NESUS cost IC1305
Network for Sustainable Ultrascale Computing
Second NESUS Workshop • September 2015 • Vol. 1, No. 1

Evaluating data caching techniques in DMCF workflows using Hercules

Francisco Rodrigo Duro†, Fabrizio Marozzo∗, Javier Garcia Blas†

frodrigo@arcos.inf.uc3m.es, fmarozzo@dimes.unical.it, fjblas@arcos.inf.uc3m.es

Jesus Carretero†, Domenico Talia∗, Paolo Trunfio∗

jesus.carretero@uc3m.es, talia@dimes.unical.it, trunfio@dimes.unical.it

† ARCOS, University Carlos III, Spain
∗ DIMES, University of Calabria, Italy

Abstract

The Data Mining Cloud Framework (DMCF) is an environment for designing and executing data analysis workflows in cloud platforms. Currently, DMCF relies on the default storage of the public cloud provider for any I/O related operation. This implies that the I/O performance of DMCF is limited by the performance of the default storage. In this work we propose the usage of the Hercules system within DMCF as an ad-hoc storage system for temporary data produced inside workflow-based applications. Hercules is a distributed in-memory storage system highly scalable and easy to deploy. The proposed solution takes advantage of the scalability capabilities of Hercules to avoid the bandwidth limits of the default storage. Early experimental results are presented in this paper, they show promising performance, particularly for write operations, compared to the performance obtained using the default storage services.

Keywords DMCF, Hercules, data analysis, workflows, in-memory storage, Microsoft Azure

I. Introduction

In the last decade, most of the scientific computing problems are increasing their needs to process large quantities of data. Large simulations, data visualization, and big data problems are some of the application areas leading the trends in scientific computing. This evolution is moving needs from a computing-centric power point of view to a data-centric approach. Current trends in High Performance Computing (HPC) also include the use of cloud infrastructures as a flexible approach to virtually limitless computing resources. Given this current scenario, a solution that combines HPC, data analysis, and cloud computing is becoming more and more necessary.

According to their elastic feature, cloud computing infrastructures can serve as effective platforms for addressing the computational and data storage needs of most big data applications that are being developed nowadays. However, coping with and gaining value from cloud-based big data requires novel software tools and advanced analysis techniques. Indeed, advanced data mining techniques and innovative tools can help users to understand and extract what is useful in large and complex datasets for making informed decisions in many business and scientific applications.

The Data Mining Cloud Framework (DMCF), developed at University of Calabria, is an environment for designing and executing data analysis workflows in cloud platforms. Currently, DMCF uses the storage provided by the cloud provider for any I/O related job. This implies that the I/O performance of DMCF is limited by the performance of the default storage. Moreover, it is influenced by the contention that occurs when other I/O tasks are concurrently executed in the same region. Finally, the cost of using persistent storage service to store temporary data should be also taken into account.

The solution proposed here consists in using Hercules as the default storage system for temporary data produced in workflows. Hercules is a distributed in-memory storage system, easy to deploy and highly scalable. This system has been developed in the ARCOS research group, at University Carlos III Madrid, and it has also been proved in traditional HPC cluster with promising results.

This novel approach has three main objectives. The first one is taking advantage of the scalability of Hercules to avoid the bandwidth limits of the default storage. When the number of Hercules I/O nodes increases, the total available aggregated bandwidth usable by worker nodes is enhanced. The second objective is to allow the deployment, thanks to the easy deployment of Hercules, of an ad-hoc and independent in-memory storage system to avoid the contention produced during peak-loads in the cloud storage service. The last objective is the independence from the cloud platform used. While each cloud infrastructure have different APIs to access their storage services, Hercules has interfaces for commonly used APIs (like POSIX-like, put/get, MPI-IO) in order to imply minor modifications to existing code.

The main focus of this work is to deploy Hercules on a cloud infrastructure together with DMCF and to evaluate their performance with respect to the cloud storage service in different scenarios. This preliminary evaluation is aimed at demonstrating the capabilities of Hercules to be used as temporary storage of data analysis applications developed using DMCF.

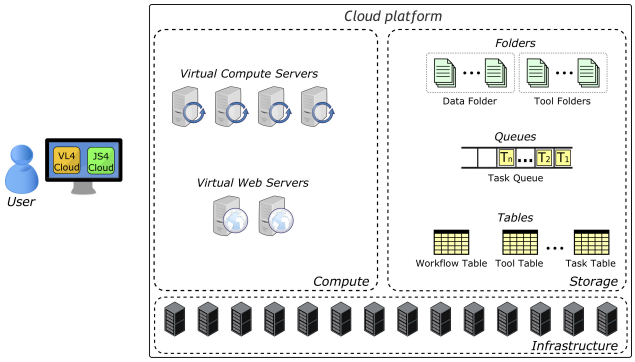


Figure 1: Architecture of Data Mining Cloud Framework.

