Distributed Cloud Storage Services with FleCS Containers

Hobin Yoon, Madhumitha Ravichandran, Ada Gavrilovska, Karsten Schwan
Center for Experimental Research in Computer Systems
Georgia Institute of Technology

Abstract

There are limits to the ability to migrate or deploy applications across geographically distributed/loosely coupled cloud resources, requiring substantial data movement and/or uniformly visible and accessible storage services across such distributed infrastructure. To address these issues, we propose and explore the utility of *FleCS* – an approach for providing FLExible Cloud Storage services in distributed systems. FleCS provides *storage containers* as a cloud-level abstraction that uniquely identifies a subset of storage resources and their associated *attributes*. Attributes determine certain container properties, including those concerning data replication and consistency, thereby creating opportunities to pay those costs only for the state/data which require them. FleCS exports to cloud applications an object-based storage API that allows them to request the 'right' types of storage, and to correspondingly group/classify their data. Sample uses go beyond the established notions of application-provided or derived hints to classify the 'hotness/coldness' of data and/or to provide better energy-efficient storage services, to also include application-specific notions of data consistency and update strategies.

FleCS and several types of storage containers are realized for a prototype platform consisting of groups of nodes, virtualized with the Xen hypervisor, with distinct storage targets, each managed by a separate NFS server. Evaluations use benchmarks based on popular cloud applications. A future target platform for evaluation is a distributed OpenCirrus cloud infrastructure spanning multiple data centers.

1 Introduction

Today's cloud stacks are typically configured to provide uniformly visible storage services to VMs running on any node in the cloud platform. Internally, this property may be enabled through use of centralized storage services and by file systems such as NFS, or achieved on top of distributed storage services using multiple physical storage targets [7], including across individual disks present on each datacenter server node, and distributed file system layers like HDFS [2]. This view simplifies storage management services and the various management tasks that rely on them, such as VM migration. However, it requires cloud applications to tolerate variable storage access latencies and bandwidths, creates potential unfairness in terms of storage accessibility across different applications, and prevents cloud platforms from scaling across geographycially distributed locations.

Our research advocates an approach in which cloud applications are provided with explicit knowledge and guarantees about the storage resources they desire to use. Specifically and in order to address the diversity in the requirements of different applications and management policies, this paper presents and explores the utility of *FleCS* – an approach to providing FLExible Cloud Storage services in distributed cloud environments. FleCS implements the abstraction of *storage containers*, each of which uniquely identifies a subset of storage resources within a single namespace, similar to the 'buckets' in Amazon's S3 storage service. The contents of a container are uniformly visible to all VMs/applications with adequate permissions. However, containers can differ from each other – as described by attributes associated with them – in the access guarantees and properties they provide to applications. Container attributes are implemented by policy plugins associated with containers, such as those realizing data replication and consistency actions. Finally, applications can use private or shared containers, and use multiple containers in order to differentiate across the different types of data they use.

To illustrate the use of containers, consider for instance, a social networking application, like the one represented via the event calendar benchmark Olio. Multiple globally distributed user groups may use this service, therefore it is important to distribute the corresponding VMs across globally distributed cloud platforms. With FleCS, this application may use multiple instances of 'private' or 'local' storage containers that capture events private to a certain corporation or a certain social group, and instances of 'shared' or 'replicated' containers for events visible to others. 'Local' containers are confined to the storage resources of individual cloud platform, whereas 'shared' ones are replicated across storage resources in each of the distributed clouds. These both im-

proves the efficiency of storage access, while also allowing us to deal with the slower and more limited cross-cloud network links.

Containers permit applications to specialize storage for the different types of data they use and consequently, to efficiently operate across diverse storage resources, including in distributed clouds spanning different datacenters. Containers support such diversity via *policy plugins* that can implement general or application-specific methods to cope with physical constraints like limited storage access bandwidths or highly variable access latencies [11]. Plugins, therefore, enrich containers to provide applications with desirable properties, one example being the use of replication to provide higher levels of availability, another being improved energy efficiency through storage consolidation [2] or even by migrating computations to where energy costs are currently lowest.

