Monitoring Resilience in a Rook-managed Containerized Cloud Storage System

Louis Baumann, Stefan Benz
Abraxas, St.Gallen, Switzerland
Email: louis.baumann@abraxas.ch, stefan.benz@abraxas.ch

Leonardo Militano, Thomas Michael Bohnert
Zurich University of Applied Sciences, Switzerland
Email: leonardo.militano@zhaw.ch, thomas.bohnert@zhaw.ch

***Abstract*—Distributed cloud storage solutions are currently gaining high momentum in industry and academia. The enterprise data volume growth and the recent tendency to move as much as possible data to the cloud is strongly stimulating the storage market growth. In this context, and as a main requirement for cloud native applications, it is of utmost importance to guarantee *resilience* of the deployed applications and the infrastructure. Indeed, with failures frequently occurring, a storage system should quickly recover to guarantee service availability. In this paper, we focus on containerized cloud storage, proposing a resilience monitoring solution for the recently developed *Rook* storage operator. While, Rook brings storage systems into a cloud-native container platform, in this paper we design an additional module to monitor and evaluate the resilience of the Rook-based system. Our proposed module is validated in a production environment, with software components generating a constant load and a controlled removal of system elements to evaluate the self-healing capability of the storage system. Failure recovery time revealed to be 41 and 142 seconds on average for a 32GB and a 215GB object storage device respectively.**

***Keywords*—Distributed Cloud Storage, Resilience, Monitoring, Ceph, Rook, Kubernetes, Cloud-native**

I. Introduction

The interest for distributed cloud storage is growing at a very fast pace as witnessed by its numbers in the worldwide market. The global storage market has an annual growth of 25.8% and it is predicted to reach $74.94 billion of value in 2021 [1]. Companies and end-users interested in moving their storage to the cloud are growing day by day. As a consequence, the digital universe is expected to reach the impressive number of 40,000 exabytes in 2020 with about 40% of the digital data being stored or processed in a Cloud somewhere in its life journey [2]. The observed evolutionary trend towards cloud storage is also strongly motivated by the rapid volume growth of enterprise data and unstructured data [3]. This would have the consequence of highly increased investment and maintenance costs when local storage is adopted. If on the one hand storing data on the cloud comes with a lot of advantages for the end-users, on the other hand it also introduces new challenges for cloud storage providers.

With companies storing all or part of their data in the cloud, it becomes of utmost importance for cloud providers to guarantee service continuity [4]. From the cloud providers perspective, any failure would correspond to important revenue losses. As an example, losses were estimated at $273 million in 2007-2013, for 28 cloud service providers, given 1,600 hours of disruptions [5]. Several metrics and performance indicators can be used to measure how healthy the system is. Among the most often talked about metrics in the cloud storage industry is the concept of *data durability*, with nearly every provider quoting some number of nines for this[1]. If guaranteeing that the data is not lost is important, key requirements for cloud native applications (CNA) running on top of a cloud computing infrastructure services are the inherent support for self-management, scalability and *resilience* [8]. Resilience can be defined as the ability of a server, a storage system or an entire data center to recover and to get back to an operational status after that a failure occurs. Resilience is particularly in focus for this paper as it is a major challenge to support the massive migration of business services and storage to the cloud [7]. One motivation for the high focus on resilience is that hardware or software failures in the cloud elements may cause severe effects on business revenues, even when the failure has a short duration in number of hours [5]. As reported in [7], a viable approach to guarantee or at least improve resilience is to *forecast*, when possible, the threatening situations that may cause a failure. Based on measurements and/or estimations, a failure threat can be identified so that countermeasures can be applied before the failure actually occurs. Moreover, it is a requirement for cloud native applications to anticipate failures and fluctuations in quality of both cloud resources and third-party services for an application [8].

