Scaling storage systems for future eXascale environments

Christos Filippidis[*]

National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
cfjs@outlook.com

Abstract. High performance computing (HPC) has crossed the Petaflop mark and is reaching the Exaflop range quickly. The exascale system is projected to have millions of nodes, with thousands of cores for each node. At such an extreme scale, 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.

1 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 [1]:

(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).

[*] Dissertation Advisor: Yiannis Cotronis, Associate Professor

(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 that may significantly affect performance is the architecture of the storage system, 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 [2]:

(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 three (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) [2]. 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 and functionality 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 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 provides more bandwidth, but without offering globally shared file system behavior [2].

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. KM3NeT is a future European deep-sea research infrastructure hosting a new generation neutrino detectors that - located at the bottom of the Mediterranean Sea - will open a new window on the universe and answer fundamental questions both in particle physics and astrophysics.

This kind of experiments are generating datasets which are increasing exponentially in both complexity and volume, making their analysis, archival, and sharing one of the grand challenges of the 21st century. These experiments, in their majority, adopt computing models consisting of Tiers (each Tier is made up of several computing

Centers and provides a specific set of services) and for the different steps of data processing (simulation, filtering, calibration, reconstruction and analysis) several software packages are utilized. The computational requirements are extremely demanding and, usually, spans from serial to multi-parallel or GPU-optimized jobs.

In order to confront those challenges we introduced IKAROS as a framework that enables us to create ad-hoc nearby storage formations, able to use a huge number of I/O nodes 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.

2 Dissertation Summary

IKAROS 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. By referring to the I/O requirements/demands of the application we mean that IKAROS is not using a static/fixed algorithm for data placement. Due to the numerous configuration parameters offered the users and applications are able to choose the preferred strategy for each workload. In figure 1 we show an overview of the IKAROS framework: the input parameters used and the resources managed by IKAROS (I/O nodes and storage media) in all the tiers of the computing model.

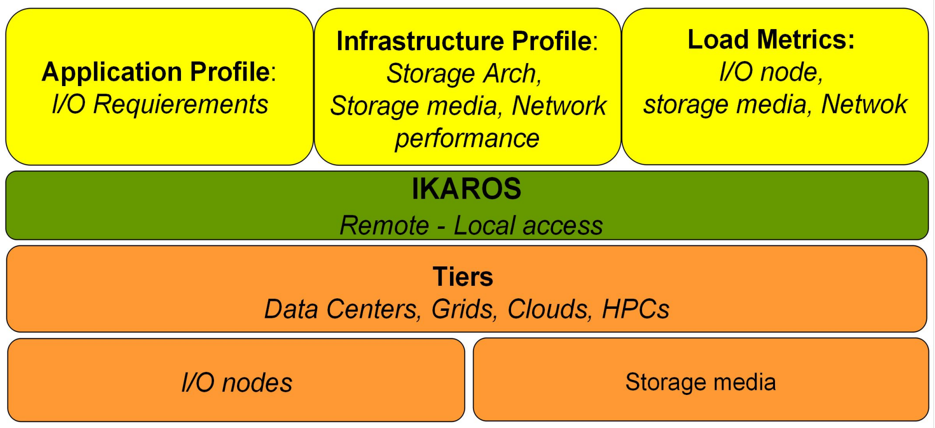
Application Profile: I/O Requierements
Infrastructure Profile: Storage Arch, Storage media, Network performance
Load Metrics: I/O node, storage media, Netwok
IKAROS
Remote - Local access
Tiers
Data Centers, Grids, Clouds, HPCs
I/O nodes
Storage media

Fig 1. IKAROS Framework

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).

The UI/Client node type is not a typical client but rather is more like 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. In a typical scenario, a user puts a request to his preferred IKAROS client (e.g: browser, other HTTP client, or UI/Client node) in order to read data from the storage facility. This request triggers the IKAROS module to interact with the meta-data node in order to fetch the necessary information regarding the file partition distribution schema. The client, then, establishes communication with the appropriate I/O nodes.

The IKAROS meta-data service (iMDS) holds a key role in the IKAROS architecture. The iMDS allows us to handle the meta-data sub-systems 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 [5] (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 [5].

2.1 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 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 the 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.