Containers can be enriched in arbitrary ways. They may provide cross-data center storage functionality by fully mirroring all state across cloud boundaries, so as to provide services for disaster recovery. They may use policies that carefully place cloud state for shared use by multiple virtual machines across multiple storage targets [6, 1]. They may even be built on top of distributed cloud file systems [7]. However and in contrast to such approaches, containers operate at a higher level of abstraction, in ways that can be visible to applications, so that they can avoid the potential costs incurred by lower level approaches that may have to move large volumes of potentially unnecessary data across inter-cloud links, or that may apply replication and consistency actions to applications that does not require them. Further, containers can be constructed to take advantage of prior work on mechanisms that specialize storage services to certain sharing and consistency requirements [4], or to better support certain management objectives like energy-efficiency [2], including via use of application-level hints [9]. Finally, containers are similar to existing work in that they offer higher-level, non-POSIX compliant storage APIs [7, 9, 4, 5]

To use FleCS containers, applications must explicitly request the 'right' types of storage and then use such storage with appropriate types of data. This generalizes upon the notions of application-provided or derived hints to classify the 'hotness/coldness' of data and provide more energy-efficient storage services [9], application-provided handlers for customized key-value stores [5] or the use of modified application-specific APIs [4] used to otherwise establish data groupings. Toward this end, FleCS exposes to cloud applications an object-based file system interface, similar to other object-based storage interfaces [3, 12], including popular cloud stores such as Amazon's S3 storage service, which allows placement of entire objects, not low-level storage blocks, in specific storage containers. Internally, the current implementation of FleCS containers uses a standard file system to represent objects, using a one-to-one mapping of objects to files.

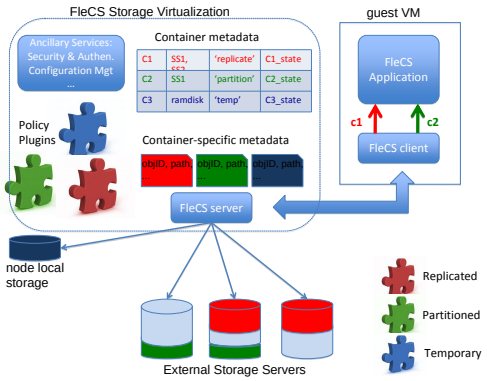

Figure 1: FleCS architecture.

Summarizing, this paper makes the following contributions:

- We introduce *FleCS* – an architecture for supporting flexible and diverse storage services in virtualized cloud environments – and its accompanying *typed storage container* abstraction and API, which allow applications to express data groupings and classify data that need to be handled in a certain manner. This is particularly important in distributed cloud environments, where the costs of uniformly making all data visible and accessible from all nodes in the (potentially geographically) distributed cloud infrastructure can be prohibitive.
- We describe our prototype realization of FleCS and several types of storage containers (e.g., temporary, replicated, partitioned) on a platform consisting of groups of nodes virtualized with the Xen hypervisor and with multiple and separate storage targets, each managed by a separate NFS server.
- We demonstrate the feasibility and utility of the FleCS approach by evaluating its overheads and showing opportunities for improved operating costs and efficiency, for different types of containers, and with benchmarks using the FleCS API.

2 FleCS Architecture

Figure 1 illustrates the components of the FleCS architecture. Each node in a cloud environment is running a FleCS server, deployed as part of the node-level virtualization layer – e.g., in dom0 in Xen. Guest VMs interact with the FleCS server via a VM-resident FleCS client, exporting to guest applications the FleCS API. Applications, running in VMs, use the FleCS API to request storage containers with a certain property by specifying one of multiple supported attributes. The current model assumes that storage containers with different attributes are made available to cloud users as different storage services provided by the cloud platform, and details regarding their implementation are hidden from VMs.

The FleCS server relies on one or more policy plug-

ins, each of which specifies the implementation of the FleCS operations in a manner that is consistent with the attribute(s) described by the policy. For instance, a plugin for a container that provides "reliability" may rely on replication to realize the attribute/property. As a result, creating this type of container will result in physical storage allocation on multiple physical storage servers (perhaps even the distribution/distance of those physical locations is further specified by the target attribute), and will enforce execution of appropriate consistency protocol with each put/get access to container data, again as specified by the policy plugin. Stated technically, policy plugin provide concrete implementations for the FleCS API, including any necessary translations from the FleCS abstracts to those required by the underlying policy implementation.

The model can support existing cloud solutions as well, for instance via a 1-to-1 mapping exists between a VMs disk image and a FleCS container with a "persistent" property. Each VM may use one or more containers, and each container may be used by one or more VMs/applications.

