Middleware in Modern High Performance Computing System Architectures*

Christian Engelmann, Hong Ong, and Stephen L. Scott

Computer Science and Mathematics Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6164, USA
{engelmannc,hongong,scottsl}@ornl.gov
http://www.fastos.org/molar

Abstract. A recent trend in modern high performance computing (HPC) system architectures employs "lean" compute nodes running a lightweight operating system (OS). Certain parts of the OS as well as other system software services are moved to service nodes in order to increase performance and scalability. This paper examines the impact of this HPC system architecture trend on HPC "middleware" software solutions, which traditionally equip HPC systems with advanced features, such as parallel and distributed programming models, appropriate system resource management mechanisms, remote application steering and user interaction techniques. Since the approach of keeping the compute node software stack small and simple is orthogonal to the middleware concept of adding missing OS features between OS and application, the role and architecture of middleware in modern HPC systems needs to be revisited. The result is a paradigm shift in HPC middleware design, where single middleware services are moved to service nodes, while runtime environments (RTEs) continue to reside on compute nodes.

Key words: High Performance Computing, Middleware, Lean Compute Node, Lightweight Operating System

1 Introduction

The notion of "middleware" in networked computing systems stems from certain deficiencies of traditional networked operating systems (OSs), such as Unix and its derivatives, *e.g.*, Linux, to seamlessly collaborate and cooperate. The concept of concurrent networked computing and its two variants, parallel and distributed computing, is based on the idea of using multiple networked computing systems collectively to achieve a common goal. While traditional OSs contain networking features, they lack in parallel and distributed programming models, appropriate system resource management mechanisms, remote application steering and

* This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

user interaction techniques, since traditional OSs were not originally designed as parallel or distributed OSs. Similarly, traditional OSs also do not differentiate between various architectural traits, such as heterogeneous distributed or massively parallel.

Since the emergence of concurrent networked computing, there have been two different approaches to deal with these deficiencies. While one approach adds missing features to an existing networked OS using middleware that sits in-between the OS and applications, the other approach focuses on adding missing features to the OS by either modifying an existing networked OS or by developing a new OS specifically designed to provide needed features. Both approaches have their advantages and disadvantages. For example, middleware is faster to prototype due to the reliance on existing OS services, while OS development is a complex task which needs to deal with issues that have been already solved in existing OSs, such as hardware drivers.

Software development for high performance computing (HPC) systems is always at the forefront with regards to both approaches. The need for efficient, scalable distributed and parallel computing environments drives the middleware approach as well as the development of modified or new OSs. Well known HPC middleware examples are the Parallel Virtual Machine (PVM) [1], the Message Passing Interface (MPI) [2], the Common Component architecture (CCA) [3], and the Grid concept [4]. Examples for modifications of existing OSs for HPC include the Beowulf Distributed Process Space (BProc) [5], cluster computing toolkits, like OSCAR [6] and Rocks [7], as well as a number of Single System Image (SSI) solutions, like Scyld [8] and Kerrighed [9]. Recent successes in OSs for HPC systems are Catamount on the Cray XT3/4 [10] and the Compute Node Kernel (CNK) on the IBM Blue Gene/L system [11].

A runtime environment (RTE) is a special middleware component that resides within the process space of an application and enhances the core features of the OS by providing additional abstraction (virtual machine) models and respective programming interfaces. Examples are message passing systems, like PVM and implementations of MPI, but also component frameworks, such as CCA, dynamic instrumentation solutions, like Dyninst [12], as well as visualization and steering mechanisms, such as CUMULVS [13].

This paper examines a recent trend in HPC system architectures toward "lean" compute node solutions and its impact on the middleware approach. It describes this trend in more detail with regards to changes in HPC hardware and software architectures and discusses the resulting paradigm shift in software architectures for middleware in modern HPC systems.

2 Modern HPC System Architectures

The emergence of cluster computing in the late 90's made scientific computing not only affordable to everyone using commercial off-the-shelf (COTS) hardware, it also introduced the Beowulf cluster system architecture [14, 15] (Fig. 1) with its single head node controlling a set of dedicated compute nodes. In this ar-

