Unified Execution Environment

Geoffroy R. Vallée*, Thomas Naughton, Christian Engelmann and David Bernholdt
* (865) 574-3152 / valleegr@ornl.gov
Oak Ridge National Laboratory
Computer Science and Mathematics Division
Oak Ridge, TN 37831, USA

The design and development of new system software for HPC (both operating systems and run-times) face multiple challenges, including scalability (high level of parallelism), efficiency, resiliency, and dynamicity. Guided by these fundamental design principles, we advocate for a unified execution environment, which aims at being scalable, asynchronous, dynamic, resource efficient, and reusable. The proposed solution is based on the following core building blocks, (i) events, (ii) agents, and (iii) enclaves. We use these building blocks to support composable environments that may be tailored to combine appropriate system services as well as user jobs. Additionally, for resilience and scalability the proposed design encourages localized or regional operations to foster autonomy of execution contexts. We advocate this approach for exascale systems, which include a massive number of heterogeneous computing resources, since it enables architecturally informed structures (topologies) as well as encouraging efficient grouping of functionality/services.

Events trigger notifications and interactions inside the system. This enables the implementation of asynchronous capabilities and therefore improves scalability of the system. In order to support a wide range of interactions between system entities, multiple event types are included: real time (time bounded delivery), standard (delivery guaranteed without any time constraint), and optional (event can be dropped). These different types of events also limit the risk of overflowing the event system when many events are triggered simultaneously. The event system also implements a progress model to ensure that events are dispatched and delivered correctly, and be used to throttle events to avoid overload.

Agents instantiate system services as well as user applications. This composition promotes a more efficient allocation of resources due to the fact that applications and system services are combined into a single entity that may work more tightly on resource utilization, etc. Agents also include the concept of contract. A contract involves the definition of common interfaces and contextual properties or dependencies [4]. These additional dependencies allow common interfaces to be supplemented for specialization to take advantage of unique capabilities in an underlying implementation. For instance, it is possible to define a specific agent for the management of memory that guarantees reliability. In fact, based on the underlying hardware, this agent could implement replication techniques or simply rely only on the hardware. This also enables the definition of system services that support varied levels of service guarantees (reliability), which ultimately facilitate scalability and resiliency. Finally, a system library can be used either to implement new tools, run-times, system services or even applications (unified system), or encapsulate existing software. Because agent setup is based on input from the user, resource manager, job scheduler, and even compilers, it enables the optimization of resource allocation for the system services required for the execution of a given job, as well as the deployment of new system services that assist running applications (for instance, a system service could be deployed to enable background compression of application data).

Enclaves are the last key concept. An enclave is a logical group of agents that have tight interactions. An enclave is designed to be light-weight (low system footprint), sparse, and dynamic (a enclave can dynamically grow and shrink). Each enclave instantiate its own event engine, which is used for interactions between agents in the same enclave. They also support a simple event API for inter-enclave interactions, such as dynamic agent management between enclaves (i.e., agent migration). Enclaves can also be organized hierarchically to manage multiple enclaves without limiting the scalability of the overall system. Finally, default enclaves are setup based on the hardware characteristics and the overall system configuration. For instance, service nodes could host a I/O enclave. As a result, enclaves are the basic objects used for the deployment of the composition representing a user job; agents being the low-level instantiation of the execution contexts representing the job, as well as the required system services.

Ultimately the proposed solution provides a unified execution environment which aims at: (i) ease scalability at extreme scale by incorporating the locality constraints directly into the key system services, as well as using only sparse representations of system objects and using asynchronous operations; (ii) ease fault tolerance by avoiding any central point of failure and abstracting all execution contexts and any global synchronization or notification in the

context of failures; as well as including the concept of contract (in other terms quality of service) for system services; (iii) improve the integration between the different system software tools and run-times for efficiency; (iv) improve resource management, including memory, by revisiting the design of basic system-level resource management in the context of distributed computing, including by a fine-grain management of resource affinity and locality.

