# Introduction

MarFS is a file system that presents a near-POSIX interface to the user, but whose data is stored efficiently into both POSIX File Systems and Object Storage Systems

More and more, data dominates the computing world. There is a “sea” of data out there in many different formats that needs to be managed and used. “Mar” means “sea” in Spanish. Thus, this product is dubbed MarFS, a file system for a sea of data.

## Rationale

Many may question why a new effort like MarFS is necessary. While there are products that are developing, none provide a scalable near-POSIX interface with adequate performance, cost, and efficiency, at this time.

Current object storage systems use erasure based disk systems to store the data, which is a positive thing. RAID solutions, such as RAID-6, are not adequate data protection for truly massive scale. Products such as Cleversafe, Scality, and EMC ViPR are moving towards the “sea of data” concept where data can have multiple personalities including POSIX, Object, and HDFS. Currently, these object storage systems are immature and don’t support efficient near-POSIX interfaces. MarFS assumes you want a first class near-POSIX interface to your files. MarFS is trying to be the best of both worlds, allowing data scaling like an object storage system, metadata scaling like N POSIX name spaces, and multiple kinds of access to the same data (Posix, Object, HDFS, etc.), the true “sea of data” concept. In time, it is certainly possible that other products/efforts will fill MarFS’s role.

It is possible to put object storage systems under scalable file systems like GPFS using a block interface over the object storage system, but the block write patterns of these PFSes (parallel file systems) are not well suited to benefit from these object storage systems’ high performance. MarFS will be able to use any object storage system, including potentially using cloud-based services, as a back end storage repo.

The team has investigated existing open source projects, and there doesn’t appear to be one that provides the needed functionality. Ceph provides a file system on objects, but isn’t known for scaled out metadata service. GlusterFS is probably the closest thing to an alternative, and indeed GlusterFS can be a global name space combining multiple file systems into one mount point. It also hashes the file names across the file systems, which is something MarFS is not currently designed to do. The main difference is the approach to what GlusterFS documentation refers to as unified file and object. GlusterFS has been integrated to be object storage for OpenStack Swift (for objects) and block storage for OpenStack Cinder (for blocks). Conversely, MarFS is designed to put a near-POSIX interface over any object storage system, including OpenStack Swift.

Some may question why a HSM (hierarchical storage management) tool like HPSS (High Performance Storage System) or DMF (Data Migration Facility) is not used. These systems currently don’t take advantage of the enormous industry investments in object storage. HPSS metadata performance is likely 1/10th or less of what MarFS metadata performance is expected to be. MarFS will leverage existing tools and be a small amount of code to combine these tools. To be fair, MarFS is not a HSM, although batch utilities could be written to move data around inside of MarFS to various kinds of storage systems that would be most appropriate for the data based on policy configurations. HSM systems, like HPSS, are not generally highly parallel. MarFS is designed for dozens to hundreds of metadata servers/name spaces and thousands or even tens of thousands of parallel data movement streams. HPSS is designed for about an order of magnitude less parallelism.

There are products that are optimized for WAN and HSM metadata rates. For example, General Atomics Nirvana Storage Resource Broker, iRODS (Integrated Rule Oriented Data Systems). There are some capabilities for putting POSIX files over objects, but these methods are largely via NFS or other methods that try to mimic full file system semantics including update in place. These methods are not designed for massive parallelism in a single file, etc.

The team has looked at name space solutions. EMC’s Maginatics is in its infancy and targeted at enterprise. An open source name space project called Camlistore appears to be targeted and personal storage. Bridgestore is a POSIX name space over objects, but it puts its metadata in a flat space so rename of a directory is horribly painful. Avere NFS over objects is focused at NFS so shared file N-1 will not be high performance.

We need an open source solution to deploy in production now that enables the described functionality. It is our hope that MarFS will set the bar high for fully integrated solutions to replace it.

# Requirements and Design

This section defines the requirements and design elements that were crafted to meet the requirements.

