OctopusFS in Action: Tiered Storage Management for Data Intensive Computing

Elena Kakoulli
Cyprus University of Technology
elena.kakoulli@cut.ac.cy

Nikolaos D. Karmiris
Cyprus University of Technology
nd.karmiris@edu.cut.ac.cy

Herodotos Herodotou
Cyprus University of Technology
herodotos.herodotou@cut.ac.cy

ABSTRACT

The continuous improvements in memory, storage devices, and network technologies of commodity hardware introduce new challenges and opportunities in tiered storage management. Whereas past work is exploiting storage tiers in pairs or for specific applications, OctopusFS—a novel distributed file system that is aware of the underlying storage media—offers a comprehensive solution to managing multiple storage tiers in a distributed setting. OctopusFS contains automated data-driven policies for managing the placement and retrieval of data across the nodes and storage tiers of the cluster. It also exposes the network locations and storage tiers of the data in order to allow higher-level systems to make locality-aware and tier-aware decisions. This demonstration will showcase the web interface of OctopusFS, which enables users to (i) view detailed utilization information for the various storage tiers and nodes, (ii) browse the directory namespace and perform file-related actions, and (iii) execute caching-related operations while observing their performance impact on MapReduce and Spark workloads.

PVLDB Reference Format:
Elena Kakoulli, Nikolaos D. Karmiris, Herodotos Herodotou. OctopusFS in Action: Tiered Storage Management for Data Intensive Computing. *PVLDB*, 11 (12): 1914-1917, 2018.
DOI: https://doi.org/10.14778/3229863.3236223

1. INTRODUCTION

Over the past decade, a plethora of platforms have emerged to support data intensive computing, including Hadoop [2], Spark [9], Flink [1], and their growing ecosystems. Such systems have been optimized for running on large *clusters of commodity hardware* and processing data residing on *distributed file systems* such as HDFS [8]. Commodity machines have seen significant improvements over the years in terms of memory, storage devices, and network technologies, which have been exploited recently in order to meet the increasing data storage and I/O demands of modern large-scale data analytics. New data-processing systems are utilizing memory or SSDs for primary storage [9] or for actively caching data [7], while others are using local disks for caching data from remote or cloud storage [6].

Whereas past work is exploiting storage tiers in pairs or for specific applications, OctopusFS [4] offers a comprehensive solution to managing multiple storage tiers in a distributed setting. Specifically, *OctopusFS is a novel distributed file system that is aware of the underlying storage media (e.g., memory, SSDs, HDDs) with different capacities and performance characteristics.* Hence, data can be distributed and stored on one or multiple storage tiers, enabling efficient support of diverse workloads (e.g., batch processing, iterative, interactive) executed from higher-level systems. OctopusFS offers a spectrum of usage patterns ranging from fully automating data management to providing explicit data placement and retrieval control to users and applications.

The system *automates data management* in terms of how to replicate and distribute data both across nodes and across tiers for increasing throughput and cluster utilization. It includes a variety of pluggable policies for automating data placement, retrieval, and caching across the storage tiers and cluster nodes. The policies employ multi-objective optimization techniques for making intelligent data management decisions based on the requirements of fault tolerance, data and load balancing, and throughput maximization.

At the same time, the *storage media are explicitly exposed* to users and applications, allowing them to choose the distribution and placement of replicas in the cluster based on their own performance and fault tolerance requirements. Hence, higher-level processing systems can build their own types of automation in order to improve their efficiency and effectiveness in analyzing large-scale data. For example, systems such as Hadoop and Spark can improve their task scheduling, Hive and Pig can improve their query processing, while Oozie and Airflow can improve their workload management.

Contributions and Demo: We demonstrate OctopusFS, a multi-tier distributed file system built via making significant modifications and additions to HDFS, one of the most widely used file systems in cluster deployments [8]. Yet, OctopusFS remained backwards compatible with HDFS and can, therefore, be used as its drop-in replacement. Our demonstration of OctopusFS aims at (i) introducing the OctopusFS architecture and methodology through a system implementation, and (ii) demonstrating the immediate benefits of using OctopusFS with (unmodified) data-intensive processing systems, such as Hadoop and Spark, in terms of both increased performance and better cluster utilization. We visually demonstrate the behavior of its core compo-

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236223

nents in a range of scenarios, giving the audience members a complete visual insight into the behavior of OctopusFS. In particular, the audience will have the ability to interact with the system through a graphical web interface for:

- viewing detailed information about the utilization of the different storage tiers and nodes, while workloads are executing on the cluster;
- browsing the directory namespace of the file system, uploading and downloading files, viewing file placement information, and modifying the file replication;
- performing caching-related operations such as adding, modifying, and deleting cache directives, while observing their impact on MapReduce and Spark workloads.

