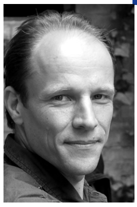
CARLOS MALTZAHN, ESTEBAN MOLINA-ESTOLANO, AMANDEEP KHURANA, ALEX J. NELSON, SCOTT A. BRANDT, AND SAGE WEIL

Ceph as a scalable alternative to the Hadoop Distributed File System

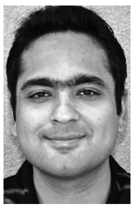


Carlos Maltzahn is an associate adjunct professor at the UC Santa Cruz Computer Science Department and associate director of the UCSC/Los Alamos Institute for Scalable Scientific Data Management. His current research interests include scalable file system data and metadata management, storage QoS, data management games, network intermediaries, information retrieval, and cooperation dynamics.

carlosm@soe.ucsc.edu

Esteban Molina-Estolano is a PhD student at UC Santa Cruz. His research interests include parallel file and storage systems, and their modeling and simulation. He received his BS from Harvey Mudd College.

eestolan@cs.ucsc.edu



Amandeep Khurana completed his master's in computer science from UC Santa Cruz, specializing in distributed systems. He has been dabbling with Hadoop and HBase and is now at Amazon, helping them scale their systems. Prior to this, he worked with Cisco on building scalable data integration frameworks.

akhurana@soe.ucsc.edu

Alex Nelson is a PhD student at UC Santa Cruz. His research interests include secure, large-scale, and long-term storage.

ajnelson@cs.ucsc.edu

Scott Brandt is a professor of computer science at the University of California, Santa Cruz, and director of the UCSC/Los Alamos Institute for Scalable Scientific Data Management (ISSDM) and the UCSC Systems Research Laboratory (SRL), in which Ceph was developed. His current research includes high-performance storage systems, real-time systems, and distributed system performance management.

scott@cs.ucsc.edu

Sage Weil built Ceph as part of his PhD research in storage systems at UC Santa Cruz. Prior to his graduate work, Sage helped found New Dream Network, the company behind Dreamhost Web hosting (dreamhost.com), which now supports a small team of Ceph developers.

sage@newdream.net

THE HADOOP DISTRIBUTED FILE System (HDFS) has a single metadata server that sets a hard limit on its maximum size. Ceph, a high-performance distributed file system under development since 2005 and now supported in Linux, bypasses the scaling limits of HDFS. We describe Ceph and its elements and provide instructions for installing a demonstration system that can be used with Hadoop.

Hadoop has become a hugely popular platform for large-scale data analysis. This popularity poses ever greater demands on the scalability and functionality of Hadoop and has revealed an important architectural limitation of its underlying file system: HDFS provides only one *name-node*, which has to store the entire file system namespace in main memory. This puts a hard limit on the amount of metadata, in particular the number of files, that HDFS can store. The single name-node limitation is well-recognized in the Hadoop user and developer community (see, for example, [8, 16]). Large clusters frequently run out of capacity at the name node to track new files even though there is plenty of storage capacity at the data nodes. The single name-node also creates a single point of failure and a potential performance bottleneck for workloads that require relatively large amounts of metadata manipulations, such as opening and closing files.

Ceph [11] is an object-based parallel file system with a number of features that make it an ideal storage system candidate for Hadoop:

- Ceph's *scalable metadata server* [14] can be distributed over hundreds of nodes while providing consistent, reliable, high-performance metadata service using dynamic subtree partitioning with near-linear scalability.
- Each file can specify its own *striping strategy* and object size. Flexible striping strategies and object sizes are important tuning parameters for Hadoop workloads [2, 6, 10].
- Data is stored on up to 10,000 nodes which export a single, *reliable object service* [13] with a flat namespace of object IDs, not unlike Amazon's Simple Storage Service (S3) [1]. Changes in the storage cluster size cause automatic (and fast) failure recovery and rebalancing of data with no interruption of service and minimal data movement, making Ceph suitable for very large deployments.
- The state of the entire storage cluster, includ-

ing data placement, failed nodes, and recovery state, has a very compact representation due to calculated data placement [12] as opposed to allocation tables, and is known in every part of Ceph. As in HDFS, Hadoop's scheduler can take advantage of this information to place mapping close to where the data resides.

