~okeanos
Large-Scale Cloud Service Using Ceph

FILIPPOS GIANNAKOS, VANGELIS KOUKIS, CONSTANTINOS VENETSANOPOULOS, AND NECTARIOS KOZIRIS

Filippos Giannakos is a cloud storage engineer at the Greek Research and Technology Network. His research interests include distributed and software-defined storage. Giannakos is a PhD candidate in the Computing Systems Laboratory at the School of Electrical and Computer Engineering, National Technical University of Athens. philipgian@grnet.gr

Vangelis Koukis is the technical lead of the ~okeanos project at the Greek Research and Technology Network (GRNET). His research interests include large-scale computation in the cloud, high-performance cluster interconnects, and shared block-level storage. Koukis has a PhD in electrical and computer engineering from the National Technical University of Athens. vkoukis@grnet.gr

Constantinos Venetsanopoulos is a cloud engineer at the Greek Research and Technology Network. His research interests include distributed storage in virtualized environments and large-scale virtualization management. Venetsanopoulos has a diploma in electrical and computer engineering from the National Technical University of Athens. cven@grnet.gr

Nectarios Koziris is an associate professor in the Computing Systems Laboratory at the National Technical University of Athens. His research interests include parallel architectures, interaction between compilers, OSes and architectures, OS virtualization, large-scale computer and storage systems, cloud infrastructures, distributed systems and algorithms, and distributed data management. Koziris has a PhD in electrical and computer engineering from the National Technical University of Athens. nkoziris@cslab.ece.ntua.gr

~Okeanos is a large-scale public cloud service powered by Synnefo and run by GRNET. Ceph is a distributed storage solution that targets scalability over commodity hardware. This article focuses on how we use Ceph to back the storage of ~okeanos. More specifically, we will describe what we aimed for in our storage system, the challenges we had to overcome, certain tips for setting up Ceph, and experiences from our current production cluster.

The ~okeanos Service

At GRNET, we provide ~okeanos [2, 4], a complete public cloud service, similar to AWS, for the Greek research and academic community. ~okeanos has Identity, Compute, Network, Image, Volume, and Object Storage services and is powered by Synnefo [3, 5]. ~okeanos currently holds more than 8000 VMs.

Our goals related to storage are to provide users with the following functionality:

- The ability to upload files and user-provided images by transferring only the missing data blocks (diffs).
- Persistent VMs for long-running computationally intensive tasks, or hosting services.
- Thin VM provisioning (i.e., no copy of disk data when creating a new VM) to enable fast spawning of hundreds of VMs, with zero-copy snapshot functionality for checkpointing and cheap VM backup.

We aim to run a large-scale infrastructure (i.e., serve thousands of users and tens of thousands of VMs) over commodity hardware.

A typical workflow on ~okeanos is that a user downloads an image, modifies it locally (e.g., by adding any libraries or code needed), and reuploads it by synchronizing it with the server and uploading only the modified part of the image. The user then spawns a large number of persistent VMs thinly, with zero data movement. The VM performs some computations, and the user can then take a snapshot of the disk as a checkpoint; the snapshot also appears as a regular file on the Object Storage service, which the user can sync to his/her local file system to get the output of the calculations for further offline processing or for backup.

To achieve the described workflow, however, we had to overcome several challenges regarding storage.

Storage Challenges

We stumbled upon two major facts while designing the service: (1) providing persistent VMs conflicts with the ability to scale and (2) using the same storage entities from different services requires a way to access them uniformly without copying data.

Providing persistent VMs while being able to scale is a demanding and difficult problem to solve. Persistence implies the need to live migrate VMs (change their running node while keeping the VM running) or fail them over (shut them down and restart them on another node) when a physical host experiences a problem. This implies shared storage among the nodes because the VM, and consequently the VM's host, needs to access the virtual disk's data. The most common solution to provide shared storage among the nodes is a central NAS/SAN exposed to all nodes.