FleCS Containers. The storage containers are the main storage abstraction in FleCS. Similarly to the Amazon's S3 'buckets', they represent a single namespace containing data objects. In our current prototype there is one-to-one mapping between these data objects and files, but our next steps are relaxing this requirement, for instance to better deal with files of arbitrarily large sizes. A key distinction of the FleCS containers, is that we associate with them properties – attributes – that go beyond the simple 'availability zone' notion provided in S3, which indicates high-level information regarding the grouping of the physical storage backing a certain bucket.

Each container is uniquely identified with its identifier. Support for user-readable names will be added in the future; the current prototype uses numeric container identifiers only. Internally, FleCS maintains several types of information regarding each container, including the plugin policy associated with the container attribute, provenance and access rights, policy-specific information regarding physical storage involved in the realization of the container, policy-specific metadata (container metadata in Figure 1), etc. There may be more than one container of certain type, each of which may be allocated on different physical resources.

For each container accessed by any of the VMs deployed on a cloud node, FleCS creates an instance for the corresponding policy plugin (currently accomplished through use of separate threads). Internally, each instance maintains its own policy-specific metadata needed for locating objects and performing actions related to data placement, consistency and update strategies, necessary to maintain the desired attribute.

Figure 1 illustrates several examples of storage containers. These include (i) 'temporary' – e.g., mapped to node-local storage or even tmpfs, and useful for maintaining volatile state, (ii) 'replicated' across different physical storage servers, e.g., for critical state for which we require greater reliability, where replication with greater consistency is needed, or even for shared state being accessed from different sites, where replication techniques relying on weaker/eventual consistency methods may be acceptable, and (iii) 'partitioned' across multiple physical servers, e.g., for state which requires high-availability and scalability. In these examples, the policy plugin specifies the manner in which physical cloud storage is allocated for different containers. Other types of containers may specify other functionality, as described further below in this section.

objects	opaque data objects
container	attribute, size
object access operations	put and get, delete
container operation	create, load, list, delete
container metadata	ID, name, policy, storage servers
attributes	local, replicate-*, partition

Table 1: FleCS API elements.

FleCS API. Table 1 shows the basic components, their features and supported operations, which constitute the FleCS API. Containers are created by supplying to the create operation one of several storage attributes provided by the cloud infrastructure. The resulting unique identifier is used to tag any subsequent container accesses, based on which they are routed to the appropriate plugin instance in the FleCS server. FleCS exports to VMs an object-based API, represented via put and get operations on data objects. Each container represents a single name space, and objects are uniquely identified within a container. Internally, the implementation of the container may use different identifiers, and it is the plugin's responsibility to implement the necessary translations from the FleCS object identifier to the container-internal object identifier (e.g., server IP address and pathname). In our current implementation, objects correspond to files stored on one or more NFS servers, and each of the container instances translate the object ID to the corresponding NFS pathname. Similarly, operations on objects currently operate on entire file boundary. The object maximum size is a configurable parameter.

Benefits of FleCS containers. The ability to group cloud storage into semantically meaningful groups, creates opportunities to enhance the cloud-level storage management tasks. For instance, substantial research has been focused on determining optimal, or improved data placement in distributed environments where multiple physical servers are shared and accessed from multiple distinct locations [8, 1, 6]. With FleCS, such methods can be made more efficient, by specifically applying them to those containers – i.e., data and state – which require them. Similar arguments apply to the opportunities to simplify other storage management methods, such as QoS and performance isolation, or those concerned

with matching properties of the storage service, such as replication degree, to the workload requirements or load levels [5, 10, 9].

FleCS creates opportunities to improve the cost and the efficiency of cloud-level storage management tasks, and to permit the support of richer storage management policies. We make these claims due to multiple factors. First, by exporting the availability of different types of storage services to cloud applications, FleCS allows applications to specify and explicitly use distinct types of storage – i.e., different containers. In this manner, applications provide *hints* regarding the management tasks which are suited for their storage needs. The benefits of using application-provided information to improve system behavior are well-understood, and have been demonstrate in context ranging from operating system and networking services, to management of large-scale systems, and beyond In addition, such *hints* may be necessary to express certain application-specific constrains, such as regulatory policy which restrict data placement locations for financial or health applications.

