Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016)
Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

A generic I/O architecture for data-intensive applications based on in-memory distributed cache

Francisco Rodrigo Duro, Javier Garcia Blas, Jesus Carretero

University Carlos III, Spain

frodrigo@arcos.inf.uc3m.es, fjblas@arcos.inf.uc3m.es, jesus.carretero@uc3m.es

Abstract

The evolution in scientific computing towards data-intensive applications and the increase of heterogeneity in the computing resources, are exposing new challenges in the I/O layer requirements. We propose a generic I/O architecture for data-intensive applications based on in-memory distributed caching. This solution leverages the evolution of network capacities and the price drop in memory to improve I/O performance for I/O-bounded applications adaptable to existing high-performance scenarios. We have showed the potential improvements of our proposed solution applied on three scenarios: clusters, cloud, and mobile cloud computing environments.

Keywords Ultrascale systems, NESUS, generic I/O architecture, distributed I/O, data-intensive applications, workflow, cloud computing, in-memory storage

I. Introduction

In the last decade, the scientific computing scenario is greatly evolving in two main areas. First, the focus in scientific computation is changing from CPU-intensive jobs like large scale simulations or complex mathematical applications towards a data-intensive approach. This new paradigm greatly affects the underlying architecture requirements, slowly vanishing the classical CPU bottleneck and exposing bottlenecks in current I/O systems.

Second, the evolution in computing technologies and science funding restrictions are changing the computing resources available in the scientific community. Cloud computing offers a virtually limit-less pool of computing resources in a pay-per-use approach, but most of the research institutions still have access to clusters or supercomputing resources. This heterogeneity in the nature of the available resources leads to new demands in the flexibility of the I/O layer, requiring a more generic approach.

Current trends in bandwidth and latency improvements in high-speed networks in conjunction with the RAM price drop and the near advent of non-volatile memory, present a bright opportunity for improving I/O performance through the use of in-memory I/O solutions. The possibility of using spare memory in compute nodes, and the performance offered by state-of-the-art network technologies, can lead to distributed in-memory solutions where the number of I/O nodes deployed can be flexibly adjusted depending on the performance required by each application, or even by each different experiment. This flexibility in the number of I/O nodes can tackle the I/O bottleneck present in current parallel file systems using fixed configurations.

We propose a new generic I/O architecture for data intensive applications based on in-memory distributed cache targeting both the I/O bottlenecks and the heterogeneity of computing resources. The architecture design is guided by four main objective: flexibility, scalability, performance, and ease of deployment. In an effort to demonstrate the flexibility and capabilities of our solution, we present three different successful scenarios where our proposed solution has been applied: a workflow engine running on a cluster infrastructure,

a data mining framework running on a cloud infrastructure, and a mobile cloud computing scenario.

II. Thesis idea

The main goal of this thesis is to propose a novel generic I/O architecture design for an in-memory storage system based on distributed caching [2]. As shown in Figure 1, the front-end of the architecture is a user-level library and the back-end consists of Memcached servers enhanced with persistence and other performance tweaks. The memory distributed among the server nodes is offered to the user as a unified storage space that can be accessed through the use of easy-to-use APIs: POSIX-like, MPI-IO, and put/get.

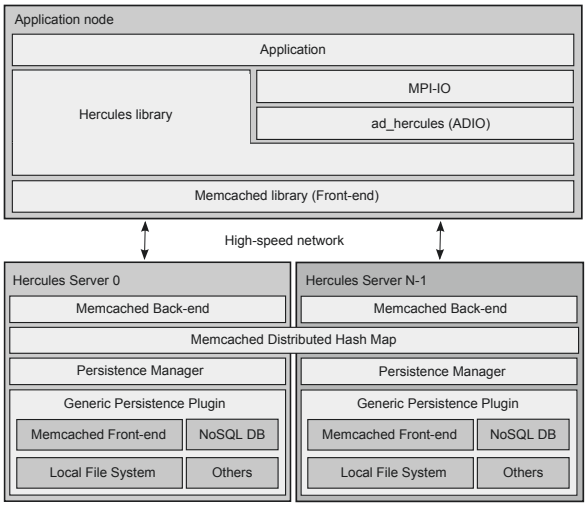

Figure 1: Current version of our proposed generic I/O architecture, namely Hercules [3]

Internally, the I/O nodes behave as stateless servers composing a distributed key-value store where data and metadata are completely distributed. The unified memory space is used as a virtual device. In every key-value pair stored, the key acts as the block ID, and the value represents the block contents. Thanks to this approach, every block ID can be calculated instead of being stored, simplifying the algorithms for data placement and retrieval.