2. TIERED STORAGE MANAGEMENT

OctopusFS [4] facilitates scalable and efficient data storage on compute clusters by utilizing directly-attached memory, SSDs, HDDs, and remote (network-attached or cloud) storage. The main operations of the system are to store and retrieve *files* broken into blocks, which are distributed across nodes and storage tiers, and replicated for fault tolerance.

2.1 System Architecture

OctopusFS uses a multi-master/slave architecture shown in Figure 1 (similar to HDFS [8]) that consists of multiple *Primary* and *Backup Masters*, *Workers*, and *Clients*.

Primary Masters: Each Primary Master maintains two metadata collections, the directory namespace and the block locations. The directory namespace contains a traditional hierarchical file organization and offers typical operations like opening, closing, renaming, and deleting files and directories. The file content is split into large blocks (128MB by default) and each block is independently replicated at multiple Workers. In addition, each Primary Master is responsible for maintaining the mapping of file blocks to Workers and storage media. To scale the name service horizontally, there are multiple Masters that form a federation and are independent from each other.

Backup Masters: Each Backup Master maintains an in-memory, up-to-date image of the directory namespace of some Primary Master and is standing by to take over in case the Primary fails. Moreover, it is responsible for periodically creating a checkpoint of the namespace metadata so that the system can restart from the most recent checkpoint quickly.

Workers: The main responsibilities of the Workers include storing and managing the file blocks on the various storage media they contain, as well as serving read and write requests from the file system's Client. The Workers are typically run one per node in the cluster and perform the block creation, deletion, and replication upon instructions from the Masters, in a similar way as HDFS [8]. The storage media available across the Workers are logically grouped into *storage tiers* based on the I/O characteristics of the devices. Specifically, a storage tier (e.g., the "SSD" tier) will encompass all Workers in a cluster that are associated with the same storage media type (e.g., SSDs) of similar performance. Figure 1 shows an example of four storage tiers, namely "Memory", "SSD", "HDD", and "Remote". The file blocks can be replicated and stored in one or more storage tiers, based on requests from the Client or pluggable management policies. For example, a block may have 3 replicas

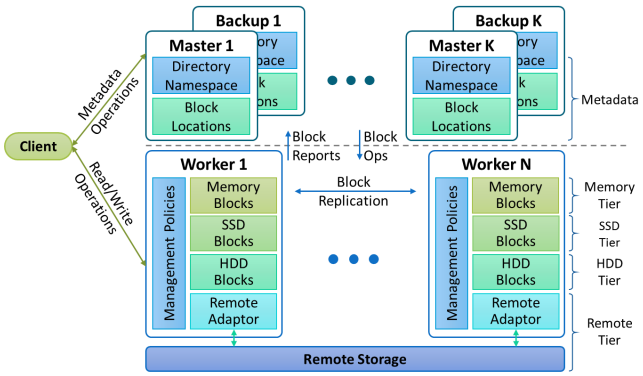

Figure 1: OctopusFS architecture configured with four storage tiers

on the "HDD" tier (on 3 different nodes); or it may have 1 replica on each of the "Memory", "SSD", and "HDD" tier (on 1, 2, or 3 different nodes); or any other combination. Hence, users and applications have tremendous flexibility on how to place and move data in the file system.

Clients: The Client exposes APIs for all typical file system operations like reading, writing, and deleting files, as well as creating and deleting directories. In addition, the APIs expose locality and storage-media information to applications, enabling the scalability and performance benefits of compute-data co-location. In order to enable tiered storage, OctopusFS expanded the Apache Commons FileSystem API v2.7.0 with minimal extensions, as presented in [4].

