future internet

MDPI

Article

Dependable and Secure Voting Mechanism in Edge Computing

Pedro A.R.S. Costa * and Marko Beko

Copelabs, Informatics Department, Universidade Lusófona, Campo Grande 376, 1749-024 Lisbon, Portugal; beko.marko@ulusofona.pt

* Correspondence: pedrosacosta@ulusofona.pt

Received: 1 November 2019; Accepted: 12 December 2019; Published: 15 December 2019

Abstract: Edge computing is a distributed computing paradigm that encompasses data computing and storage and is performed close to the user, efficiently guaranteeing faster response time. This paradigm plays a pivotal role in the world of the Internet of Things (IoT). Moreover, the concept of the distributed edge cloud raises several interesting open issues, e.g., failure recovery and security. In this paper, we propose a system composed of edge nodes and multiple cloud instances, as well as a voting mechanism. The multi-cloud environment aims to perform centralized computations, and edge nodes behave as a middle layer between edge devices and the cloud. Moreover, we present a voting mechanism that leverages the edge network to validate the performed computation that occurred in the centralized environment.

Keywords: byzantine fault-tolerance; IoT; cloud computing

---

1. Introduction

Cloud computing has tremendously impacted society by allowing users to use infrastructure, platforms, and software provided by cloud providers at a low cost. Current services use multiple clouds to increase data availability and performance. Contrarily, the complexity of an application increases, along with the risks in security and privacy [1]. In conventional cloud computing, the process of uploading data to centralized servers creates significant pressure on the network, specifically in the data transmission costs of bandwidth and resources. The network performance worsens with increasing data size [2].

Edge computing is used as a method for optimizing cloud applications by taking a certain portion of data or services away from a central cloud to nodes closer to the end-users. While everyday devices take advantage of cloud computing, Internet of Things (IoT) manufacturers and application developers are only just starting to discover the benefits of doing more computing and analytics on the devices themselves [3]. For critical solutions, edge computing is used to reduce latency and for real-time computation, to better manage the massive deluge of data being generated by the IoT devices, and to lower the dependence on network connectivity into the cloud. For instance, the agriculture industry requires real-time monitoring of the soil properties to determine the soil quality, moisture, and type of crop production [4]. In support of this, all agriculture entities need to be connected to have a decision-making system that can help increase production and ease the distribution of agricultural products from farmers to marketing agencies and from vendors to farmers.

Yet, edge computing cannot defeat the benefits of using a centralized cloud, which includes better overall management of applications and data since they are centralized. Companies are always looking for the perfect cloud environment, which is secure and easy to configure, to overcome the various cloud limitations: Loss of availability and privacy, data corruption, and vendor lock-in. Using multiple clouds to replicate services can be viewed as a solution, but it does not come for free. Initially, using multiple clouds is more expensive in every sense than using one cloud. Even so, interoperability and portability are important factors to the success of this architecture [5]. By using multiple clouds, companies can reduce long-term costs, improve resilience and security, and avoid vendor lock-in.

Cloud computing can improve the quality of smart-city services, offer support to store, and analyze and extract knowledge from raw data [6]. In a recent vision of computing for smart environments, academics and experts published a blueprint to help build smart cities by adopting a secure hybrid cloud architecture [7]. Edge computing can support time-sensitive requirements of IoT applications and exploit the constrained resources of edge devices, while cloud-based programming models can strongly secure sensitive data, perform cloud backups for disaster recovery, and reduce costs in general.

A generic IoT network (see Figure 1) can be viewed as a three-tier network with the existing infrastructure networks (e.g., cloud and cellular networks). This architecture consists of 'cloud tier', 'edge tier', and 'device tier'. The device tier corresponds to the endpoints of the networks, where IoT devices act as data sources. Optionally, some IoT devices can be connected to edge gateways. The edge intelligence resides in the second tier: The edge computing layer. This layer comprises network devices, such as routers, gateways, and switches, that are capable of processing, computing, and temporarily storing the received information. The edge tier is composed of a group of edge nodes structured hierarchically and located between the cloud servers and the IoT devices. Edge nodes are devices capable of routing network traffic and usually also possess high computing power. They can range from base stations, routers, or switches to small-scale data centers. The cloud tier corresponds to cloud intelligence and is capable of storing and processing an enormous amount of data, depending on the capability of the data centers.