The remainder of the paper is structured as follows. Section II describes the main features of DMCF. Section III introduces Hercules architecture and capabilities. Section IV emphasizes the advantages of integrating DMCF and Hercules and outlines how this integration will work. Section V presents preliminary results of the performance achieved by Hercules in the Azure cloud infrastructure and compares the results with Azure Storage. Section VI briefly presents other research work in the same field. Finally, section VII concludes the work and give some future research related to the presented work.

II. Data Mining Cloud Framework

The Data Mining Cloud Framework (DMCF) [6] is a software system designed for designing and executing data analysis workflows on Clouds. A Web-based user interface allows users to compose their applications and to submit them for execution to the Cloud platform, following a Software-as-a-Service (SaaS) approach.

The architecture of DMCF includes different components that can be grouped into storage and compute components (see Figure 1). The storage components include:

- A *Data Folder* that contains data sources and the results of knowledge discovery processes. Similarly, a *Tool Folder* contains libraries and executable files for data selection, pre-processing, transformation, data mining, and evaluation of results.
- *Data Table*, *Tool Table* and *Task Table* contain metadata information associated with data, tools, and tasks.
- The *Task Queue* contains the tasks that are ready for execution.

The compute components are:

- A pool of *Virtual Compute Servers*, which are in charge of executing the data analysis tasks.
- A pool of *Virtual Web Servers* that host the Web-based user interface.

The DMCF architecture has been designed to be implemented on top of different Cloud systems. The implementation used in this work is based on Microsoft Azure[1].

[1] http://azure.microsoft.com

A user interacts with the system to perform the following steps for designing and executing a knowledge discovery application:

1. The user accesses the Website and designs the workflow through a Web-based interface.
2. After submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the workflow.
3. Each idle Virtual Compute Server picks a task from the Task Queue, and concurrently executes it.
4. Each Virtual Compute Server gets the input dataset from the location specified by the workflow. To this end, a file transfer is performed from the Data Folder where the dataset is located to the local storage of the Virtual Compute Server.
5. After task completion, each Virtual Compute Server puts the results on the Data Folder.
6. The Website notifies the user as soon as her/his task(s) have completed, and allows her/him to access the results.

The set of tasks created on the second step depends on how many data analysis tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue. All the potential parallelism of the workflow is exploited by using all the needed Virtual Compute Servers.

DMCF allows to program data analysis workflows using two languages:

- VL4Cloud (Visual Language for Cloud), a visual programming language that lets users develop applications by programming the workflow components graphically.
- JS4Cloud (JavaScript for Cloud), a scripting language for programming data analysis workflows based on JavaScript [6].

Both languages use two key programming abstractions:

- Data elements, denoting input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
- Tool elements, denoting algorithms, software tools or complex applications performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

Another common element is the Task concept, which represents the unit of parallelism in our model. A task is a Tool invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a data-driven task parallelism. This means that, as soon as a task does not depend on any other task in the same workflow, the runtime asynchronously spawns it to the first available virtual machine (VM). A task Tj does not depend on a task Ti belonging to the same workflow (with $i \neq j$), if Tj during its execution does not read any data element created by Ti.

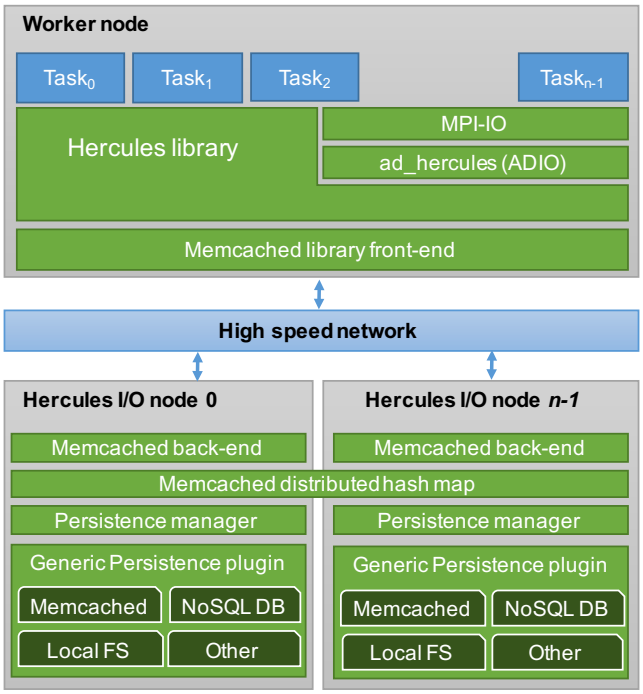


Figure 2: Hercules architecture. On the top the worker side, a user-level library. On the bottom the server side with the Hercules I/O nodes divided in modules.

III. Hercules

Hercules [3] is a distributed in-memory storage system based on the key/value Memcached database [4]. The distributed memory space can be used by the applications as a virtual storage device for I/O operations and has been specially adapted in this work for being used as in-memory shared storage for cloud infrastructures. Our solution relies on an improved version of Memcached servers, which provides an alternative storage solution to the default storage service.