- Ceph is an open source project (ceph.newdream.net) written in C++ and C that started as a PhD research project at UC Santa Cruz over four years ago and has been under heavy development ever since. A Hadoop module for integrating Ceph into Hadoop has been in development since release 0.12, and Hadoop can also access Ceph via its POSIX I/O interface, using ioctl calls for data location information.
- Since Ceph is designed to serve as a general-purpose file system (e.g., it provides a Linux kernel client so Ceph file systems can be mounted), if it supported Hadoop workloads well, it could also be a general solution to other storage needs.

In this article we describe the Ceph file system architecture, how to install Ceph on a 10-node cluster, and how to use Ceph with Hadoop.

Ceph

Ceph was designed to fulfill the following goals specified by three national laboratories (LLNL, LANL, and Sandia) back in 2005:

- Petabytes of data on one to thousands of hard drives
- TB/sec aggregate throughput on one to thousands of hard drives pumping out data as fast as they can
- Billions of files organized in one to thousands of files per directory
- File sizes that range from bytes to terabytes
- Metadata access times in μsecs
- High-performance direct access from thousands of clients to
    - different files in different directories
    - different files in the same directory
    - the same file
- Mid-performance local data access
- Wide-area general-purpose access

The challenges of such a file system are that it needs to be able to deal with huge files and directories, coordinate the activity of thousands of disks, provide parallel access to metadata on a massive scale, handle both scientific and general-purpose workloads, authenticate and encrypt at scale, and grow or shrink dynamically because of frequent device decommissioning, device failures, and cluster expansions. You can't just shut down the system because of a disk failure or shortage of space.

Ceph is an *object-based parallel file system* whose design is based on two key ideas. The first key idea is object-based storage, which splits the traditional file system architecture into a client component and a storage component. The storage component manages disk scheduling and layout locally, relieving clients and servers from low-level per-disk details and increasing scalability. This design allows clients to communicate with storage nodes via a high-level interface and manage data in terms of objects, which are chunks of data much larger than 512-byte blocks. The T10 standard of the SCSI Object Storage Device (OSD) command set [5] is an example of an object interface specification. Ceph uses and significantly extends the concept of OSDs. For all practical purposes, think of a Ceph OSD as a process that runs on a cluster node and uses a local file system to store data objects.

The second key idea in the Ceph design is the separation of data and metadata. Management of data differs fundamentally from management of metadata: file data storage is trivially parallelizable and is limited primarily by the network infrastructure. Metadata management is much more complex, because hierarchical directory structures impose interdependencies (e.g., POSIX access permissions depend on parent directories) and the metadata server must maintain file system consistency. Metadata servers have to withstand heavy workloads: 30–80% of all file system operations involve metadata, so there are lots of transactions on lots of small metadata items following a variety of usage patterns. Good metadata performance is therefore critical to overall system performance. Popular files and directories are common, and concurrent access can overwhelm many schemes.

The three unique aspects of Ceph's design are:

- distributed metadata management in a separate metadata server (MDS) cluster that uses dynamic subtree partitioning to avoid metadata access hot spots and that is robust against non-byzantine failures;
- calculated pseudo-random data placement that allows for very compact state that can be shared easily throughout the system (CRUSH); and
- distributed object storage using a cluster of intelligent OSDs which forms a reliable object store that can act autonomously and intelligently (RADOS) (see Figure 1).