The architecture design targets four objectives: scalability, flexibility, easy deployment, and performance. **Scalability** is achieved by fully distributing data and metadata among all the available I/O nodes, avoiding any possible bottleneck derived from centralized services. Data placement is fully calculated client-side by a hashing algorithm, minimizing storage and communications for data retrieval.

Flexibility is tackled in both client and server sides. On the front-end, the APIs offered to the user are widely used in existing applications, facilitating the use of existing applications with minimum changes. The layered design simplifies the addition of new APIs and persistence plugins. On the back-end, the servers are completely state-less, permitting the deployment of any number of I/O nodes depending on the characteristics of the infrastructure, even on different levels of the I/O hierarchy if necessary. The only information needed by the clients are the IP addresses of the I/O nodes. The servers, on the other end, do not need any information about other servers running on the same hierarchy level.

Ease of deployment is especially important in order to design an architecture as generic as possible. Both the user-level library and the I/O nodes can be deployed on any Linux system in user mode, without requiring any special privileges.

Performance-wise, our solution supports parallel I/O accesses to enhance applications throughput. Each I/O node available can be accessed independently, multiplying the maximum throughput peak performance. Furthermore, the multi-threading implementation increases the level of parallelism for serving requests.

Scalability, flexibility, and easy deployment work together to adjust the system for the best possible performance required by each situation. The user can deploy as many I/O nodes as necessary depending on the throughput requirements of each application, or even for different runs of the same application.

III. Application scenarios

This work presents an I/O architecture design aiming to be generic. In order to demonstrate the capabilities of our I/O solution for adapting to different infrastructures, we present three different scenarios where our proposed architecture has been successfully applied.

III.1 Workflow engine over cluster infrastructure

The first scenario consists of deploying our in-memory architecture as an I/O accelerator for the Swift/T workflow engine [3] in collaboration with Argonne National Laboratory (USA), developer of the Swift/T workflow engine and runtime.

This scenario is motivated by the I/O contention suffered by classic parallel file systems available in HPC infrastructures, in applications with a high number of worker nodes accessing concurrently to the shared file system. Classic parallel file systems are deployed in a static configuration, thus number of I/O nodes available for the applications can not be dynamically configured. The aggregated bandwidth of the I/O nodes is shared among all the workers accessing concurrently, which is translated in high I/O contention during peak I/O loads.

As shown in Figure 2 our solution (labeled as Hercules) is deployed as an alternative storage space for temporary files in the workflow life-cycle. Most of the files generated by each task of the workflow are consumed by other task. Deploying one Hercules I/O node sharing resources with each worker node, we target two main objectives.

First, the number of I/O nodes scales with the number of worker nodes available. This is translated into a better scalability in the maximum available bandwidth available for I/O operations, especially when compared with the default shared file system.

Second, the possibility of exposing and exploiting data locality. Our storage space is allocated using spare memory of the worker nodes. Offering information about data placement to the scheduler can expose data locality. Co-locating tasks and data in the same node, data locality can be exploited. Additionally, the data placement policy is also optimized for data locality purposes. Another advantage offered by this approach is the isolation from the shared file system noise obtained through the deployment of I/O nodes dedicated to one specific application.

Evaluated against GPFS, our solution scales better when the number of available worker nodes is increased. In the most extreme cases, our proposed solution was able of converting an I/O bounded problem (where the total execution time increased when scaling the worker nodes as a result of I/O contention) into a CPU-bounded application (where the execution time always decreased while increasing the number of worker nodes available).

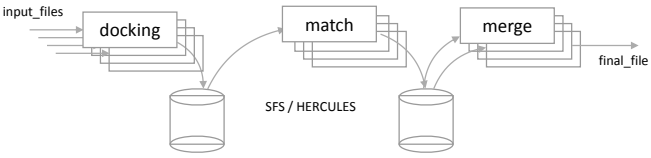

Figure 2: Example of workflow. Temporary files can be stored in the default shared file system or in Hercules for improving maximum throughput and data locality [3]

III.2 Data mining framework over cloud infrastructure

The objective targeted by this second scenario is shared with the previous one, aiming to accelerate the I/O accesses over temporary files in a data mining workflow through the use of in-memory storage. The main difference is the infrastructure where the workers are deployed, using cloud resources instead of a cluster. The idea behind this scenario is a collaboration with the DIMES group at University of Calabria (Italy), developers of the Data Mining Cloud Framework (DMCF) [5].