As can be seen in the Figure 2, Hercules architecture has two levels: worker library and servers. On top is the worker user-level library with a layered design. Back-ends are based on the Memcached server, extending its functionality with persistence and tweaks. Main advantages offered by Hercules are: scalability, easy deployment, flexibility, and performance.

Scalability is achieved by fully distributing data and metadata information among all the nodes, avoiding the bottlenecks produced by centralized metadata servers. Data and metadata placement is completely calculated in the worker-side by a hash algorithm. The servers, on the other hand, are completely stateless.

Easy deployment and flexibility at worker-side are tackled using a POSIX-like user-level interface (open, read, write, close, etc.) in addition to classic put/get approach existing in current NoSQL databases. The existing software requires minimum changes to run with Hercules. The layered design allows for performing any future change with the minimum required effort. Servers can be deployed

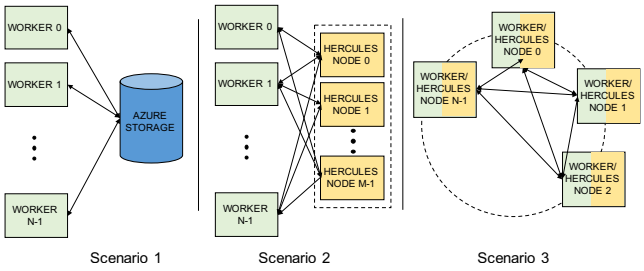


Figure 3: Deployment scenarios for the combination of Hercules and DMCF infrastructures.

in any kind of Linux systems at user level. Persistence can be easily configured using the existing plugins or developing new ones. An MPI-IO interface is also available for legacy software relying on MPI as communication system.

Finally, performance and flexibility at server-side are targeted by exploiting the parallel I/O capabilities of Memcached servers. Flexibility is achieved by Hercules due to its easiness to be deployed dynamically on as many nodes as necessary. Each node can be accessed independently, multiplying the total throughput peak performance. Furthermore, each node can serve requests in a concurrent way thanks to a multi-threading approach. The combination of these two factors results in full scalability: both when the number of nodes increases and when the number of workers running on each node increases.

IV. Integration between DMCF and Hercules

The final objective of this joint research work is the integration of DMCF and Hercules. As can be seen in Figure 3, Hercules and DMCF can be configured in more than one deployment scenarios to achieve different levels of integration.

The first scenario shows the current approach of DMCF, where every I/O operation is done against the cloud storage service offered by the cloud provider, which is Azure Storage in this work. While this storage service is suitable for persistent data, it could be inefficient for temporary data. The main benefits of a cloud storage service are the convenience of using every tool offered by the same provider and the persistence options offered, even in different geographical regions. Nevertheless, there are, at least, four disadvantages about this approach. First, proprietary interfaces and tools to access the storage service offered by different providers. Second, the performance offered by this services could have limitations that can not be avoided and performance could not be stable when there are peaks of use by other users. Third, the storage services are offered in a closed configuration, and can not be customized to the necessities of users at any time. Fourth, the cloud philosophy is tightly related with the pay-per-use concept. However, it does not make sense to pay for temporary data as if it was persistent data.

The second scenario, and the first contribution of this paper, is to use Hercules as the default storage for temporary generated data. Temporary data is becoming more and more popular in data analysis and many-task based applications. Most of these applications are developed as a sequence of tasks that communicate by using temporary files. Hercules I/O nodes can be deployed on as many VM instances

as needed by the user depending on the required performance and the characteristics of data. Even the instance type can be configured according to the necessities of each different application. As stated in Section III, Hercules offers different user-level interfaces such as POSIX-like, put/get, and MPI-IO, allowing a more flexible deployment of legacy applications than the default cloud storage service. Cost-wise it is needed to better study the competition between using a persistence-focused service against launching Hercules I/O node instances as temporary storage.

The third scenario shows an even tighter integration of DMCF and Hercules infrastructures. In this scenario Hercules I/O nodes share virtual instances with the DMCF workers. If the data needed by the DMCF worker is stored inside the Hercules I/O node running in the same instance, it will not be necessary to use the network for accessing data, and every I/O operation will be completely local. This functionality, paired with the improved data placement algorithm that stores all the data related with one file in the same Hercules I/O node, and with a DMCF scheduler that co-locates the tasks in the nodes where the data is stored, can lead to even better performance, exposing and exploiting data locality.

Before implementing the system integration, we need to analyze the potential performance improvement that Hercules can offer on a public cloud infrastructure, specially against Azure Storage, which is the storage service chosen in the current DMCF implementation. This preliminary evaluation is presented in Section V.