Next, by grouping state with different storage requirements or properties, FleCS makes it more feasible to develop storage management policies which would otherwise rely on the ability on "learn" and maintain, dynamically and in a black-box manner, some information regarding the VM access patterns and needs. Examples include policies requiring isolation guarantees to be provided across different datasets, such as when certain datasets contain more important or more frequently access data. In order to provide different timing and performance for map-reduce tasks accessing the "hot" vs. "cold" dataset [2], needed are runtime methods to identify the different type of data and the applications that use them. Although related research is focused on developing such methods, the FleCS approach can simplify the problem and lead to more effective and lower overhead mitigation and management techniques that reduce such 'heat'. Similarly, distributed cross-cloud applications, such as the event calendaring application described in Section 1, can benefit from use of containers labeled as "private" or "shared", where the replication overheads are incurred for accesses to contents of the "shared" container only.

In summary, FleCS creates opportunities to pay additional storage costs – in terms of physical storage resources, runtime overheads of storage accesses, or dollars – only for the state/data which require them. As a result, it not only improves the efficiency of different types of storage services, but it may also eliminate critical overheads from service implementations, which would otherwise be rendered useless.

3 Implementation

We have implemented a prototype of the FleCS architecture for platforms virtualized with the Xen hypervisor. The FleCS server is implemented as a multi-threaded user-level process in dom0, with separate threads managing the accesses to separate containers, as specified by the policy plugin corresponding to the container attribute. The FleCS client is implemented as a user-level library, and, in the current implementation, the interaction between the client and the server process in dom0 is performed via TCP sockets. Currently, we support plugins for the three types of containers shown in Figure 1 – 'temporary', 'replicated', and 'partitioned'. Each container is uniquely identified via the Xen ID of the VM requesting the container creation. Finally, we currently make simplifying assumptions regarding the metadata management for containers, and assume that it is replicated at each node.

Our future extension of this implementation will improve the efficiency of the VM-FleCS interactions, allow support for human-readable container names, provide support for scalable metadata operations, e.g., via use distributed hash tables and caching, and add mechanisms for access control and configuration management (as shown in Figure 1).

Finally, our prototype cloud platform is a very simplistic one. A future target platform for evaluation is a distributed OpenCirrus cloud infrastructure spanning multiple data centers.

4 Initial Results

Testbed. We next present the results for the preliminary evaluation of the feasibility and utility of the FleCS approach described in this paper. The results are gathered on a small datacenter prototype consisting of four Xeon nodes, virtualized with the Xen 3.0.3 hypervisor, running RHEL 5.6, kernel version 2.6.18. Two nodes serve as NFS servers, each with a 6-disk RAID5 storage array with 400GB HDDs. All components are interconnected via 1Gbps Ethernet. To emulate wide-area delays for access to remote storage, we insert configurable delays on one of the FleCS server - NFS server datapaths.

Feasibility. First, we compare the costs (i.e., latency) of the `put` and `get` operations of the different types of FleCS containers currently supported in our prototype, with a baseline case corresponding to direct access to one of the NFS server managing one storage array. The benchmark application running in a guest VM performs repeated accesses to objects/files of different sizes, and measures the latencies associated with each data size. In all cases we disable client-side caching and ensure I/O operations interact with the storage servers. Figure 2 shows the results of these measurements. First, we observe that containers can be efficiently realized – the replicated and partitioned containers provide similar I/O access properties as the NFS-based baseline case. Furthermore, FleCS creates opportunities to seamlessly use different type of storage services, including those accessing local disks, such as for containers holding temporary,

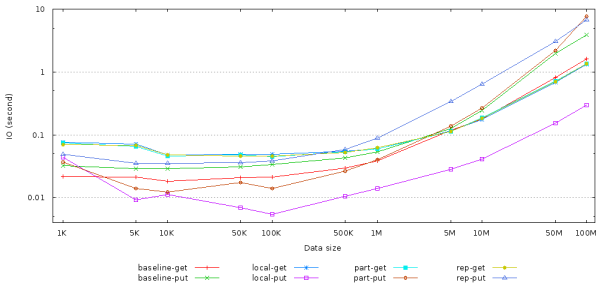

Figure 2: Feasibility of FleCS containers.

