JAWS – Just Another Workspace Suite
Data Orchestration for HPC Environments

DOCTORAL DISSERTATION COLLOQUIUM

EXTENDED ABSTRACT

Mehmet Soysal
Karlsruhe Institute of Technology (KIT),
Steinbuch Centre for Computing (SCC),
Karlsruhe, Germany
mehmet.soysal@kit.edu

Dissertation Advisor: Achim Streit

***Abstract*—The data volume in HPC environments is increasing rapidly. To meet this demand, different storage systems are installed hierarchically. A large number of storage systems and various access methods overwhelm users.**

A data management system is required to ensure the efficient use of storage systems and infrastructure. This system should simplify the use of the various storage systems. A concept for JAWS (Just Another Workspace Suite) has been developed. JAWS abstracts the different storage types and technologies as well as the infrastructure. As a centralized data management service, JAWS orchestrates the movement of data between the different storage systems. It can be easily integrated into HPC environments to facilitate scientific work.

By using a batch system for moving data, the efficient use of the infrastructure is increased. It helps to orchestrate the enormous amounts of data generated in scientific computing, for operators and users. The developed prototype is available under MIT license: https://github.com/mehsoy/jaws.

***Index Terms*—datasets, workspace, data mangement, data movement, metadata, HPC, supercomputing, data orchestration**

I. Introduction

High-performance computing (HPC) systems become larger and more powerful. This allows users to perform simulations with more complex models, higher resolution, larger domains, and more simulation runs. Consequently, a lot of data is generated and the demand for storage increases enormously, which poses problems for users and operators of the HPC Systems. Nowadays, simulations that create millions of files and/or terabyte of data are not uncommon [1], [2]. Other scientist develop methods to reduce the amount of data, so only the mandatory parts of the results are written [3]. These can limits the possibilities of follow-up examinations. Furthermore, new user communities are emerging, which generate, process and analyze huge amounts of data. This trend, known as data-intensive computing, is changing the demands on the systems. Data-intensive communities have also an increased capacity requirement. It can be assumed, that the demand for data storage will continue to grow. To meet this demand with the usual high-performance storages, like Lustre [4], GPFS [5] or BeeGFS [6], a significant financial investment is required. A common solution to solve this challenge is to install different type of storages in a hierarchy [7]. Each storage tier in this hierarchy focus a specific requirement. The range from this hierarchy is from expensive, fast and parallel, low-capacity storage to cheaper, slower, high-capacity storages. As required, a HPC center can introduce several layers to the storage hierarchy. For example, on-demand file systems are offered at SCC. These are created with the node-local SSDs if required, thus representing a further tier at the top [2]. In a deeper hierarchical storage environments, more cost and performance efficient data storage is possible. But this is at the expense of simplicity. As a result, the user often only uses the fast parallel storage. A movement of data between the hierarchical layers are often avoided. However, in some scientific domains, there are community built tools that can help with the management and migration from storage to storage. These are usually tailored to specific workflows and often not prepared for generic usage in HPC centers.

The current situation leads to the fact that the expensive storage is preferred by the users. Therefore the capacity of these storage devices is often unnecessarily occupied. To address this issue a concept has been developed, that helps to automate the task of data orchestrating. Furthermore, the prototype JAWS – Just Another Workspace Suite, has been developed. JAWS can be used as a central service and reduce the complexity of storage hierarchies.

To help automate data management, JAWS provides the ability to move data in a batch queued manner. JAWS can also be seen as the bridge between user and operator. Users can add metadata to the workspaces, e.g., a DOI number could show that the data has been published. So that the operator knows whether this data should still be kept.

The main contribution of this work is the concept for data management, within HPC environments, which led to the

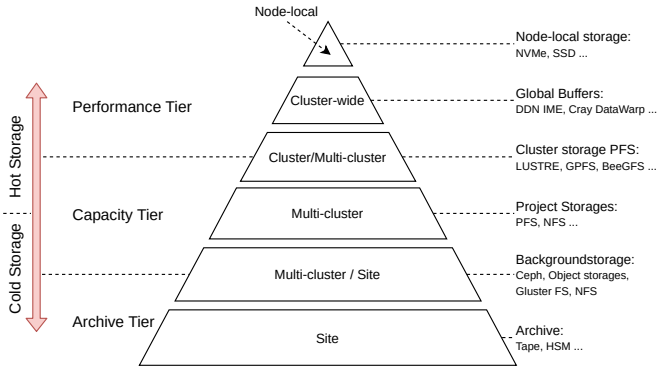

Fig. 1. Different storage tiers available in HPC centers.

implementation of JAWS. In this paper, the background is explained and the concept for the decisions. Furthermore the design and implementation is explained. The implementation is online available and should be considered as work in progress. The paper is structured as follow; in Chapter 2 the current state and the motivation for this work is explained. Chapter 3 explains the concept, design and architecture of the data management framework. A brief introduction in the related work is given in Chapter 4 and ate the end a conclusion and future work is given.