This collaboration shows the potential performance of our proposed solution deployed over the Microsoft Azure infrastructure and evaluated against the Azure Storage, the default storage provided by Microsoft. The collaboration has followed with the full integration of DMCF and Hercules, and it is still active for exposing and exploiting data locality.

In order to show the flexibility of our solution, additionally, it has been deployed over another cloud provider, Amazon AWS in this case. Hercules was deployed on Amazon EC2 instances and evaluated against S3 using S3FS and I/O performance was evaluated through specifically designed micro-benchmarks, with successful results [4].

III.3 Mobile cloud computing scenario

In 2013 we developed CoSMiC, a version of our proposed architecture especially adapted for the emerging Mobile Cloud Computing field. Leveraging the ease of deployment and flexibility of our architecture, the objective of this work was improving the storage capabilities of mobile devices, especially on public places and limited connectivity scenarios.

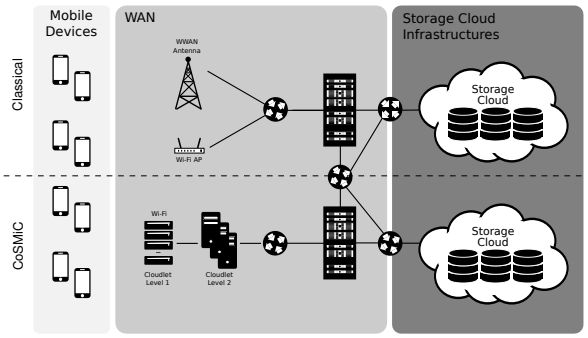

Figure 3: Application of our generic architecture into a Mobile Cloud Computing scenario, based on the cloudlet concept [1]

As shown in Figure 3 our solution presents an alternative data path for mobile device users based on the cloudlet concept. The advantage of this approach is a result of the proximity of the storage in contrast with the classic cloud approach. Due to this proximity, mobile device storage is expanded, latency is significantly reduced, and energy-efficiency is improved through the use of Wi-Fi instead of 3G/HSDPA/4G. MNOs are also benefited, relieving the pressure in their WAN infrastructures by caching popular contents in public places, especially on highly crowded scenarios, leading to a win-win situation for every participant.

IV. Conclusions and future work

This Thesis presents a new generic I/O architecture for data intensive applications based on in-memory distributed cache. Our solution tackles the I/O system bottlenecks exposed by new trends of scientific computing while tends to be generic in order to be usable in legacy HPC infrastructures and other resources gaining popularity such as public clouds.

The flexibility and performance capabilities of our proposed solution are presented as four heterogeneous scenarios where our solution has been successfully applied, supported by publications on prestigious international journals, conferences, and workshops.

Acknowledgment

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] Francisco Rodrigo Duro, Francisco Javier García Blas, Daniel Higuero, Oscar Pérez, and Jesús Carretero. CoSMiC: A hierarchical cloudlet-based storage architecture for mobile clouds. *Simulation Modelling Practice and Theory*, 50:3–19, 2015.

[2] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. A Hierarchical parallel storage system based on distributed memory for large scale systems. EuroMPI '13, pages 139–140, New York, NY, USA, 2013. ACM.

[3] Francisco Rodrigo Duro, Javier Garcia Blas, Florin Isaila, Justin Wozniak, Jesus Carretero, and Rob Ross. Exploiting data locality in Swift/T workflows using Hercules. NESUS 2014, pages 71–76, Porto, Portugal, 2014. UC3M.

[4] Francisco Rodrigo Duro, Javier Garcia-Blas, Florin Isaila, and Jesus Carretero. Experimental evaluation of a flexible I/O architecture for accelerating Workflow engines in cloud environments. DISCS '15, pages 6:1–6:8, New York, NY, USA, 2015. ACM.

[5] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Jesus Carretero, Domenico Talia, and Paolo Trunfio. Evaluating data caching techniques in DMCF workflows using Hercules. NESUS 2015, pages 95–106, Krakow, Poland, 2015.