V. Evaluation

As mentioned before, to demonstrate the capabilities of Hercules in accelerating the I/O operations of DMCF workers, we evaluated the performance of the Azure Storage service against our proposed solution. For this purpose, we have designed and implemented a simple benchmark, referred from now on as *Filecopy Benchmark*. In this benchmark, a configurable number of workers perform two simple tasks per worker: the first one is writing files to the configured storage (Azure Storage or Hercules) and, after the write task is complete, a read task starts over the data written previously. The benchmark is fully configurable in terms of:

- *Number of worker nodes*: each worker node is a VM deployed in Azure.
- *Number of workers per node*: worker processes running in the same node in parallel. This parameter is important to evaluate how the storage solutions will behave in multi-core architectures and how they perform when different worker processes share the same network interface.
- *File size*: the total size in MegaBytes (MB) of the file can be configured to simulate different problem sizes.
- *Chunk size*: in Azure storage, a BLOB object is divided into blocks (maximum block size of Azure Storage is 4 MB, not enough for large files). The Java library used for accessing to Azure Storage, automatically divides a block object in the required number of block objects. In addition to this behavior, our implementation divides a file into different BLOB objects. Chunk size parameter is the size of each of the block objects that are part of a complete file. In Hercules, it corresponds to the buffer size of the POSIX write operation. Internally, Hercules divides the files in blocks adapted to the key-value hashmap of Memcached.

Table 1: Azure instance type characteristics.

Type	Cores	RAM (GB)	Bandwidth (Mbps)[1]	Price (€/h)
A0	1	0.75	<100	0.017
A1	1	1.75	~240	0.050
A2	2	3.50	~480	0.101
A3	4	7.00	~960	0.202
A4	8	14.00	~1700	0.408
D1	1	3.50	~480	0.097
D2	2	7.00	~900	0.194
D3	4	14.00	~1600	0.388
D4	8	28.00	~2000	0.776

The computing resources used during the evaluation are completely based on Microsoft Azure. Table 1 shows the characteristics of the different instance types used during our evaluation. All the resources used were located on the "Western Europe" region and the OS installed on the VMs was Ubuntu 14.04 LTS. It is also worth to be noted that, as the objective of the research work is to use Hercules as temporary storage, persistence features are disabled.

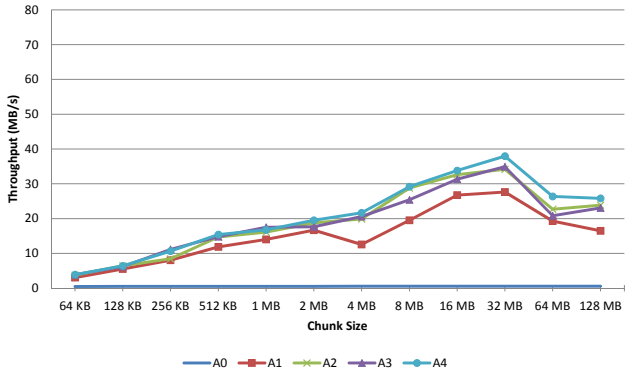
V.1 Chunk size evaluation

For the first evaluation case, we have fixed the file size to 128 MB, to have a file size that is big enough to show the performance with different chunk sizes. The chunk size will vary during the evaluation and we have used the five standard (A0-A4) instance types. Figure 4(a) shows the performance achieved during the write operations and Figure 4(b) the read operations performance. As it can be seen in these figures, Azure Storage performs much better for read (up to 72 MB/s) than for write operations (up to 38 MB/s). Also, the performance increases with the chunk size, achieving the best performance around the 32 MB mark. Finally, it is interesting to note how the performance varies with the instance type used: as expected, the most expensive instances have the better performance.

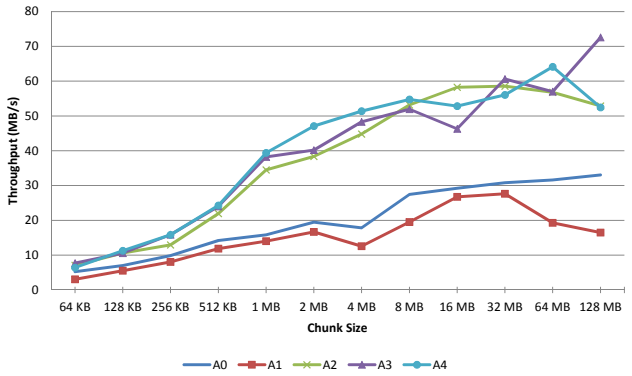
V.2 Hercules I/O nodes scalability

The next phase in the evaluation process is the measurement of the performance difference between Azure Storage and Hercules using different configurations. Also, we evaluate how Hercules scales its performance as the number of deployed I/O nodes increases. Based on the preliminary nature of this evaluation, our budget was limited to VMs running with a maximum number of 25 cores in total. After some quick bandwidth evaluation cases (results showed in Table 1), we selected D1 and D2 instances as the best performers in network bandwidth per core ratio. D1 instances achieve a peak performance of 60 MB/s using one core while D2 tops at around 115 MB/s with two cores, managing to reach almost the best possible performance of the available Gigabit virtual network interface. This is 2x the bandwidth available per core compared with

[1] Bandwidth measured experimentally using *iperf* tool between two VMs of the same instance type in the same region.