II. Background and Motivation

a) Storages in HPC centers: Storage systems in HPC environments are distinguished based on the requirements of the intended use. Figure 1 presents a typical hierarchy pyramid of storages and a possible classification. The top of the pyramid is classified as the *performance-tier*, where the fastest storages are located, in terms of bandwidth and latency. But due to the costs, these usually offer a low capacity. The bottom of the pyramid can be classified as an *archive-tier*. The storage here provides high capacity for long term storage, but has a reduced performance, e.g., high latency with tape storages. The middle layer is filled with a compromise of speed and capacity, this is often called *capacity-tier*.

A second view to the pyramid provides the location of the storage. It starts from the top with the node-local storages, which is only accessible from the corresponding compute node. The end at the bottom is built of storage systems, which are site-wide accessible. The left side in Figure 1 annotate the distinction between cold and hot storages. Hot denotes a suitable data location for starting jobs. Starting jobs with data in cold storages is inefficient because the delivery of data is too slow. To reduce the waste of compute resources it is preferred that data is always located on a hot storage. For example, parallel file systems like Lustre, GPFS or BeeGFS are often used as hot storage. On the right side of Figure 1 some more examples of storage and file systems are present.

Of course, it is important to note that different storage types and file system should be used in a different manner for efficient usage. For example, it is best practice to move data in a archived format to a tape based storage, e.g., tarball or as a zip-archive. The various storages and different access options can easily overwhelm users.

III. JAWS

JAWS is designed to automate the data management process within HPC environments. The key design aspects are (i) simplify the usage of the different storage systems, (ii) efficient use of infrastructure and storage systems, (iii) a simple way to identify unused data, (iv) a role based system, and (v) an easy integration to HPC environments. To archive this first some elements of JAWS are introduced.

a) Workspace: A workspace is defined as a set of directories and files. The workspace has an associated expiration date and a global unique identifier. Furthermore, additional metadata can be included to provide a more detailed description of the content.

b) Target: A target describes the storage type, the storage location and properties. Moreover, the way to access the storage are configured. So it is possible to configure two different targets on the same storage system, e.g, one target could be configured so that the workspaces are stored here as one compressed archive. Targets have also configured properties, such as a maximum lifetime of workspaces, and number of allowed extensions of this lifetime. A special property classifies the target as hot or cold. This indicates whether the workspace is located on a suitable storage for a batch job start.

c) Data Job: For physical operations like moving and deleting a workspace, a data job is created. The data job is submitted to a queuing system and will be processed by workers.

d) Queuing System: A simple queuing system to efficiently use the infrastructure and storage systems is implemented. The master controls the queue and the workers. It allocates data jobs to free workers and keeps track of the progress.

e) Roles: A role define the privileges of different user groups. Beside a user and admin role, a group manager role is also provided. A group manager is associated with users and has configurable privileges over these users' workspaces.

A. Design and Architecture

JAWS is developed using python 3 and the Flask framework [8]. Flask is a web micro-framework, which helps developing web based application with a template engine. Figure 2 illustrates the components of the JAWS framework. The main daemon is offering a REST API [9]. JAWS provides different clients, a web-interface and a Command Line Interface (CLI), both use the same API calls.

a) JAWS Daemon: The main service is composed of several components. The two components workspace manager and application/controller contain the complete logic for managing and controlling the data management.

Another important component is the queuing system consisting of the job queue and the corresponding master. The master is the controller for the job queue and communicates with the workers. All jobs that make any changes on

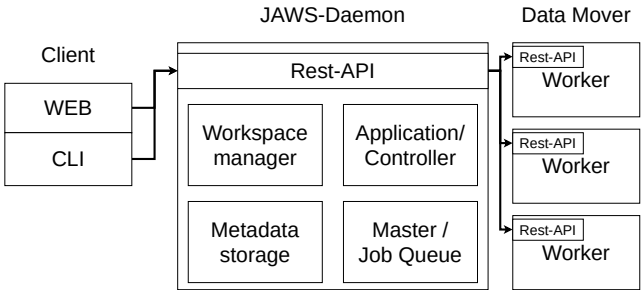

Fig. 2. Design of the JAWS framework. Clients (Web/CLI) access the main daemon via REST API. JAWS Daemon communicates via REST API with distributed worker.

```
~$ jaws --help
usage: jaws <command> [<args>]

JAWS - CommandLineTool

----- Currently available Command -----
move, activate, active, cancel, configuration,
resume, queue, job, log, resume, rights,
deactivate, setPriority, workers, share,
shareInfo, workspaces, storages

positional arguments:
  command     Subcommand to run

optional arguments:
  -h, --help  show this help message and exit
```

