Distributed Monitoring and Management of Exascale Systems in the Argo Project

Swann Perarnau[1](✉), Rajeev Thakur[1], Kamil Iskra[1], Ken Raffenetti[1],
Franck Cappello[1], Rinku Gupta[1], Pete Beckman[1], Marc Snir[1],
Henry Hoffmann[2], Martin Schulz[3], and Barry Rountree[3]

[1] Argonne National Laboratory, Lemont, IL 60439, USA
[2] University of Chicago, Chicago, IL 60637, USA
[3] Lawrence Livermore National Laboratory, Livermore, CA 94550, USA
schulzm@llnl.gov

Abstract. New computing technologies are expected to change the high-performance computing landscape dramatically. Future exascale systems will comprise hundreds of thousands of compute nodes linked by complex networks—resources that need to be actively monitored and controlled, at a scale difficult to manage from a central point as in previous systems.

In this context, we describe here on-going work in the Argo exascale software stack project to develop a distributed collection of services working together to track scientific applications across nodes, control the power budget of the system, and respond to eventual failures. Our solution leverages the idea of enclaves: a hierarchy of logical partitions of the system, representing groups of nodes sharing a common configuration, created to encapsulate user jobs as well as by the user inside its own job. These enclaves provide a second (and greater) level of control over portions of the system, can be tuned to manage specific scenarios, and have dedicated resources to do so.

1 Introduction

Disruptive new computing technology has already begun to change the scientific computing landscape. Hybrid CPUs, many-core systems, and low-power system-on-a-chip designs are being used in today's most powerful high-performance computing (HPC) systems. As these technology shifts continue and exascale machines emerge, the *Argo* research project aims to provide an operating system and runtime (OS/R) designed to support extreme-scale scientific computations. To this end, it seeks to efficiently leverage new chip and interconnect technologies while addressing the new modalities, programming environments, and workflows expected at exascale. At the heart of the project are four key innovations: dynamic reconfiguring of node resources in response to workload changes, allowance for massive concurrency, a hierarchical framework for management of nodes, and a cross-layer communication infrastructure that allows resource managers and optimizers to communicate efficiently across the platform. These innovations will result in an open-source prototype system that is expected to form the basis of production exascale systems deployed in the 2020 timeframe.

A. Bessani and S. Bouchenak (Eds.): DAIS 2015, LNCS 9038, pp. 173–178, 2015.
DOI: 10.1007/978-3-319-19129-4_14

Argo is designed with a hierarchical approach. The system is organized in *enclaves*, a set of resources that share the same configuration and can be controlled as a whole. Enclaves can monitor their performance, respond to failures, and control power usage according to a budget. These enclaves form a hierarchy: the top enclave is acting over the whole system, while jobs are contained in their own enclave and the user can subdivide its job enclave further.

We describe here the early stages of an ongoing effort, as part of Argo, to design the services that will take care of creating, keeping track of, and destroying enclaves, as well as monitoring and controlling the resources and failures happening in those enclaves. Our design is influenced by two factors. First, we expect that future machines will differ significantly from current HPC systems in size, failure rate, and fine-grained access to resources. Second, we don't expect the Argo system to behave as commonly available distributed systems such as peer-to-peer or cloud infrastructures. Indeed, an HPC machine will still be composed of known, dedicated resources over which we have total control (as low level as necessary). As a result, we believe that several services typical of a distributed infrastructure can benefit from this unique setup and from being distributed across the hierarchy of enclaves.

In Section 2 we describe more fully the components of the Argo system. In Section 3 we detail typical services we identified as critical to Argo, while also representing distinct communication and control patterns that could benefit from our unique setup. In Section 4 we review related work, and in Section 4 we briefly discuss future work.

2 The Argo Machine: Architecture, Enclaves and Underlying Services

While predicting the exact architecture of future exascale systems is difficult, the Argo project bases its designs on general trends such as those highlighted in the Exascale Software Project Roadmap [1]. We expect the Argo machine to be composed of hundreds of thousands of compute nodes, with each node containing hundreds of cores. Furthermore, those nodes will be linked together by dedicated and highly efficient networks, integrating smart control and monitoring interfaces. We also expect that, in order to meet the U.S. DOE exascale budget limits, complex power management interfaces will be available at all levels of the machine.

The software stack designed by the Argo project to manage such a machine is divided into four key components. First, each compute node will use a customized operating system derived from Linux (*NodeOS*). Second, a runtime taking advantage of massive intranode parallelism (*Argobots*) will be available. Third, a global information bus or *backplane* will provide advanced communication services on top of the native network. In particular, a distributed key-value store and a pub-sub system will be available to other components. Fourth, the *GlobalOS*—the focus of this paper—will manage enclaves and their services.