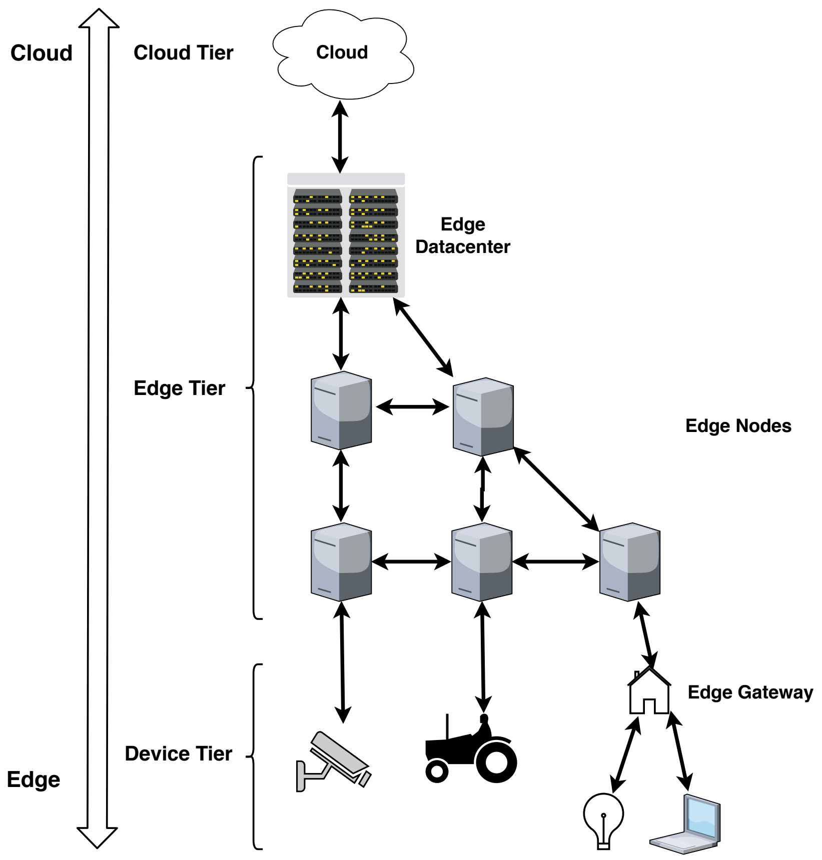

Figure 1. Generic three-tier edge network architecture.

The three-tier architecture retains the core advantage of using clouds as a support infrastructure and places the control and trust decisions on the edge devices. Other solutions have expanded this architecture to use more layers. These additional layers are designed to add business requirements to the traditional architecture [8,9]. The capabilities of a generic three-tier architecture satisfy the requirements for this solution.

Byzantine fault tolerance (BFT) relates to the dependability of computer systems, especially in distributed systems where components may fail arbitrarily and provide erroneous information. The problem of Byzantine generals was first proposed by Lamport et al. [10] and is still of significant interest in the present day [11]. Distributed computing that occurs in environments such as centralized clouds or edge nodes are prone to Byzantine failures. These are the most challenging class of failure modes, which imply no restrictions and make no assumptions about the kind of behavior of the system [10].

In today's networks, the data that pass through the edge network are frequently unattended and unsupervised. Sometimes edge nodes may fail, and there is imperfect information about whether a particular node has failed. The lack of fundamental security features exacerbates the privacy risks of edge devices and edge computing, and the decentralized management of a highly distributed environment can lead to security or privacy violations [3,12,13]. Consequently, authentication and access control mechanisms must be extended from current cloud deployments and customized for the edge ecosystem. The need to create methods to ensure 'correctness' in an unsafe environment has become a critical issue. There are risks associated with edge computing that are different from the threats that occur in a centralized environment. Ensuring edge computing security is much more challenging than providing cloud security because it involves distributed data processing. One step to improve safety is to use a set of Byzantine agreement protocols to detect data or computation corruption.

Byzantine failures also affect cloud services running in data centers. First, arbitrary faults are known to happen in the data center, corrupting the processing and affecting the 'correctness of the results' [14–16]. Second, malicious attacks perpetrated by cloud insiders or external hackers can also cause corruption of the processing and of its results [17–19]. Third, cloud outages may lead to the unavailability for cloud services. Experience shows that these events are frequent, with cases of unavailability of hours in services like Facebook or Cloudflare, to name just a few, that can impact the global traffic [20].

In this paper, we propose a voting scheme that works in a set of edge nodes located in the same geographical region that validates the computation that occurred in the multi-cloud environment. The novelty of this proposal arises from the combination of using multiple clouds and edge nodes to not only parallelize computation but also to tolerate Byzantine faults transparently. This solution can be useful for information-centric IoT applications that require the following requisites: (i) IoT data rely on the network for further computational analysis, (ii) it is necessary to ensure the correctness of the computation, and (iii) applications are not subject to reduced latency. The nature of most IoT applications tends to possess these characteristics because consumers desire to receive meaningful knowledge from the data.

Our solution addresses several non-trivial challenges toward this purpose in the edge environment. First, it aims to be transparent for the user or any IoT device. Second, this is a general framework that aims to relay computation from edge nodes into the cloud. Third, this framework uses a voting scheme that validates cloud computation in the edge network.