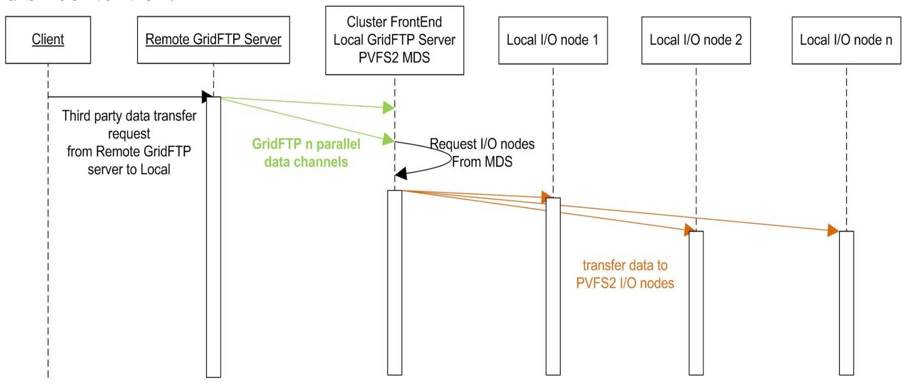
Client
Remote GridFTP Server
Cluster FrontEnd
Local GridFTP Server
PVFS2 MDS
Local I/O node 1
Local I/O node 2
Local I/O node n
Third party data transfer request from Remote GridFTP server to Local
GridFTP n parallel data channels
Request I/O nodes From MDS
transfer data to PVFS2 I/O nodes

Fig 2. Overall data flow-write request, 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 figure 2 we are using only one remote GridFTP server and the parallel data channels technique. 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 the 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 servers.

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 in order to avoid the unnecessary network and disk contention. Techniques like the reverse read implementation of the write request, introduced by IKAROS, 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.

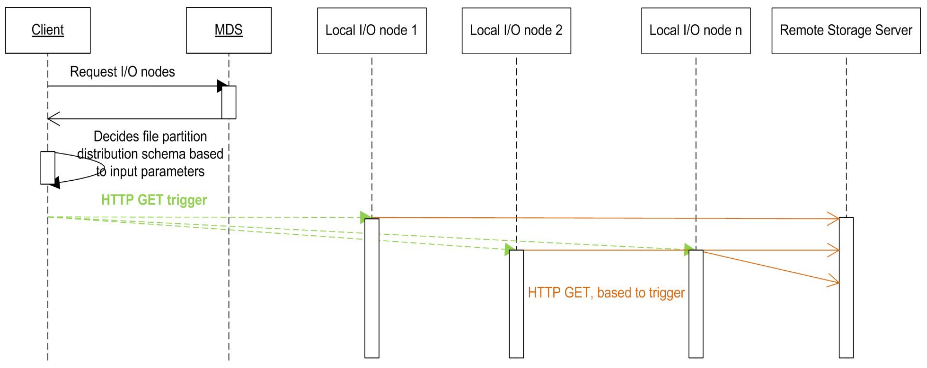

Fig 3. IKAROS overall data flow-write request

3 Results

For the evaluation we are using two different testbeds: the ZEUS computing cluster located at the National Center for Scientific Research "Demokritos" (soho-NAS environment) and the Cy-tera machine which is a Regional European HPC facility located at the Cyprus Institute (HPC environment). In figure 6 we show the two testbeds. The ZEUS computing Cluster provides an infrastructure composed of ten AMD Opteron CPU based systems (each with 8 CPU cores and 16 GB of RAM), four 800Mhz CPU based soho-NAS devices (each with 256 MB of RAM, 1000 Mbps Ethernet controller and 3 TB of storage capacity) and ten 200Mhz CPU based soho-NAS devices (each with 32 MB of RAM, 100 Mbps Ethernet controller and 2 TB of storage capacity). The ZEUS computer Cluster also provides a 1000 Mbps full duplex link between the nodes and an 2.5 Gbps WAN connectivity. In this study we are using the four 800Mhz CPU based soho-NAS devices and the frontend.

The Cy-tera machine consists of 100 nodes (96 Compute nodes) each with 12 Intel Xeon CPU cores, 48 GBs of RAM and one 15K rpms SATA local HDD. The nodes are connected over a QDR (40Gbit/s) infiniband. The GPFS file system is implemented by 4 storage servers. It is supported by 360 TBs raw disk space in 18 Raid-6 arrays each with 10 7200 rpms SATA HDD. The GPFS meta-data is provided by 4 Raid-10 arrays (one associated at each storage server).

As we mentioned before, its extremely important to obtain information like the storage architecture profile, the network topology, the I/O node and storage media performance in order to configure/customize properly each data transfer request. In figure 4 we show the performance of writing an 80 GBs file from one node to 1,2,4 and 8 storage servers. All nodes are equipped with only one hard disk. In this scenario we are writing/partitioning the data file, which is located at the client side at 1 local HDD, to 1-8 storage servers (1-8 HDDs).

The purpose of this measurement is to help us make an estimation of the optimum load (read/write requests) that a hard disk can handle when we are transferring big data files that do not fit to memory. This estimation applies to both testbeds because the hard disks at the Zeus frontend and the disks at the Cy-tera computing nodes are having similar performance. We can use this estimation in order to decide the optimal number of the I/O nodes that the parallel file system should use in order to stripe/partition the data file.