As stated in Section 1, enclaves are logical groups of nodes sharing the same configuration. They are organized in a tree, whose root is the enclave containing all nodes. Inside an enclave, at least one node (*master*) hosts the services specific to this enclave. Masters communicate with each other to distribute control of the nodes in the system. We note that nodes are members of all enclaves above them in the hierarchy. In other words, a master node controls all nodes in its subtree in the hierarchy of enclaves, and not just the masters of its subenclaves. Enclaves cannot be created inside a compute node: they are logical constructs intended to manage only the distributed part of the Argo machine. Figure 1 gives an example of a hierarchy of four enclaves distributed across a system, with each enclave having its master replicated across several nodes.

This enclave concept is critical to the design of the distributed services managing the Argo machine. In particular, we organize responsibilities so that as much of the node management as possible is delegated to the deepest master in the hierarchy. Conversely, the higher in the hierarchy, the less the interaction between the master and nodes, and the more coarse-grained this interaction is. We give typical examples in the next section.

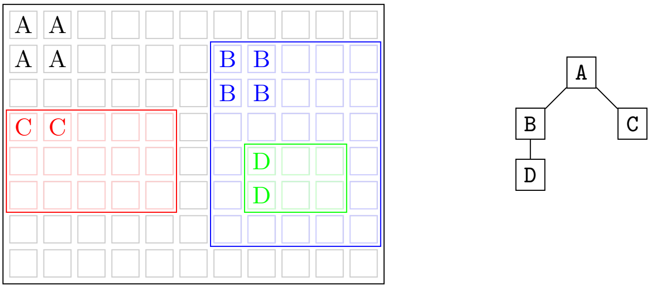

Fig. 1. Example of a hierarchy of 4 enclaves. On the left, the squares represent compute nodes, with the filled ones representing the master and its replica. On the right, the hierarchy of enclaves is represented as a tree.

3 Typical Services in the Argo Machine

Three services critical to GlobalOS can serve as examples of the different ways the enclave hierarchy is used in Argo.

3.1 Hierarchical Control Bus for Enclaves

The Argo system is used like any other HPC system: a user submits to the system a job comprising at least the number of nodes required, the time duration, the configuration (to be deployed across the job), and the script or application to run. In our infrastructure, each job is managed as an enclave, a direct child of the *root* enclave (which contains all nodes). Furthermore, the user can create enclaves inside the job, if he needs distinct parts of the allocated nodes to have different features or different configurations. This process distributes the responsibilities of node management between the masters at different levels of the hierarchy. The root master has the exclusive role of creating enclaves at its level, and does so only when receiving commands from the job scheduler. A master at the *job enclave* level might be responsible only for creating subenclaves; and so on. Similarly, since the user has control over all nodes in a job enclave, commands to reconfigure an enclave, a subtree of an enclave, or just one node might happen at different levels of the hierarchy.

Thus, the chain of command between the root master and a node can be seen as a bus, where commands flow from master to master, going deeper into the hierarchy, with each master affecting the process or injecting new commands until they reach a node. Some commands might be of higher priority (e.g., the root enclave asking for job termination) or be altered (e.g., a command that must span an entire enclave instead of just one node). Nevertheless, all enclave masters are responsible for the continuous working of their enclaves, from the moment the enclave management is delegated to them until the enclave exits (all nodes gracefully exit the enclave, including the master) or is destroyed (massive failure, forceful exit). We are designing this control bus on top of two mechanisms: a logical naming scheme that allows a message or command to be sent to different stages of the control bus (an enclave, a master, or a specific node) and a message broker dedicated to the control bus, present on each node, and forwarding messages across the bus in the right order.

The naming scheme is similar to paths in a filesystem: each enclave is a directory with the root enclave at the top, and nodes are files in those directories. A few special names also exist: **'..'** (parent enclave), **'.'** (closest enclave), and **'*'** (all enclaves/nodes). This naming scheme simplifies sending a command to a specific part of the system, even from a node having only partial knowledge of the hierarchy: it is enough to know the next master in the path. This knowledge is kept in the key-value store available in the Argo system, so that every node knows its own path and how to contact parent and children enclaves. As an example, a node in the enclave D will have the path `/B/D/node`.

