# EbbRT

*Jonathan Appavoo, Dan Schatzberg, James Cadden, Orran Krieger, Boston University, jappavoo@bu.edu*

We are building a multi-node library runtime, called EbbRT, to support exascale applications. It provides core system software functionality, similar to today's per-node lightweight kernels, yet a single EbbRT instance can be distributed across many heterogeneous nodes. EbbRT also exposes primitives that we find useful to build scalable software; higher-level libraries and applications can be built using these primitives. Features of the design were chosen to make the runtime: 1) customizable so that it can be extended and tuned to adapt to HPC systems and applications, 2) applicable to a broad set of applications, including emerging high performance applications, 3) scalable across many nodes and many cores of a single node, and 4) adaptive so that it can grow and shrink in reaction to changes in load and to deal with contention and failures. At the same time, EbbRT supports applications that require rich legacy OS functionality; allowing them to be incrementally modified to exploit new EbbRT specific functionality.

Three key elements of the EbbRT architecture are: 1) a distributed library OS architecture, 2) software structured using objects called *Elastic Building Blocks* (EBBs), and 3) a system-wide event driven programming model.

**Distributed Library OS:** EbbRT is a runtime for constructing efficient multi-node software. Rather than having a fixed interface, the runtime functionality is provided as a library of components. Applications link with selected components of EbbRT to provide the necessary runtime. Developers can extend, optimize, reimplement, and port components of the library. The same model is provided for what would be traditionally OS functionality (e.g., a device driver), library functionality (e.g., distributed hash table), and application functionality (e.g., a diffusion matrix).

**Elastic Building Blocks (EBBs):** EBBs are the component model adopted by EbbRT. Developers can introduce new EBBs that conform to an existing interface, or introduce whole new libraries of EBB interfaces and implementations. An EBB is internally composed of a set of distributed representatives.[1] Each representative provides an identical interface and locally services requests, possibly collaborating with one or more other representatives. The state of the object can be distributed, cached, replicated, and partitioned to different representatives. To the clients, the EBB appears and behaves like a traditional object. New representatives are created on demand as an EBB is accessed on a new core or node.

**Event Driven Programming:** EBB developers are guided to use an event-driven programming model in which they bind event occurrences to an invocation on an EBB instance. Events can be tied directly to hardware interrupts (e.g., network packet arrivals); events can also be created synthetically by software. In this way, all execution occurs in the context of handling an event rather than a long running thread. Event bindings and EBB instances can be dynamically changed in response to activity and conditions. In this manner, a developer constructs software that is reactive in nature.

The library OS model is possible in modern HPC [3, 6] systems because hardware nodes are generally allocated in their entirety to one application at a time. [2] EbbRT provides us with a much simpler model than previous distributed operating systems [4, 7, 16, 20]; since EbbRT is only required to support a single application at runtime, it need not provide traditional OS level scheduling, multiplexing, resource management, or even protection. This lack of complexity simplifies many system level optimizations, e.g., multi-core scaling, multi-node communication, I/O staging, prefetching, specialty hardware support, etc.

An EbbRT application may span a heterogeneous mix of nodes running full legacy OSes and other nodes with no underlying OS. Software running on a legacy OS can exploit the rich functionality of that OS. On a node without an underlying OS, EBBs providing highly-optimized system-level OS functionality are used. Legacy applications can be incrementally modified, as needed, to take advantage of highly-optimized nodes. Also, optimized nodes can offload operations to nodes with full legacy OSes for functionality or compatibility reasons.

One of the key advantages to a library model is that the library functionality can be customized to the application it is linked to. We want to enable a broad community of system vendors, library developers, and application developers to all be able to modify, customize and extend EbbRT to better match their needs. The software engineering advantages of EBBs (objects) are critical to us. They provide us a model for constructing well defined interfaces to enable customization; a new implementation can be introduced and exploited without change to the clients as long as it conforms to the same interface. Objects are defined per physical or logical resource (e.g., memory region, matrix, key-value store), allowing each resource instance's implementation to be customized separately. We can also *hot swap* EBBs from one implementation to another while the system is running to adjust to changing application demands.