(a) Throughput of Azure Storage by using the Filecopy Benchmark (128 MBytes) for evaluating the block size for writes.

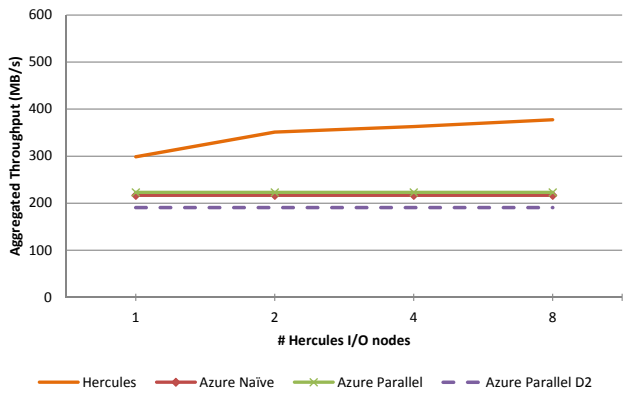


(b) Throughput of Azure Storage by using the Filecopy Benchmark (128 MBytes) for evaluating the block size for reads.

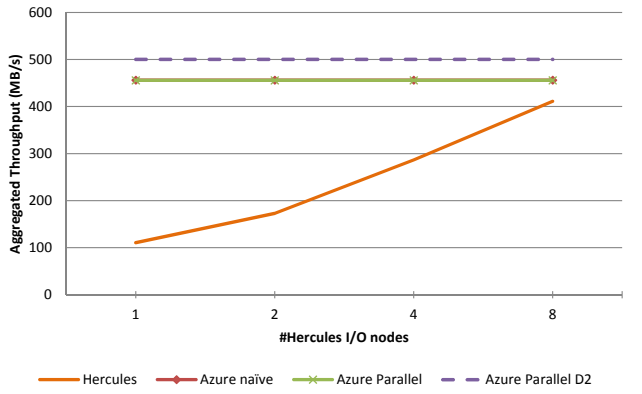
Figure 4: File copy benchmark configured for evaluating the Azure Storage performance depending on the block object size.

Standard 'AX' instance types. In the future, it would be interesting to evaluate the performance achieved by Hercules running in the A8 and A9 network optimized instances with Infiniband network, and 56 and 112 Gigabytes of RAM respectively. This network optimized instances should be the optimal option for running Hercules I/O nodes.

The final selection for this test is 8 VMs (D1 instances) as worker nodes and up to 8 VMs (D2 instances) as Hercules I/O nodes. Figure 5 plots the filecopy benchmark results, configuring the experiment with a file size of 512 MB, with 32 MB of chunk size and executing one read/write operation per worker node (one worker process per node) which implies a 4096 MB problem size (512 MB x 8 worker nodes). We have compared four different cases. The first one is the performance obtained by Hercules using between 1 and 8 I/O nodes. The second case is Azure Storage baseline approach, using the default access pattern offered by the Java API, without any optimizations. Third case is Azure Storage applying some optimizations to the code, specially important is setting up the



(a) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for writes. We set up the experiment with 8 worker nodes, writing 512 MBytes each one (4 GBytes in total).



(b) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for reads. We set up the experiment with 8 worker nodes, reading 512 MBytes each one (4 GBytes in total).

Figure 5: File copy benchmark configured for evaluating the Hercules I/O nodes scalability. 8 worker processes running on 8 worker nodes access 4 Gigabyte of data. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in the best read cases.

BlobRequestOptions object property *setConcurrentRequestCount* with 8 threads per process, using 8 concurrent threads to parallel access to Azure Storage. The last case can not be directly compared with the performance achieved by Hercules, because it uses the reserved D2 instances as worker nodes, instead of using them as I/O nodes, to show the peak performance achievable by Azure Storage with fully working Gigabit interface, hence the dotted line. In the Hercules case, the peak performance is limited by the aggregated bandwidth available worker-side (8x60 MB/s ~480 MB/s) not by the server-side 8x115 MB/s (~920 MB/s).

Figure 5(a) shows the performance evolution as the number of Hercules I/O nodes increase compared to the different Azure Stor-

age approaches. The figure clearly demonstrates how Hercules performance tops near the 400 MB/s mark, which is near the maximum theoretical peak performance of 8x60 MB/s (~480 MB/s). This peak performance achieved using 8 I/O nodes for parallel access is nearly 2x the performance achieved by Azure Storage in any of the configurations. Some interesting sights in the Azure Storage side are how both the baseline and the parallel approach performance is nearly identical caused by only being one core available in D1 instances. Also, it is interesting how the D2 instances performance using parallel accesses is even lower, exposing the deficiencies of Azure Storage performance in write operations.