2.2 Replication and Cache Management

The management of files with respect to the storage tiers is achieved through the notion of the *replication vector*, which specifies the number of replicas for each storage tier. The representation of the replication vector is V = <M, S, H, R, U>, which is a shorthand notation for the tiers <"Memory", "SSD", "HDD", "Remote", "Unspecified">. For example, the replication vector V = <1, 2, 0, 0, 0> for a file F indicates that F has 1 replica in the "Memory" tier and 2 replicas in the "SSD" tier. The special entry "U" in the replication vector indicates the number of replicas to be placed on *any* storage tier. Thus, by using the replication vector mechanism, OctopusFS enables the *full spectrum of choices* between the user explicitly setting all storage tiers and the system automatically selecting the storage tiers.

The replication vector can be specified at file creation and modified (via the API or the web interface) to achieve various functionalities, such as moving a file between tiers, copying a file between tiers, modifying the number of replicas within a tier, and deleting a file from a tier. Note that each time the replication vector of a file changes, a network-aware and tier-aware placement policy is invoked for deciding where the addition or deletion of a replica will take place. Finally, the Client exposes the Workers and storage tiers of the block replicas as well as useful information per tier (e.g., total/remaining capacity, read/write throughput, etc.), all of which are also presented in the web interface.

Furthermore, OctopusFS provides explicit multi-tier cache management, allowing users and applications to cache files (i.e., create extra replicas) to higher storage tiers such as "Memory". This feature is important for various scenarios such as caching frequently used data or the working set of

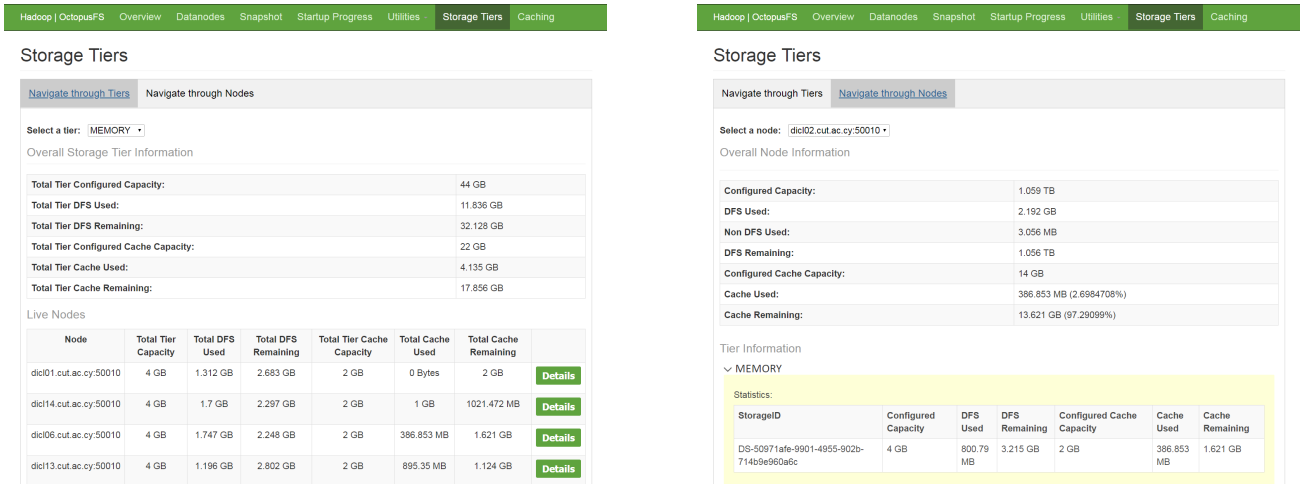

Figure 2: Screenshots of the *Storage Tiers* view presenting storage utilization organized at the level of (A) a single storage tier and (B) a single node

a high-priority workload. For example, Pegasus [5], a graph mining system, can cache the relevant graph data needed by an iterative graph application during its duration, leading to great performance benefits [4]. The Client can be used to issue *cache directives* containing the file or directory to be cached along with a replication vector specifying the caching tiers. Cache directives are grouped into *cache pools*, which are used for resource management and permission checking.