EBBs provide developers with a programming model to scale their applications across many cores and nodes, and a model for programmers to encapsulate and reuse complex parallel optimization. The EbbRT infrastructure provides the EBB developer with information about where an object is accessed, and provides a framework for the caching, replication, and distribution that is critical to achieving good performance in parallel systems. Also, the level of indirection afforded

[1] An EBB is a light-weight construct that can be used without sacrificing performance. The cost of an EBB invocation is one extra pointer dereference over a traditional object invocation. In our current implementation on AMD64 hardware, the overhead is 6 cycles more than a C function invocation.

[2] If network isolation is not available in HW, it can be provided by virtualization.

by EBBs enables rich tooling to introspect on EBB usage patterns to identify performance problems. Implementations can then be selected or developed to improve performance.

An event driven programming model enables an application to take over handling of system events, such as power/thermal alerts, notifications of new packets arriving, disk I/O completing, fault notifications, and other communication and synchronization events. Even the lowest level exception paths of EbbRT are written as event driven EBBs; a customized application EBB can be invoked in our current prototype within 100 cycles of an interrupt being dispatched.

For core computation, an application could optionally use events much like threads. However, we expect a more natural use of events in the runtime of applications that adopt data flow and task models of processing. These more adaptive models are gaining increasing interest for exascale systems, where resource failures will be the common case rather than an exceptional condition. EbbRT is designed to be a foundation for developers of such adaptive runtimes, as well as more traditional libraries and runtimes like MPI (or even Java).

### Related work and assessment

The event driven model and Elastic Building Blocks of EbbRT borrow heavily from our earlier OS work [10, 12]; in our current work, we extend these models across multiple nodes and beyond the OS to application level code. Library OSes have been demonstrated by us and others to be practical, performant, and able to support rich runtimes [2, 9, 17].

This work will borrow heavily from the rich history of HPC system software research. It is complementary to dataflow and task programming research that is constructing dynamic runtimes in order to cope with asymmetric performance or potentially failures [8, 15]. We will exploit the extensive work to optimize Linux [13] for our full OS nodes. Conceptually, our layering of EbbRT on bare hardware is quite similar to LWK work [**?**, 11, **?**, **?**, **?**]. Further, the approach of using both legacy and bare hardware nodes is similar to a number of projects [11, **?**, **?**].

The criteria for assessing the project described in the call for proposal are:

**Challenges addressed:** The EbbRT research will directly address many parts of the resilience, parallelism (within and across node), OS structure and Legacy challenges described in the call for proposal. It also provides a framework to reason about fundamental new challenges, different from those addressed through today's legacy OSes (e.g., elasticity, scale, heterogeneity); many of these challenges become much simpler in a single-tenant, single-application library OS model. We believe that this research will not only be relevant to the broad scientific computing community, but also to the emerging high-performance commercial computing community. To enable this project to be sustainable, we are releasing it under an open source license that is compatible with proprietary vendor extension. We are also investing a large effort to develop well-defined EBB interfaces around which a larger community can innovate.

**Maturity:** While the synthesis of the library OS approach, event driven programming, and EBBs is new, each of them have been demonstrated independently. While there are significant research challenges, we are dealing with proven technologies.

**Uniqueness** : We believe that it is critical that we pursue this work with a focus on exascale systems and applications. While it is of broader utility, if the work was focused first on general purpose systems it would be unlikely to meet the extreme needs of exascale systems. Scalability is very difficult to retrofit into a system.

**Novelty** : While there is much work going on with event driven systems for cloud systems, it has made little traction for HPC requirements. The library OS model has been explored by researchers in various institutions [2, 9, 17], but our focus on multi core/node scalability, and the software engineering use of EBBs is novel. In the past, the Elastic Building Block approach has been explored in the context of single node, shared memory, and only within an OS [10, 12]. Distributed objects [**?**, **?**, **?**] of various sorts have been explored, but only for performance insensitive applications on commodity networks. Most importantly, while we are evolving each of these models in novel ways, the greatest novelty of EbbRT is in how it can combines these approaches for the use in exoscale applications.