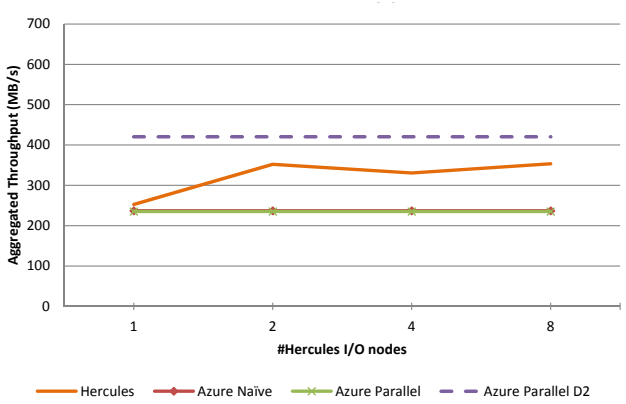
In Figure 5(b), which depicts the read operations performance, can be clearly seen how the Hercules performance evolves as the number of I/O nodes available increases. With only one I/O node available, the performance is ~100 MB/s, the maximum offered by the network interface of the I/O node (D2 instance). As the number of I/O nodes increases, the performance evolves, reaching a peak performance of ~400 MB/s, again near the theoretical up mark of 480 MB/s and near the performance of Azure Storage that slightly outperforms Hercules in this case. Azure Storage performs at the peak performance of the available network, with same performance in naive and parallel approaches using D1 instances while performing marginally better when D2 instances are used as worker nodes.

Third evaluation case is an evolution of the previous test for a scenario with higher congestion using the same infrastructure (8 D1 instances as worker nodes and 8 D2 instances as Hercules I/O nodes). In this case, instead of having 1 worker running on each node, we launched 4 workers running in parallel on each of the worker nodes, keeping the problem size in 4096 MB. For this purpose, each worker process writes, and then reads, a 128 MB file, with the same chunk size of 32 MB.

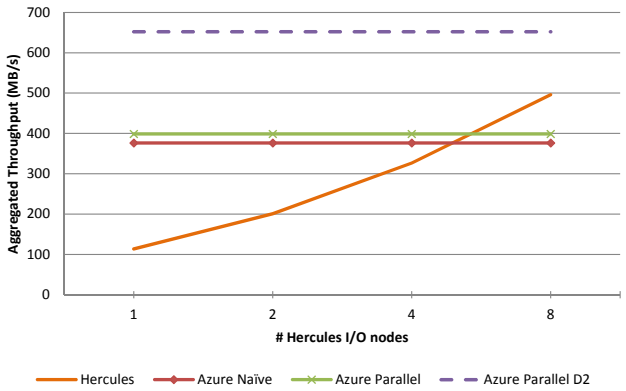
Figure 6(a), showing the performance in write operations, reports a very similar behavior of Hercules compared to the previous test case, but achieving a lower peak performance. At the same time, Azure Storage performance with D1 instances increases and the difference between Hercules and Azure Storage is narrowed to a 50% difference in favor of Hercules. Furthermore, using more than one process per node in the dual-core D2 instances, doubles the performance obtained by Azure Storage than Hercules in this special case.

On the other hand, on Figure 6(b), related with read operations, the peak performance of Hercules is even higher than the previous case, fully utilizing the ~480 MB/s of the available aggregated bandwidth at client-side and surpassing the peak throughput performance of Azure Storage accessed from D1 instances. When Azure Storage is accessed by D2 instances with more than one process reading in parallel from different files, the performance is almost doubled, in a similar way seen in the write operations.

As conclusions of the last two cases, we can emphasize how the aggregated throughput of the workers accessing to the Hercules storage system approaches the theoretical maximum bandwidth available in every studied case, showing the scalability capabilities of our proposed solution. The performance in write operations is between 1.5x and 2x the performance achieved by Azure Storage with a similar architecture, while the performance in read operations in first case is marginally in favor of Azure and in the second case is comparable.



(a) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for writes. We set up the experiment with 8 clients and 4 process per node, writing 128 MBytes each process (4 GBytes in total).

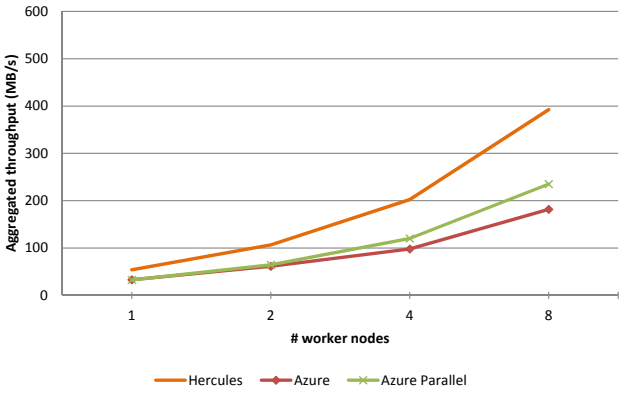


(b) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for reads. We set up the experiment with 8 clients and 4 process per node, reading 128 MBytes each process (4 GBytes in total).