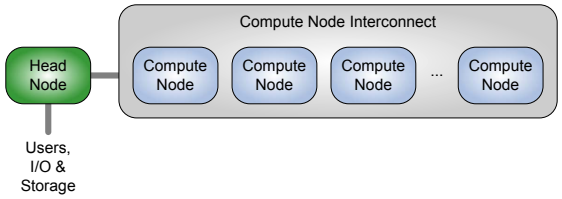


Fig. 1. Traditional Beowulf Cluster System Architecture

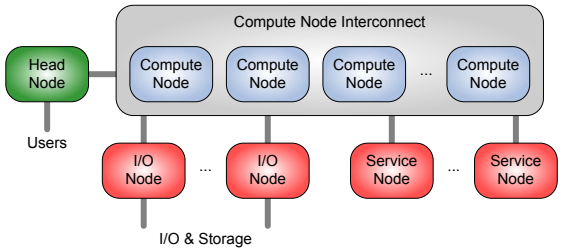


Fig. 2. Generic Modern HPC System Architecture

chitecture, head node, compute nodes, and interconnects can be customized to their specific purpose in order to improve efficiency, scalability, and reliability. Due to its simplicity and flexibility, many supercomputing vendors adopted the Beowulf architecture either completely in the form of HPC Beowulf clusters or in part by developing hybrid HPC solutions.

Most architectures of today's HPC systems have been influenced by the Beowulf cluster system architecture. While they are designed based on fundamentally different system architectures, such as vector, massively parallel processing (MPP), single system image (SSI), the Beowulf cluster computing trend has led to a generalized architecture for HPC systems. In this generalized HPC system architecture (Fig. 2), a number of compute nodes perform the actual parallel computation, while a head node controls the system and acts as a gateway to users and external resources. Optional service nodes may offload specific head node responsibilities in order to improve performance and scalability. For further improvement, the set of compute nodes may be partitioned (Fig. 3), tying individual service nodes to specific compute node partitions. However, a system's architectural footprint is still defined by its compute node hardware and software configuration as well as the compute node interconnect.

System software, such as OS and middleware, has been influenced by this trend as well, but also by the need for customization and performance improvement. Similar to the Beowulf cluster system architecture, system-wide management and gateway services are provided by head and service nodes. However, in contrast to the original Beowulf cluster system architecture with its "fat" compute nodes running a full OS and a number of middleware services, today's HPC systems typically employ "lean" compute nodes (Fig. 4) with a basic OS and

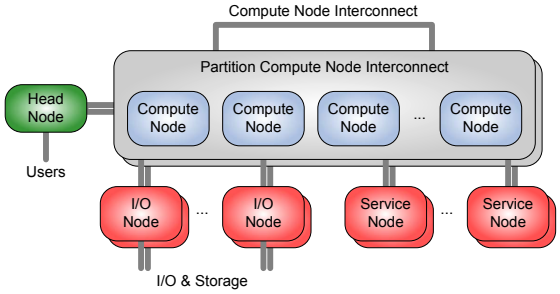
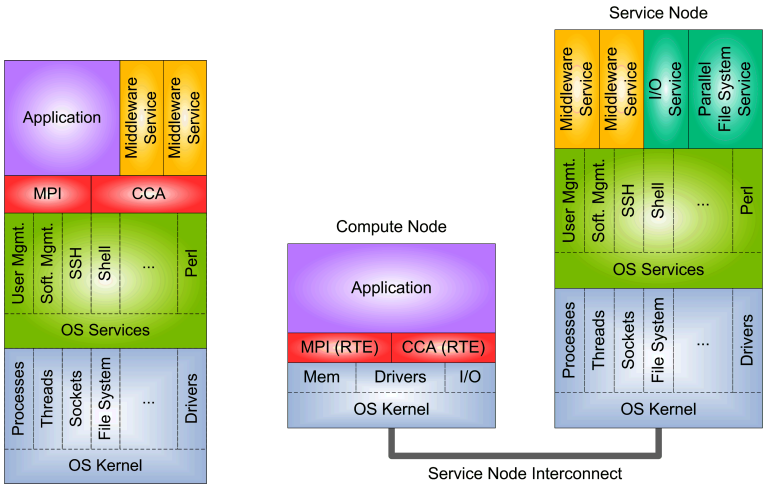


Fig. 3. Generic Modern HPC System Architecture with Compute Node Partitions



(a) Fat (b) Lean

Fig. 4. Traditional Fat vs. Modern Lean Compute Node Software Architecture

only a small amount of middleware services, if any middleware at all. Certain OS parts and middleware services are provided by service nodes instead.

The following overview of the Cray XT4 [16] system architecture illustrates this recent trend in HPC system architectures.

The XT4 is the current flagship MPP system of Cray. Its design builds upon a single processor node, or processing element (PE). Each PE is comprised of one AMD microprocessor (single, dual, or quad core) coupled with its own memory (1-8 GB) and dedicated communication resource. The system incorporates two types of processing elements: compute PEs and service PEs. Compute PEs run a lightweight OS kernel, Catamount, that is optimized for application performance. Service PEs run standard SUSE Linux [17] and can be configured for I/O, login, network, or system functions. The I/O system uses the highly scalable Lustre™ [18, 19] parallel file system. Each compute blade includes four compute

PEs for high scalability in a small footprint. Service blades include two service PEs and provide direct I/O connectivity. Each processor is directly connected to the interconnect via its Cray SeaStar2^{TM}routing and communications chip over a 6.4 GB/s HyperTransport^{TM}path. The router in the Cray SeaStar2^{TM}chip provides six high bandwidth, low latency network links to connect to six neighbors in the 3D torus topology. The Cray XT4 hardware and software architecture is designed to scale steadily from 200 to 120,000 processor cores.

