Parallel Storage Systems for Large Scale Machines

Christos Filippidis*
Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens,
Athens, Greece
cfjs@outlook.com

***Abstract*—I/O has become a bottleneck in application performance as processor speed skyrockets, leaving storage hardware and software struggling to keep up. The substantial amount of concurrency can cause a critical contention issue for I/O system. This study proposes a dynamically coordinated I/O architecture for addressing some of the limitations that current parallel file systems and storage architectures are facing with very large-scale systems. The fundamental idea is to coordinate I/O accesses according to the topology/profile of the infrastructure, the load metrics, and the I/O demands of each application. The measurements have shown that by using IKAROS approach we can fully utilize the provided I/O and network resources, minimize disk and network contention, and achieve better performance.**

Keywords—Data management; storage; distributed systems; parallel file systems; high performance computing; Exascale; GPFS; GridFTP; PVFS2.

I. INTRODUCTION

Large-scale scientific computations tend to stretch the limits of computational power and parallel computing is generally recognized as the only viable solution to high performance computing problems. I/O has become a bottleneck in application performance as processor speed skyrockets, leaving storage hardware and software struggling to keep up. Parallel file systems have be developed in order to allow applications to make optimum use of available processor parallelism. The most important factors affecting performance are the number of parallel processes participating in the transfers, the size of the individual transfers and of course the access patterns. The I/O access patterns are generally divided into the following subgroups [5]:

1. Compulsory (consist of I/Os that must be made to read a program's initial state from the disk and write the final state back to disk when the program has finished. For example, a program might read a configuration file and perhaps an initial set of data points, and then write out the final set of data points along with graphical and textual representations of the results).
2. Checkpoint/restart (are used to save the state of a computation in case of a hardware or software error which would require the simulation to be restarted).
3. Regular snapshots of the computation's progress.
4. Out-of-core read/writes for problems which do not fit to memory.
5. Continuous output of data for visualization and other post-processing.

Another important factor affecting performance is the storage architecture on which we apply the file system. Nowadays, a typical HPC facility uses a small portion of the available nodes for storage purposes (I/O nodes acting as storage servers). Normally each storage server provides a huge number of hard disks through a RAID system. Current globally shared file systems, being deployed at the aforementioned facilities using current storage architectures, have several performance limitations when used with large-scale systems, because [1]:

1. Bandwidth does not scale economically to large-scale systems.
2. I/O traffic on the high speed network can be affected by other unrelated jobs.
3. I/O traffic on each storage server can also be affected by other unrelated jobs.

The (3) above problems are generally recognized as the most limiting factors for developing future exascale storage infrastructures. Exascale systems will require I/O bandwidth proportional to their computational capacity and it seems that current file systems and storage architectures will not be able to fulfill this requirement. One approach is to configure multiple instances of smaller capacity, higher bandwidth storage closer to the compute nodes (nearby storage) [1]. The multiple instances can provide exascale size bandwidth and capacity in aggregate and can avoid much of the impact on other jobs.

This approach does not provide the same file system semantics as a globally shared file system. In particular, it does not provide file cache coherency or distributed locking, but there are many use cases where those semantics are not required. Other globally shared file system semantics are required, such as a consistent file name space, and must be provided by a nearby storage infrastructure. In cases where the usage or lifetime of the application data is constrained a

*Dissertation Advisor: Yiannis Cotronis, Associate Professor.

We acknowledge the support of Special Account for Research Grants of the National and Kapodistrian University of Athens

globally shared file system provides more functionality than the application's requirements while at the same time limits the bandwidth which the application can use. Nearby storage as described above reverses that, providing more bandwidth but not providing globally shared file system behavior [1].

The factors affecting performance are increasing if we consider the overall data flow (remote-local access) within an international collaborative scientific experiment, like the Large Hadron Collider (LHC) at CERN and KM3NeT [7,8]. KM3NeT is a future European deep-sea research infrastructure hosting a new generation neutrino detectors located at the bottom of the Mediterranean Sea.

In order to confront the aforementioned problems and limitations we introduced IKAROS [3] as a framework that enables us to create ad-hoc nearby storage formations and is able to use a huge number of low spec low power consumption I/O nodes in order to increase the available bandwidth (I/O and network) [3,4]. It unifies remote and local access in the overall data flow by permitting direct access to each I/O node, regardless of the Tier. In this way we can handle the overall data flow at the network layer, limit the interaction with the operating system and minimize disk and network contention.

II. Dissertation Summary

IKAROS is a write optimized system that provides a dynamically coordinated I/O architecture for I/O accesses according to the topology/profile of the infrastructure, the load metrics, and the I/O demands of each application. In Figure 1 we show an overview of the IKAROS framework: the input parameters used by the IKAROS (application's I/O requirements, load metrics, infrastructure profile/topology) and the resources that IKAROS manages (I/O nodes and storage media), in all the tiers of the computing model. Currently, we feed the appropriate input parameters at the IKAROS framework manually.