However, having a central storage to rely on cannot scale and can be a single point of failure for the storage infrastructure. DRBD is another well-known enterprise-level solution to provide persistent VMs and scale at the same time. DRBD mirrors a virtual disk between two nodes and propagates VM writes that occur on one node to the replica. DRBD is essentially a RAID-1 setup over network. However, this choice imposes specific node pairs where the data is replicated and where the VM can be migrated, narrowing the administrator's options when performing maintenance. Moreover, it does not allow for either thin VM provisioning or for sharing blocks of data among VM volumes and user uploaded files, which takes us to the second problem.

The second problem we had to overcome involved presenting the different storage entities to the cloud layer uniformly. For example, a snapshot should be treated as a regular file to be synced and also as a disk image to be cloned. The actual data, though, remain the same in both cases, and only the cloud layer on top of the storage should distinguish between the two aspects. Therefore, we needed a way to access the same data from different cloud services uniformly, through a common storage layer.

To solve these problems, we needed a storage layer that would abstract the cloud aspect of a resource from its actual data and allow the ability to present this data in various ways. Additionally, we needed this mechanism to be independent from the actual data store. Because no suitable solution existed, we created Archipelago.

Archipelago

Archipelago [1] is a distributed storage layer that unifies how storage is perceived by the services, presenting the same resources in different ways depending on how the service wants to access them. It sits between the service that wants to store or retrieve data and the actual data store. Archipelago uses storage back-end drivers to interact with any shared data store. It also provides all the necessary logic to enable thin volume provisioning, snapshot functionality, and deduplication over any shared storage. Therefore, Archipelago allows us to view the cloud resources uniformly, whether these are images, volumes, snapshots, or just user files. They are just data stored in a storage back end, accessed by Archipelago.

Synnefo uses Archipelago to operate on the various representations of the same data:

- From the perspective of Synnefo's Object Storage service, images are treated as common files, with full support for remote syncing and sharing among users.
- From the perspective of the Compute service, images can be cloned and snapshotted repeatedly, with zero data movement from service to service.
- And, finally, snapshots can appear as new image files, again with zero data movement.

Backing Data Store

Archipelago acts as a middle layer that presents the storage resources and solves the resource unification problem but does not actually handle permanent storage. We needed a data store to host the data. Because we were not bound by specific constraints, we had various shared storage options. We decided to start with an existing large central NAS, exposed to all nodes as an NFS mount. This approach had several disadvantages:

- It could not scale well enough to hold the amount of users and data we aimed for.
- Having a large enterprise-level NAS imposed geographical constraints. The data reside in only one datacenter.
- It imposes a centralized architecture, which is difficult and costly to extend.

Because ~okeanos had exponential growth, we needed a different storage solution.

Ceph

Ceph is a distributed storage solution. It offers a distributed object store, called RADOS [6], block devices over RADOS called RBD, a distributed file system called CephFS, and an HTTP gateway called RadosGW. We have been following its progress and experimenting with it since early 2010.

RADOS is the core of Ceph Storage. It is a distributed object store comprising a number of OSDs, which are the software components (processes) that take care of the underlying storage of data. RADOS distributes the objects among all OSDs in the cluster. It manages object replication for redundancy, automatic data recovery, and cluster rebalancing in the presence of node failures.

RBD provides block devices from objects stored on RADOS. It splits a logical block device in a number of fixed-size objects and stores these objects on RADOS.

CephFS provides a shared file system with near-POSIX semantics, which can be mounted from several nodes. CephFS splits files into objects, which are then stored on RADOS. It also consists of one or more metadata servers to keep track of file-system metadata.

Finally, **RadosGW** is an HTTP REST gateway to the RADOS object store.

Ceph seemed a promising storage solution that provided scalable distributed storage based on commodity hardware, so we decided to evaluate it. Ceph exposes the data stored in RADOS in various forms, but it does not act on them uniformly like Archipelago does. Because we already had an HTTP gateway and VM volume representation of the data by Archipelago, we needed RADOS, which also happens to be the most stable and mature part of Ceph, to store and retrieve objects. We used Ceph's librados to implement a user-space driver for Archipelago to store our cloud data on RADOS and integrate it with Synnefo.