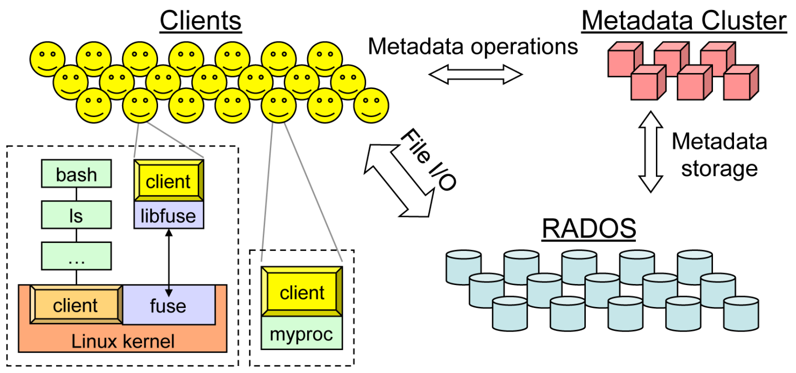
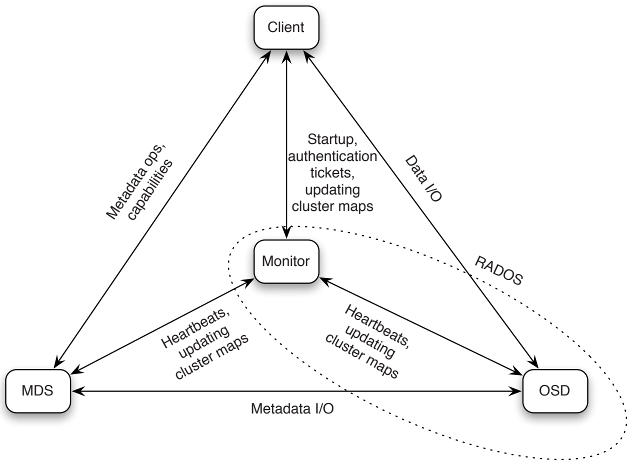


FIGURE 1: ARCHITECTURE. CEPH CONSISTS OF FOUR SUBSYSTEMS: (1) FILE SYSTEM CLIENTS, (2) DATA PLACEMENT USING A FORM OF CONSISTENT HASHING (CONTROLLED REPLICATION UNDER SCALABLE HASHING, OR CRUSH), (3) A CLUSTER OF METADATA SERVERS (MDS), AND (4) A RELIABLE AUTONOMIC, DISTRIBUTED OBJECT STORE (RADOS), WHICH INCLUDES THE MONITOR SERVICE AND OBJECT STORAGE DEVICES (OSDS).

The high-level interaction of these components is shown in Figure 2.



Client
Metadata ops, capabilities
Startup, authentication tickets, updating cluster maps
Data I/O
RADOS
Monitor
Heartbeats, updating cluster maps
Heartbeats, updating cluster maps
MDS
OSD
Metadata I/O

FIGURE 2: CEPH COMPONENT INTERACTIONS

CLIENT

A user or an application interacts with Ceph through the client component. The client exposes a POSIX file system interface. The interface also supports a subset of POSIX I/O extensions for high-performance computing, including the `O_LAZY` flag for the POSIX open command, which allows performance-conscious applications to manage their own consistency [15]. Ceph has a user-level client as well as a kernel client. The user-level client is either linked directly to the application or used via FUSE [9]. The kernel client is now available in the mainline Linux kernel (since 2.6.34) and allows the Ceph file system to be mounted.

For a simple example of how a client interacts with the rest of the Ceph system, consider a client opening a file /foo/bar for reading: the client sends an "open for read" message to the MDS. The MDS reads directory /foo from the appropriate OSDs (unless the directory is already cached in memory) and returns the capability for reading "/foo/bar" to the client. The client then calculates the name(s) and location(s) of the data object(s) for the file and reads the data from the corresponding OSD(s). Finally, the client closes the file by relinquishing the capability to the MDS. Remember that all these messages between these different components of Ceph are invisible to the application: the application simply issues POSIX open, read, and close commands.

The client is the only place in Ceph where metadata meets data: the capability returned by the MDS contains the inode number, the replication factor, and information about the *striping strategy* of a file, which can be file-specific and is set at file creation time. The striping strategy, the inode number, and an offset allow the client to derive the object identifier. A simple hash function maps the object identifier (OID) to a *placement group*, a group of OSDs that stores an object and all its replicas. There are a limited number of placement groups to create an upper bound on the number of OSDs that store replicas of objects stored on any given OSD. The higher that number, the higher the likelihood that a failure of multiple nodes will lead to data loss. If, for example, each OSD has replica relations to every other OSD, the failure of just three nodes in the entire cluster can wipe out data that is stored on all three replicas. Placement groups are mapped to OSDs by CRUSH, a consistent hashing function that takes as parameters (1) the placement group ID, (2) the replication factor, (3) the current cluster map (see section on RADOS, below), and (4) placement rules (see section on CRUSH, below). CRUSH then returns an ordered list of OSD IDs, one for each replica. The client picks the first OSD (the "primary") on the list as a location of the object.

