Evaluating a Data-Aware Scheduling Approach to Reduce Processing Costs of DMCF Workflows

Fabrizio Marozzo*, Francisco Rodrigo Duro†, Javier Garcia Blas†,
Jesus Carretero†, Domenico Talia*, Paolo Trunfio*

* DIMES, University of Calabria, Italy
Email: [fmarozzo, talia, trunfio]@dimes.unical.it

†ARCOS, University Carlos III of Madrid, Spain
Email: [frodrigo, fjblas, jesus.carretero]@arcos.inf.uc3m.es

***Abstract*—As scientific data analysis applications become more and more complex, there is a great need to simplify the definition and execution of such applications, particularly when dealing with large datasets. The Data Mining Cloud Framework (DMCF) is a system allowing domain experts to design and execute complex data analysis workflows on cloud platforms, relying on cloud storage services for every I/O operation. In order to enhance I/O operations, we propose the integration of DMCF with Hercules, an in-memory I/O solution that can be used in combination with DMCF as an alternative to cloud storage services to improve the I/O performance of workflow executions. The integration between DMCF and Hercules is based on a data-aware scheduler that exploits data locality and in-memory I/O to reduce run time of workflows. The goal of this work is to evaluate the reduction of processing costs obtained by using the proposed data-aware scheduler, compared to the costs obtained with the original DMCF solution on the same workflow. The evaluation performed on a 32-node cloud cluster results in 52% reduction of I/O time, which results in 8% total execution time reduction, and 9% of cloud services cost reduction.**

***Index Terms*—Workflows, cost reduction, in-memory storage, data locality, Microsoft Azure.**

I. Introduction

Workflow management systems are computing platforms widely used today for designing and executing data-intensive applications over High-Performance Computing (HPC) systems or distributed infrastructures. Data-intensive workflows consist of interdependent data processing tasks, often connected in a DAG style, which communicate through intermediate storage abstractions, typically files [1]. While workflow management systems deployed on HPC systems (e.g., parallel machines) typically exploit a monolithic parallel file system that ensures highly efficient data accesses, workflow systems implemented on a distributed infrastructure (most often, a public Cloud) must borrow techniques from the Big Data computing (BDC) field, such as exposing data storage locality and scheduling work to reduce data movement in order to alleviate the I/O subsystems under highly demanding data access patterns.

Our previous work [2] has been focused on improving the I/O performance of the Data Mining Cloud Framework (DMCF) [3], a system allowing users to design and execute data analysis workflows on Cloud platforms. The improvement is based on the use of an in-memory I/O accelerator, known as Hercules [4], which is used in DMCF as an alternative to typical cloud storage services. This approach aims to reduce the impact of I/O load on workflows execution times. The integration between DMCF and Hercules is based on a data-aware scheduler that exploits data locality and in-memory I/O accessess, with the goal of reducing run time of workflows [5].

The main goal of this work is to evaluate the reduction of processing costs by using a data-aware scheduler, compared to the costs obtained with the original DMCF solution on the same workflow. Multiple simulations have been carried out on a reference 32-node cloud-based cluster to evaluate the cost reduction of the proposed data-aware scheduling strategy executing data analysis workflows. The simulation results show a 52% reduction in I/O time, which results in 8% total execution time reduction, and 9% of cloud services cost reduction. Monetary cost reduction is achieved for two reasons: *i*) by exploiting data locality, we reduce the total execution time and the price paid for the corresponding CPU time; *ii*) by exploiting in-memory storage, we reduce the amount of cloud storage necessary to stage temporary data and so the corresponding price paid to the cloud provider.

The remainder of the paper is structured as follows. Section II describes the main features of DMCF. Section III discusses Hercules architecture and capabilities. Section