The message broker infrastructure is also simple. A message broker runs on each node and uses the key-value store to route messages to the various masters on the bus. This broker also inspects each message to decide whether commands need to be triggered locally or whether the message needs to be altered or redirected. This design avoids specifying how those messages should be transported. Indeed, as several communication services are made available by the backplane, several communication channels might be available at the same time. For example, it might be more efficient to distribute a command across all nodes of an enclave by using a reliable publish-subscribe interface rather than using point-to-point communication. Such choices are being evaluated but might depend on the specific architecture our system is deployed on.

3.2 Distributed Power Management

Since we expect the size of an exascale machine to involve hundreds of thousands of compute nodes, controlling the total power consumption of the system is critical. Indeed, such systems represent tens of megawatts in consumption and a significant cost to any organization. Therefore, idle parts of the system or underutilized resources should be put in low-power modes as much as possible. To do so, we expect the architecture to provide meters distributed across the system to measure the current power usage and interfaces, similar to dynamic frequency scaling or Intel's RAPL [4] on each node. We therefore are designing a distributed power management service as part of GlobalOS.

This service comprises two components: a reader service, running on each node, that periodically reports power consumption to its enclave master and a power control service, installed on enclave masters, that distributes a global power budget across the enclave hierarchy. The power consumption reader design is straightforward. On each node, periodically the local measuring interface is used to gather information, which is then sent to the closest master. Each master then aggregates the data coming from its enclave and sends it to its parent, and so on. This information gathering is made possible by the publish-subscribe service of the backplane, and likewise for the aggregation. The power control service reacts to information coming from the consumption reader as well as global power budget limits set by the machine administrators. The exact algorithms used to distribute this power budget across the hierarchy will be derived from previous work [3,2].

3.3 Managing Failures as Exceptions

Given the expected size of an exascale machine, failures—in both hardware and software—will have a statistically greater chance of occurring than in previous systems. Consequently, all components of Argo are being designed with faults in mind. This effort includes replication of masters across the enclave hierarchy, for example.

While failure detection is a complex issue in itself, our focus here is on reactions to failures. We designed a service distributed across the enclave hierarchy that, when notified of failures, acts like an exception system: each master on the path between the failed component and the root enclave will receive the notification one by one, from the deepest in the hierarchy to the highest, and have a chance to act on it. If a master cannot act or resolve the failure, the parent master will take control, and so on.

The specific action a master takes will depend on the component that failed and on the recovery/reaction strategy active on the failure manager. We plan to design several strategies, ranging from restarting the component to destroying the enclave, that users will be able to configure inside their enclaves. We expect the root enclave to have the most complex strategy, with the additional role of notifying administrators in case of unrecoverable hardware failures, and having sole control of powering off and on nodes, for example.

4 Related Works and Conclusion

Similar projects on a new software stack for exascale systems have recently emerged, including the Hobbes project directed by Sandia National Laboratories in the United States and the post-K project directed by RIKEN in Japan. We are collaborating with them on the design of several components, and we expect these efforts to result in robust and versatile components to manage an entire machine. Recent cloud technologies, such as OpenStack, have also started to address the issues in orchestrating and monitoring distributed resources. These technologies are, however, targeted to systems with smaller numbers of compute nodes, no dedicated high-performance networks, and no static knowledge of the available hardware.

We are still in the design phase of our distributed framework for the provisioning, management, and monitoring of an exascale machine. Our goal is to build on the features of the future architecture and Argo's communication component, focusing on a hierarchical and lightweight solution that is tuned as issues become apparent as the project advances. We hope that in the coming year, with an integrated Argo prototype implemented, our research will move its focus to the study and design of efficient management strategies to be implemented into the various services of the GlobalOS.

Acknowledgement. This work was supported by the U.S. Dept. of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357. Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-669278). The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

References

1. Dongarra, J., Beckman, P., et al.: The International Exascale Software Project Roadmap. International Journal of High Performance Computing Applications 25(1), 3–60 (2011)
2. Ellsworth, D., Malony, A., Rountree, B., Schulz, M.: POW: system-wide dynamic reallocation of limited power in hpc. To appear in International ACM Symposium on High Performance Distributed Computing, HPDC 2015, Portland, OR, USA (2015)
3. Hoffmann, H., Maggio, M.: PCP: A generalized approach to optimizing performance under power constraints through resource management. In: International Conference on Autonomic Computing, ICAC 2014, Philadelphia, PA, USA (2014)
4. Rountree, B., Ahn, D.H., de Supinski, B.R., Lowenthal, D.K., Schulz, M.: Beyond DVFS: A first look at performance under a hardware-enforced power bound. In: International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPSW 2012, Shanghai, China (2012)