IKAROS allows data in a file to be striped across multiple disk volumes on multiple heterogeneous nodes and provides the utility for the storage system to access and transfer a data file in parts and in parallel mode, without a specific order according to client requests. IKAROS defines three types of nodes: The User Interface (UI)/Client node, the Meta-data node and the I/O node.

IKAROS first version was designed as an Apache Dynamic Shared Object (DSO) [3]. The latest versions are written in NodeJS, which provides more flexibility and interoperability with web 2.0 platforms. IKAROS node types are peers with the ability to act in any mode, driven by client requests (i.e. any node can act at the same time, as a Client/UI, meta-data node or I/O node).

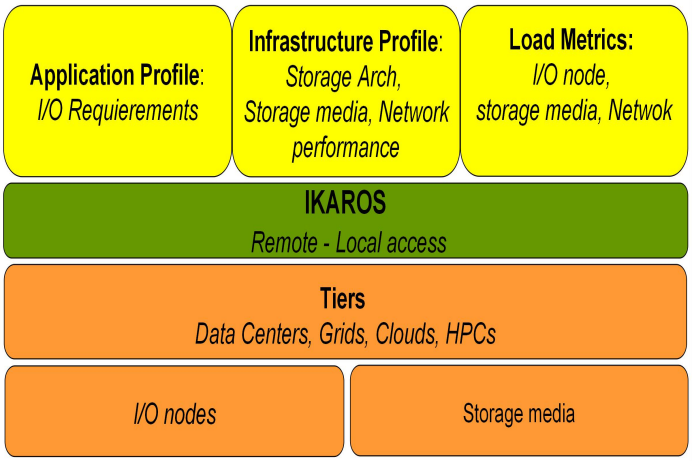

Fig 1. IKAROS Framework

The UI/Client node type is not a typical client but rather a gLite UI [3]. This node type provides services to many users. The users are not forced to use the UI/Client node type. Alternatively, they can access IKAROS by using their own browser or any other HTTP client such as curl and wget.

The IKAROS meta-data management service (iMDS) holds a key role in the IKAROS architecture. The iMDS allow us to handle the meta-data sub-system differently based on the needs. It may respond to a client/application request with three different ways. The client may find the answer: 1) within his own "cache", 2) locally at a nearby iMDS utility or 3) at an external platform/utility. This approach, focusing on flexibility, can scale both up and down and so can provide more cost effective infrastructures for both large scale and smaller size systems.

Thus, we are able to use existing external infrastructures [6] (as the top level MDS utility, in a tiered system), such as Facebook and Gmail, in order to dynamically manage, share and publish meta-data. In this way we do not have to build our own utilities for searching, sharing and publishing. Additionally, we are enabling users to dynamically use the infrastructure, by creating on demand storage formations and virtual organizations [6].

A. IKAROS overall data flow scenario-write request (remote-local access)

In this scenario we analyze a data transfer from a remote storage server to the local parallel file system. In Figure 2 we show the implementation of this action by combining GridFTP with the PVFS2. The client initiates a third party data transfer in order to transfer the data file from the remote GridFTP server to the local parallel file system, in this case we are using PVFS2. We implement the local GridFTP server and the PVFS2 MDS at the frontend machine of the local computing cluster.

Due to the client request, the remote GridFTP server starts sending the data file to the local GridFTP server by using N parallel data channels. The local GridFTP server moves data chunks to the PVFS2 I/O nodes. The combined use of GridFTP with the PVFS2, for implementing the overall data flow, forces us to initiate many independent transfers incurring much overhead to set up and release connections. This approach can significantly impact performance due to the unnecessary network and disk contention.

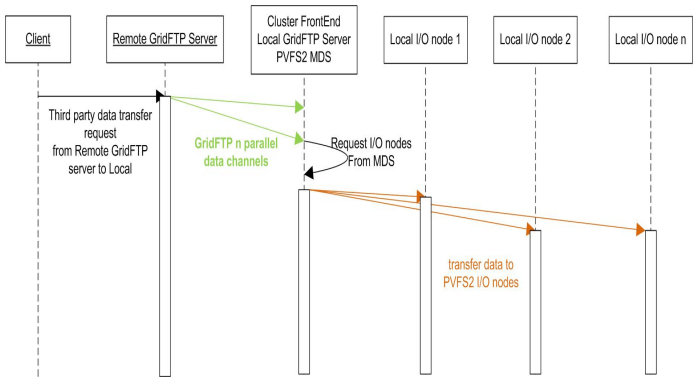

Fig 2. Overall data flow, GridFTP + PVFS2

The network and disk contention mainly appears due to the lack of proper coordination between the two systems, GridFTP+PVFS2. There is no guarantee that the remote access protocol and the local parallel file system, in this case GridFTP and PVFS2, have the same stripe size and the same stripe mapping. At its latest versions PVFS2 provides these kind of information (stripe size and stripe mapping) so we may achieve the required synchronization manually, but this in not also the case for other parallel file systems like GPFS.

