterwards, we define architecture extensions, API, and the behavior of the reconfiguration-related operations. We note that no introduced changes should break the existing primitives for misconfiguration prevention [4], and the existing direct push strategy [3]. The extension should support push patterns suitable for different use cases, but also handle partitioning and dissemination of large configuration files, and their subsequent merge in large-scale DCs.

We implement a prototype based on the proposed design to demonstrate its feasibility and usability in a multi-tenant DC environment [5][6]. And lastly, we discuss scalability differences between two delivery models: (1) centralized (direct) configuration delivery, and (2) decentralized where we rely on peer to peer (P2P) and gossip mechanisms to disseminate new configuration in the system.

In Section 2 we provide an overview of the existing reconfiguration solutions, focusing on those that are decentralized and designed for distributed and cloud environments. Section 3 explores the DC model, its main properties and requirements for the decentralized reconfiguration solution.. In Section 4 we propose the general architecture of the solution and discuss its implementation, while Section 5 concludes the paper and presents directions for future work.

2. Related work

In this section, we'll give an overview of configuration management solutions for different systems and environments, both centralized and decentralized.

Based on these, we extracted requirements for the DC platform and detected different strategies that need to be addressed in various DC scenarios.

EdgePier is a fully decentralized container registry developed by Becker et al. [12] that can be deployed across edge sites and is able to decrease container deployment times by utilizing peer-to-peer connections between participating nodes. Image layers are shared without the need for further centralized orchestration entities. The conducted evaluation shows that the provisioning times are improved by up to 65% in comparison to a baseline registry, even with limited bandwidth to the cloud. While interesting as an idea, this tool is restricted only to container images. But nonetheless the research shows promising ideas to do decentralized dissemination of container images that could be large in size, and therefore reduce overall latency in the system.

In [11] researchers looked at overlay networks such as BitTorrent and Avalanche that are increasingly used for disseminating potentially large files from a server to many end users via the Internet. The authors provided the analytic performance analysis that is based on a new uplink-sharing version of the well-known broadcasting problem and separation of one large file in multiple small parts. They also investigate the performance of a decentralized strategy, providing evidence that the performance of necessarily decentralized P2P file dissemination systems should be close to this bound and therefore that it is useful in practice. Even though this is not directly related to the system we are trying to develop, it shows a promising option on splitting one large file into parts and disseminating them over P2P networks.

The Akamai Configuration Management System [13] was developed to facilitate fault-tolerant, consistent updates with efficient and secure delivery. The system consists of publishers, who can concurrently push new versions of configuration files, and receivers, that run on each node and fetch configuration from storage points. Applications connect to their receiver and subscribe to a certain configuration file. The pull-based solution is fully optimized for HTTP download and caching capabilities but it might not be best suited for large-scale and dynamic DCs. But it shows an interesting approach on reacting to changes (filtering content) in the system.

Kubernetes [14] follows the principle of separating code from configuration and uses the ConfigMap primitive to achieve so. Users first specify the ConfigMap resource, and later bind it to a pod by referencing it in its specification. Kuberentes uses labels as a lightweight binding mechanism for efficient filtering artifacts in cloud deployments. However, no Kubernetes component will notify the application of the change, so a pull model has to be employed, which may introduce more traffic in the large-scale DCs. But the system of labels allow efficient filtering artifacts, while ConfigMaps allow easier ways for users to interact with the system. This can be a useful idea when extending existing centralized options [3] to provide more decentralized configuration in large-scale DC environments.

3. Distributed clouds

As a technology with certain shortcomings, cloud computing is constantly improving, and it is believed that next-generation technologies, such as DC, will successfully overcome these challenges [9]. DC improves global communication of services, enabling faster and more efficient communication in specific regions [9]. DC offers more security, faster content delivery, and cost-effectiveness, but faces the risk of hacker attacks during transmission, while its popularity is growing with the development of the Internet of Things [9].

Researchers and companies are increasingly interested in applications in the cloud, while large companies are considering the transition to hybrid clouds. Parallel processing is required to efficiently execute complex applications [10]. Distributed systems enable communication between computers and other devices through message exchange. With the reduction of hardware costs and advances in networking technologies, the use of distributed systems is expanding [10].

The most defining characteristic of the DC model lies in the hierarchical structure. This structure directly supports scalable and modular deployment. At the lowest level, each cluster is composed of multiple nodes. Furthermore, clusters are aggregated into regions, which are typically co-located to minimize communication delays and enhance cooperation. Multiple regions form the final level, topology, enabling high-level orchestration tasks such as data replication, inter-region load balancing, and application failover. This multi-layered organization not only supports geographic and operational flexibility, it instigates it. This model introduces opportunities for region offloading and high availability mechanisms across distributed zones.