2. Related Work

Edge computing has been useful in supporting applications in several domains. Ha et al. used cloudlets to offload computing tasks for wearable cognitive assistance [21]. Chun et al. presented CloneCloud that enables mobile applications to leverage from the cloud [22]. Both solutions reduce energy consumption by offloading part of the execution onto the cloud.

Wang et al. proposed an in-network distributed processing for IoT data based on the widely deployed MapReduce framework called MR-IoT [23]. MR-IoT is built upon information-centric networking architecture to offer better network support for generic IoT applications. The whole network is responsible for deploying and executing MapReduce tasks based on the naming scheme. This solution is constrained to the computational power of the nodes in the network and security issues. MR-IoT is not practical for a consumer to send large volumes of data since it will overload the network. Additionally, MR-IoT does not concern itself with authentication, or privacy, ignoring the challenge of providing security to the network.

Few works address fault-tolerance in edge computing. In the work [24], researchers proposed a new fault-tolerant architecture for IoT applications that allows data to be computed in edge nodes and later sent to the cloud for further computation and storage, if necessary. This architecture consists of three layers—(i) application isolation, (ii) data transport, and (iii) multi-cluster management—to allow the placement of the computation on either the edge or the cloud. The authors considered surveillance systems where edge nodes can operate under hardware faults. Nevertheless, this solution tolerates crash faults by replicating data so that a fault in the edge or cloud will not disrupt the data processing.

EdgeCons [25] is the only known consensus protocol, with a Paxos-based approach, that achieves fast ordering in an edge network. EdgeCons runs a sequence of Paxos instances on the edge network, but it distributes the leadership of the Paxos instances based on the recently running history of the consensus process. The purpose of this protocol is to guarantee the ordering of concurrent requests by the edge network. This protocol achieves a fast event ordering for large-scale distributed applications in edge computing networks. Nonetheless, this protocol must assume that there is a cloud behind the edge network that never fails or becomes network-partitioned, which is unrealistic.

Blockchain systems use different consensus mechanisms to achieve necessary agreement on a single data value or a single state. Stanciu has investigated the use of blockchain as a platform for edge computing to execute critical distributed algorithms in a secure environment [26]. This ongoing project uses Hyperledger Fabric in the edge network to validate transactions and to connect the client to validating peers. Hyperlegder Fabric is an authorised blockchain platform, which means that, similar to our proposal, every identity is known. In terms of performance, there is a limitation regarding the load that can be processed in real time.

There could exist a debate between blockchain and BFT state machine replication. Vukolic compared PoW-based blockchain limitations with BFT state machine replication in terms of scalability and performance [27]. While BFT solutions are known for its poor scalability, in terms of performance, these protocols sustain thousands of transactions with practically equal network speeds. In contrast, blockchain performance is dependent on 'block size' and 'block frequency'. Noticing the blockchain limitations, Decker et al. proposed a new system called PeerCensus that acts as a certification authority and relies on blockchain to regulate entities joining the system and in Chain Agreement Protocol (CA) to commit transactions [28]. The system ensures that at any time t, $\frac{2}{3}$ of the nodes are in 'secure state', keeps track of system membership, resolves conflicts in case of a blockchain fork, and augments the system with strong consistency. Consequently, CA ensures that Byzantine fault-tolerance protocols such as PBFT [29] or Zyzzyva [30] can function accurately in the system.

3. System

This section begins by presenting the system architecture and then defines the system model by providing examples of possible attacks.

3.1. Architecture

The system is composed of a set of distributed processes that run in edge and cloud nodes. Each edge node has a service that deals with edge device requests. The processes in the edge nodes have the objective of receiving the client requests, the responses from the cloud instances, as well as performing a voting mechanism with all its peers to validate the output from the cloud instances. In this architecture, there will also exist cloud services waiting for requests from the edge nodes. The cloud instances aim to receive the requests from the edge nodes, perform the computation from the request, produce a cryptographic digest from the output of the computation, and send the digest to the edge node. Each edge node is connected to a single cloud instance, and a single edge device sends the request to a set of edge nodes. From the client request, each edge node can identify each peer that the client also submits the request. All the entities in the system are connected by reliable channels, so no messages are lost, duplicated, or delivered out of order (e.g., TCP). Data integrity and privacy are maintained through cryptographic hash functions (e.g., SHA-512) and public-key cryptography (e.g., TLS).

A cryptographic hash function is applied to the output of the computation performed in the cloud. To prevent any disclosure of the computation, only the produced message digest, or digest, is sent to the edge nodes. All messages that circulate between each party are encrypted with a session key to preventing eavesdropping and authenticated to guarantee that each party correctly belongs to the system.

Figure 2 illustrates the multi-layer architecture composed by a cluster of edge nodes E (e.g., e_0, e_1, and e_2) that are in the same geographical region and are connected to a set of cloud instances C (e.g., c_0, c_1, and c_2) through the Internet Backbone. Contrary to the edge nodes, there is no requirement regarding the geographical distribution of cloud instances. When the client submits an operation σ, the operation is replicated to the nearest cluster E and forwarded to C.