METADATA SERVICE (MDS)

Ceph provides a cluster of metadata servers which continually load-balances itself using dynamic subtree partitioning [14]. The responsibility for managing the namespace hierarchy is adaptively and intelligently distributed among tens or even hundreds of metadata servers. The key to the MDS cluster's adaptability is that Ceph metadata items are very small and can be moved around quickly. This would be impossible if Ceph were to use the approach of many file systems, in which file byte streams are mapped to disk blocks using allocation tables.

Each directory is stored as an object. Inodes are embedded in directories and stored with the corresponding directory entry (file name). This organization optimizes the common access pattern of listing a directory and retrieving metadata for each file. Embedded inodes complicate the management of hard links, but it turns out that hard links are rare and exhibit useful locality properties (most hard links are referring to entries within the same or parallel directory).

To enable failure recovery, the MDS journals metadata updates to OSDs. The journals are striped across large objects, which leads to efficient, sequential writes. All in-memory metadata that is updated and journaled but not written as an updated directory object is marked dirty and pinned to main memory until the corresponding entries in the journal must be trimmed from the tail of the journal. Journals are allowed to grow very large (hundreds of megabytes). By the time updates need to be trimmed, many metadata updates have been invalidated by subsequent updates and can be consolidated into a small number of directory updates. With this approach the MDS acts as an intelligent metadata cache that turns random updates of small metadata items into efficient data I/O.

The file system's namespace is partitioned across the cluster of MDS nodes along directory subtrees, as shown in Figure 3. This *dynamic subtree partitioning* ensures that the distribution of subtrees is as coarse as possible to preserve locality (which is often aligned along subtrees) and that it is continually adapted to keep the MDS workload-balanced. Subtrees are migrated between MDSes as workload changes. To alleviate hot spots, heavily read directories are replicated on multiple MDSes so that the number of clients knowing about a particular replica can be bounded. Large or heavily written directories are fragmented for load distribution and storage. Rebalancing of the MDS at even extreme workload changes is usually accomplished within a few seconds. Clients are notified of relevant partition updates whenever they communicate with the MDS.

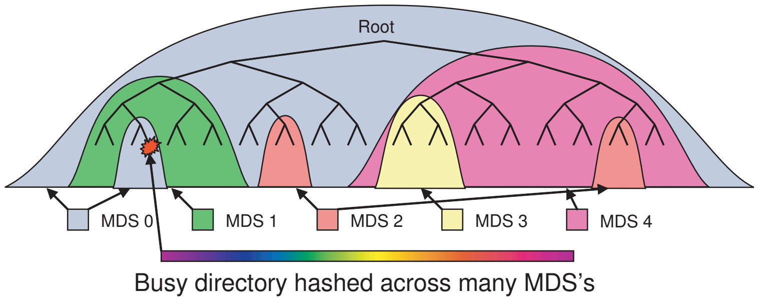


FIGURE 3: DYNAMIC SUBTREE PARTITIONING

RELIABLE AUTONOMIC DISTRIBUTED OBJECT STORAGE (RADOS)

Data in Ceph is stored in a distributed object store that can scale up from tens to hundreds of thousands of object storage devices (OSDs). RADOS can also be used as a stand-alone system. In fact, the Ceph distribution includes a simple gateway, a fastcgi daemon, that implements Amazon's S3 API [1] for RADOS.

Initiators—in Ceph's case, clients and the MDS—see the object storage cluster as a single logical object store with a flat namespace, called RADOS. RADOS achieves its scalability by significantly expanding the T10 SCSI OSD concept: while T10 OSDs only respond to read and write, Ceph OSDs actively collaborate with peers for failure detection, data replication, and recovery.

To direct requests, initiators use a *cluster map* which they receive from RADOS. The very compact cluster map contains information about participating OSDs, their status, and the CRUSH mapping function (see CRUSH section, below). The master copy of the cluster map is managed by a distributed *monitor service* which consists of a set of monitors that maintain consistency using the Paxos algorithm [7]. When an OSD alerts the monitor service of a new failure or cluster expansion, the monitor replies with an updated version of the cluster map, which is then lazily propagated throughout the cluster. The total size of the cluster map is in the range of megabytes (depending on the size of the cluster). Propagation overhead is reduced by only communicating deltas between cluster map versions and combining them with existing inter-OSD messages.