Things to Consider with Ceph

While evaluating RADOS, we experimented with various RADOS setup parameters and drew several conclusions regarding setup methodology and RADOS performance.

OSD/Disks Setup

Ceph's OSDs are user-space daemons that form a RADOS cluster. Each OSD needs permanent storage space where it will hold the data. This space is called "ObjectStore" in RADOS terminology. Ceph currently implements its ObjectStore using files, so we will use the term "filestore." We had several storage nodes that could host RADOS OSDs. Each node had multiple disks. So the question arose how to set up our RADOS cluster and where to place the filestores. We had numerous parameters to consider, including the number of OSDs per physical node, the number of disks per OSD, and the exact disk setup. After extensive testing, we concluded that it is beneficial to have multiple OSDs per node, and we dedicated one disk to each one. This setup ensures that one OSD does not compete with any other for the same disk and allows for multiple OSDs per node, resulting in improved aggregate performance.

Journal Mode and Filestore File System

Along with the filestore, each OSD keeps a journal to guarantee data consistency while keeping write latency low. This means that every write gets written twice in each OSD, once in the journal and once in the backing filestore. There are two modes in which the journal can operate, write-ahead and write-parallel. In write-ahead mode, each write operation is first committed to the journal and then applied to the filestore. The write operation can be acknowledged as soon as it is safely written to the journal. In write-parallel mode, each write is written to both the journal and the filestore in parallel, and the write can be acknowledged when either of the two operations is completed successfully. The write-parallel mode requires btrfs as the file system of the backing filestore, because it makes use of btrfs-specific features, such as snapshots and rollbacks, to guarantee data consistency. On the other hand, the write-ahead mode can work with all the recommended file systems, such as ext4 and XFS. Because btrfs is still not production ready and showed significant instability under heavy load, we decided to proceed with ext4 and write-ahead journal mode.

Journal Placement and Size

The RADOS journal can be placed in a file in the backing filestore, in a separate disk partition on the same disk, or in a completely separate block device. The first two options are suboptimal because they share the bandwidth and IOPS of the OSD's disk with the filestore, essentially halving the overall disk performance. Therefore, we placed the RADOS journal in a separate block device.

You might think that, because writes are confirmed when they hit the journal, an OSD can sustain improved write performance for a longer period by using a bigger journal on a fast device, falling back to filestore performance when the journal gets full. However, our experiments showed that RADOS OSDs do not work like that. If the journal media is much faster than the filestore, the OSD pauses writes when it tries to sync the journal with the filestore. This behavior can result in abnormal and erratic patterns during write bursts. Thus, a small portion of a block device with performance close to the one of the filestore should be used to hold the journal. Because we do not need large journals, multiple journals can be combined in the same block device, as long as it provides enough bandwidth for all of them.

RADOS Latency

When evaluating a storage system, especially for VM virtual disks' data, latency plays a critical role. VMs tend to perform small (4 K to 16 K) I/Os where latency becomes apparent. Our measurements showed that RADOS has a non-negligible latency of about 2 ms, so you cannot expect latency comparable with local disks. This behavior can be masked by issuing multiple requests or performing larger I/O to achieve high throughput. Also, because the requests are equally distributed among all OSDs in a cluster, the overall performance remains highly acceptable.

Data Integrity Checking

Silent data corruption caused by hardware can be a big issue on a large data store. RADOS offers a scrubbing feature, which works in two modes: regular scrubbing and deep scrubbing. Regular scrubbing is lighter and checks that the object is correctly replicated among the nodes. It also checks the object's metadata and attributes. Deep scrubbing is heavier and expands the check to the actual data. It ensures data integrity by reading the data and computing checksums.