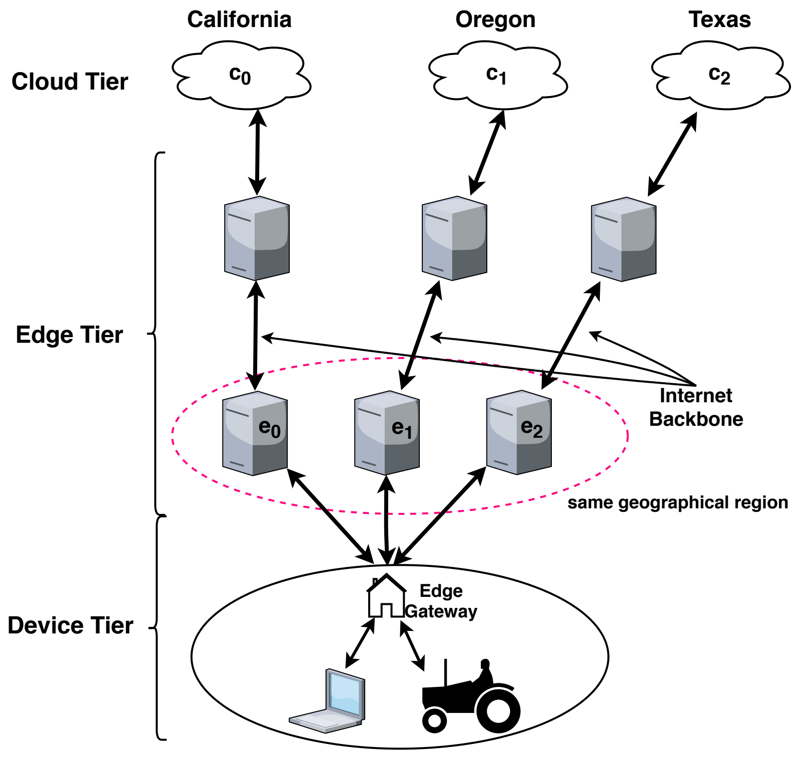

Figure 2. Edge network architecture with multiple cloud instances.

The result of the operation from each cloud V_σ, which comes in the form of a message digest, $H(V_\sigma)$, is sent to E and validated by all $e_i : i \in E$. It is assumed that V_σ is correct if E obtained at least $f + 1$ equal $H(V_\sigma)$.

Our system is configured with parameter f. In distributed systems, f is usually the maximum number of faulty results. Given an output $v_i \in V_\sigma$, our system can tolerate not only f faults but also any number of faulty replicas as long as no more than f faulty replicas return the same wrong output. It includes the possibility that up to f replicas maliciously collude and the system remains able to reach a correct output.

Consider a system of n edge nodes in an all-to-all message-passing network, $n = 2f + 1$. Each edge node ($e_i : i = \{0, 1, ..., n-1\}$) is connected to a single cloud instance ($c_i : i = \{0, 1, ..., n-1\}$). Of these

$2n$ nodes, which includes n edge nodes and n cloud instances, our system is correct if the edge nodes can still vote for the correct value v whether there are f faulty edge nodes, f faulty cloud instances, or any combination of both.

Initially, this solution seems expensive in terms of the number of resources needed. Each cloud instance is a set of several nodes with the goal to execute computer-intensive operations. The edge nodes are responsible for the voting mechanism operation, leaving the computation of complex operations to the cloud instances. While we use $2f + 1$ cloud instances with variable size to get the benefits of using a multi-cloud environment, it is necessary to add the $2f + 1$ edge nodes to this solution.

Byzantine fault tolerant systems are considered by the research community to be state-of-the-art with regard to providing reliability in distributed systems. A typical BFT system consists of $n = 3f + 1$ replicas that each provide a finite state machine and execute operations from clients in the same order [29]. This is an expensive solution, but it guarantees safety and liveness properties in harsh environments.

Cloud outages are common, and adapting services to use multiple clouds have been the solution to eliminate reliance on any single cloud provider and improve reliability [31]. Using multiple cloud instances to run services is even more expensive, but it has been shown that it could be a good strategy to tolerate BFT and cloud outages [32]. Similar to the mentioned work, this solution uses several cloud instances for the same purpose; however, we add the edges nodes to this architecture to validate the computation performed in the centralized environment. We can consider the cost of adding edge nodes to the solution as negligible when setting up a multiple cloud environment.

In terms of communication cost, traditional BFT algorithms require n^2 messages exchanged in nice executions, i.e., when there are no failures, and the system is synchronous enough for the leader not to be changed [29] and require five communication steps [33]. Our solution requires fewer communication steps comparing with other BFT algorithms. Moreover, the client is not responsible for the computation in our solution, which contrasts with [32]. The parts responsible for the system execution are the edge network and the cloud instances.