Unlike other parallel file systems, replication is managed by OSDs instead of clients, which shifts replication bandwidth overhead to the OSD cluster, simplifies the client protocol, and provides fully consistent semantics in mixed read/write workloads. RADOS manages the replication of data using a variant of primary-copy replication. As mentioned in the Client section, above, replicas are stored in placement groups. Each placement group includes a primary OSD which serializes all requests to the placement group. In case of writes, the primary forwards the request to the other OSDs of the placement group. The primary applies the write locally after the other replicas have applied theirs. Only then does the client receive an acknowledgment from the primary.

Writes are applied in two phases, as shown in Figure 4. This two-phase approach separates writing for the purpose of sharing with other clients from writing for the purpose of durability and makes sharing data very fast. The client receives an *ack* from the primary after the data has been replicated in memory on all of the replicas. At this point, the client still has the data in its buffer cache. Once the data is committed to the disk on all replicas, the primary sends a *commit* to the client, confirming that the data is now durable. The client may then delete the data from its buffer cache.

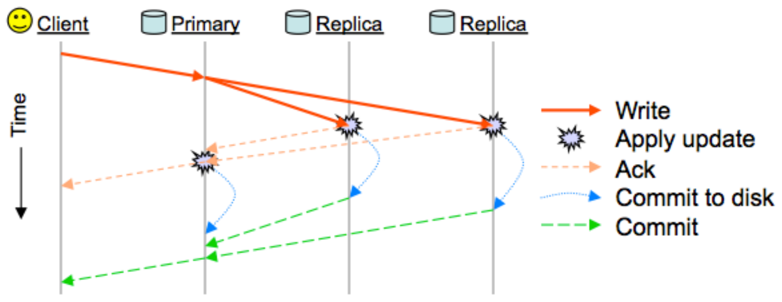


FIGURE 4: WRITE SEMANTICS

By default, OSDs use Btrfs [3] as their local file system (but ext3 works too). Data is written asynchronously using copy-on-write, so that unsuccessful write operations can be fully rolled back. Each OSD maintains a log of object versions for each placement group in which it participates. If one OSD fails, the remaining OSDs can quickly identify stale or missing objects by comparing these logs; OSDs which intermittently fail can quickly recover.

An OSD monitors the state of other OSDs in the same placement groups using heartbeat messages, which are usually piggybacked on replication traffic. OSDs that discover an unresponsive OSD alert the monitor service and receive a new cluster map that marks the unresponsive OSD as *down*. Other OSDs temporarily take over any primary roles the unresponsive OSD might have had. If within a configured time the OSD does not come back up, the monitor service issues another cluster map that marks it as *out*, some other OSD is elected to be the new primary, and the re-establishment of the full complement of replicas begins.

In summary, Ceph's failure detection and recovery are fully distributed and the monitor service is only used to update the master copy of the cluster map. The monitor service does not broadcast these updates to the cluster. Instead, the cluster map updates are communicated by OSDs using epidemic-style propagation that has bounded overhead. This procedure is used to respond to all cluster map updates, whether due to OSD failure, cluster contraction, or expansion. OSDs always collaborate to realize the data distribution specified in the latest cluster map while preserving consistency of read/write access.

DATA DISTRIBUTION WITH CRUSH

The small size of metadata items in the MDS and the compactness of cluster maps in RADOS are enabled by CRUSH (Controlled Replication Under Scalable Hashing) [12]. Ceph uses this hash function to calculate the placement of data instead of using allocation tables, which can grow very large and unwieldy. CRUSH is part of the cluster map and behaves like a consistent hashing function in that failure, removal, and addition of nodes result in near-minimal object migration to re-establish near-uniform distribution.

As mentioned in the Client section, CRUSH maps a placement group ID to an ordered list of OSDs, using a hierarchically structured cluster map and placement rules as additional input. The length of the list of OSDs depends on the replication factor. The first available OSD in the list is the primary. Any list output by CRUSH meets the constraints specified by *placement rules*. These rules are defined over cluster maps which an administrator can hierarchically structure according to, say, failure domains such as racks or shelves (since they often share the same electrical circuit or power supply). Thus, placement rules can prevent two replicas from being placed in the same failure domain. This awareness of failure domains during data placement is critically important for the overall data safety of very large storage systems where correlated failures are common.