Related Work

Configurable and composable system software has been studied by several research projects and at a very general level by granularity, degree of configurability and phase/time when the configuration takes place [5]. The techniques used to perform the composition vary based on these characteristics. A common technique in recent years has been to decompose the system into components with clear interfaces to enable composition. The ConfigOS [6] project investigated a framework to support OS construction via a component-based approach. A similar approach has been taken to compose HPC services at the runtime level [1, 3]. There have These library based approaches allow for dynamic configuration of required capabilities and simplify loading of alternate versions at run time, (e.g., hardware accellerated communication implementations, debug-enabled services, etc.). The ParalleX project [2] is combining lightweight execution threads with an active message based communication system for use in their runtime system. This approach to communication provides efficient use of hardware and is well suited for asynchronous communication.

Assessment

Challenges addressed: The event-based approach we advocate is very well suited to accommodate the needs to support asynchronous and adaptive messaging layers. Also, the unified execution environment aids with reliability and fault tolerance by allowing for common services to be replicated over a set of Agents (i.e., Enclave). This can then be combined with the event-based approach to allow for different levels of service to fulfill differing reliability requirements.

Maturity: The different concepts gathered by the proposed solution have been previously and individually proposed in different studies. However, this is the first time a proposed solution is combining them and applied to HPC. As a result, the proposed concepts are recognized as good candidates to address the identified challenges.

Uniqueness: To the best of our knowledge, this is the only solution that combines all the proposed concepts. Because it enables scalability as well as composition based on characteristics of the underlying hardware, we believe this is a excellent candidate for exascale. However, since the key idea is to provide a solution that is "hiding" ever changing hardware characteristics by providing more abstract system services and enabling application composition for adaptation to the execution platform, we claim that the proposed solution is suitable for any HPC platforms.

Novelty: To the best of our knowledge, this is the only proposition that supports dynamic application composition without compromising scalability and minimizing the resources used by the system. It is also the only solution that could unify all traditional system services available on HPC platforms.

Applicability: Since a unified execution environment abstracts the hardware and provide a simple but yet extensible interfaces and semantics for the implementation of new run-time systems, it is a good candidate for the implementation of a new programming language. This includes cases where the compiler may use input from the run-time system in order to perform optimizations at compile time. Furthermore, because of the event-based and distributed nature of the architecture, a unified execution environment provides all the characteristics required for the implementation of resiliency capabilities, dynamic adaptation, as well as new policies and techniques for load balancing.

Effort: Because of the concept of agent and composition, it is possible to reuse existing software (legacy libraries, run-times, and tools) or develop new solutions that will better suite the unified execution environment. In fact, only the core concepts need to be define during the first phase of the project , as well as the associated system library. Then, the team will be able to identify required system services and target applications for the development of a more extensive unified execution environment, based on the latest identified characteristics of exascale platforms. In other terms, the proposed solution is not limited by any technological choices.

References

[1] Darius Buntinas, George Bosilica, Richard L. Graham, Geoffroy Vallée, and Gregory R. Watson. A Scalable Tools Communication Infrastructure. In *Proceedings of the 22nd International High Performance Computing Symposium (HPCS'08)*. IEEE Computer Society, June 9-11, 2008. Session track: 6th Annual Symposium on OSCAR and HPC Cluster Systems (OSCAR'08).

[2] Hartmut Kaiser, Maciek Brodowicz, and Thomas Sterling. ParalleX: An advanced parallel execution model for scaling-impaired applications. In *Proceedings of the International Conference on Parallel Processing Workshops*, ICPPW '09, pages 394–401, Washington, DC, USA, 2009. IEEE Computer Society.

[3] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[4] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[5] Jean-Charles Tournier. A survey of configurable operating systems. Technical Report TR-CS-2005-43, University of New Mexico, 2005.

[6] Jean-Charles Tournier, Patrick G. Bridges, Arthur B. Maccabe, Patrick M. Widener, Zaid Abudayyeh, Ron Brightwell, Rolf Riesen, and Trammel Hudson. Towards a framework for dedicated operating systems development in high-end computing systems. *ACM SIGOPS Operating Systems Review*, 40(2):16–21, April 2006.