3.2. System Model

This solution does not need a leader to propose values as with other BFT algorithms [30,34]. Each cloud instance is connected to a single edge node. Each edge node accepts the proposed value from the correct cloud instance. We detail the algorithm from the client and edge node perspective.

3.2.1. Algorithm

The execution is replicated in $2f + 1$ cloud instances for ensuring the existence of $f + 1$ identical outputs even in the presence of arbitrary or malicious faults. Each cloud instance proposes a value v to the particular edge node. Each edge node begins the voting mechanism. The algorithm must enforce a set of properties to ensure correct operation:

1. **Voting:** All correct processes must terminate with single value v.
2. **Agreement:** All correct processes must choose the same value v.
3. **Termination:** Processing or communication delays are bounded to a value Δ.

The first two properties are concerning safety. The algorithm must guarantee that no correct process should ever make a decision that is contrary to what the remaining correct processes have chosen. The third property is concerning liveness.

The algorithm that is depicted in the Figure 3 is composed of three phases—PROPOSE, PREPARE, COMMIT. In this example, we are considering $f = 1$, which means that three edge nodes and three cloud instances are necessary. In the next paragraph, we detail the execution of the system from the client and server perspectives.

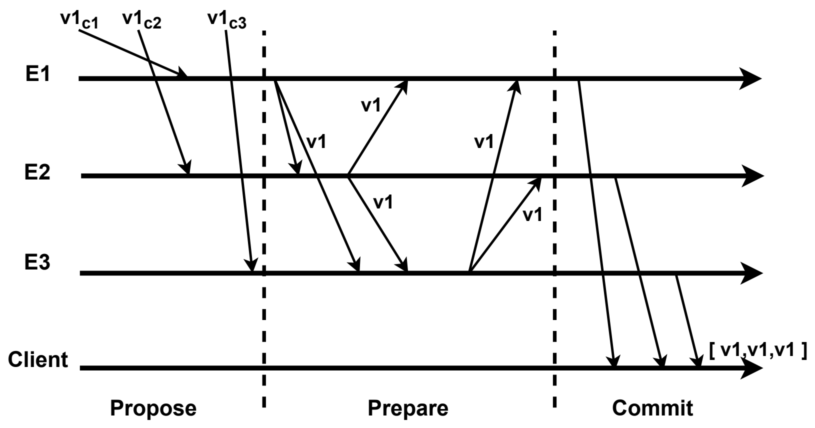

Figure 3. Correct execution of voting mechanism.

3.2.2. Client

A client c sends a request of the execution to a set of $2f+1$ edge nodes by sending a message $\langle \text{REQUEST}, seq, \sigma \rangle_{pc}$. Requests are signed with session keys pc created for the operation. Requests with invalid signature are ignored. seq is the request identifier that is used to ensure that the REQUEST will only compute once. σ refers to the operations performed in the cloud instances.

The client expects to receive $\langle seq, H(V_\sigma) \rangle$ from the edge nodes. $H(V_\sigma)$ is a message digest from the outcome of the computation σ, V_σ. The client knows that the computation ended successfully, if it receives at least $f+1$ equal $H(V_\sigma)$.

3.2.3. Edge Node

The client request is forwarded to the respective cloud instance with the message $\langle \text{REQUEST}, seq, \sigma, e_i \rangle_{pk}$. e_i identifies the edge node that forwarded the request and that will execute the voting mechanism. Requests are signed with session keys pk created for the operation.

The output of the computation performed in the cloud c_i comes in the form of $\langle c_i, H(V_\sigma \rangle_{pk}$. $c_i \in C$ identifies the cloud that performed the computation. Lets refer to $v_i = \langle c_i, H(V_\sigma \rangle_{pk}$ as the result of the computation in cloud instance c_i.

The voting algorithm comprises three phases as depicted in Figure 3, which begin from the moment the first reply is received by an edge node e_i from the cloud instance c_i. First, in the PROPOSE phase, each edge node will receive $\langle \text{PROPOSE}, seq, v \rangle$ that represents the result from the computation. PROPOSE is the type of message sent from the c_i to e_i. seq is the initial sequence number. v_i is the proposed value that resulted from the computation σ in cloud c_i.

After an edge node e_i receives v_i, it forwards during the PREPARE phase $\langle \text{PREPARE}, seq, v_i, e_i \rangle$ to $2f$ peers. v_i is the proposed value from c_i. e_i is the edge node that received the value. The COMMIT phase begins when all the values have been broadcast.

In the COMMIT phase, the edge nodes apply a simple majority scheme to find a single value v that identifies the computation. Finally, each node will relay the result to the client with the message $\langle \text{RESPONSE}, seq, v, e_i \rangle_{pc}$.