User interaction with the system is primarily done through specification interfaces. Users define the configuration and desired behavior of their DCs, while the rest of the system manages processes that drive the DC to the desired state. The control plane takes the responsibility for provisioning infrastructure, orchestrating workloads, and maintaining compliance with user-defined constraints.

Properties of DCs that influence the design of our reconfiguration system the most are:

- **Heterogeneity:** nodes in the same cluster can, and mostly will, have diverse characteristics, many of them are commodity hardware with limited resources,
- **Highly dynamic nature:** users can form, alter and dispose of the infrastructure based on their needs, we expect that infrastructure

continuously transform with many nodes joining or leaving at the same time,

- **Large scale:** it is possible for a cluster to contain thousands or even tens of thousands of nodes. This behavior is anticipated as the expected capabilities of individual nodes are low,
- **Communication over the network:** No specialized network providing high speed or reliability is intended for use.

Our main objective is to develop a reconfiguration system that meets the stated requirements. However, we must also take into account that not all installments of the DCs are going to be large-scale in nature, and for some of them, prompt delivery is crucial [3].

4. Decentralized reconfiguration

The reconfiguration API should serve diverse clients and their potentially conflicting needs, e.g. smaller installments of DCs could benefit more from the centralized strategy [3], while the decentralized one could be more suitable for large-scale DCs with multi-tenant properties [5][6]. The API must also provide information about the current state of the entire cluster(s), displaying what configuration the nodes have received and are yet to receive. This could be especially useful when DCs span vast geographical areas, and direct insight might take a lot of time for manual lookup [1][2]. But also, preventing misconfiguration in the system, and delivery of multiple identical requests [4]. Here we can detect two main components: (1) control plane, designed as an entry point to the system, holds state and acts as a mediator between users and DCs, and (2) DCs that users want to interact with.

Every configuration and reconfiguration of the DCs is done using YAML configuration files. Users specify what is the new configuration and what strategy should be employed to disseminate it. After the submission, the control plane then performs all required actions. Choosing a proper strategy can have a large impact on both scalability and time required to receive new configuration. Therefore, users should be aware of their DC installments' capabilities.

When the centralized strategy is preferable, users specify nodes to which new configuration should be delivered. All selected nodes will directly receive configuration from the control plane, as done in the previous work presented in [3]. If configuration is intended for the entire cluster(s), a query selecting the nodes should be empty indicating no node filtering is needed.

The communication pattern for this strategy is displayed in Fig. 1. Here we can clearly see that the control plane has to communicate excessively with the existing DC layer. For large-scale DCs, this could present a major

scalability issue, but can be a good solution for small DCs or for scenarios when immediate reconfiguration is needed.

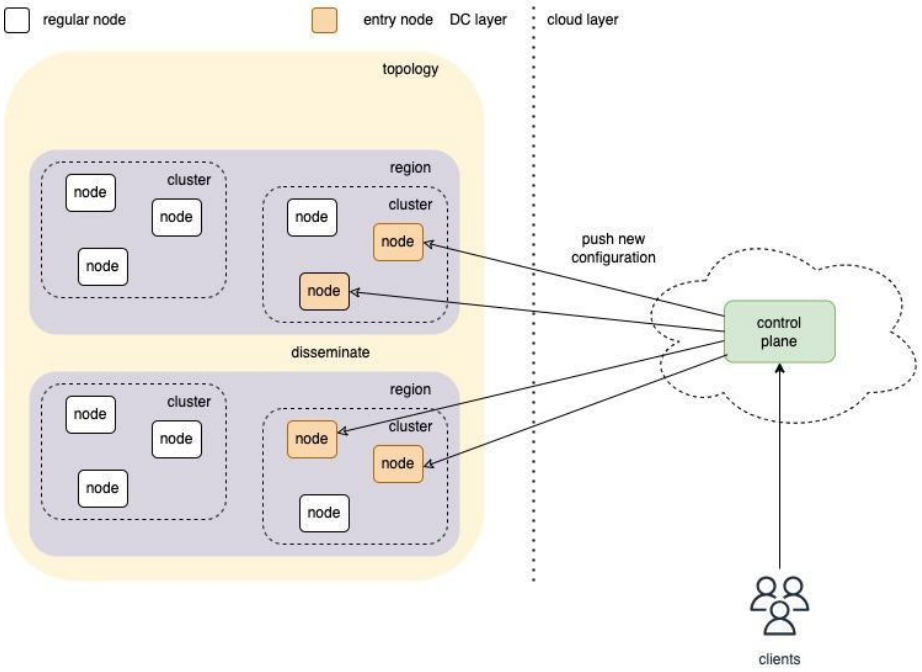

Fig. 1. Centralized strategy for delivering new configuration.