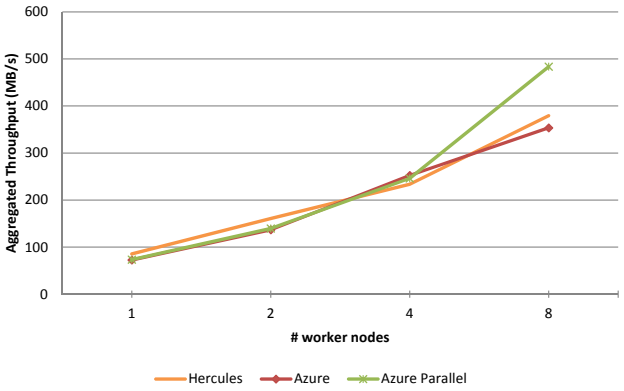
Figure 6: File copy benchmark configured for evaluating the Hercules I/O nodes scalability. 32 worker processes running on eight worker nodes (4 processes per node) access 4 Gigabyte of data. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in the best read cases.

V.3 Worker nodes strong scalability

The last test cases focus on evaluating the behavior of our solution with an increasing number of worker nodes accessing the Hercules storage system. The objective is to evaluate the impact of the congestion against Azure Storage. The test cases are equivalent to the previous test cases, with Hercules using always 8 I/O nodes, while Azure Storage is evaluated using the native approach and the optimized parallel implementation. The aim of this test is to study a strong scalability scenario, where an increasing number of worker nodes perform the same total work: writing 8x512 MB files, a total problem size of 4096 MB, and then reading them. As expected, as the number of worker nodes increases, the total available band-



(a) Throughput varing the worker nodes from 1 to 8, writing 8 files (512 MBytes) per node. We set up the experiment with 8 I/O nodes in case of Hercules.



(b) Throughput varing the worker nodes from 1 to 8, reading 8 files (512 MBytes) per node. We set up the experiment with 8 I/O nodes in case of Hercules.

Figure 7: File copy benchmark configured for comparing Azure Storage and Hercules performance with an increasing number of worker nodes accessing to the storage concurrently. Hercules is configured with 8 I/O nodes and from 1 to 8 worker nodes access to the storage systems concurrently. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in most cases.

width increases at the same pace, leading to better peak throughput performance, but the bottleneck continues at client-side.

Figure 7 shows the same trends already explained in the previous test cases. In Figure 7(a), which represent the aggregated throughput in write operations, can be seen how Hercules is always reaching the theoretical peak performance of each configuration, and how its performance is better than Azure Storage in every case, even doubling the performance in the most favorable one.

In the read operations performance case, Figure 7(b), again Hercules takes advantage of the available bandwidth in every case and competes really well with Azure Storage but the case of 8 clients where the Azure Parallel performance is better.

From the results of the evaluation, we can conclude that Hercules is capable of fully utilize the available bandwidth of every infrastructure where it has deployed. Furthermore, the scalability is assured in any case, on one hand when the number of I/O nodes deployed increases and, on the other hand, when the number of concurrent worker nodes scales and the congestion is higher. Compared to Azure Storage, our proposed solution is up to 2x better in performance during write operations and competes on equal conditions on read operations. Furthermore, should be noted that every test case evaluated in this work uses the best possible configuration for Azure Storage, as explained at the beginning of this section, and it could be predicted the same performance for Hercules in other scenarios while Azure Storage is expected to be penalized.

The potential of our proposed solution is clearly exposed in this preliminary benchmark evaluation. However, we are still working on test cases with a greater number of workers and I/O nodes to better show the scalability capabilities of the Hercules storage system deployed on a cloud infrastructure. Our final objective is to find the limitations in performance of Azure Storage and to evaluate how many number of Hercules I/O nodes are needed to achieve a comparable performance.

VI. Related work

The continued growth in popularity of *many-task computing* has caused many researchers to focus on research to improve the performance of storage systems, one of the major bottlenecks in this type of paradigms.

Previous solutions for providing in-memory storage are Parrot, Chirp, and AHPIOS. Parrot [7] is a tool to adapt existing systems using a remote I/O through the POSIX interface and Chirp [8]. Chirp is a user-level filesystem for collaboration across distributed platforms such as clusters, clouds, and grid computing systems.

AHPIOS (Ad-Hoc Parallel I/O system for MPI applications) [5] is a fully scalable system for I/O parallel MPI applications. AHPIOS relies on dynamic partitions and elastic demand partitions for distributed deployment applications. AHPIOS provides different memory caches levels. Hercules shares many of its features: (1) the user-level deployment without special privileges, transparency using a widely and easy deployment by using simple commands, (2) Hercules is designed to achieve high scalability and performance by leveraging many compute nodes as possible for I/O nodes, (3) Hercules uses main memory for temporal storage in order to improve performance in access. Costa et al. [1, 2] propose using the file attributes of MosaStore to provide communication between the workflow engine and file system by using hints. The workflow engine can provide these hints directly to the file system or file system can infer patterns by analyzing the data. The MosaStore approach is radically different from Hercules, because it uses a centralized metadata server rather than a focus on easy deployment and fully distributed as is our proposal. This server could became a bottleneck in large-scale systems.