2.3 Data Placement and Retrieval

OctopusFS proposes automated placement and retrieval policies for improving the I/O performance of the cluster. An application or a user adds data to OctopusFS by creating a new file and writing the data to it using the Client. At file creation, the Client can optionally specify a block size as well as a replication vector. The Client then writes the data one block at a time. Upon block creation, the Client first contacts the Master and obtains a list of locations (i.e., Worker and storage tier pairs) that will host the replicas of that block. This list is determined using a *pluggable block placement policy*. The default one (explained in [4]) makes placement decisions by formulating a *multi-objective optimization problem* and solving it to find a Pareto optimal solution that optimizes four objectives: (i) fault tolerance for avoiding data loss due to corruption or failures; (ii) load balancing for distributing I/O requests across storage media; (iii) data balancing for distributing data blocks across storage media; and (iv) throughput maximization for optimizing the overall I/O throughput of the cluster.

An application or a user accesses data from OctopusFS via the Client, which contacts the Master and obtains a list of locations (i.e., Worker and storage tier pairs) to read from. This list is ordered using a *pluggable block retrieval policy*. The default one (presented in [4]) takes into consideration the Client location, the replica locations, the network topology, the average data transfer rates from each Worker, the read throughput rate of each media, the number of active network connections per Worker, and the number of active I/O connections per storage media. The Client then contacts the first Worker directly and requests the transfer of the desired block. In case of a read failure, the Client contacts the next Worker on the list for reading the data.

3. DEMONSTRATION PLAN

The demonstration will use the web interface of OctopusFS, which is an extension of the HDFS web interface. Hence, users accustomed to HDFS will find the OctopusFS interface familiar and easy to use. In addition to general information about the cluster utilization and health already provided by the HDFS interface, the new interface presents information related to storage tier management and supports new functionalities exposed by OctopusFS. Users can use the new web interface to (i) view detailed information about the utilization of the different storage tiers and nodes in the *Storage Tiers* view; (ii) browse the directory namespace of the file system and perform file-related actions in the *Browse Directory* view; and (iii) perform caching-related operations such as adding, modifying, and deleting cache directives in the *Caching* view. For the purposes of the demonstration, OctopusFS will be running on our 12-node in-house cluster. Hadoop and Spark workloads from the HiBench benchmark [3] and Pegasus [5] will be executed on the cluster so that the audience can experience the behavior of OctopusFS and get a better understanding of its benefits. At the same time, a poster will be used to introduce the audience to the OctopusFS architecture and inner workings.

3.1 Monitoring Storage Tier Utilization

The *Storage Tiers* view is important for users, especially system administrators, to monitor the storage utilization of the various storage tiers and nodes. The provided information includes total, used, and available storage capacity as well as total, used, and available cache capacity, presented at different granularities. In particular, the storage utilization can be viewed from two different perspectives: one organized at the level of storage tiers as shown in Figure 2(A) and one organized at the level of Worker nodes as shown in Figure 2(B). In the first case, users will see aggregate information for each storage tier and for each node containing storage media of that tier. It is also possible to drill down to individual storage media of a particular storage type for each node (notice the "Details" button in Figure 2(A)). In the second case, users can see storage utilization aggregated for each node available in the system as well as for each storage media of each tier on that node.

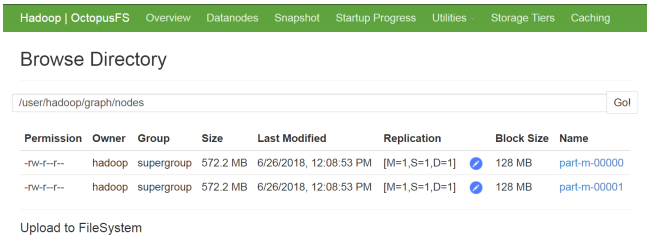

Figure 3: Screenshot of the *Browse Directory* view

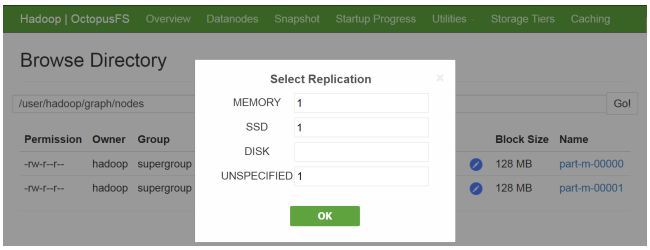

Figure 4: Screenshot of the *Browse Directory* view when modifying the replication vector of a file

In this part of the demonstration, we plan to execute a write-intensive workload, while varying the size of the generated datasets. We will then navigate the *Storage Tiers* view in order for the audience to observe how the different storage tiers and nodes are utilized for storing the data. In this way, the audience will get a better understanding of how our automated data placement policy distributes and replicates data to the available storage media.