Fig. 3. JAWS cli-frontend listing the available commands.

workspaces are placed in the job queue. The job queue ensures that the infrastructure can be used efficiently and that no congestion situations can occur.

The metadata of workspaces, such as lifetime, is stored in the metadata storage. Modifications to the metadata of workspaces are done within the main daemon.

b) Workers: Workers are responsible for physical actions on workspaces, like moving, packing or deleting. Any number of workers can be deployed. Every worker can be configured for multiple targets. At startup, workers register themselves to the master and wait until a job is allocated. During registration, workers indicate which targets they serve.

c) Client: Two different clients are provided to use JAWS. The web client is basically derived from the Flask daemon, which is the base for the JAWS daemon. Therefore, technically speaking, the web-frontend is part of the JAWS daemon. As already mentioned, the web and cli, both uses the same REST API. A complete list of available command is shown in Figure 3. An important difference between both variants is the way of authentication to the main daemon (see next paragraph). More information and detailed manual of the tools can be found in the documentation.

d) Authentication: JAWS offers different ways to authenticate users with the main daemon, based on the access methods. The command line interface offers two optional methods. (i) A random string is created and stored as a file to the user home and to the JAWS database. This string then can be used as a credential to authenticate the user by appending it as a key to the REST API calls. (ii) A more secured method uses the authentication service MUNGE [10] (MUNGE Uid 'N' Gid Emporium). It is highly scalable and often used in HPC cluster environments. When enabling MUNGE, the whole communication between the CLI-clients, the daemon, and the workers are encrypted by MUNGE. Furthermore, additional safety features offered by MUNGE are also used, e.g., replay protection or time-to-live.

To login to the web interface a username and password is needed. After successful authentication a session key is created and stored in the browser. For the user password two optional methods are implemented. (iii) The user password from the system. (iv) The string from (i).

Please note that the solution (i) and (iv) are fallback methods. The generation of the string could be insecure if the JAWS daemon is running on a different host. Therefore, this method should be only used in testing environments. For (iii) the JAWS daemon requires extended privileges. This might be too insecure for some HPC centres, but adding additional method for authentication for a web-based login is easy to implement.

IV. Related work

Even with current solutions the task of data management in HPC environments are still a challenge. There are several solutions available, however, we can only give a brief introduction.

The utility collection HPC-Workspace [11] also uses the basic idea of workspaces, but it has no data movement components implemented. The fields to enrich metadata are very limited. Stork [12] is a scheduler for data placement activities in the Grid. Stork queues, monitors and manages data placement in a fault tolerant manner. However, it does not offer any further metadata to organize data.

The tools Data Jockey [13] and Rucio [14] are similar to JAWS. Both aggregate file and directories to a logic unit – called datasets. Data Jockey was developed to automate the task of bulk data movement and placements of scientific workflows within HPC centres. Rucio is a project that provides services and associated libraries for allowing scientific collaborations to manage large volumes of data spread across facilities at multiple institutions and organisations. Rucio has been developed by the ATLAS [15] experiment. Both tools are very complex and need a tight integration to the system while JAWS is a very lightweight framework. Pegasus [16] is a framework for mapping complex scientific workflows onto distributed systems. It uses catalogs to map files to their physical locations but do not allow any advanced data placement. JAWS in contrast moves data to new locations. It does not have a catalogue for every file, instead it returns the absolute path of the workspace.

Pwrake [17] and ADIOS [18] are introduced for efficiently transferring data over wide area networks. Our aim is the man-

agement of data within a computing center. If the workspaces have to be transferred over remote networks, we could benefit from these tools, so they are not in contrast to this approach.

V. Conclusion and Future Work

In this work, a lightweight framework JAWS to manage data within HPC centers is presented. Files and directories are logically combined to workspaces. These workspaces can be annotated and easily moved between storages. JAWS empowers the users to manage their scientific data across various storage architectures within a compute-site. It transparently provide access to these storages, reducing the need to get familiar with the heterogeneous storage types.