In the simple use case, the sequence of events on the server-side is the following: (1) Each cloud instance sends the result of the computation to the respective edge node; (2) after all edge nodes receive the proposed value v, it is propagated to every other peer; (3) each edge node will apply a voting scheme for value v; (4) each edge node will send the voted value to the client; (5) the client c waits for $f+1$ matching replies with the content $\langle \text{RESPONSE}, seq, v, e_i \rangle_{pc}$ to know that the computation was performed successfully.

From the client perspective, it receives a message from each e_i. If the client finds a majority of equal results, it knows that the computation performed successfully.

There is no synchronization between edge nodes. In other words, each edge node can be in different phases at the same time, or have a Byzantine behavior. With this algorithm, the following scenarios are possible: (i) One node already finished the PREPARE phase, but the other peers did not complete the PROPOSE phase yet. The only edge node that has already propagated its value cannot continue execution because it still does not have enough values to vote; (ii) e_i cannot wait indefinitely if it did not receive v_i from a c_i, or from its peers. Thus, it is necessary to implement a simple timeout mechanism to guarantee liveness [29].

By using the following inequality in $|Tc_j + Tk_j - Tc_i + Tk_i| < \Delta$, the system calculates the parameter Δ that binds processing and communication delays. Tc refers to the processing time, and Tk is the latency of a message. After $e_i \in E$ receives v_i from $c_i \in C$, the system expects that any $e_j \in E$ receives a value within a Δ interval. Namely, it is expected that the time difference between the first process to finish, $Tc_i + Tk_i$, and any other process j, $Tc_j + Tk_j$ is less than Δ. If e_j does not receive any message after Δ time, it is considered that e_j failed.

We model an attacker from the Byzantine failure model. The cloud and edge nodes are prone to arbitrary and malicious faults. The cloud outages are considered by the system as an 'omission failure'. In Figure 4, we depict a scenario where e_1 is compromised by a malicious insider and tampers the value v_1. In this case, e_1 distributes the wrong value v_2 to the remaining nodes. Nevertheless, the remaining $2f$ nodes can still vote correctly and send the correct result to the client.

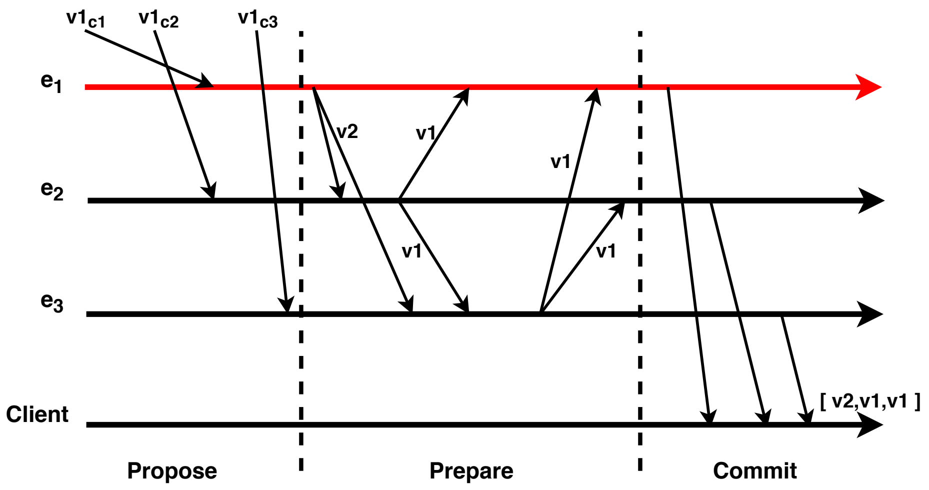

Figure 4. Execution with one fault.

In Figure 5, we depict another example where the system suffered two faults: c_2 suffered an arbitrary fault that altered the outcome to v_3, and e_1 is compromised by a malicious attacker. c_1 and c_3 are the only clouds that proposed the correct value, and the e_2 and e_3 nodes are in 'correct' state.

In the PROPOSE phase, a set of mixed values is distributed to the edge nodes, and it will be propagated to all edge nodes in the PREPARE phase. The nodes e_1 and e_2 have the set of values $\{v_1, v_2, v_3\}$, and the node e_3 has the set $\{v_1, \emptyset, v_3\}$. The $\emptyset$ in e_3 is a consequence of e_1 not sending the value v_2. Consequently, in the COMMIT phase, the simple voting scheme cannot find the correct value, and $\emptyset$ will be sent to the client. In this case, the system suffered two faults which are higher than what was configured initially, $f = 1$. If we want to tolerate $f = 2$, five edge nodes and five cloud instances are necessary.