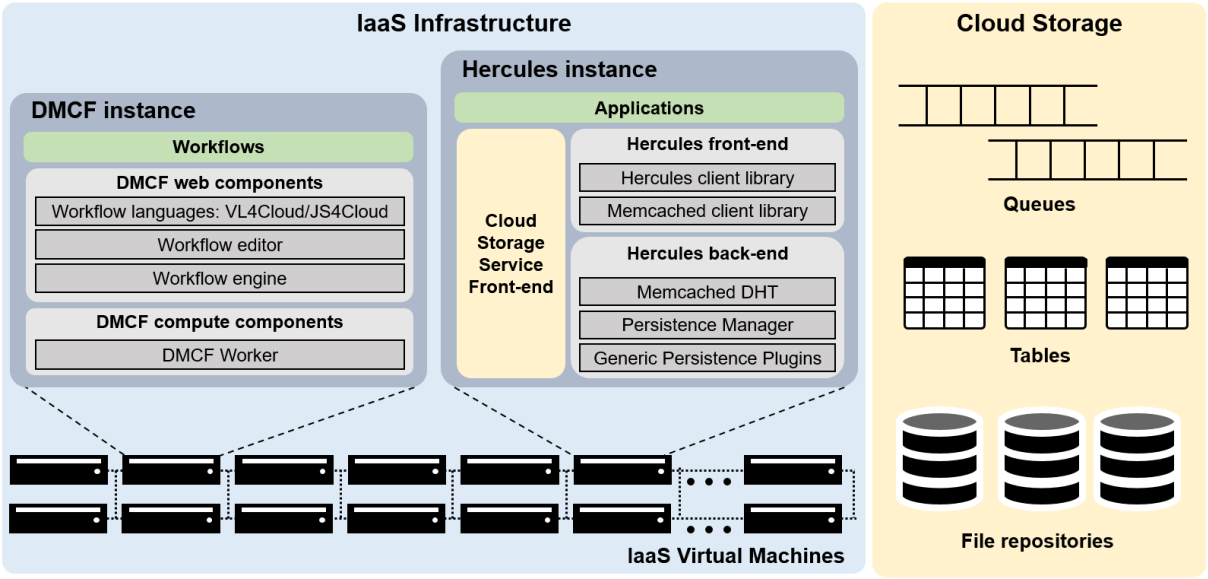


Fig. 1: DMCF and Hercules architecture.

IV emphasizes the advantages of integrating DMCF and Hercules and outlines how the integration works. Section V introduces the cost model used for the evaluation. Section VI presents the evaluation results. Section VII discusses related work. Finally, Section VIII concludes the work and outlines future work.

II. Data Mining Cloud Framework overview

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

The DMCF architecture has been designed to be deployed on different Cloud settings. Currently, there are two different deployments of DMCF: *i*) on top of a Platform-as-a-Service (PaaS) cloud, i.e., using storage, compute, and network APIs that hide the underlying infrastructure layer; *ii*) on top of an Infrastructure-as-a-Service (IaaS) cloud, i.e., using virtual machine images (VMs) that are deployed on the infrastructure layer. In both deployment scenarios, we use Microsoft Azure[1] as cloud provider.

The DMCF software modules can be grouped into *web components* and *compute components* (see top-left part of Figure 1). DMCF allows users to compose, check, and run data analysis workflows through a HTML5 web editor. The workflows can be defined using two languages: *VL4Cloud* (Visual Language for Cloud) [3] and *JS4Cloud* (JavaScript for Cloud) [7]. Both languages use three key abstractions:

- *Data* elements, representing input files (e.g., a dataset to be analyzed) or output files (e.g., a data mining model).
- *Tool* elements, representing software tools used to perform operations on data elements (partitioning, filtering, mining, etc.).
- *Tasks*, which represent the execution of Tool elements on given input Data elements to produce some output Data elements.

The DMCF editor generates a JSON descriptor of the workflow, specifying what are the tasks to be executed and the dependency relationships among them. The JSON workflow descriptor is managed by the DMCF workflow engine that is in charge of executing workflow tasks on a set of workers (virtual processing nodes) provided by the Cloud infrastructure. The workflow engine implements a data-driven task parallelism that assigns workflow tasks to idle workers as soon as they are ready to execute. Further details on DMCF execution mechanisms are given in Section IV.

III. Hercules overview

Hercules [4] is a distributed in-memory storage system based on the key/value Memcached database [8]. The distributed memory space can be used by the applications as a virtual storage device for I/O operations. Hercules has been adapted in this work for being used as an alternative to cloud storage service, offering in-memory shared storage for applications deployed over cloud infrastructures.