A solution could be to use the GridFTP striped server technique, which is not exactly the case we show in Figure 2. In the striped server scenario the data file will be striped at several remote GridFTP servers and the local I/O nodes will have to act both as PVFS2 I/O nodes and local GridFTP servers. This means that we must assign public IP addresses to all local I/O nodes, which in most cases is not desirable. We may bypass that by using a pNFS+PVFS2 combination instead of the GridFTP+PVFS2 combination, but still we will not be able to properly conFigure the local parallel file system on the fly. Another issue that we will have to consider is that there is no guarantee that the GridFTP servers will stripe the data across the data nodes in the same sequence as PVFS2 does across the I/O nodes.

In Figure 3 we analyze the same overall data flow scenario, a data transfer from a remote storage server to the local parallel file system, but now we are using IKAROS for the overall data flow. Our goal is to avoid the unnecessary network and disk contention. Techniques like the reverse read implementation of the write request, introduced by IKAROS [3], combined if necessary with reverse HTTP tunneling techniques can help us provide proper coordination between remote and local access and achieve better performance.

This approach allows us to mainly route the data at the network level and minimize the usage of the operating system. In Figure 3 the IKAROS client follows the procedure demonstrated in [3] in order to trigger each local I/O node, which participates in the transfer. Then each I/O node, based on this trigger, makes a request to the remote storage server for the corresponding data chunk. In this way we apply only coordinated parallel data transfers in contrast with the GridFTP+PVFS2 case where we must manually synchronize the stripe size and the stripe mapping between them.

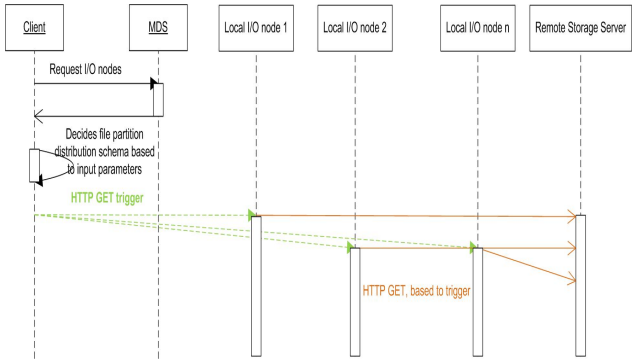

Fig 3. IKAROS Overall data flow

III. RESULTS

In this study, our main concern is to provide the appropriate framework in order to fully utilize the available I/O and Network resources. Our approach focusing on creating clusters of balanced network and I/O bandwidth at the first level of the parallel storage system. The available storage hardware (I/O nodes, storage media, network) at current HPC machines are underutilized due to: 1) the storage architecture being implemented, and 2) the failure of software to keep up with the I/O requirements. E.G: the Cytera machine [2].

This HPC facility consists of 100 nodes (96 Compute nodes) each with 12 Intel Xeon CPU cores, 48 GBs of RAM and one SATA local HDD. The nodes are connected over a QDR (40Gbit/s) infiniband. Cytera is using GPFS as the file system, which is implemented by 4 storage servers. It is supported by 360 TBs raw disk space in 18 Raid-6 arrays. This 180 HDD facility in theory could provide a throughput close to 4200 MB/s which is way higher than the measured GPFS peak performance at the Cytera machine (~1600 MB/s) [2]. We are referring in write requests.

In [2] we show how we can fully utilize the provided I/O and network resources, minimize disk and network contention by creating a cluster of dedicated (4:1-HDD:client) or semi-dedicated (4:2) storage facility for each client, where the I/O traffic will not be affected by other client requests. In this way we managed to improve performance by 33% with the 1/3 of the available hard disks (Figure 4). The fundamental idea is to coordinate I/O accesses according to the topology/profile of the infrastructure, the load metrics, and the I/O demands of each application.