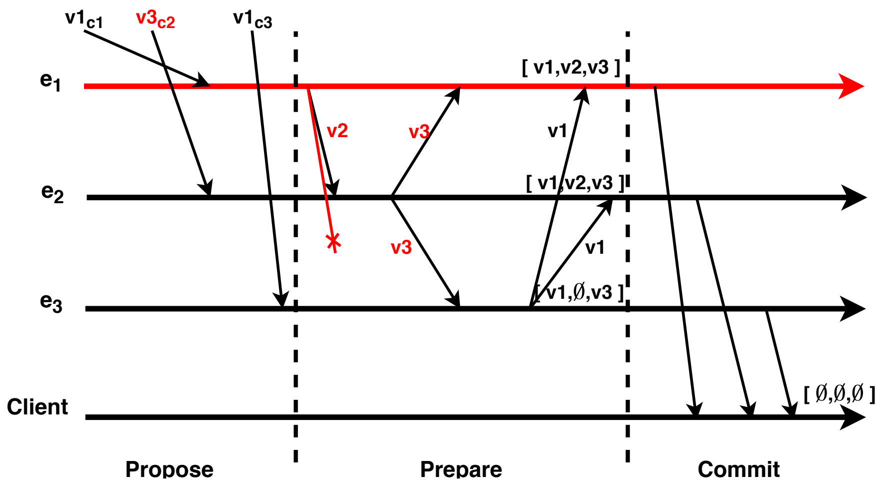

Figure 5. Execution with two faults.

4. Conclusions

In this paper, we presented a voting scheme that validates computation in a multi-cloud environment by leveraging edge-computing technology. The proposed solution uses edge nodes for two purposes. First, to forward operations initiated by the client to a set of cloud instances. Second, to validate the computation performed in the multi-cloud environment. Information-centric IoT applications that rely on the network for further computational analysis can benefit from this solution because the system guarantees dependability in the computation, even in the presence of Byzantine faults. In future work, we intend to compare our methods with related work to understand how latency can affect the functioning of the system.

Author Contributions: Conceptualization, P.A.R.S.C.; Methodology, P.A.R.S.C.; Investigation, P.A.R.S.C. and M.B.; Writing—Original Draft, P.A.R.S.C. and M.B.; Writing—Review & Editing, P.A.R.S.C. and M.B.; Funding Acquisition, M.B.

Funding: This research was partially supported by Fundação para a Ciência e a Tecnologia under Projects UID/MULTI/04111/2020, UID/EEA/00066/2020, foRESTER PCIF/SSI/0102/2017 and IF/00325/2015.

Acknowledgments: The authors would like to thank Marko Vukolic of IBM Zurich for sharing his insight and answering all our queries.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Singh, S.; Jeong, Y.S.; Park, J.H. A Survey on Cloud Computing Security. *J. Netw. Comput. Appl.* **2016**, *75*, 200–222. [CrossRef]
2. Persico, V.; Botta, A.; Montieri, A.; Pescape, A. A First Look at Public-Cloud Inter-Datacenter Network Performance. In Proceedings of the IEEE Global Communications Conference, Washington, DC, USA, 4–8 December 2016; pp. 1–7.
3. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
4. Solutions, I. IoT Transforming the Future of Agriculture. 2019. Available online: https://www.iotsworldcongress.com/iot-transforming-the-future-of-agriculture/ (accessed on 13 December 2019).
5. Rani, B.K.; Rani, B.P.; Babu, A.V. Cloud Computing and Inter-Clouds—Types, Topologies and Research Issues. *Procedia Comput. Sci.* **2015**, *50*, 24–29. [CrossRef]
6. Horwitz, L. Can Smart City Infrastructure Alleviate the Strain of City Growth? Available online: https://www.cisco.com/c/en/us/solutions/internet-of-things/smart-city-infrastructure.html (accessed on 13 December 2019).
7. Zhang, K.; Ni, J.; Yang, K.; Liang, X.; Ren, J.; Shen, X. Security and Privacy in Smart City Applications: Challenges and Solutions. *IEEE Commun. Mag.* **2017**, *55*, 122–129. [CrossRef]