The Cray XT4 system architecture with its lean compute nodes is not an isolated case. For example, the IBM Blue Gene/L solution also uses a lightweight compute node OS in conjunction with service nodes. In fact, the CNK on the IBM Blue Gene/L forwards most supported POSIX system calls to the service node for execution using a lightweight remote procedure call (RPC).

System software solutions for modern HPC architectures, as exemplified by the Cray XT4, need to deal with certain architectural limitations. For example, the compute node OS of the Cray XT4, Catamount, is a non-POSIX lightweight OS, *i.e.*, it does not provide multiprocessing, sockets, and other POSIX features. Furthermore, compute nodes do not have direct attached storage (DAS), instead they access networked file system solutions via I/O service nodes.

The role and architecture of middleware services and runtime environments in modern HPC systems needs to be revisited as compute nodes provide less capabilities and scale up in numbers.

3 Modern HPC Middleware

Traditionally, middleware solutions in HPC systems provide certain basic services, such as a message passing layer, fault tolerance support, runtime reconfiguration, and advanced services, like application steering mechanisms, user interaction techniques, and scientific data management. Each middleware layer is typically an individual piece of software that consumes system resources, such as memory and processor time, and provides its own core mechanisms, such as network communication protocols and plug-in management. The myriad of developed middleware solutions has led to the "yet another library" and "yet another daemon" phenomenons, where applications need to link many interdependent libraries and run concurrent to service daemons.

As a direct result, modern HPC system architectures employ lean compute nodes using lightweight OSs in order to increase performance and scalability by reducing compute node OS and middleware to the absolute necessary. Basic and advanced middleware components are placed on compute nodes only if their function requires it, otherwise they are moved to service nodes. In fact, middleware becomes an external application support, which compute nodes access via the network. Furthermore, single middleware services on service nodes provide support for multiple compute nodes via the network. They still perform the same role, but in a different architectural configuration. While middleware services, such as daemons, run on service nodes, RTEs continue to run on compute nodes either partially by interacting with middleware services on service nodes or com-

pletely as standalone solutions. In both cases, RTEs have to deal with existing limitations on compute nodes, such as missing dynamic library support.

While each existing HPC middleware solution needs to be evaluated regarding its original primary purpose and software architecture before porting it to modern HPC system architectures, new middleware research and development efforts need to take into account the described modern HPC system architecture features and resulting HPC middleware design requirements.

4 Discussion

The described recent trend in HPC system architectures toward lean compute node solutions significantly impacts HPC middleware solutions. The deployment of lightweight OSs on compute nodes leads to a paradigm shift in HPC middleware design, where individual middleware software components are moved from compute nodes to service nodes depending on their runtime impact and requirements. The traditional interaction between middleware components on compute nodes is replaced by interaction between lightweight middleware components on compute nodes with middleware services on service nodes.

Functionality Due to this paradigm shift, the software architecture of modern HPC middleware needs to be adapted to a service node model, where middleware services running on a service node provide essential functionality to middleware clients on compute nodes. In partitioned systems, middleware services running on a partition service node provide essential functionality to middleware clients on compute nodes belonging to their partition only. Use case scenarios that require middleware clients on compute nodes to collaborate across partitions are delegated to their respective partition service nodes.

Performance and Scalability The service node model for middleware has several performance, scalability, and reliability implications. Due to the need of middleware clients on compute nodes to communicate with middleware services on service nodes, many middleware use case scenarios incur a certain latency and bandwidth penalty. Furthermore, central middleware services on service nodes represent a bottleneck as well as a single point of failure and control.

Reliability In fact, the service node model for middleware is similar to the Beowulf cluster architecture, where a single head node controls a set of dedicated compute nodes. Similarly, middleware service offload, load balancing, and replication techniques may be used to alleviate performance and scalability issues and to eliminate single points of failure and control.

Slimming Down The most intriguing aspect of modern HPC architectures is the deployment of lightweight OSs on compute nodes and resulting limitations

for middleware solutions. While the native communication system of the compute node OS can be used to perform RPC calls to service nodes in order to interact with middleware services, certain missing features, such as the absence of dynamic linking, are rather hard to replace.