On the other hand, when the decentralized strategy is employed, the control plane or the user selects a subset of nodes from the cluster(s), and the message is propagated only to them. Subsequently, those nodes start gossiping the message to all the other nodes in P2P manner, and eventually every node in the cluster(s) will receive information and react upon its arrival. Here, we piggyback the information on already present protocols in the system. These protocols run continuously, grouping nodes into clusters.

The communication pattern for this strategy is displayed in Fig. 2. Here we can see that there is little communication between the control plane and the existing DCs (depicted with full body arrows), and that the gossip mechanisms disseminate new configuration throughout the DCs (depicted with the dotted body arrows). With this approach, large-scale DCs can be reconfigured without introducing too much communication with the central point, which improves scalability dramatically.

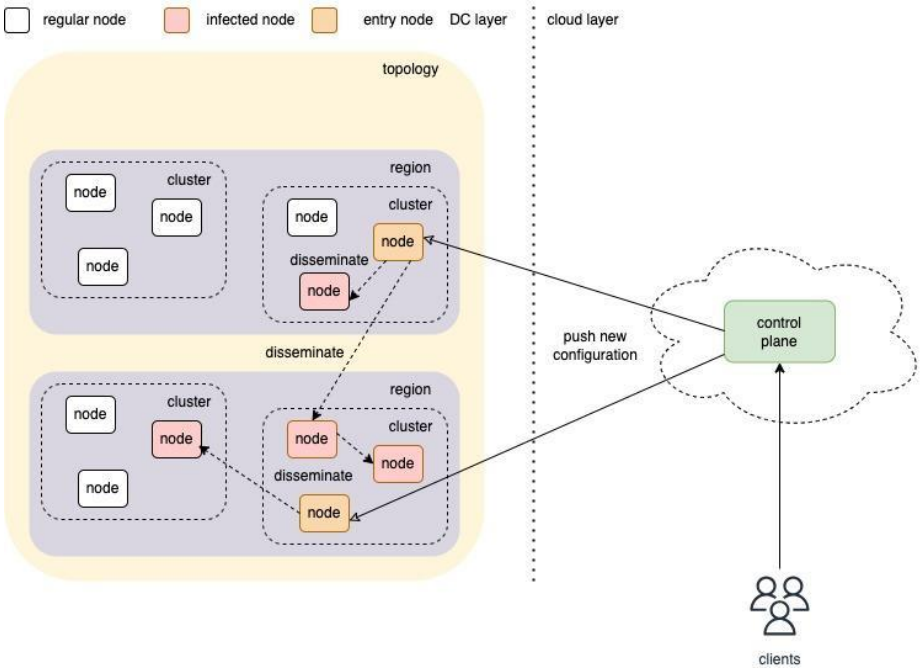

Fig. 2. Decentralized strategy for delivering new configuration.

However, we don't always want every node to react to the new configuration. Not every configuration is desired for every node in the cluster(s). Because of that, we allow users to specify which nodes are affected by what configuration using a set of filters. This can be done easily without changing the existing infrastructure and introducing more complexity into the system. By utilizing already existing DCs labels for every node [3], we can do the filtering process. Labels represent a group of textual key-value pairs assigned by the users when node is added to the system [1][2]. When a node receives configuration via gossip, if labels from that message match its own labels, it acknowledges the new configuration. Otherwise, it discards it, but still propagates it to its peers.

We implement the described strategy[1] by extending the control plane and the node agent, both written in Go programming language. The control plane can now support the new strategy, while agents can disseminate information using a gossip protocol, react to it, and perform filtering based on labels. Agents are also able to partition large configuration files and disseminate its parts over the gossip protocol, as well as reconstruct them into the full configuration.

[1] https://github.com/c12s/star

Communication between the control plane and clusters is facilitated by the NATS message broker[2], and operation of the peer-to-peer network inside clusters is performed by the Hashicorp Serf library[3], implementing the Lifeguard protocol [7], an enhancement of the SWIM gossip protocol [8].

Apart from the reconfiguration module, other parts of the system can also benefit from our solution. As now nodes in the cluster run a gossip-style protocol with the ability to disseminate information, other data can be piggybacked by reusing the existing primitives. One limitation we emphasize here is the risk of high bandwidth consumption, which can be reduced by employing compression techniques to reduce the traffic overhead.

Besides the idea that direct reconfiguration [3] is tailored to small cluster(s), and decentralized reconfiguration is tailored to large-scale cluster(s), there are two more properties that we mustn't overlook, and those are (1) consistency, and (2) reaction time. In the centralized approach [3], users prefer a more strongly consistent approach, while in the decentralized one they prefer scalability, therefore an eventually consistent approach. Regarding reaction time, when using the centralized strategy, cluster(s) will receive new configuration messages instantaneously, while with the decentralized strategy, we rely on P2P protocols and their internal strategy of information dissemination which can be slower in large-scale cluster(s). One more situation where a decentralized solution might be better suited than a centralized one is when we have big configuration elements. This can strain the network even more, therefore introducing more latency. These two properties represent additional factors to take into account when users and/or other parts of the system plan to optimize future reconfigurations.