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

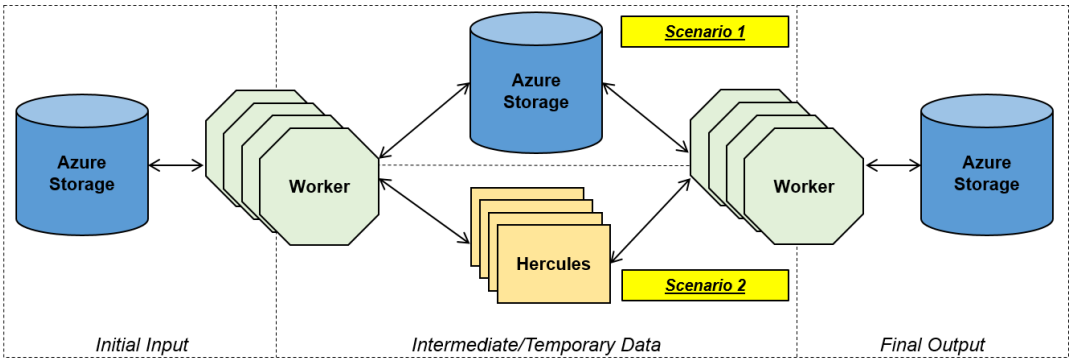


Fig. 2: Workflow execution scenarios between DMCF and Hercules.

Hercules architecture (see top-center part of Figure 1, labeled Hercules instance) has two main layers: front-end (Hercules client library) and back-end (server layer). The user-level library is used by the application (or DMCF workers) for accessing to the Hercules back-end. The library features a layered design, while back-end components are based on enhanced Memcached servers that extend basic functionality with persistence and tweaks.

Hercules offers four main advantages: scalability, easy deployment, flexibility, and performance.

Scalability is achieved by fully distributing data and metadata information among all the available nodes, avoiding the bottlenecks produced by centralized metadata servers. Data and metadata placement is completely calculated at client-side by a hashing algorithm. The servers are completely stateless.

Easy deployment and flexibility at worker-side is provided by a POSIX-like user-level interface in addition to the classic put/get approach existing in current NoSQL databases. This approach supports legacy applications with minimum changes. Servers can also be deployed without root privileges.

Performance and flexibility are targeted at server-side by exploiting I/O parallelism. The capacity of dynamically deploying as many Hercules nodes as necessary provides the flexibility feature. The combination of both approaches results on each node being accessed independently, multiplying the total throughput peak performance.

IV. Integration between DMCF and Hercules

DMCF and Hercules can be configured according with two main deployment scenarios to achieve different levels of integration (see Figure 2):

- *Scenario 1*: Every I/O operation is performed against the cloud storage service offered by the cloud provider (Azure Storage, in the current implementation). There are, at least, four disadvantages in using this approach: proprietary interfaces, I/O contention in the service, lack of configuration options, and persistence-related costs unnecessary for temporary data.
- *Scenario 2*: Initial input and final output are stored on persistent Azure storage, while intermediate data are stored on Hercules in-memory nodes. Hercules I/O nodes share virtual instances with the DMCF workers.

Figure 3 describes, with further implementation details, the second scenario of integration between DMCF and Hercules. Four main components are present: DMCF Worker daemon, Hercules daemon, Hercules client library, and Azure client library.

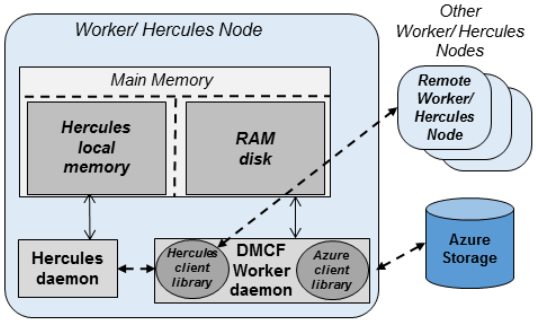


Fig. 3: DMCF and Hercules daemons.

The DMCF workers are in charge of executing the tasks of the workflow (data analysis tools/applications), Hercules daemons act as I/O nodes (storing data in-memory and managing data accesses), the Hercules client library is intended to be used by the applications to access to the data stored in Hercules (query Hercules daemons), and the Azure client library is used to read/write data from/to the Azure storage.

We designed data placement mechanisms that combine DMCF load-balancing capabilities and Hercules data distribution for implementing a data-aware scheduling strategy [5]. Data placement mechanisms focus on group-

ing data related to the same task, while the data-aware scheduler policy targets the co-location of compute task in the nodes where the data can be found in-memory.