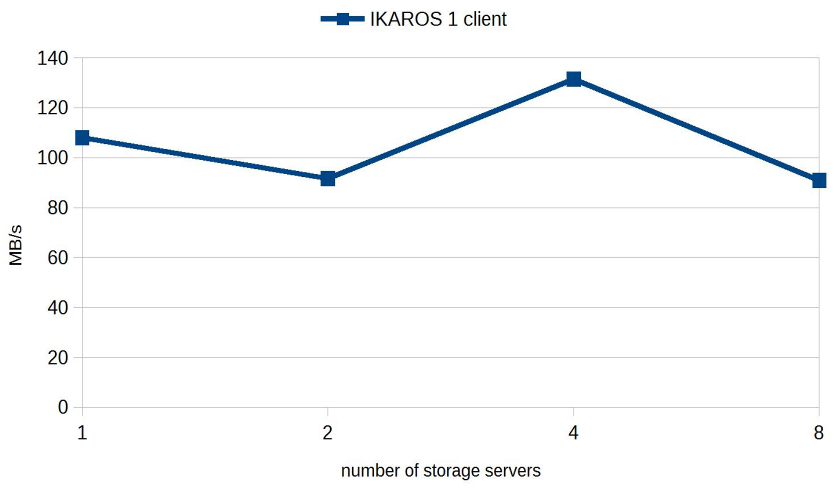

Fig 4. HDD Performance

Figure 4 shows that if the data file is located at a single disk, which is the case most of the times (the compute nodes used by the applications are equipped with a single disk), the best split ratio is 1:4. At the next section we show that if we maintain this ratio (4 dedicated HDDs serving each client for writing/partitioning the file) we can fully utilize the available I/O and network bandwidth.

In figure 4 we observe that by choosing an 1:4 ratio we achieve better performance than by choosing the 1:2 ratio. This is expected because most controllers and data media have queuing mechanisms, which when processing several parallel requests ensure under certain circumstances a higher performance than when processing fewer parallel or even only single requests. This, however, is always done at the price of higher response times. If many parallel requests only entail an increase in response time, and no longer in throughput, the disk subsystem is overloaded [6]. It seems that the 1:4 ratio is the most sufficient ratio for writing a big data file, located in a single disk, to the parallel file system.

Here we compare IKAROS with GPFS in data transfers with 80 GBs file size (in total) and multiple clients, using the Cy-tera machine. In figure 5 we measure the GPFS performance in the Cy-tera machine in order to properly choose the number of concurrent clients at the experiments in figure 6.

For a given GPFS file system, the most important factors affecting performance (aside from the access pattern) are the number of parallel processes participating in the transfers, and the size of the individual transfers. Figure 10 shows that we achieve the highest performance when the ratio of client processes to server nodes is near to 5:1 (though the nodes running our experiments have 12 processors per node, we ran only one client task per node). This ratio is slightly better but close to the 4:1 ratio measured at Lawrence Livermore National Laboratory at 2000 [7] by using 38 servers and 152 clients.

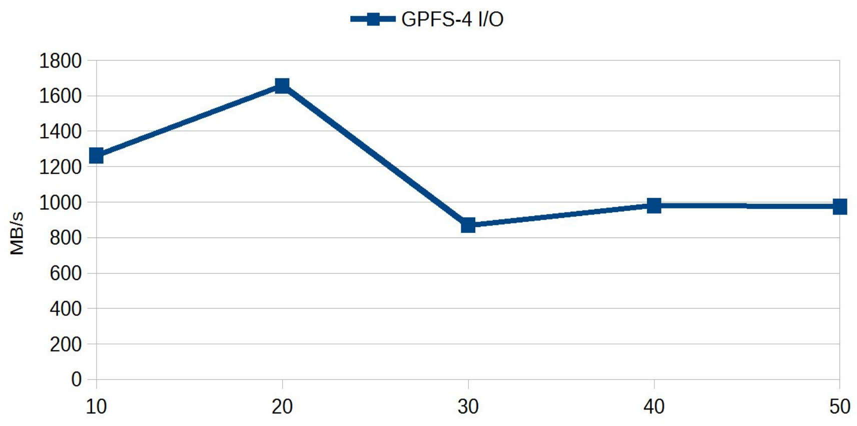

Fig 5. Cy-tera-GPFS Performance

In figure 5 we wrote a 80 GBs file, splitted into separate files, one file for each client process. As mentioned at [7] when the client:server ratio is too low the servers are starved for data; when it is too high the receiving buffers fill up faster than they can be drained, eventually causing packets to be dropped and retries initiated, reducing performance. From figure 10 we conclude that for the Cy-tera machine we must not initiate more than 20 simultaneously data transfers.