Based on the above considerations, in this paper we design a failure monitoring solution for a containerized distributed cloud storage solution based on the Rook project[2]. In particular, we adopt a Ceph distributed storage implemented using Rook acting as a Kubernetes operator. The objective of our work is to implement a methodology to monitor and prove the containerized storage system to be *resilient*. There are different levels of failure, e.g., software failure, equipment failure, power outage, and all of them have to considered, for a system to be called resilient. We propose a solution to monitor the storage cluster health and the infrastructure metrics that may indicate a reduced performance and a threat of failure. An alerting system is also designed, to inform the system administrator about health warning or error status in case of system misbehaviour or failures. The proposal has been validated in the production environment running at the company Abraxas Informatik AG[3], where a Ceph [13] storage cluster is implemented using Rook on Kubernetes [14].

[1]https://www.backblaze.com/blog/cloud-storage-durability/
[2]https://rook.io/
[3]https://www.abraxas.ch/

Through selected test scenarios we showcase whether and to which extent the system is resilient. To this scope software modules to provide controlled load and removal of components are designed to verify whether the service for the end-user is interfered. To the best of our knowledge, the proposed solution is the first implementation of this kind for resilience monitoring service in a containerized cloud storage system.

The remainder of the paper is organized as follows. In Section II we browse the related work w.r.t. distributed storage in the cloud, whereas in Section III we briefly report on the relevant projects for this paper. In Section IV the problem and the proposed solution is described. The validation of the implemented proposal is reported in Section V, whereas concluding remarks are given in Section VI.

II. Related Work

Resilience is identified in [8] as the first goal to be attained in order to achieve a functioning and available cloud native application. This goal is typically reached with several strategies and by using redundant resources being pursued on different levels. However, resilience goes well beyond the borders of cloud native applications. For instance, in [15] the focus is on the carrier cloud and the networking aspects of 5G mobile networks in need of high availability and system reliability. Focus on distributed storage systems is in [10] where the authors propose an approach to replica management to ensure data availability and durability. Accounting for the availability history of nodes a solution to improve replica placement and repair is proposed, while keeping an eye on the trade-off between data availability, load-balancing and bandwidth consumption. In a similar context, the authors in [11] analyze replication strategies for storage systems aggregating disks of nodes spread across the Internet. In [12] instead, an analysis is proposed on the Self-Monitoring, Analysis and Reporting Technology (SMART) that modern hard disk drives support. The basic monitoring of internal attributes for the drives to predict impending failures is improved adopting neural network models to better predict drive failures.

More related to the focus of our paper, a survey on categories and techniques for cloud computing infrastructure resilience can be found in [7]. As the paper highlights, disruptions due to failures see three major components in the cloud architecture as the origin: the servers hosting the application, the network interconnecting them, or the the application itself. With the advent of containerized solutions in the cloud, an additional component has to be added to the mentioned three components in [7], namely a container. Actually in a Kubernetes environment this is more precisely a pod, i.e., a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers. Indeed, recent cloud environments adopt container-based virtualization using Docker for container packaging and Kubernetes for multihost Docker container management. In such a container-based environment, it is important that Kubernetes can dynamically monitor the resource requirements and/or usage of the running applications, and then accordingly adjust the resource provisioned to the managed containers. Accurately predicting a failure is a problem in itself and much work in this area when it comes to cloud systems, e.g. [9]. The authors show that it is not easy to even find the right indicators for accurately predicting that something is going wrong in the system.

As the authors in [8] state, cloud applications should be continuously monitored to achieve resilience. Application-specific and infrastructural metrics should be monitored to provide automated and responsive reactions to failures, minimizing human intervention. To this aim, either services from the infrastructure provider or a third-party service can be adopted. Both of these options would lead to vendor lock-in and are also paid services. Ad-hoc built solutions may solve these issues, but come at the cost of engineering work. Therefore, in [8] it is stated that monitoring, health management, and scaling features should be developed within the managed cloud native applications itself, so that they naturally adapt to the dynamic nature of the application. Based on these observations, for the scope of our research we adopt open-source software that helps avoiding vendor lock-in issues and additional costs.

Currently, Kubernetes provides a naive dynamic resource-provisioning mechanism which only considers CPU utilization and thus is not effective. In [16] a generic platform to facilitate dynamic resource-provisioning based on Kubernetes is proposed where the solution relies on the monitoring of the system resources. The monitoring solution we propose in this paper goes beyond the built-in mechanism provided in Kubernetes taking into account both system resource utilization and quality of storage metrics.