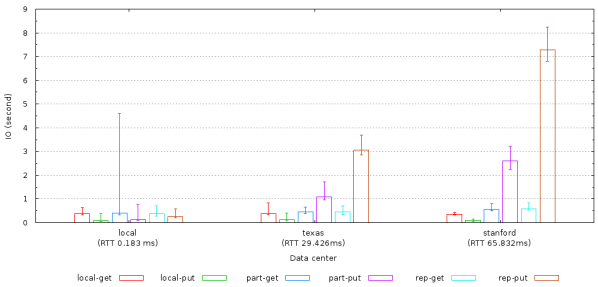

Figure 3: FleCS containers in distributed datacenters.

volatile data. Concretely, in these experiments, we realize temporary containers via tmpfs, which may be suitable for certain temporary data. This has the potential of resulting in significant performance improvements for certain types of workloads, and we plan to further investigate this in the future.

Clearly, the exact performance levels provided by various types of containers vary based on the network latencies associated with accessing the storage targets involved in the container realization. To show these effects we artificially introduce delays on the I/O path for one of the storage targets available in our system, so as to model accesses to remote datacenters – specifically using delays measured when pinging servers at UT Austin and Stanford University. Figure 3 demonstrates that the effects of the remote access have linear impact on the I/O performance provided by the given container, as expected, and that FleCS does not introduce any hidden overheads.

Utility. In order to demonstrate the utility of FleCS, we conduct a simple experiment evaluating the performance of multiple map-reduce word-count applications implemented for FleCS, operating across two different data sets, stored in two separate containers. One application uses dataset 1, the remaining applications access dataset 2 only. Due to the small size of our testbed, we cannot create sufficient number of map-reduce VMs that create contention for the storage resource. Therefore we also use a 'load generator' VM running the fio benchmark, which performs random accesses to dataset 2 only. The results compare the performance observed in the following scenarios: (1) both datasets are placed in two containers stored at different storage servers, (2) dataset 2 is realized with a 'replicated' FleCS container, and distributed across both storage targets, and (3) both datasets are replicated with the same degree of replication, which is common for current cloud storage solutions which do not differentiate between the different storage needs. The goal of the experiment is to demonstrate that with FleCS we can achieve performance improvements while limiting the storage costs and complexities only to those workloads which require them. The final version of this paper will include the results from these measurements.

5 Conclusions and Future Work

This paper presents FleCS and its associated abstractions, and explores their utility in supporting diverse and flexible cloud storage services. Through use of storage containers, and their associated attributes, FleCS provides applications with explicit knowledge about the properties of the storage resources they desire to use, and, in turn, uses that knowledge to maintain such properties in a manner which limits the costs only to data/state that require them. FleCS enables realizations of range of storage services with acceptable costs, include those targeting cross-cloud storage interactions. We are continuing to evolve the FleCS prototype and to develop and experiment with realistic applications and use cases, which we hope to include in the final version of this paper.

References

[1] S. Agarwal, J. Dunagan, et al. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *NSDI*, 2010.

[2] H. Amur, J. Cipar, et al. Robust and Flexible Power-Proportional Storage. In *SOCC*, 2010.

[3] A. Azagury, V. Dreizin, et al. Towards an object store. In *11th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003.

[4] S. Das, D. Agrawal, and A. E. Abbadi. G-Store: A Scalable Data Store for Transactional Multi key Access in the Cloud. In *SOCC'10*, 2010.

[5] R. Geambasu, A. A. Levy, et al. Comet: An active distributed key-value store. In *OSDI*, 2010.

[6] A. Gulati, C. Kumar, et al. BASIL: Automated IO Load Balancing Across Storage Devices. In *FAST'10*, 2010.

[7] J. G. Hansen and E. Jul. Lithium: Virtual Machine Storage for the Cloud. In *SOCC'10*, 2010.

[8] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained Mirror Placement on the Internet. In *INFOCOM*, 2001.

[9] R. Kaushik, L. Cherkasova, et al. Lightning: Self-Adaptive, Energy-Conserving, Multi-Zoned, Commodity Green Cloud Storage System. In *HPDC'10*, 2010.

[10] L. Princehouse, H. Abu-Libdeh, and H. Weatherspoon. RACS: A Case for Cloud Storage Diversity. In *SOCC'10*, 2010.

[11] Y. Saito, B. Bershad, and H. Levy. Manageability, Availability, and Performance in Porcupine: A Highly Scalable Cluster-based Mail Service. In *SOSP*, 1997.

[12] F. Wang, S. A. Brandt, et al. OBFS: A File System for Object-Based Storage Devices. In *Conf on Mass Storage Systems and Technologies*, 2004.