8. Tang, B.; Chen, Z.; Hefferman, G.; Wei, T.; He, H.; Yang, Q. A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities. In Proceedings of the ASE Big Data & Social Informatics 2015, Kaohsiung, Taiwan, 7–9 October 2015.
9. Ogino, T.; Kitagami, S.; Shiratori, N. A multi-agent based flexible IoT edge computing architecture and application to ITS. *Int. J. Netw. Comput.* **2019**, *14*, 47–52. [CrossRef]
10. Lamport, L.; Shostak, R.; Pease, M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* **1982**, *4*, 382–401. [CrossRef]
11. Sousa, J.; Bessani, A.; Vukolic, M. A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg, 25–28 June 2018; pp. 51–58. [CrossRef]
12. Infosec Institute. Security Issues in Edge Computing and the IoT. Available online: https://resources.infosecinstitute.com/security-issues-in-edge-computing-and-the-iot (accessed on 23 August 2018).
13. Roman, R.; Lopez, J.; Mambo, M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]
14. Meza, J.; Wu, Q.; Kumar, S.; Mutlu, O. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In Proceedings of the IEEE/IFIP 45th International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015; pp. 415–426.
15. Nightingale, E.B.; Douceur, J.R.; Orgovan, V. Cycles, Cells and Platters: An Empirical Analysisof Hardware Failures on a Million Consumer PCs. In Proceedings of the 6th Conference on Computer Systems (EuroSys), Salzburg, Austria, 10–13 April 2011; pp. 343–356. [CrossRef]
16. Schroeder, B.; Pinheiro, E.; Weber, W.D. DRAM Errors in the Wild: A Large-scale Field Study. In Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems, Seattle, WA, USA, 15–19 June 2009; pp. 193–204. [CrossRef]
17. Cloud Security Alliance. Cloud Security Alliance Warns Providers of 'The Notorious Nine' Cloud Computing Top Threats in 2013. Available online: https://cloudsecurityalliance.org/articles/ca-warns-providers-of-the-notorious-nine-cloud-computing-top-threats-in-2013/ (accessed on 25 February 2013).
18. Chen, A. GCreep: Google Engineer Stalked Teens, Spied on Chats. Available online: http://gawker.com/5637234/gcreep-google-engineer-stalked-teens-spied-on-chats (accessed on 13 December 2019).
19. Kandias, M.; Virvilis, N.; Gritzalis, D. The Insider Threat in Cloud Computing. In *Critical Information Infrastructure Security*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 6983, pp. 93–103. [CrossRef]
20. Whittaker, Z. It Was a Really Bad Month for the Internet. Available online: https://techcrunch.com/2019/07/05/bad-month-for-the-internet/ (accessed on 13 December 2019).
21. Ha, K.; Chen, Z.; Hu, W.; Richter, W.; Pillai, P.; Satyanarayanan, M. Towards Wearable Cognitive Assistance. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, Bretton Woods, NH, USA, 16–19 June 2014; pp. 68–81.
22. Chun, B.G.; Ihm, S.; Maniatis, P.; Naik, M.; Patti, A. CloneCloud: Elastic execution between mobile device and cloud. In Proceedings of the Sixth Conference on Computer Systems, Salzburg, Austria, 10–13 April 2011.
23. Wang, Q.; Lee, B.; Murray, N.; Qiao, Y. MR-IoT: An information centric MapReduce framework for IoT. In Proceedings of the 15th IEEE Annual Consumer Communications Networking Conference, Las Vegas, NV, USA, 12–15 January 2018; pp. 1–6.
24. Javed, A.; Heljanko, K.; Buda, A.; Främling, K. CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 813–818.
25. Hao, Z.; Yi, S.; Li, Q. EdgeCons: Achieving Efficient Consensus in Edge Computing Networks. In Proceedings of the USENIX Workshop on Hot Topics in Edge Computing, Boston, MA, USA, 10 July 2018.
26. Stanciu, A. Blockchain Based Distributed Control System for Edge Computing. In Proceedings of the 21st International Conference on Control Systems and Computer Science, Bucharest, Romania, 29–31 May 2017; pp. 667–671. [CrossRef]
27. Vukolić, M. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In Proceedings of the International Workshop on Open Problems in Network Security, Zurich, Switzerland, 29 October 2015; Volume LNCS-9591, pp. 112–125.
28. Decker, C.; Seidel, J.; Wattenhofer, R. Bitcoin Meets Strong Consistency. In Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, 4–7 January 2016; pp. 1–10.

29. Castro, M.; Liskov, B. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* **2002**, *20*, 398–461. [CrossRef]
30. Kotla, R.; Alvisi, L.; Dahlin, M.; Clement, A.; Wong, E. Zyzzyva: Speculative Byzantine Fault Tolerance. In Proceedings of the Twenty-First ACM Symposium on Operating Systems Principles, Stevenson, WA, USA, 14–17 October 2007; pp. 45–58.
31. Rouse, M. Multi-Cloud Strategy. Available online: https://searchcloudcomputing.techtarget.com/definition/multi-cloud-strategy (accessed on 13 December 2019).
32. Costa, P.A.; Ramos, F.M.; Correia, M. On the Design of Resilient Multicloud MapReduce. *IEEE Cloud Comput.* **2018**, *4*, 74–82. [CrossRef]
33. Verissimo, P.; Lung, L.C.; Bessani, A.N.; Correia, M.; Veronese, G.S. Efficient Byzantine Fault-Tolerance. *IEEE Trans. Comput.* **2013**, *62*, 16–30.
34. Bessani, A.; Sousa, J.; Alchieri, E.E.P. State Machine Replication for the Masses with BFT-SMART. In Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, GA, USA, 23–26 June 2014; pp. 355–362.