III. Overview of Adopted Technologies

Kubernetes is a container orchestration and management platform designed by Google [14]. It is under open source license and is the foundation of many popular PaaS solutions such as Openshift, CloudFoundry, Tectonic or Juju. Kubernetes follows the controller-worker model, where a controller manages Docker containers across multiple Kubernetes workers. The controller and its controlled nodes constitute a cluster. Kubernetes offers solutions for automated provisioning and scaling of containers, and also orchestrating computing, networking, and storage infrastructure for user workloads. The containers are isolated from each other and through the consolidation on operating-system-level, the overhead produced by a hypervisor falls away. Kubernetes supports the microservice approach in the software development and every component of a software can be managed and scaled. Typically, an application is divided into one or more tasks executed in one or more containers. Each container is generally restricted in terms of the maximum resource quantity that it can consume.

Rook is an open source cloud-native storage framework designed to manage storage solutions, and is natively integrated with cloud-native environments. Since 2018 the Cloud Native Computing Foundation (CNCF) hosts Rook as the first project in the cloud-native storage category. The main objective of Rook is to bring File, Block and Object storage systems into the Kubernetes cluster, where other applications and services may run that are actually consuming the storage. It runs as a Kubernetes operator, which makes storage software a self-managing, self-scaling, and self-healing service using Kubernetes primitives. Kubernetes applications can mount block devices and file-systems, use the S3/Swift API for object storage managed by the Rook operator. The operator is running as a container, which automates configuration and monitors the cluster to ensure the storage remains available and healthy.

Instead of building a new storage system, Rook focuses on turning existing battle-tested storage systems into cloud-native services on-top of Kubernetes. The initial efforts were put on Ceph, with the Ceph Custom Resource Definitions (CRDs) recently promoted to Beta version in the Rook v0.8 release. However, also other storage systems have been integrated in Rook (like Minio object storage, CockroachDB, Cassandra, Network File System) and others are expected to come in the near future. Noteworthy, the Rook deployment includes several additional components that support the monitoring of the cluster, like the Prometheus exporter and the Grafana dashboard. These allow to display the status of the Rook cluster and all the storage components which definitely come in handy in the design of our proposed monitoring solution for resilience monitoring.

Ceph is a unified, distributed storage system that offers high performance, reliability, and scalability. It promotes self-managing, self-healing and no single point of failure for block, file and object storage [13]. The concept Ceph promotes is to support primarily consistency and partition tolerance over availability (see [6] for more details on the related CAP theorem notions). This practically means that for every data write operation an acknowledgment is sent to the client only after all the replicas are correctly written. A key element of Ceph is the implementation of a pseudo-random data distribution function (CRUSH) to determine where and how to store the data on the storage nodes. The Ceph storage cluster is made up of several software daemons taking care of unique Ceph functions. The main components are:

- **Reliable Autonomic Distributed Object Store (RADOS):** responsible for storing data in the Ceph cluster in the form of variably sized objects;

- **Object Storage Devices (OSDs):** store user data in form of objects with OSD Daemons handling the read/write operations on the storage disks;

- **Monitors (MONs):** track the health of the entire cluster by keeping a map of the cluster state. All the cluster nodes report to monitor nodes and share information about every change in their state;

- **Librados:** library to get access to RADOS, providing a native interface to the Ceph storage cluster, and a base for other services such as RBD, RGW, as well as the POSIX interface for Ceph file system;

- **Ceph** (previously Rados) **Block Device (RBD):** provides block storage, which can be mapped, formatted, and mounted just like any other disk to the server;

- **Ceph Metadata Server (MDS):** keeps track of file hierarchy and stores metadata only for CephFS;

- **Ceph File System (CephFS):** offers a POSIX-compliant, distributed file system;

- **Rados Gateway (RGW):** offers object storage on top of the Ceph storage and implements an S3/Swift interface;

- **Manager (MGR):** provides further monitoring features and an interface for external monitoring and management.