Exploiting the new data-aware scheduling strategy, the DMCF Worker cyclically checks whether there are tasks ready to be executed in the Task Queue. If so, a task is removed from the Task Queue and its status is changed to 'running'. To take advantage of data locality, the task removed from the queue is the one having the highest number of input data that are available on the local storage of the worker. This differs from the original data-locality agnostic scheduling policy adopted in DMCF [3], in which each worker picks and executes the task from the queue following a FIFO policy. Then, the transfer of all the needed input resources (files, executables and libraries) is performed from their location (Hercules local or remote node) to two local folders and the Worker locally executes the task and waits for its completion.

V. Costs Model for In-Memory Storage on Clouds

This section presents an overview of the cost model used for evaluating the processing costs of workflows in DMCF and Hercules deployed over the Azure platform. The most characteristic feature of this model is its focus on the cost of I/O-related operations, taking into account every cost related with data access (storage, I/O operations, persistence, etc.) that is a key point for data-intensive applications.

The model requires information about the application and the platform where that workflow is going to be executed.

- *Application.* The model requires details about the application: the CPU time needed for each task in the workflow benchmarked in the cloud platform, the I/O operations (put/get) performed by each task, and the size of these operations (input, output, and temporary files).
- *Platform.* It is necessary to know the main cost concepts: VM deployment, I/O operations, storage of data, and persistence. In addition, it is necessary to measure the performance of the I/O operations in the platform to calculate the time spent on I/O operations.

The *total execution cost* of a given application (C_{TOTAL}) is denoted as the sum of the costs of the cloud storage services (C_{CSS}) and the costs of compute instances (C_{CCI}):

$$C_{TOTAL} = C_{CSS} + C_{CCI} \quad (1)$$

Both costs depend on the characteristics of the application, the configuration of the infrastructure, and the execution time, i.e. the time needed for executing an application is lower using two computing instances than using one, but the cost of deploying two virtual machines is greater than deploying one during the same amount of time. Although the model was initially based on the Amazon AWS cost concepts, it is completely compatible with Microsoft Azure pricing. The main relevant difference between Amazon AWS and Azure is the CPU time pricing strategy, with hour slots in Amazon EC2 and minute slots in Azure. Details on the model are available in [9].

C_{CSS} is obtained by calculating the total execution time based on the characteristics of the infrastructure allocated (number of VMs for DMCF workers and Hercules I/O nodes). The total execution time includes CPU time and I/O time, which is dynamically calculated taking into account the I/O performance of the platform for each specific configuration. On the other hand, C_{CCI} depends on the number of I/O operations, the size of the data stored, and, as in the previous case, the total execution time. We include only the cost of storing the data in the cloud storage service during the execution of the workflow. This is the minimum possible cost and other approaches could be discussed based on the characteristics of the workload (requirements of persistence, iterations over the same data, etc.).

VI. Evaluation of computation costs

This section presents the cost-based evaluation of the DMCF and Hercules integration using the model introduced in Section V. The evaluation is focused on emulating the execution time and processing cost of a real workflow executed in the Microsoft Azure IaaS platform. We have simulated the execution using three alternative configurations:

- *Azure-only*: every I/O operation of the workflow is performed by DMCF using the Azure storage service (Scenario 1 in Figure 2).
- *Locality-agnostic*: a full integration between DMCF and Hercules is exploited, where each intermediate data is stored in Hercules, while initial input and final output are stored on Azure. DMCF workers and Hercules I/O nodes share resources (they are deployed in the same VM instance), however, every I/O operation is performed over remote Hercules I/O nodes through the network (Scenario 2 in Figure 2).
- *Data-aware*: based on the same deployment as in the previous case, this scenario is based on a full knowledge of data location, and executes every task in the same node where data is stored, leading to fully local accesses over temporary data. Based on this locality exploitation, most I/O operations are performed in-memory rather than through the network (Scenario 2 in Figure 2).