As we mentioned previously, the Cy-tera machine uses 180 hard disks distributed at the 4 storage servers in 18 Raid-6 arrays. This Raid-6 configuration in theory could provide a throughput close to 4200 MB/s which is way higher than the measured GPFS peak performance at the this machine (around 1600 MB/s). At the following experiments we show how IKAROS can fully utilize the available resources (I/O nodes and storage media) and achieve better performance.

The following experiments have been chosen because they show the effects of varying the I/O characteristics of application programs. We measured how aggregate throughput varied depending on the number and configuration of client processes and the size of individual transfers. We also show how GPFS and IKAROS performance scale with system size and throughput of parallel tasks creating and writing a single large file, and of reading an existing file.

To measure the throughput of writes, the benchmark performs a barrier, then each task records a “wall clock” starting time, process 0 creates the file and all other processes wait at a barrier before opening it, then all processes write their data according to the chosen application characteristics (in the tests shown here, always independently of each other, filling the file without gaps and without overlap); finally, all processes close the file and record their ending time.

The throughput is calculated as the total number of bytes written in the total elapsed wall clock time (the latest end time minus the earliest start time). This approach is very conservative, but its advantages are that it includes the overhead of the opening and closing and any required seeks, etc., and measures true aggregate throughput rather than, for example, an average of per process throughput rates. For implementing IKAROS we are using the computing nodes and the local hard disks provided by them. All the nodes being used were on exclusive mode (only used by our process).

The results, for the GPFS, indicate the peak performance the file system is capable of delivering rather than what a user would see in the presence of other jobs competing for the same resources. The I/O performance seen in real applications will depend on complex interactions between the system and the application’s run time behavior. Competition with other applications for I/O resources, wild access patterns, and some randomness in the file system can cause performance to be lowered. IKAROS framework enables us to create concurrent client requests through a coordinated I/O architecture in order to prevent I/O system contention [8].

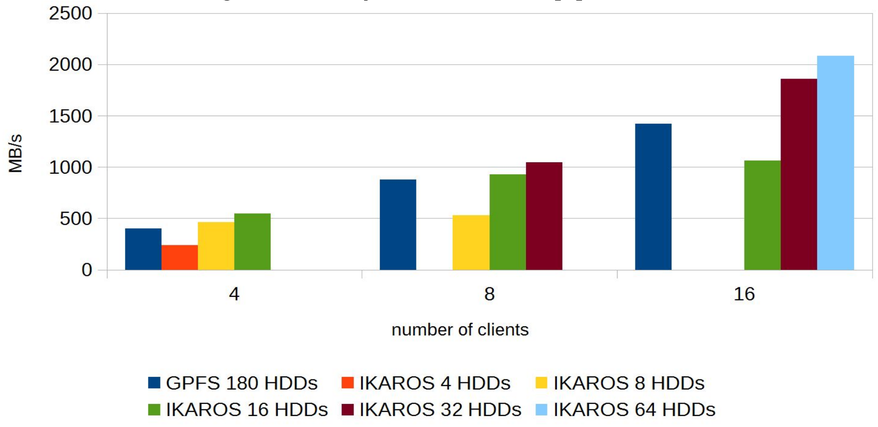

Fig 6. IKAROS vs GPFS by using several numbers of HDDs

In figure 6 we are using up to 16 concurrent clients due to the available nodes at the Cy-tera machine and because the measurements in figure 10 clearly shows that the

GPFS performance, for the current system, is seriously impacted by using more than 20 concurrent clients. Each time we measure the performance of GPFS versus IKAROS on 4:4, 4:2 and 4:1 write ratios (HDDs:clients). In this way we are able to create a virtual cluster of dedicated (4:1) or semi-dedicated (4:2, 4:4) storage facility for each client, where the I/O traffic can not be impacted by other client requests.

In figure 6 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. IKAROS improves performance by 33% with the 1/3 of the available hard disks [8].

4 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 for 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 virtual 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[46] and HTCondor[47] and or a Message Passing Interface (MPI).

References

1. 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).

2. 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.
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. 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.
6. WHITE PAPER, FUJITSU PRIMERGY SERVER BASICS OF DISK I/O PERFORMANCE, http://sp.ts.fujitsu.com/dmsp/Publications/public/wp-basics-of-disk-io-performance-ww-en.pdf.
7. Terry Jones, Alice Koniges, and R. Kim Yates, Performance of the IBM General Parallel File System, Proceedings of the International Parallel and Distributed Processing Symposium, Cancun, Mexico, May 2000.
8. 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