IV. Monitoring Cloud Storage for Resiliency

As introduced in Section I, *resilience* is of utmost importance in distributed cloud storage to guarantee data durability and service availability. However, proving a service to be resilient for every combination of software, service and platform is not trivial. Nonetheless, a well-designed *monitoring* of the provided cluster infrastructure can be of great support. We will focus our attention on the *Rook* framework with a Ceph implementation as a Kubernetes operator. We build our monitoring solution upon the following three key elements:

1) **Selection and monitoring of key metrics:** We identify a set of parameters, metrics and sampling intervals that best fit to the scope of monitoring the status of the different components in the deployed Rook cluster and the underlying infrastructure;
2) **Controlled system load and removal of components:** To best evaluate the resilience of a system, a controlled load needs to be provided. In particular, with a constant load a controlled removal of components can be designed to evaluate the time to react and recover from a failure in the system;
3) **Health status forecasting and alerting:** Provide means to evaluate whether the functionality of the service is not interfered with the removal of a component and whether a noticeable change in feel occurs in using the service for the end-user. An alerting service will inform the system administrator in order to react to system misbehaviour or threatening conditions for the infrastructure.

Algorithm 1 Cluster health status control

```
 1: procedure VerifyClusterStatus(M,W,E)    ▷ M vector of metrics, W vector of warning thresholds, E vector of error thresholds
 2:     status = 'healthy'
 3:     i=0
 4:     while (status≠'error')&(i<length(M)) do
 5:         if ($M_i > W_i$) then
 6:             status = 'warning'
 7:             send warning message about $M_i$
 8:         end if
 9:         if ($M_i > E_i$) then
10:             status = 'error'
11:             send error message about $M_i$
12:         end if
13:         i++
14:     end while
15:     return status
16: end procedure
```

As it concerns the parameters and metrics of interest for our solution, these are derived both from the Ceph cluster and from the infrastructure itself. A Ceph cluster can raise several health messages, but we limit our interest to the subset that best meet our objectives. For a detailed description of the Ceph health messages please refer to the official Ceph documentation [13]. The status of the Ceph cluster can change between three different possible states, i.e., *Healthy*, *Warning* and *Error*, and is determined by Ceph itself as a combination of the health status of all the components. When the system is not in a healthy state, an action from an administrator or from the system itself is necessary to recover.

Besides the Ceph health messages, we consider additional metrics for the underlying infrastructure the Rook cluster is running on. Some of the selected metrics are already calculated by the integrated Grafana dashboard, other metrics instead, have to be calculated separately. The additional metrics we consider are the following: *OSD Commit Latency, OSD Apply Latency, IOPS, CPU Usage, Cores usage, Network usage, Disk IOPS usage, Available storage, OSD down*, *OSD orphan, Monitor down, PG undersized*, *PG stale*, *PG degraded*. A weighted function of these parameters determines the infrastructure health status which in combination with the Ceph health state, determines the final health state for the storage cluster. Whenever a metric does not meet the constraints set for a warning or an error message to be sent, an alerting message is sent to the administrator (see Algorithm 1).

The overall health states are displayed in the Grafana dashboard, with three different health levels for each metric. For monitoring purposes and to produce a constant load in the system we adopted a load generator based on *fio*, which is a highly flexible tool for bench-marking I/O. All the metrics are monitored and measured in terms of number of actions per second. Consequently, the adopted sampling intervals are less or equal to one second. Based on an initial health status, we get a good overview of the cluster before running any further tests. In particular, to run a heavy load test, the cluster should be in healthy state, because the tests should not interfere with the functionality of an in production storage cluster. Therefore, a first step of our system resilience analysis is to verify the status of the Ceph cluster and the Kubernetes infrastructure, as described in the flow diagram in Fig. 1. Only if the status is HEALTHY we can proceed with further tests as wished. In the latter case, whenever any of the considered metrics does not meet the health constraints as defined in Table I, an alert message is sent to the system administrator using a web-hook in order to take the required countermeasures. What the exact countermeasures to be taken are, is not in the scope of this paper. Our scope is indeed to provide the required tools to demonstrate resilience and the self-healing capability of the Rook-managed Ceph cluster, and at the same time, quickly report to the system administrator whenever a threat or misbehavior is recognized so that actions can be timely taken.