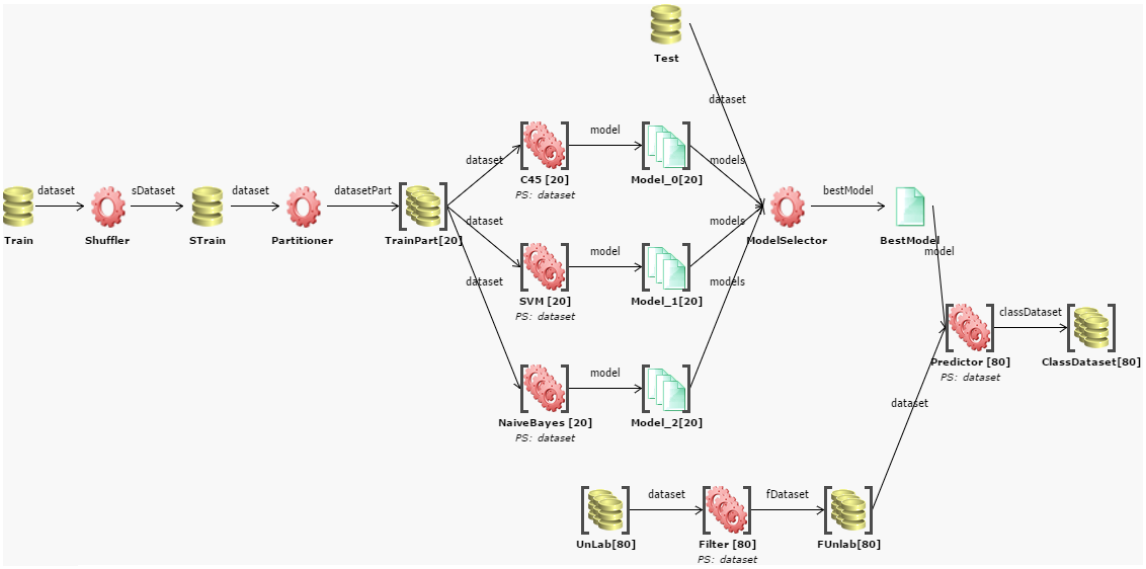


Fig. 4: Classification VL4Cloud workflow.

This evaluation focuses on the cost of execution of DMCF improved with Hercules over the Azure-only approach, which is the default scenario in the pay-per-use model of cloud platforms.

The evaluation is based on a data mining workflow that analyzes n partitions of a training set using k classification algorithms so as to generate kn classification models. The kn models generated are then evaluated against a test set by a model selector to identify the best model. Then, n predictors use the best model to produce in parallel n classified datasets. The classification algorithms used in the workflow are C4.5 [10], Support Vector Machine (SVM) [11] and Naive Bayes [12], that are three major classification algorithms [13]. The training set, test set, and unlabeled dataset, that represent the input of the workflow, have been generated from the *KDD Cup 1999*'s dataset[2], which contains a wide variety of simulated intrusion records in a military network environment.

Figure 4 shows the VL4Cloud version of the data mining workflow. The visual formalism clearly highlights the level of parallelism of the workflow, expressed by the number of parallel paths and the cardinality of tool array nodes (*20* for the three data classifiers and *80* for the Filter and the Predictor).

Table I lists all the read/write operations performed during the execution of the workflow on each data node. Each row of the table describes: *i*) the number of files included in the data array node; *ii*) the total size of the data array; *iii*) the total number of read operations performed on the files included in the data array; and *iv*) the total number of write operations performed on the files included in the data array. Similarly, Table II lists the number of tasks and their average execution time associated to each tool node.

TABLE I: Read/write operations performed during the execution of the workflow.

Data node	N. of files	Total size	Number of read operations	Number of write operations
Train	1	100MB	1	-
Strain	1	100MB	1	1
TrainPart	20	100MB	60	20
Model	60	≈20MB	60	60
Test	1	50MB	1	-
BestModel	1	300KB	80	1
UnLab	80	8GB	80	-
FUnLab	80	≈8GB	80	80
ClassDataset	80	≈6GB	-	80

TABLE II: Computing operations performed during the execution of the workflow.