Operators can move orphaned or expired workspaces to back-end storage, this allows to increase the free capacity on the hot storage. The operators can extend their site with new storage technologies. JAWS enable transparent deployment of storage systems and eliminates the need for extensive user training. JAWS helps to optimize storage efficiency by storing only active data in hot storage and moving unused data to less expensive storage. With its lightweight user interface, the user experience is not affected. In the end, the financial resources can be used in a better and more targeted manner, instead of reserving fast storage for inactive data.

For the prototype a simple queue manager is implemented. This queue manager is only provided with the most necessary functions. For future versions, it is planned to use a well established HPC job scheduler. JAWS would then benefit from the comprehensive features of an HPC job scheduler.

Large scale storages are going to get more heterogeneous and more complex. A tool to abstract the usage of these different storages will help user and operators with the usage of such systems. The complexity of a storage system can be hidden by frameworks like JAWS.

VI. Acknowledgements

Parts of the JAWS Framework were developed during a student course. The task at the course was to develop an independent service for copy jobs. This ended up in the Data Movement Daemon (DMD), which has become part of the JAWS project. The students are hereby sincerely thanked for their work and they are of course also listed as authors of the software. This work as part of the project ADA-FS is funded by the DFG Priority Program “Software for Exascale Computing” (SPPEXA, SPP 1648), which is gratefully acknowledged. Also this developemt is funded by the Ministry of Science, Research and the Arts Baden-Württemberg.

References

[1] Thorsten Zirwes, Feichi Zhang, Jordan Denev, Peter Habisreuther, and Henning Bockhorn. Automated code generation for maximizing performance of detailed chemistry calculations in OpenFOAM. In W.E. Nagel, D.H. Kröner, and M.M. Resch, editors, *High Performance Computing in Science and Engineering ’17*, pages 189–204. Springer, 2017.

[2] Mehmet Soysal, Marco Berghoff, Thorsten Zirwes, Marc-André Vef, Sebastian Oeste, Andre Brinkman, Wolfgang E. Nagel, and Achim Streit. Using On-demand File Systems in HPC Environments. *The 2019 International Conference on High Performance Computing and Simulation*, 2019.

[3] Martin Bauer, Johannes Hötzer, Marcus Jainta, Philipp Steinmetz, Marco Berghoff, Florian Schornbaum, Christian Godenschwager, Harald Köstler, Britta Nestler, and Ulrich Rüde. Massively parallel phase-field simulations for ternary eutectic directional solidification. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.

[4] Peter J Braam and Philip Schwan. Lustre: The intergalactic file system. In *Ottawa Linux Symposium*, page 50, 2002.

[5] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST ’02, Berkeley, CA, USA, 2002. USENIX Association.

[6] Jan Heichler. An introduction to BeeGFS, 2014.

[7] Wing N Toy and Benjamin Zee. *Computer Hardware-Software Architecture*. Prentice Hall Professional Technical Reference, 1986. pg. 30.

[8] Miguel Grinberg. *Flask web development: developing web applications with python*. ” O’Reilly Media, Inc.”, 2018.

[9] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine, 2000.

[10] C Dunlap. Munge uid n grid emporium. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2004.

[11] Holger Berger. HPC Workspaces. https://github.com/holgerBerger/hpc-workspace, 2020.

[12] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 342–349. IEEE, 2004.

[13] Woong Shin, Christopher D. Brumgard, Bing Xie, Sudharshan S. Vazhkudai, Devarshi Ghoshal, Sarp Oral, and Lavanya Ramakrishnan. Data Jockey: Automatic Data Management for HPC Multi-tiered Storage Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 511–522, May 2019. ISSN: 1530-2075.

[14] Vincent Garonne, R Vigne, G Stewart, M Barisits, M Lassnig, C Serfon, L Goossens, A Nairz, Atlas Collaboration, et al. Rucio–the next generation of large scale distributed system for atlas data management. In *Journal of Physics: Conference Series*, volume 513, page 042021. IOP Publishing, 2014.

[15] Georges Aad, JM Butterworth, J Thion, U Bratzler, PN Ratoff, RB Nickerson, JM Seixas, I Grabowska-Bold, F Meisel, S Lokwitz, et al. The atlas experiment at the cern large hadron collider. *Jinst*, 3:S08003, 2008.

[16] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, and et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, July 2005.

[17] Masahiro Tanaka and Osamu Tatebe. Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC ’10, page 356–359, New York, NY, USA, 2010. Association for Computing Machinery.

[18] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, CLADE ’08, page 15–24, New York, NY, USA, 2008. Association for Computing Machinery.