**Applicability** : The model of EbbRT is designed to be general. We believe that commercial data-center scale systems will increasingly adopt architectures that look more and more like today's HPC systems [1, 6, 18], and that these systems will share the same dynamic, fault tolerant and scale requirements of exascale systems. Existing legacy OSes are increasingly mismatched for the requirements of the next generation of commercial applications.

**Effort:** We currently have a primitive EbbRT prototype running on Linux and MacOS and on bare-metal for PPC32/PPC64 and x86-64 machines. It will take three more person years to produce a prototype that can handle a couple of interesting application on a large-scale HPC system. Once the model is proven, ongoing refinement as we gain experience with more applications, and supporting a broader community of vendor and application developers, will require a five person team.

*This material is based upon work supported in part by the Department of Energy Office of Science under its agreement number DE-SC0005365 and upon work supported in part by National Science Foundation award #1012798.*

## References


[1] Hp project moonshot: Changing the game with extreme low-energy computing. 2012.

[2] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. M. Da Silva, O. Krieger, M. A. Auslander, D. J. Edelsohn, B. Gamsa, G. R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis. Libra : A Library Operating System for a JVM in a Virtualized Execution Environment Libra Libra Hypervisor. *System*, 2007.

[3] J. Appavoo, A. Waterland, and V. Uhlig. Project Kittyhawk: building a global-scale computer. *ACM SIGOPS Operating Systems Review*, 42(1):77, January 2008.

[4] Amnon Barak and Ami Litman. Mos: a multicomputer distributed operating system. *Softw. Pract. Exper.*, 15(8):725–737, August 1985.

[5] Adrian M. Caulfield, Joel Coburn, Todor Mollov, Arup De, Ameen Akel, Jiahua He, Arun Jagatheesan, Rajesh K. Gupta, Allan Snavely, and Steven Swanson. Understanding the impact of emerging non-volatile memories on high-performance, io-intensive computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[6] D. Chen, J. J. Parker, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, and B. Steinmacher-Burow. The IBM Blue Gene/Q Interconnection Network and Message Unit. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, page 1, 2011.

[7] D. R. Cheriton. The v kernel: A software base for distributed systems. *IEEE Softw.*, 1(2):19–42, April 1984.

[8] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 53:1–53:11, New York, NY, USA, 2009. ACM.

[9] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, 1995.

[10] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pages 87–100, Berkeley, 1999. USENIX Association.

[11] Mark Giampapa, Thomas Gooding, Todd Inglett, and Robert W. Wisniewski. Experiences with a lightweight supercomputer kernel: Lessons learned from blue gene's cnk. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[12] O. Krieger, M. Mergen, A. Waterland, V. Uhlig, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, D. Da, S. Michal, and O. Jonathan. K42: Building a Complete Operating System. *ACM SIGOPS Operating Systems Review*, 40(4):133, October 2006.

[13] Argonne National Labs. Zeptoos.

[14] Teng Ma, George Bosilca, Aurelien Bouteiller, and Jack J. Dongarra. Hierknem: An adaptive framework for kernel-assisted and topology-aware collective communications on many-core clusters.

[15] Wenjing Ma and Sriram Krishnamoorthy. Data-driven fault tolerance for work stealing computations. In *Proceedings of the 26th ACM international conference on Supercomputing*, ICS '12, pages 79–90, New York, NY, USA, 2012. ACM.

[16] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The sprite network operating system. *Computer*, 21(2):23–36, February 1988.

[17] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt. Rethinking the library os from the top down. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 291–304, New York, NY, USA, 2011. ACM.

[18] A. Rao. Seamicro technology overview. 2010.

[19] Mikiko Sato, Go Fukazawa, Kiyohiko Nagamine, Ryuichi Sakamoto, Mitaro Namiki, Kazumi Yoshinaga, Yuichi Tsujita, Atsushi Hori, and Yutaka Ishikawa. A design of hybrid operating system for a parallel computer with multi-core and many-core processors. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, pages 9:1–9:8, New York, NY, USA, 2012. ACM.

[20] A. S. Tanenbaum and S. J. Mullender. An overview of the amoeba distributed operating system. *SIGOPS Oper. Syst. Rev.*, 15(3):51–64, July 1981.