Installing and Configuring Ceph

A Ceph installation requires at least one monitor, one OSD, and one metadata server. These may all be on the same node, although the use of additional nodes allows for greater performance, robustness, and capacity. Here we give a configuration walk-through for a multi-node Ceph installation. For a single-node installation, adapt this example to start up one of each daemon, all on the same node.

Our hypothetical cluster has 10 nodes, with hostnames node0 through node9. Each node has a raw, unused partition at /dev/sdb1 and another at /dev/sdb2. We will configure the nodes as follows:

- Three monitors, on nodes 0–2
- Three MDSes, on nodes 0–2
- Eight OSDs, on nodes 2–9

Less hardware could be used for a multi-node cluster. There should always be an odd number of monitors. For robustness, two MDSes (active and standby) are sufficient. For data replication, two OSDs would be sufficient.

We use a file system setup user named `setupuser`, whose public key grants access to `setupuser` on all nodes and root accounts on the storage nodes, due to the need to format partitions. This key should be password-locked and used with SSH-Agent or an equivalent session manager—it will only be used to set up and tear down the Ceph cluster. However, if you use the same account and key for later Hadoop setup, this key cannot have a password due to Hadoop's passwordless SSH requirement.

These instructions are sufficient to set up and run a Ceph cluster. If you want further documentation, see the Ceph wiki (http://ceph.newdream.net/wiki/), and src/sample.ceph.conf within the downloaded Ceph server code. Alternate packaging is available for Debian and Fedora Core from Ceph's wiki; also for Fedora, there is a `ceph.spec` file in the source. This tutorial works with Ceph 0.20.1 from source and Linux kernel 2.6.34.

INSTALLING CEPH DAEMONS

First, we install the prerequisite packages on each node. For Fedora Core 12 or Ubuntu 9.10, run these commands:

```
#Ubuntu 9.10
apt-get install autoconf automake libtool libboost-dev libedit-dev libssl-dev

#Fedora Core 12
yum install rpm-build fuse-devel libtool libtool-ltdl-devel boost-devel
libedit-devel openssl-devel gcc-c++ btrfs-progs
```

Second, we configure and build the source. For simplicity, we will assume that you are building the source in a shared NFS directory, visible on all nodes. We will run the daemons from <ceph directory>/src/ in the build tree without installation. We will also create the ceph.conf configuration file in the source tree.

To avoid relying on NFS, you can `make install` Ceph on each node, or install Ceph using the Debian packages (see http://ceph.newdream.net/wiki/Debian). In this case, you will also need to deploy ceph.conf to /etc/ceph/ceph.conf on each node, instead of keeping it in the source tree.

```
cd <ceph directory>
CXXFLAGS="-g" ./configure
make
```

The cluster initialization script detects when these are run out of a local directory instead of being fully installed; thus there is no need to set the Ceph executables' directory in a configuration file. If you do want to have a local installation on each node, run `sudo make install` after `make`. This places the Ceph binaries into /usr/local/bin and /usr/local/sbin as necessary.

CONFIGURING

Ceph's configuration is stored in a single file, `ceph.conf`, which is identical across all nodes. There is a section in the configuration file for each daemon and a common section for each type of daemon. For instance, configuration parameters common across all monitors are placed in the [mon] section, and per-monitor settings are placed in [mon0], [mon1], and so on.

For logging and other local data purposes, create a local directory on all nodes, /data, readable and writeable as setupuser. Then create src/ceph.conf as follows.

```
[global]
   user = setupuser
   ; where the mdses and osds keep their secret encryption keys
   keyring = /data/keyring.$name

; monitors
[mon]
   ;Directory for monitor files
   mon data = /data/mon$id

[mon0]
   host = node0
   mon addr = 192.168.0.100:6789
[mon1]
   host = node1
   mon addr = 192.168.0.101:6789
[mon2]
   host = node2
   mon addr = 192.168.0.102:6789

; metadata servers
[mds]

[mds0]
   host = node0
[mds1]
   host = node1
[mds2]
   host = node2

; OSDs
[osd]
   ; osd data is where the btrfs volume will be mounted;
   ; it will be created if absent
   osd data = /data/osd$id
   ; osd journal is the regular file or device to be used for journaling
   osd journal = /dev/sdb2
   ; The 'btrfs devs' partition will be formatted as btrfs.
   btrfs devs = /dev/sdb1
   host = node$id

[osd2]
[osd3]
[osd4]
[ . . .}
[osd9]
```

Observe that by setting host = node$id, and by instantiating OSDs 2 through 9 (skipping 0 and 1), we can exploit the sequential hostnames and avoid explicitly setting the hostname for every OSD.