## Design Overview

This design will require:

* Linux system(s) with C/C++ and FUSE support
* MPI for parallel communication in pftool (a parallel data transfer tool, see https://github.com/pftool/pftool). Thus, most any MPI library with a C interface can be used.
* Communications with the MPI library can utilize many communications methods like TCP/IP, Infiniband OFED, etc.
* If you plan to use MarFS only to combine multiple POSIX file systems into one mount point, any set of POSIX file systems can be used.
* If you plan on using multi-node parallelism for the FUSE daemon, pftool, or the batch utility programs (MarFS software), all file systems, including MarFS file systems, must be globally mounted on all nodes running MarFS software. This includes NFS and other global file systems.
* If you plan to store data on an object store, that object store needs to be accessible by all nodes running MarFS software. The MarFS metadata component must be capable of POSIX extended attributes (xattr) and must support sparse files (files that have a non-zero size but that occupy no space).

The planned MarFS implementation will use multiple GPFS file systems as the metadata component (although any POSIX file system that supports XATTR and Sparse files would work) and Scality and/or ECS ViPR object stores as the data storage component.

The interactions with the GPFS-based metadata component are via the normal POSIX interface. GPFS has some ILM (Information Lifecycle Management) capabilities for managing massive amounts of metadata that helps immensely with batch processing for management of the system. MarFS will stitch together many GPFS file systems to allow for scaling of metadata size and operation. MarFS will utilize the POSIX extended attributes to store with each file in the metadata file systems where the backing object store data is for that file.

The interactions with the Scality- and/or ECS ViPR-based data storage component are via the most efficient object protocols, such as Amazon S3 and CDMI. MarFS can put a file per object, pack many small files into one object, and spread a large file across many objects. Although the design does not call for using POSIX file systems as the data storage component, the design does not preclude it. If MarFS were configured to use a POSIX file system as the data storage component, then any such file system would work including PFSes (parallel file systems) like GPFS, Lustre, Panasas, etc., or non-PFSes, like NFSv3.

The GPFS file systems and object stores will be hidden from the users so that they cannot use them directly, but the MarFS components will know about them and how to use them efficiently.

A FUSE daemon will provide the system mount point and interactive use component of MarFS. This daemon will know that it will use the GPFS file systems for metadata operations and the specified object stores as the data storage component. The FUSE daemon enforces writing only serially from byte zero.

The parallel data movement utility, pftool, will likewise be modified to use the GPFS file systems as the metadata component and the object stores as the data storage component. pftool is a load balanced, highly parallel utility on one node or across multiple nodes. It can walk the file system tree in parallel, move data between file systems, and move small files in parallel or break-up big files to move them in parallel for any POSIX file system, including MarFS. It will be possible to write data to MarFS in parallel using pftool, or by writing one’s own parallel data movement utility using the library.

There will be some MarFS utility programs that will be run periodically to free deleted storage space and ensure that users do not exceed their assigned quotas. Other utility programs may be implemented in the future to manage other aspects of the file system that can be performed on an as-needed or periodic basis.

The interfaces to MarFS are an interactive interface provided by a FUSE file system. This interface will be near-POSIX. It won’t support things like hard links and other problematic parts of POSIX. It also won’t support update in place for files that use object stores as backing store. What this means is that if you call truncate on a file you have to truncate to zero and you cant seek around and update parts of a file randomly. The batch interface will be through pftool (the parallel file copy/sync tool) or a library that one could write one’s own parallel application.

Security in MarFS is POSIX security enforced by the metadata file system(s) and efforts are made to ensure one cant get around the metadata system to access objects that hold data via standard access control mechanisms.

The MarFS is an Open Source project targeted at sites that want to leverage massive scale erasure based object stores but need an efficient near-Posix interface to the data. It is not intended to be a file system that is general purpose like an NFS file system or Parallel File System. It is intended for the Data Lake use case, a very massive place to hold data and read data enforcing some modest limits on how data can be ingested.