5. Conclusion

This paper presents a cloud computing model of the DC with the ability to form ad-hoc, geographically dispersed DCs that complement the existing cloud infrastructure. This decentralized approach ensures that modern application requirements, such as low latency and enhanced privacy, are effectively met.

We also expanded the current reconfiguration architecture by adding the strategy that minimizes the need for centralized control and communication with the DCs, which then enables efficient and reliable reconfiguration of DC systems and applications, and the propagation of information across clusters using gossip protocols and P2P communication. By combining the existing centralized solution with a newly developed decentralized approach for large-

[2] https://nats.io/
[3] https://github.com/hashicorp/serf

scale deployments, we can accommodate a wider range of potential requirements.

Future work will focus on two main directions: First, we aim to remove reliance on a single gossip protocol by offering other possibilities that might show better properties for specific use cases, and allow for dynamic protocol switching. Second, we plan to extend the system with compression techniques that would further reduce the size of messages sent over the network in both ways: (1) control plane to DCs, and (2) between peers in the DCs.

Acknowledgements

 Funded by the European Union (TaRDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not nec- essarily reflect those of the European Union. Neither the European Union nor the grant- ing authority can be held responsible for them.

References

1. Simić, M., Sladić, G., Zarić, M., Markoski, B. (2021). Infrastructure as Software in Micro Clouds at the Edge. Sensors, 21(21), 7001. https://doi.org/10.3390/s21217001.
2. Simić, M., Prokić, I., Dedeić, J., Sladić, G., & Milosavljević, B. (2021). Towards edge computing as a service: Dynamic formation of the micro data-centers. IEEE Access, 9, 114468-114484.
3. Ranković, T., Kovačević, I., Maksimović, V., Sladić, G., Simić, M. (2024). Configuration Management in the Distributed Cloud. In: Trajanović, M., Filipović, N., Zdravković, M. (eds) Disruptive Information Technologies for a Smart Society. ICIST 2024. Lecture Notes in Networks and Systems, vol 860. Springer, Cham. https://doi.org/10.1007/978-3-031-71419-1_20.
4. T. Ranković, F. Šiljić, J. Tomić, G. Sladić and M. Simić, "Misconfiguration Prevention and Error Cause Detection for Distributed-Cloud Applications," 2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY), Pula, Croatia, 2024, pp. 000297-000302, doi: 10.1109/SISY62279.2024.10737513.
5. Simić, M., Dedeić, J., Stojkov, M., Prokić, I. (2024). A Hierarchical Namespace Approach for Multi-Tenancy in Distributed Clouds. IEEE Access. 10.1109/ACCESS.2024.3369031.
6. Simić, M., Dedeić, J., Stojkov, M., Prokić, I. (2025). Data Overlay Mesh in Distributed Clouds Allowing Collaborative Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2024.3525336.
7. Dadgar, A., Phillips J., and J. Currey, "Lifeguard: Local Health Awareness for More Accurate Failure Detection," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Luxembourg, Luxembourg, 2018, pp. 22-25, doi: 10.1109/DSN-W.2018.00017.
8. Das, A., Gupta, I., Motivala, A. (June 23, 2002). "SWIM: Scalable weakly-consistent infection-style process group membership protocol". Proceedings International Conference on Dependable Systems and Networks. pp. 303–312. doi:10.1109/DSN.2002.1028914. ISBN 0-7695-1597-5. S2CID 11094028.

9. Ali T Atieh. “The next generation cloud technologies: a review on distributed cloud, fog and edge computing and their opportunities and challenges”. In: ResearchBerg Review of Science and Technology 1.1 (2021), pp. 1–15.
10. Nashma, M., Zeebaree SRM., and Rashid, ZN. “Distributed Cloud Computing and Mobile Cloud Computing: A Review”. In: Qalaai Zanist Scientific Journal 7.2 (2022).
11. Mundinger, J., Weber, R., and Weiss, G. 2008. Optimal scheduling of peer-to-peer file dissemination. J. of Scheduling 11, 2 (April 2008), 105–120. https://doi.org/10.1007/s10951-007-0017-9.
12. Becker, S.m Schmidt, F., and Kao O., "EdgePier: P2P-based Container Image Distribution in Edge Computing Environments," 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, 2021, pp. 1-8, doi: 10.1109/IPCCC51483.2021.9679447.
13. A. Sherman, P. A. Lisiecki, A. Berkheimer, and J. Wein, “Acms: The akamai configuration management system,” in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2, pp. 245–258, 2005.
14. Kelsey Hightower, Brendan Burns, and Joe Beda. 2017. Kubernetes: Up and Running Dive into the Future of Infrastructure (1st. ed.). O'Reilly Media, Inc.