Tool Node	Number of instances	Average execution time in secs
Shuffler	1	1
Partitioner	1	1
C45	20	288
SVM	20	600
NaiveBayes	20	791
Filter	80	104
ModelSelector	1	9
Predictor	80	2,321

The simulation results are based on synthetic bandwidth measurements performed over the Azure infrastructure and the Azure cost concepts[3]. The benchmark

[2]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99

[3]https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/ and
https://azure.microsoft.com/es-es/pricing/details/storage/blobs/

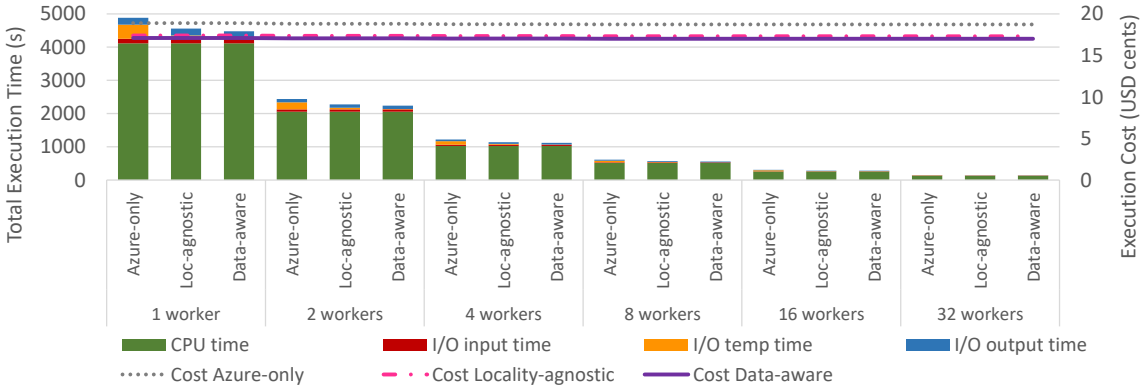


Fig. 5: Breakdown of the estimated execution time of the workflow deployed over different configurations, using up to 32 DMCF workers. The secondary axis shows the cost of execution of the workflow.

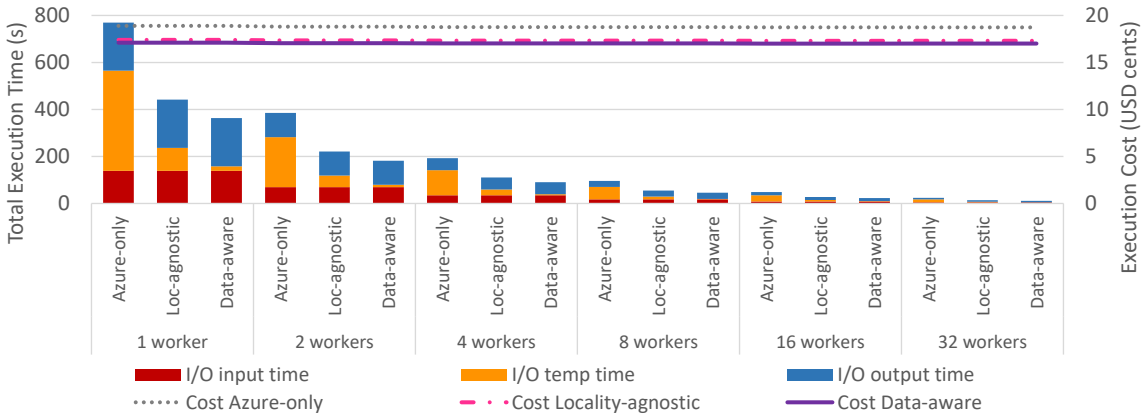


Fig. 6: Detailed view of the results present in Figure 5. This figure discards CPU time, and focuses on the time spent in I/O operations. In the secondary axis the results are enclosed between 14 and 19 to better show the cost differences between cases.

application performs write and read operations over a 256 MB file with a 4 MB chunk size. We have deployed the application on Azure D2_v2 VM instances in the West Europe region. The results are summarized in Table III and represent the expected I/O performance of the application when deployed over each evaluated configuration. Table IV presents the cost concepts taken into account in the simulation.

TABLE III: Synthetic bandwidth measurements performed over the Azure IaaS platform.

Solution	Read op.	Write op.
Azure storage	60 MB/s	30 MB/s
Hercules remote	175 MB/s	180 MB/s
Hercules local	800 MB/s	1,000 MB/s

Figures 5 and 6 plot the breakdown of the total execution time and the cost of execution of the previously introduced workflow deployed over configurations ranging from 1 to 32 D2_v2 VM instances in the Azure platform. Both figures represent the same information with a different level of detail.

TABLE IV: Azure platform Cost concepts considered in the simulation.