To demonstrate the resilience of the Ceph cluster, the strategy we follow is to showcase that the loss of instances does not interfere with its functionality. To this aim we define a software element, called *monkey* as part of the implemented solution, that randomly destroys a running cluster component during its execution. The set of components out of which to select are a Rook operator, a Rook agent, an OSD and a Ceph monitor. As we see from the values reported in Table I the loss of one OSD instance generates a Yellow state (which is acceptable), but as soon as two OSD instances fail, the cluster will change to error state. The same applies for the Ceph monitors. The remaining Rook components instead, have no influence on the functionality of the Ceph cluster. From our tests the Ceph monitors restarted almost instantaneously. For this reason, in our validation we report on the OSD case only.

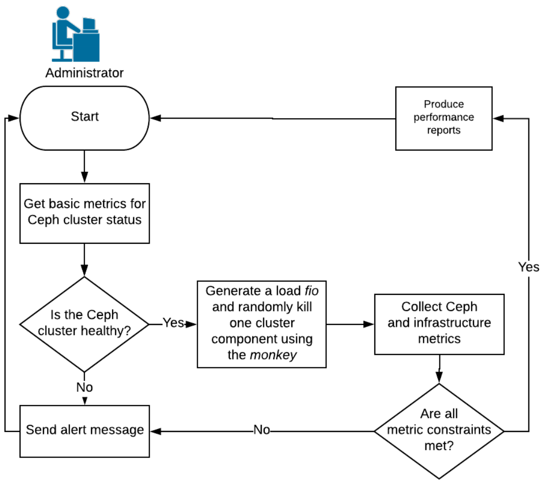

Administrator
Start
Produce performance reports
Get basic metrics for Ceph cluster status
Yes
Is the Ceph cluster healthy?
Yes
Generate a load *fio* and randomly kill one cluster component using the *monkey*
Collect Ceph and infrastructure metrics
No
Send alert message
No
Are all metric constraints met?

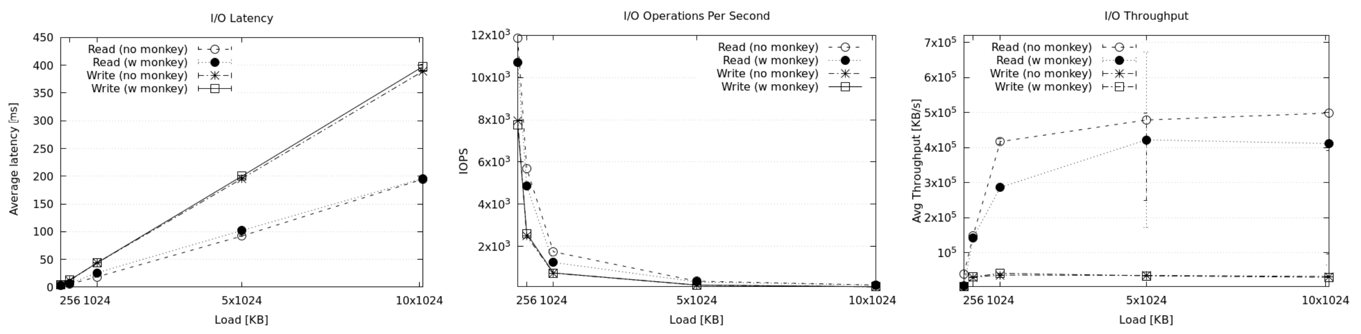
Fig. 1. Flowchart of the proposed solution.