Storage vs. Compute Nodes

Another setup choice we had was to keep our storage nodes distinct from our compute nodes, where the VMs reside. This has two main advantages. First, OSDs do not compete with the VMs for compute power. OSDs do not require a lot of CPU power normally, but it can be noticeable while scrubbing or rebalancing. Normal operations will also require more CPU power with the upcoming erasure coding feature, where CPU power is used to reconstruct objects in order to save storage space. Second, we can use the storage container RAM as cache for hot data using the Linux page-cache mechanism, which uses free RAM to cache recently accessed files on a file system. Hosting VMs on the same node would leave less memory for this purpose.

Experiences from Production

Ceph has been running in production since April 2013 acting as an Archipelago storage back end. As of this writing, Ceph's RADOS backs more than 2000 VMs and more than 16 TB of user-uploaded data on the Object Storage service.

Our current production setup comprises 20 physical nodes. Each node has

- 2 × 12-Core Xeon(R) E5645 CPU
- 96 GBs of RAM
- 12 × 2 TB SATA disks

Our storage nodes can provide more CPU power than Ceph currently needs. We are planning to use this extra power to seamlessly enable future Ceph functionality, like erasure coding, and to divert computationally intensive storage-related tasks (e.g., hashing) to the storage nodes, using the RADOS "classes" mechanism.

Each physical node hosts six OSDs. Each OSD's data resides on a RAID-1 setup between two 2 TB disks. RADOS replicates objects itself, but because it was under heavy development, we wanted to be extra sure about the safety of our data. By using this setup along with a replication level 2, we only use one fourth of our overall capacity, which covers our current storage needs. As our storage needs grow and RADOS matures, we aim to break the RAID setups and double the cluster capacity.

Using Ceph in production has several advantages:

- It allows us to use large bulk commodity hardware.
- It provides a central shared storage that can self-replicate, self-heal, and self-rebalance when a hardware node or a network link fails.
- It can scale on demand by adding more storage nodes to the cluster as demand increases.
- It enables live migration of VMs to any other node.

Using Ceph in a large-scale system also revealed some of its current weaknesses. Scrubbing and especially deep scrubbing can take a lot of time to complete. During these actions, there is significant performance drop. The cluster fills with slow requests and VMs are affected. This is a major drawback, and we had to completely disable this functionality. We plan to re-enable it as soon as it can be used without significant performance regression. Performance also drops when rebuilding the cluster after an OSD failure or when the cluster rebalances itself after the addition of a new OSD. This issue is closely related to the performance drop during deep scrubbing, and it occurs because RADOS does not throttle traffic related to recovery and deep scrubbing according to the underlying disk utilization and the rate of the incoming client I/O.

On the other hand, Ceph has survived numerous hardware failures with zero data loss and minimal service disruption. Its overall usage experience outweighs the hiccups we endured since we began using it for production purposes.

In conclusion, our experience from running a large-scale Ceph cluster shows that it has obvious potential. It can run well over commodity hardware, scale without any visible overhead, and helped us to deploy our service. However, in its current state, running it for production purposes has disadvantages because it suffers from performance problems when performing administrative actions.

Ceph is open source. Source code, more info, and extra material can be found at http://www.ceph.com.

References

[1] Archipelago: http://www.synnefo.org/docs/archipelago/latest.

[2] The ~okeanos service: http://okeanos.grnet.gr.

[3] Synnefo software: http://www.synnefo.org.

[4] Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris, "~okeanos: Building a Cloud, Cluster by Cluster," *IEEE Internet Computing*, vol. 17, no. 13, May-June 2013, pp. 67–71.

[5] Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris, "Synnefo: A Complete Cloud Stack over Ganeti," USENIX, *;login:*, vol. 38, no. 5 (October 2013), pp. 6–10.

[6] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn, "RADOS: A Scalable, Reliable Storage Service for Petabyte-Scale Storage Clusters," in *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07,* PDSW '07 (ACM, 2007), pp. 35–44.