Concept	Cost
D2_v2 VM Instance	0.136 USD/h
PUT op. (per 10,000)	0.108 USD (GRS-HOT)
GET op. (per 10,000)	0.004 USD (GRS-HOT)
GB of data (per month)	0.0392 USD (GRS-HOT)

Figure 5 clearly shows how the performance of the application scales with the number of worker nodes available, reducing the total execution time. However, it should be noted how the total cost of execution remains constant, because the VM instances are used during a smaller amount of time. In fact, the efficiency of the execution scales linearly, paying the same money and receiving the results in less time.

Figure 6 increases the level of detail of Figure 5 to better show the differences between configurations. The figure avoids showing the time spent in CPU operations because it is not affected by the use of Hercules, and zooms in the

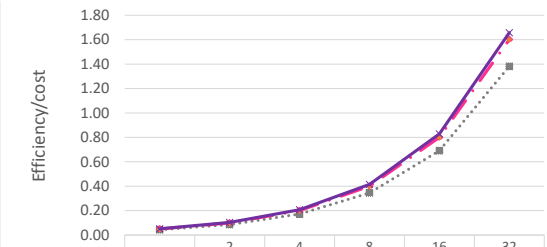
associated execution cost. The execution time breakdown now is focused only on I/O-related operations, and clearly shows how Hercules greatly benefits the I/O operations performed over temporary data, the only operations affected by the deployment of the in-memory infrastructure. It should be noted how in the case of locality-aware cases, the time spent in data accesses over temporary data is reduced to almost a negligible time. Following the results presented in [5], the total execution time is reduced by 8% (6% in locality-agnostic cases), I/O-related time is reduced by 52% (42% in locality-agnostic cases) and, focusing only in temporary data, where Hercules is specifically applied, the time is reduced by up to 95% (77% in locality-agnostic cases).

In Figure 5, the execution cost remains almost linear, but Figure 6 better shows how the application of Hercules reduces costs by around 9% (8% for locality-agnostic cases) thanks to the reduction of total execution time and the reduction of costs associated with I/O operations performed over temporary data.

We can conclude that, on the basis of this evaluation, our solution not only reduces execution time, but thanks to this phenomenon plus to the reductions in I/O operations performed over temporary data through the Azure Storage service, our solution also provides cost cuts for users. We would like to highlight how the time spent in I/O operations over temporary data in the application evaluated, represents a small fraction of the total execution time, which implies that these results can be further improved in other data-intensive applications with a stronger use of temporary data.

Finally, we consider that the implications of reducing the execution time and cost at the same time in Figures 5 and 6 can be improved. In order to better show the combination of both improvements, Figure 7 depicts a new metric labeled as efficiency/cost. As its name suggests, this metric divides the efficiency of execution of the application by the incurred execution costs. Efficiency is calculated by dividing ideal execution time by the real execution time. Ideal execution time is calculated as the total CPU time needed by one CPU to execute the application, excluding any overhead (load balance, I/O-related time, etc.), while the real execution is the total execution time calculated in the simulation. As the number of worker nodes deployed increases, the total execution time is reduced, and the efficiency shows both the speedup and the overhead reductions, which better express the impact of deploying more DMCF workers and Hercules I/O nodes.

Figure 7, shows the combination of both efficiency and execution cost. This metric shows an improvement close to 20% using Hercules (16% in locality-agnostic scenarios) in contrast with the default Azure Storage



	1 worker	2 workers	4 workers	8 workers	16 workers	32 workers
Azure-only	0.04	0.09	0.17	0.35	0.69	1.38
Loc-agnostic	0.05	0.10	0.20	0.40	0.80	1.60
Data-aware	0.05	0.10	0.21	0.41	0.83	1.66

Fig. 7: This figure shows a metric named efficiency/cost which aims to show how the combination of time and cost reductions benefit each configuration.

only approach, which implies a better utilization of the resources available and better utilization of the budget available.

VII. Related work

Due to the increasing popularity of data-intensive workflows and the expected I/O bottlenecks, there is extensive literature focusing on solving this challenge.

Parrot and Chirp. On the one hand, Parrot [14] attaches existing applications to remote I/O solutions offering a POSIX interface. On the other hand, Chirp [15] is a user-level filesystem for collaboration in distributed systems (clusters, clouds, grids, etc.). They are usually deployed together as a distributed file system ready to be used by existing programs coded with the POSIX API. Hercules takes some hints from their design: support of legacy applications through a highly used interface, user-level deployment without root requirements, and easy deployment running a simple command per server. Hercules is designed to achieve high scalability and performance taking advantage of as many compute nodes as possible for I/O operations. Hercules uses main memory for storage improving performance in data-locality aware accesses.