V. Validation and Performance Evaluation

To validate our proposal, we implemented a software module in the open source programming language Go[4]. The objective is to be able to test and demonstrate the resilience of the Rook-managed storage cluster. We adopted this module to test the solution in a production environment for a Rook-managed Ceph cluster on Kubernetes, at the company Abraxas Informatik AG. The implemented tool is reachable from the outside through a REST interface and is connected to Grafana and to the Rook cluster. The Kubernetes cluster is running on a VMware ESXi infrastructure of virtual machines and consists of three controller nodes and six worker nodes maintained by Tectonic. Each of the worker nodes is equipped with one OSD and a dedicated disk for its OSD (this means that the loss of a node equals to the loss of an OSD). All the data and the test disks are persisted on Ceph with this leading to a balanced load on the Ceph components. Even if some constraints in the performed tests derive from the underlying infrastructure of the hosting company, we do not loose in generality in terms of the Ceph storage cluster resilience validation and monitoring capabilities.

In Table I we report the threshold values for the infrastructure metrics that have been adopted during validation. These values are specifically tailored to the specific scenario under study and derived from best practice analysis, the experience of the hosting company and application specific values. In Table II, the adopted validation settings are collected. The monitoring sampling period for the selected metrics is set to 1 second and the data sampling is performed every 10 seconds over a set of the last 30 samples. This implies that alerting messages and consequent countermeasures for the system can be performed every 10 seconds in the best case. These values can be tuned differently according to the specific system under test. However, for the reference scenario the mentioned values revealed to be satisfactory. The metrics are aggregated mostly in terms of mean values. Moreover, the 90-percentile and the Population Standard Deviation are considered for what concerns the warning state messages. This warning message is used to inform the administrator as soon as any system metric

[4]https://golang.org/



(a) Latency

(b) IOPS

(c) I/O throughput

Fig. 2. Performance comparison for a varying load value for read/write operations and with a randomly chosen component being killed by the monkey (plots are reported with 95% of confidence interval).

TABLE I. SELECTED METRICS AND THRESHOLD VALUES FOR GREEN, YELLOW, RED HEALTH STATES.

Metric	Green	Yellow	Red
Ceph health status	Healthy	Warning	Error
OSD Commit Latency [ms]	<10 ms	10-50 ms	>50 ms
OSD Apply Latency [ms]	<10 ms	10-50 ms	>50 ms
IOPS	<6000	6000-14000	>14000
CPU Usage [%]	<50%	50-85 %	>85%
Cores Usage [%]	<50%	50-80 %	>80%
CPU Usage [%]	<50%	50-80 %	>80%
Network Usage [%]	<50%	50-80 %	>80%
Disk IOPS Usage [%]	<50%	50-80 %	>80%
Available storage [%]	<50%	50-80 %	>80%
OSD down	0	1	>1
OSD orphan	0	1	>1
Monitor down	0	1	>1
PG undersized	0	1-inf	-
PG stale	0	1-inf	-
PG degraded	0	1-inf	-

gets near to the predefined thresholds.

Besides showcasing the resilience of the system, results are collected for additional parameters, namely: *latency*, which is the most important parameter as this influences the use of the service [17]; *I/O throughput* which gives information about how much data is processed; *IOPS* which tells us how many operations are used to process the data; and *recovery time* as the time between the system failure and the system being up again after a self-healing procedure.

The first analysis we present has the objective of proving that the removal of a single Rook component (i.e., agent, operator) has no relevant influence on the overall performance of the Ceph cluster. The motivation for this is that the system is able to quickly recover when a failure occurs and therefore is *resilient*. To showcase this, we run our experiments with the proposed technique comparing the case where the monkey is activated or not (labeled with *"w monkey"* and *"no monkey"* respectively in Fig. 2). We tested several blocksize values in the range $[4KB - 10MB]$ for random read/write operation using *fio* in the storage cluster. As reported in Fig. 2, we observe that in all cases there is almost no effect on the performances when the operator kills a randomly chosen OSD. At the same

TABLE II. MAIN VALIDATION PARAMETERS

	Parameter	Value
Validation settings	Data sampling period	10s
	Monitoring sample period	1s
	Network connection	10GBit/s
	Maximum infrastructure IOPS	14000
	Type of read/write operations	random
	Maximum infrastructure load	1GB
	Number of MONs	3
	Number of OSDs	6
	OSD size	32GB / 215GB
	Storage application	MongoDB
	Read/Write block size	4KB-10MB
	Test duration per run	60s

time, the monitoring and alerting solution is also informing the administrator about threatening situations. Observing the latency plots in Fig. 2 (a), we notice that the average latency increases with the load (in terms of block size) introduced in the system. This behavior is equal for reading and writing operations, with the writing operations showing an expected higher latency value. In all of the tested cases the latency ranges between an average value of 2.7 milliseconds for a reading operation and 4KB blocksize and a maximum of about 389 milliseconds for a writing operation and 10MB blocksize.