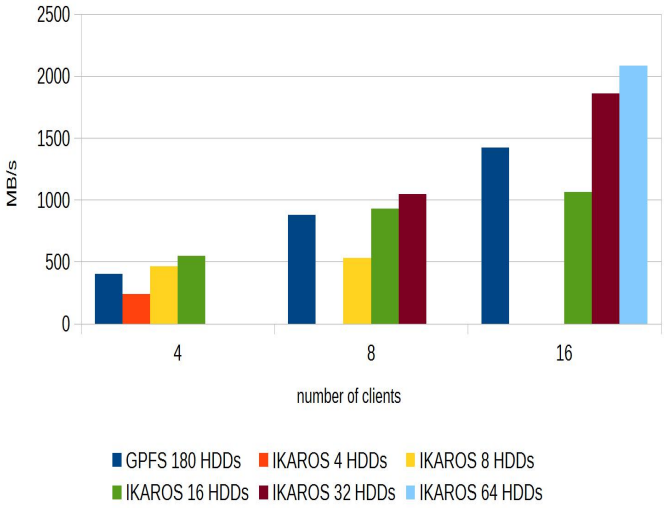

Fig 4. IKAROS vs GPFS by using several numbers of HDDs

In Figure 4 we clearly show that IKAROS can outperform GPFS by using 4:2, 4:1 ratios and that this approach can easily scale with only barrier the network bandwidth.

It seems that in order to be able to scale I/O performance against both technical and economical aspects we will have to reorganize the storage architecture in order to achieve a balance between the available network and I/O bandwidth. Following the findings in [2], in table 1 we provide "3" use cases of a balanced parallel storage system.

Table 1: Balanced parallel storage system

	Network connectivity	**I/O node type**	**# of storage media per I/O node**
Case 1	1 Gbps	soho-NAS	1
Case 2	10Gbps	typical storage server	9
Case 3	40Gbps	typical storage server	36
Cytera case	40Gbps	typical storage server	45

In these use cases we are using HDDs as the storage media, in order to be consistent with the Cytera machine, but this approach can be easily expanded to SSDs. The actual number of HDDs per I/O node depends on the HDD I/O metrics and the network connectivity.

It seems that If we are able to create clusters of storage infrastructure according to the use cases in table 1 and follow the rule of maximum 2 writes and maximum 4 read requests (concurrent) per HDD, derived from [2], we will be able to fully utilize the available storage hardware while scaling the storage facility. This approach seems to be extremely beneficial for writing large data streams that do not fit to memory.

IV. Conclusions and Future Work

This study proposes a dynamically coordinated I/O architecture for addressing some of the limitations that current parallel file systems and storage architectures are facing with very large-scale systems. The fundamental idea is to coordinate I/O accesses according to the topology/profile of the infrastructure, the load metrics, and the I/O demands of each application.

By using the IKAROS reverse read technique, for write operations, we are able to apply only coordinated parallel data transfers on the overall data flow (remote-local access). In contrast with other solutions (e.g: GridFTP+PVFS2 combination) where we are forced to initiate several independent data transfers incurring much overhead to set up and release connections. This can significantly impact performance due to the unnecessary network and disk contention.

The measurements have shown that by using IKAROS approach we can fully utilize the provided I/O and network resources, and minimize disk and network contention. We are able to achieve better performance by creating, on the fly, a cluster of dedicated (4:1) or semi-dedicated (4:2) storage facility for each client, where the I/O traffic can not be affected by other client requests. In this way we managed to improve performance by 33% with the 1/3 of the available hard disks.

As a future work we intend to develop an automated system to feed IKAROS with the appropriate input parameter (Figure 1). Probably, such a system could be part of an existing scheduling system like SLURM and HTCondor and or a Message Passing Interface (MPI).

References

[1] Dongarra, J., Beckman, P. et al. The International Exascale Software Roadmap," , Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420. Exascale Nearby Storage, Cray Position paper.

[2] Christos Filippidis, Panayiotis Tsanakas,Yiannis Cotronis, IKAROS: a Scalable I/O Framework for High-Performance Computing Systems,The Journal of Systems& Software (2016),Volume 118, pp. 277-287, DOI:http://dx.doi.org/10.1016/j.jss.2016.05.027

[3] Filippidis C., Cotronis Y.,Markou C.,(2013) IKAROS: an HTTP-based distributed File System, for low consumption and low specification devices, Journal of Grid Computing springer.

[4] Filippidis C., Cotronis Y.,Markou C.,(2012) Design and Implementation of the Mobile Grid Resource Management System. Computer Science, 13 (1). pp. 17-24. ISSN 1508-2806.

[5] Schmuck, F., Haskin, R.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the FAST 2002 Conference on File and Storage Technologies. IBM Almaden Research Center, San Jose, CA (2002).

[6] Filippidis, C., Cotronis, Y., Markou, C. (2014). Forming an ad-hoc nearby storage, based on the IKAROS and social networking services, IOP *J. Phys.: Conf. Ser.* **513** 042018.

[7] Filippidis, C., on behalf of the KM3NeT Collaboration (1/2016), Enabling Grid Computing resources within the KM3NeT computing model, Very Large Volume Neutrino Telscope workshop (VLVnT 2015).

[8] Filippidis, C., Cotronis, Y., & Markou, C. (1/2016). Using IKAROS as a data transfer and management utility within the KM3NeT computing model, Very Large Volume Neutrino Telscope workshop (VLVnT 2015).