Service-Oriented Middleware Architecture However, the shift toward the service node model for middleware has also certain architectural advantages. Middleware services may be placed on I/O nodes in order to facilitate advanced I/O-based online and/or realtime services, such application steering and visualization. These services require I/O pipes directly to and from compute nodes. Data stream processing may be performed on compute nodes, service nodes, and/or on external resources. System partitioning using multiple I/O nodes may even allow for parallel I/O data streams.

5 Conclusion

This paper describes a recent trend in modern HPC system architectures toward lean compute node solutions, which aim at improving overall system performance and scalability by keeping the compute node software stack small and simple. We examined the impact of this trend on HPC middleware solutions and discussed the resulting paradigm shift in software architectures for middleware in modern HPC systems. We described the service node model for modern HPC middleware and discussed its software architecture, use cases, performance impact, scalability implications, and reliability issues.

With this paper, we also try to engage the broader middleware research and development community beyond those who are already involved in porting and developing middleware solutions for modern HPC architectures. Based on many conversations with researchers, professors, and students, we realize that not many people in the parallel and distributed system research community are aware of this trend in modern HPC system architectures.

It is our hope that this paper provides a starting point for a wider discussion on the role and architecture of middleware services and runtime environments in modern HPC systems.

References

1. Geist, G.A., Beguelin, A., Dongarra, J.J., Jiang, W., Manchek, R., Sunderam, V.S.: PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA (1994)
2. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge, MA, USA (1996)
3. SciDAC Center for Component Technology for Terascale Simulation Software (CCTTSS): High-Performance Scientific Component Research: Accomplishments and Future Directions. Available at http://www.cca-forum.org/db/news/documentation/whitepaper05.pdf (2005)

4. Kesselman, C., Foster, I.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco, CA, USA (1998)
5. Hendriks, E.: BProc: The Beowulf distributed process space. In: Proceedings of 16^{th} ACM International Conference on Supercomputing (ICS) 2002, New York, NY, USA (2002) 129–136
6. Hsieh, J., Leng, T., Fang, Y.C.: OSCAR: A turnkey solution for cluster computing. Dell Power Solutions (2001) 138–140
7. Papadopoulos, P.M., Katz, M.J., Bruno, G.: NPACI Rocks: Tools and techniques for easily deploying manageable Linux clusters. In: Proceedings of IEEE International Conference on Cluster Computing (Cluster) 2001, Newport Beach, CA, USA (2001)
8. Becker, D., Monkman, B.: Scyld ClusterWare: An innovative architecture for maximizing return on investment in Linux clustering. Available at http://www.penguincomputing.com/hpcwhtppr (2006)
9. Morin, C., Lottiaux, R., Valle, G., Gallard, P., Utard, G., Badrinath, R., Rilling, L.: Kerrighed: A single system image cluster operating system for high performance computing. In: Lecture Notes in Computer Science: Proceedings of European Conference on Parallel Processing (Euro-Par) 2003. Volume 2790., Klagenfurt, Austria (2003) 1291–1294
10. Brightwell, R., Kelly, S.M., VanDyke, J.P.: Catamount software architecture with dual core extensions. In: Proceedings of 48^{th} Cray User Group (CUG) Conference 2006, Lugano, Ticino, Switzerland (2006)
11. Moreira, J., Brutman, M., Castanos, J., Gooding, T., Inglett, T., Lieber, D., McCarthy, P., Mundy, M., Parker, J., Wallenfelt, B., Giampapa, M., Engelsiepen, T., Haskin, R.: Designing a highly-scalable operating system: The Blue Gene/L story. In: Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2006, Tampa, FL, USA (2006)
12. Buck, B.R., Hollingsworth, J.K.: An API for runtime code patching. Journal of High Performance Computing Applications (2000)
13. Kohl, J.A., Papadopoulos, P.M.: Efficient and flexible fault tolerance and migration of scientific simulations using CUMULVS. In: Proceedings of 2^{nd} SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT) 1998, Welches, OR, USA (1998)
14. Sterling, T.: Beowulf cluster computing with Linux. MIT Press, Cambridge, MA, USA (2002)
15. Sterling, T., Salmon, J., Becker, D.J., Savarese, D.F.: How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters. MIT Press, Cambridge, MA, USA (1999)
16. Cray Inc., Seattle, WA, USA: Cray XT4 Computing Platform Documentation. Available at http://www.cray.com/products/xt4 (2006)
17. Novell Inc.: SUSE Linux Enterprise Distribution. Available at http://www.novell.com/linux (2006)
18. Cluster File Systems, Inc., Boulder, CO, USA: Lustre Cluster File System. Available at http://www.lustre.org (2006)
19. Cluster File Systems, Inc., Boulder, CO, USA: Lustre Cluster File System Architecture Whitepaper. Available at http://www.lustre.org/docs/whitepaper.pdf (2006)