When focusing on the IOPS plots in Fig. 2 (b), we observe that in all cases the number of IOPS decreases with the increase in terms of load. The exact values range between a maximum of 11860 IOPS for reading when the monkey is not active and the blocksize is set to 4KB, and a minimum of 80 IOPS on average, for write operations when the blocksize is set to 10MB. The number of IOPS reached with write operations is always less than when reading operations are performed (also this is an expected result). For the reading operations a slightly more visible influence is observed when the monkey is "killing" randomly chosen OSDs in the storage cluster.

If we observe the I/O throughput as reported in Fig. 2 (c), it becomes even more evident how reading operations are performed much faster than writing operations for all values of the blocksize. For an increasing value of the blocksize it is interesting to observe that the throughput increases until a sort of system saturation is reached for large blocksize values (please note also that a larger confidence interval is obtained in these cases as we are reaching the boundaries of the system). In particular, for the reading operations it reaches about 500MB/s for a blocksize of 10MB. Also for this metric, the influence of the monkey being active is more evident for the read operations.

In the last analysis we present, we aim at showing the ability of the Rook-managed Ceph storage cluster in recovering in case of failures. In particular, we report the CDF for the time interval between the moment that the OSD is killed and the moment where the system is back to a healthy status. In Fig. 3 two different sizes for the OSD are considered and we highlight the difference in the recovery time. Looking at the 90th percentile for the tested cases, we can observe that the recovery time for the cluster is 50 seconds for the 32GB OSD case, and a bit less than 150 seconds for the 215GB OSD case (41 and 142 seconds on average for for all the tested cases for the 32GB and the 215GB case respectively). Whenever, a failure occurs an alerting message is also sent to the system administrator so that it is informed that something happened on the system which could have influenced the usage of the service. As part of our future work, we plan an automatic system reaction to the received alert messages.

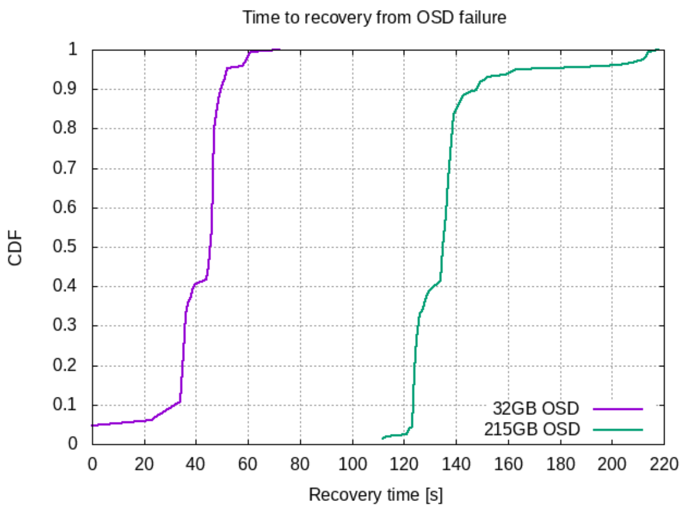


Fig. 3. CDF for failure recovery time when randomly killing an OSD of either 32GB or 215GB.