OSD journaling is more efficiently done on a raw partition than on a regular file. If you use a raw partition, setupuser should be a member of the disk group on the OSDs, so that it has write access to the device.

Next, create and start the file system as follows. The -allhosts flag makes each command work on all the nodes specified in ceph.conf, via SSH; therefore, you should have passwordless root SSH configured for each OSD. The root account's authorized_keys on each OSD should have the public key of setupuser. Once the public key is distributed, run the following commands:

```
./mkcephfs -c ceph.conf --allhosts --mkbtrfs
./init-ceph -c ceph.conf --allhosts start
```

-mkbtrfs formats the btrfs devs to Btrfs; if you wish to use another file system or regular directory instead, ensure that the resulting or underlying file system has extended attributes enabled (user_xattr for ext3).

Should you need to stop (./init-ceph -c ceph.conf -allhosts stop) and restart the Ceph services, the log directories from prior server sessions may cause a hanging failure on mount. Clearing them prevents this issue.

Mounting requires kernel support. Ceph support is in the Linux kernel as of version 2.6.34; for earlier kernel versions, see http://ceph.newdream.net/wiki/Building_kernel_client for instructions on building the Ceph kernel module. Mount as follows, using the IP address of one of the monitors:

```
mount -t ceph 192.168.0.100:/ /mnt/ceph
```

At this point, Ceph is usable like any other part of your local file system.

Using Hadoop with Ceph

There are two ways to use Ceph as the file system for Hadoop: (1) mounting Ceph as in the end of the previous section and using it as a local file system in Hadoop (file:///mnt/ceph); (2) patching the Hadoop Core with the patch available in the HADOOP-6253 JIRA [4]. This uses the user-level client of Ceph.

When starting up Hadoop, do not use bin/start-all.sh, as this will launch HDFS. Start up only the daemons you need (e.g., bin/start-mapred.sh for MapReduce).

RUNNING HADOOP ON CEPH THROUGH THE KERNEL INTERFACE

You can run Hadoop on Ceph via POSIX using Ceph's kernel interface. Using Hadoop on top of Ceph instead of HDFS requires two configuration tweaks in conf/core-site.xml (conf/hadoop-site.xml in 0.20.2):

```
<configuration>

  <property>
    <!--
    Note that in release 0.20.2 this name is fs.default.name;
    afterwards this is fs.defaultFS.
     ->
    <name>fs.defaultFS</name>
    <value>file:///mnt/ceph</value>
  </property>
</configuration>

<property>
  <name>hadoop.sharedtmp.dir</name>
  <value>/mnt/ceph/hadoop-tmp/hadoop-${user.name}</value>
</property>
```

Using Ceph through the above option will use Hadoop's Raw File System interface to communicate with Ceph as a local file system. There are optimizations that can be added to this interface in order to better leverage Ceph's performance. For example, locality information is not exposed to the Raw

File System interface; therefore Hadoop will not be able to schedule tasks close to the physical location of the data. These optimizations are being put into the HADOOP-6779 JIRA [17].

RUNNING HADOOP ON CEPH VIA JNI CODE

There is a second way to run Hadoop on Ceph which does not require the POSIX kernel interface—and hence does not require updating your cluster's kernels. You can instead use JNI code to connect Hadoop to Ceph in user-space. This feature is not yet in a Hadoop release; to use it, check out the Hadoop development trunk, and apply the patch from the HADOOP-6253 JIRA [4]. (See http://ceph.newdream.net/wiki/Hadoop_File system for more details.) Building Hadoop from the trunk source is outside the scope of this article.