The AMFS framework [9] offers programmers a simple scripting language for scripting execution of parallel applications in memory. Hercules shares with AMFS and treatment approach distributed metadata. A difference in AMFS must explicitly specify which data is to memory and what will be persistent while the goal is to be able to offer Hercules persistence transparently to the programmer.

HyCache+ [10] is a distributed storage middleware that allows

effectively use the network bandwidth of the high-end massively parallel systems. HyCache + acts as main storage of recently accessed data (metadata, intermediate results for the analysis of large-scale data, etc.), and only exchange data asynchronously with the remote file system. One of the similarities between HyCache+ and Hercules is fully distributed metadata approach, the usega of computer network rather than the network shared storage, and high scalability. HyCache+ is totally based on POSIX while Hercules offers the possibility of using a POSIX interface and *get/set* operators. HyCache+ focuses on improving parallel file systems, while Hercules is designed to accelerate workflow execution engines, facilitating the exploitation of data locality in current cloud-based applications.

There are also studies that focus on the study of performance storage platforms in the cloud. Zhao et al. [11] compares the I/O performance of S3FS, HDFS, and FusionFS [12]. As demonstrated in the experimental evaluation conducted in this paper, the performance obtained by Hercules equals or exceeds S3FS.

VII. Conclusions and future work

In this work we have presented the integration of the Hercules system and the Data Mining Cloud Framework in order to design and evaluate an ad-hoc storage system for temporary data produced inside data analysis workflow applications.

The evaluation results discussed in this paper clearly demonstrate the potential performance of Hercules, which is able to use more than 80% of the available bandwidth in every case and showing its scalability capabilities in every evaluated scenario. The performance achieved by Hercules is up to 2x the performance of Azure Storage in write operations while our proposed solution has been proved competitive in any scenario with read operations against the cloud storage service evaluated here.

Given the good results of this preliminary evaluation, our objective in the near future is to evaluate Hercules in more complex scenarios, with an increasing number of workers and I/O nodes, to better know the potential capabilities to work together with DMCF, and in addition to investigate the limitations of Azure Storage. Furthermore, it will be interesting to evaluate Hercules against Azure Storage in scenarios where Azure Storage is expected to have worse performance: changing the chunk size, changing the file size, changing the access patterns, etc.

After this first analysis of the capabilities of Hercules in complex cases, we will continue working in the integration of Hercules and DMCF, and in the evaluation of the price/performance ratio reached by Hercules in contrast with different cloud storage services. The final objective of our joint research is a fully working DMCF solution using Hercules as temporary storage for real data analysis applications.

Acknowledgement

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS). This work is partially supported by the grant TIN2013-41350-P, *Scalable Data Management Techniques for High-End Computing Systems* from the Spanish Ministry of Economy and Competitiveness.

References

[1] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. The case for a versatile storage system. *Operating Systems Review*, 44(1):10–14, 2010.

[2] L.B. Costa, H. Yang, E. Vairavanathan, A. Barros, K. Maheshwari, G. Fedak, D. Katz, M. Wilde, M. Ripeanu, and S. Al-Kiswany. The case for workflow-aware storage:an opportunity study. *Journal of Grid Computing*, pages 1–19, 2014.

[3] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. A hierarchical parallel storage system based on distributed memory for large scale systems. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 139–140, New York, NY, USA, 2013. ACM.

[4] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[5] Florin Isaila, Francisco Javier Garcia Blas, Jesús Carretero, Wei-Keng Liao, and Alok Choudhary. A Scalable Message Passing Interface Implementation of an Ad-Hoc Parallel I/O System. *Int. J. High Perform. Comput. Appl.*, 24(2):164–184, May 2010.

[6] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2015.

[7] Douglas Thain and Miron Livny. Parrot: Transparent user-level middleware for data-intensive computing. *Scalable Computing: Practice and Experience*, 6(3), 2005.

[8] Douglas Thain, Christopher Moretti, and Jeffrey Hemmes. Chirp: a practical global filesystem for cluster and grid computing. *Journal of Grid Computing*, 7(1):51–72, 2009.

[9] Zhao Zhang, Daniel S. Katz, Timothy G. Armstrong, Justin M. Wozniak, and Ian Foster. Parallelizing the execution of sequential scripts. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 31:1–31:12, New York, NY, USA, 2013. ACM.

[10] Dongfang Zhao, Kan Qiao, and Ioan Raicu. Hycache+: Towards scalable high-performance caching middleware for parallel file systems. In *IEEE/ACM CCGrid*, 2014.

[11] Dongfang Zhao, Xu Yang, Iman Sadooghi, Gabriele Garzoglio, Steven Timm, and Ioan Raicu. High-Performance Storage Support for Scientific Applications on the Cloud. *ScienceCloud'15*, June 2015.

[12] Dongfang Zhao, Zhao Zhang, Xiaobing Zhou, Tonglin Li, Ke Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu. FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 61–70, Oct 2014.