VI. Conclusion and Future Work

In this paper we have investigated on monitoring solutions for *resilience* in containerized distributed cloud storage systems. We presented a software tool that is able to test and monitor a Ceph cluster deployed using Rook on Kubernetes. As we demonstrated, the tool is able to forecast failures being about to occur and promptly alert the system administrator. In the Rook-managed storage cluster considered in our tests, failures could be recovered within 41/142 seconds on average for a 32GB and a 215GB OSD respectively. To effectively support resilience in the storage cluster a cautious monitoring of system metrics and the forecasting of failure threats that trigger alert messages is proposed. As failures correspond to revenue losses for the cloud storage providers, we believe that the proposed solution can be of high interest for the rapidly growing cloud storage market. In our future research we will extend our investigation for other scenarios and heavier load values to further test the storage cluster. Moreover, we will focus on extending the implemented tool to allow for automatic reacting to misbehaviour or failures based on known patterns and behaviors of the system.

References

[1] G. Tlili, M. F. Zhani and H. Elbiaze, ”On providing deadline-aware cloud storage services,” 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 2018.

[2] J. Gantz, and D. Reinsel, The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze the future, 1-16, 2012.

[3] R. Höppli, T. M. Bohnert, and L. Militano ”Hera Object Storage: A seamless, Automated Multi-Tiering Solution on Top of Openstack Swift”, 8th IEEE International Symposium on Cloud and Services Computing (SC2), 2018.

[4] Z., Ou, M., Song, Z. H., Hwang, A., Ylä-Jääski, R., Wang, Y., Cui, and P. Hui, ”Is cloud storage ready? Performance comparison of representative IP-based storage systems”, Journal of Systems and Software.

[5] C. Cerin, C. Coti, P. Delort, and F. Diaz, Downtime statistics of current cloud solutions, IWGCR: The International Working Group on Cloud Computing Resiliency, EU/USA, Tech. Rep. 001-en1-2013, Jun. 2013.

[6] E. Brewer, ”CAP twelve years later: How the” rules” have changed.” Computer 45.2, pp. 23-29, 2012.

[7] C. Colman-Meixner, C. Develder, M. Tornatore and B. Mukherjee, ”A Survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications,” in IEEE Communications Surveys & Tutorials, 2016.

[8] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, ”Self-managing cloud-native applications: Design, implementation, and experience”, in Future Generation Computer Systems, Vol. 72, 2017, pp. 165-179, https://doi.org/10.1016/j.future.2016.09.002.

[9] A. Srbu, and O. Babaoglu, ”Towards Data-Driven Autonomics in Data Centers”, in International Conference on Cloud and Autonomic Computing, 2015.

[10] A. M. Kermarrec, E. Le Merrer, G. Straub, and A. Van Kempen, ”Availability-based methods for distributed storage systems”, in IEEE 31st Symposium on Reliable Distributed Systems (pp. 151-160), 2012.

[11] Chun, Byung-Gon and Dabek, Frank and Haeberlen, Andreas and Sit, Emil and Weatherspoon, Hakim and Kaashoek, M Frans and Kubiatowicz, John and Morris, Robert Tappan, ”Efficient Replica Maintenance for Distributed Storage Systems”, in NSDI, vol. 6, 2006.

[12] Zhu, Bingpeng and Wang, Gang and Liu, Xiaoguang and Hu, Dianming and Lin, Sheng and Ma, Jingwei, ”Proactive drive failure prediction for large scale storage systems”, in IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), 2013.

[13] I. a. c. Red Hat, ”Ceph Documentation,” 2018. [Online]. Available: http://docs.ceph.com/docs/master/. [Accessed 26 07 2018].

[14] The Kubernetes Authors, ”What is Kubernetes?,” The Linux Foundation, 2018. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Accessed 19 08 2018].

[15] T. Taleb, A. Ksentini, and B. Sericola. ”On service resilience in cloud-native 5G mobile systems.” IEEE Journal on Selected Areas in Communications 34.3 (2016): 483-496.

[16] C. C., Chang, S. R., Yang, E. H., Yeh, P., Lin, and J. Y. Jeng, ”A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning.” GLOBECOM 2017- IEEE Global Communications Conference, 2017.

[17] G. Crump ”What is Latency? And How is it Different from IOPS?.” [Online]. Available: https://storageswiss.com/2013/12/10/what-is-latency-and-how-is-it-different-from-iops/ 2013.