Then add the following to conf/core-site.xml:

```
<property>
  <name>fs.defaultFS</name>
  <!--ip address of the mds here-->
  <value>ceph://<MDS_IPAddress:port></value>
</property>

<property>
  <name>fs.ceph.monAddr</name>
  <value><monitor_server:port></value>
  <description>The location of the Ceph monitor to connect to.  This
  should be an IP address, not a domain-based web address.</description>
</property>

<property>
  <name>fs.ceph.libDir</name>
  <value>/usr/local/lib</value>
  <description>The folder holding libceph and libhadoopceph</description>
</property>
```

After adding these configurations, Hadoop is ready to use with Ceph as its data store.

Summary

Since the Ceph kernel client was pulled into Linux kernel 2.6.34, interest in Ceph has greatly increased. Ceph is currently the only open source (LGPL licensed) parallel file system that offers a distributed metadata service that is linearly scalable to at least 128 metadata service nodes, supports the POSIX I/O API and semantics, and is able to expand and contract with low overhead without interrupting service.

The last point allows Ceph to be deployed in virtual environments such as Amazon's EC2 cloud service, where frequent and significant cluster size changes are the norm. Overall Ceph addresses a number of shortcomings of HDFS, i.e., HDFS's limited name-node scalability, its heartbeat overhead, and its highly specialized file access semantics.

As we write this, Ceph is still experimental and not yet ready for production environments. Sage Weil, Yehuda Sadeh, and Gregory Farnum are working full-time on making Ceph production-ready, with new releases coming out every 2 to 4 weeks. We hope this article will encourage people to participate in this effort by trying out Ceph with workloads they care about and reporting any bugs, performance problems, or bug/performance fixes.

REFERENCES

[1] Amazon, Simple Storage Service—Developer Guide (API Version 2006-03-01): docs.amazonwebservices.com/AmazonS3/2006-03-01/.

[2] Rajagopal Ananthanarayanan, Karan Gupta, Prashant Pandey, Himabindu Pucha, Prasenjit Sarkar, Mansi Shah, and Renu Tewari, "Cloud Analytics: Do We Really Need to Reinvent the Storage Stack?" USENIX HotCloud '09, San Diego, CA, June 15, 2009.

[3] Valerie Aurora, "A Short History of Btrfs," LWN.net, July 22, 2009: http://lwn.net/Articles/342892/.

[4] Gregory Farnum, "Add a Ceph File System Interface," ASF JIRA, May 2010: https://issues.apache.org/jira/browse/HADOOP-6253.

[5] International Committee for Information Technology Standards, SCSI Object-Based Storage Device Commands - 3 (OSD-3), project proposal for a new INCITS Standard T10/08-331r1, International Committee for Information Technology Standards, September 11, 2008.

[6] Hadoop Project, Hadoop Cluster Setup: hadoop.apache.org/core/docs/current/cluster_setup.html.

[7] Leslie Lamport, "The Part-time Parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, 1998, pp. 133–169.

[8] Konstantin V. Shvachko, "HDFS Scalability: The Limits to Growth," *;login:*, vol. 35, no. 2, 2010.

[9] Miklos Szeredi, "File System in User Space," 2006: http://fuse.sourceforge.net.

[10] Wittawat Tantisiriroj, Swapnil Patil, and Garth Gibson, "Data-intensive File Systems for Internet Services: A Rose by any Other Name..." Technical Report CMU-PDL-08-114, Parallel Data Laboratory, CMU, Pittsburgh, PA, October 2008.

[11] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, November 2006.

[12] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06)*, Tampa, FL, November 2006.

[13] Sage A. Weil, Andrew Leung, Scott A. Brandt, and Carlos Maltzahn. "Rados: A Fast, Scalable, and Reliable Storage Service for Petabyte-Scale Storage Clusters," *Proceedings of the 2007 ACM Petascale Data Storage Workshop (PDSW '07)*, Reno, NV, November 2007.

[14] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller, "Dynamic Metadata Management for Petabyte-Scale File Systems," *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, Pittsburgh, PA, November 2004.

[15] Brent Welch, "POSIX I/O Extensions for HPC," *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)*, December 2005.

[16] Tom White, "The Small Files Problem," February 2, 2009: http://www.cloudera.com/blog/2009/02/02/the-small-files-problem/.

[17] Alex Nelso, "Support for Ceph Kernel Client," ASF JIRA: https://issues.apache.org/jira/browse/HADOOP-6779, May 2010.