3.2 Browsing the File System Directory

The *Browse Directory* view, shown in Figure 3, enables users to browse the directory namespace of the file system. For each directory and file, the user can view typical metadata information such as permission settings, owner, group, and last modification timestamp. In addition, for each file the user can view its total size, block size, and replication vector showing the storage tier of each replica. A link is available for each file for drilling down to block-level information, such as the actual storage media each block is stored or cached on. Finally, users can modify the replication vector of a file and either (i) specify new tier locations for the replicas; or (ii) specify the total number of replicas in the "Unspecified" location and let OctopusFS decide the tiers; or (iii) specify both some explicit tier locations and a number of "Unspecified" locations (see Figure 4).

Continuing with the scenario of the write-intensive workload from Section 3.1, the audience will also observe how the data placement policy sets the replication vector for various files; thus getting a deeper understanding of this automated policy. In addition, the audience will be given the opportunity to modify the replication vector of files before running read-intensive workloads. The audience will then directly experience the effect that the replica locations can have on the performance of the workload as well as get a better understanding of how the data retrieval policy operates.

3.3 Performing Caching Operations

The *Caching* view presents all cache-related information and exposes all functionalities provided by the cache management feature of OctopusFS, shown in Figure 5. In particular, the user is able to add, modify, or delete both cache pools and cache directives. Each cache pool is given a name, ownership and permission restrictions, as well as cache size limits for each storage tier that it supports. Each cache directive is placed in a cache pool and is given (i) an optional expiration time in the future (after which it is removed from the cache pool) and (ii) a replication vector indicating the storage tiers to be cached on.

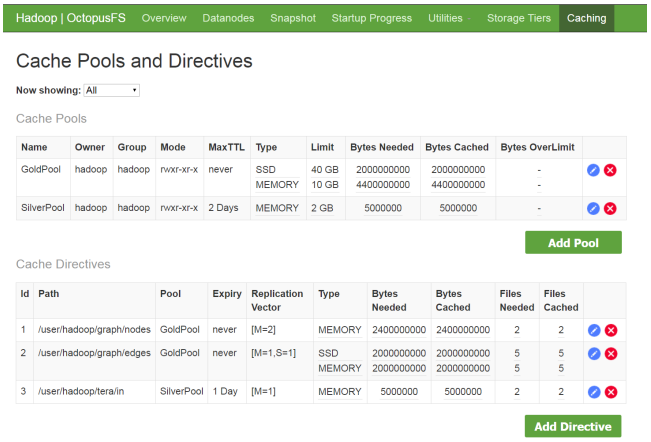

Figure 5: Screenshot of the *Caching* view

For this demonstration part, the audience will be able to interact with all cache management features of OctopusFS such as adding and removing cache pools and directives. At the same time, read-intensive and iterative workloads will be executed on the cluster so that the audience can experience first hand the impact of caching to the run-time performance of the workloads. The MapReduce and Spark applications will be monitored using the existing YARN and Spark Web UIs, respectively.

4. REFERENCES

[1] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, et al. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE TCDE*, 36(4), 2015.

[2] *Apache Hadoop*, 2018. https://hadoop.apache.org.

[3] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis. In *Proc. of ICDEW*, pages 41–51. IEEE, 2010.

[4] E. Kakoulli and H. Herodotou. OctopusFS: A Distributed File System with Tiered Storage Management. In *Proc. of SIGMOD*, pages 65–78. ACM, 2017.

[5] U. Kang, C. E. Tsourakakis, and C. Faloutsos. PEGASUS: Mining Peta-scale Graphs. *Knowledge and Information Systems*, 27(2):303–325, 2011.

[6] M. Mihailescu, G. Soundararajan, and C. Amza. MixApart: Decoupled Analytics for Shared Storage Systems. In *Proc. of FAST*, pages 133–146. USENIX, 2013.

[7] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, et al. The Case for RAMCloud. *Communications of the ACM*, 54(7):121–130, 2011.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proc. of MSST*, pages 1–10. IEEE, 2010.

[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proc. of NSDI*, pages 15–28. USENIX, 2012.