MosaStore. MosaStore [16] proposed by Costa et al. [17] extends the POSIX's attributes in order to communicate hints about the data access patterns between the workflow engine and the file system. The main difference between MosaStore and Hercules is its centralized metadata server in contrast with our fully-distributed approach with easy and flexible deployment.

AMFS. AMFS shell [18] is a simple scripting language for running parallel scripting applications, taking advantage of in-memory storage in large-scale systems. The objective of this solution is similar to the DMCF and Hercules combination, but DMCF additionally offers a Graphical User Interface (GUI) for visually designing

workflows. AMFS and Hercules share the distributed metadata approach.

HyCache+. HyCache+ [19] is a distributed storage middleware that allows I/O to effectively leverage the high bisection bandwidth of the high-speed interconnect of massively parallel high-end computing systems. HyCache+ caches hot information of the shared filesystem (e.g. metadata or intermediate or temporary results of workflows) and asynchronously swaps cold data with the shared file system. Some similarities between HyCache+ and Hercules are their fully distributed metadata approach and the high scalability capabilities. HyCache+ relies on POSIX while Hercules offers the possibility to use a POSIX-like interface in addition to put/get operations. HyCache+ is focused on enhancing parallel file systems in a generic way while Hercules has been designed to work specifically with a many-task engine, exposing and exploiting data locality in current applications. HyCache+ and Hercules share similar ideas but Hercules is ready to be deployed to improve many-task I/O performance focusing on easy and flexible deployment options.

VIII. Conclusions

This work evaluated the reduction of processing costs of data analysis workflows in the Data Mining Cloud Framework (DMCF) using the in-memory storage features of Hercules, compared to the costs registered with the original DMCF solution based on the use of standard cloud storage services. The evaluation showed a 52% improvement in I/O performance, which resulted in 8% execution time reduction, and 9% of cloud services cost reduction. The result demonstrates that the in-memory approach of Hercules, coupled with the locality-aware scheduler of DMCF, is able to exploit data locality in data-intensive applications, which results in a valuable reduction of execution time and costs.

Future work will focus on the evaluation of the approach by testing the system on workflows characterized by different workloads (i.e., compute-intensive vs data-intensive applications). Additionally, we will further study the trade-off of allocating VMs with a greater amount of RAM memory for the Hercules infrastructure, and how this change in configuration affects performance and cost. Finally, we will investigate the possibility of dropping items from cache when they are no longer available, in order to reduce the memory necessary for storing temporary data over Hercules.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data Analysis in the Cloud*. Elsevier, October 2015. ISBN 978-0-12-802881-0.

[2] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Domenico Talia, and Paolo Trunfio. Exploiting in-memory storage for improving workflow executions in cloud platforms. *The Journal of Supercomputing*, pages 1–20, 2016.

[3] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. A workflow management system for scalable data mining on clouds. *IEEE Transactions On Services Computing (IEEE TSC)*, 2017.

[4] 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.

[5] Fabrizio Marozzo, Francisco Rodrigo Duro, Javier Garcia Blas, Jesus Carretero, Domenico Talia, and Paolo Trunfio. A data-aware scheduling strategy for dmcf workflows over hercules. In *In Proceedings of the Third International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2016)*, pages 37–44, Sofia, Bulgaria, 2016.

[6] F. Marozzo, D. Talia, and P. Trunfio. A cloud framework for big data analytics workflows on azure. *Advances in Parallel Computing*, 23:182–191, 2013.

[7] 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*, 27(17):5214–5237, 2015.

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

[9] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. *I/O-Focused Cost Model for the Exploitation of Public Cloud Resources in Data-Intensive Workflows*, pages 244–257. Springer International Publishing, Cham, 2016.

[10] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[11] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[12] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[13] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007.

[14] Douglas Thain and Miron Livny. Parrot: Transparent user-level middleware for data-intensive computing. *Scalable Computing: Practice and Experience*, 6(3):9–18, 2005.

[15] 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.

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

[17] 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.

[18] 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.

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