|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |
| Flux Paper: notes  Improving flow, statement of contributions, ideas on the sections on Flux overview and description of the RT: LWJ, COMB, scalable KVS |
| Becky  January 18,2014 |

# Flux Paper: notes

## Improving flow, statement of contributions, ideas on the sections on Flux overview and description of the RT: LWJ, COMB, scalable KVS

**Abstract -- to be revised at the end**

Resource management (RM) software is crucial to HPC for efficient application execution on a set of diverse, large resources. However, growing numbers and types of compute resources and a greater interplay among various resources (e.g., between compute clusters and a shared I/O cluster) across the entire HPC center can render even the best- in-breed RM software increasingly ineffective. HPC centers are at a junction where they require a major shift in the conceptual model of resource management in order to meet the challenges of extreme scalability, a diversity of resources, and challenges such as power budgets. Our response to this critical need is FLUX, a RM software framework that can solve the key RM challenges in a simple, extensible, distributed, and autonomous fashion. It aims to manage the entire computing facility as one common pool of diverse sets of resources to provide efficient scheduling decisions and easy accommodation of site-wide constraints. Further, FLUX employs a framework approach to facilitate the seamless integration of system monitoring and administration, lightweight virtualization, and parallel programming and tools run-time systems. We discuss Flux’s vision, design challenges, and concepts, and then report our progress on building key components. Our preliminary results show that the run-time provides requisite properties such as high scalability and easy integration and interoperability among essential run-time elements such as MPI, run-time tools and middleware.

Keywords: resource management, communication framework, run-time, key value store, scalable process management services

**Introduction**

Resource management (RM) software is critical for high performance computing (HPC). It is the centerpiece that allows efficient execution of HPC applications while providing the computing facility with the main means to maximize the utilization of its diverse array of computing resources. However, several growing trends make even best-in-breed RM software systems increasingly ineffective in dealing with the diversity and size of resources fielded for HPC centers. As numbers and types of compute cores continue to grow, the key RM challenges traditionally associated only with leadership-class machines are now relevant to all computing resources, including commodity Linux clusters. An effective HPC RM must increase its purview to manage resources across the entire computing facility and to enable extreme scalability, low noise, fault tolerance, and heterogeneity management delivered within increasingly strict power bounds.

In fact, a greater interplay among various classes of clusters across the entire computing facility already makes the current paradigm of single-cluster scheduling suboptimal. An application running on a compute cluster heavily uti- lizes site-wide shared resources such as I/O and visualization clusters. Thus, avoiding any significant site-wide bottleneck requires the RM to schedule the job to all dependent resources together.

Meanwhile, greater difficulties in code development on larger systems have begun to impose far more complex requirements on the RM. For example, with- out adequate RM support, debugging, tuning, testing and verification of the applications have become too difficult and time-consuming for end-users. The next-generation code development environments require the RM to provide ef- fective mechanisms to support the reproducible results of program execution, to provide accurate correlations between user-level errors and system-level events, and to integrate and accelerate a rich set of scalable tools.

In short, an effective RM that effectively addresses all of these challengescan net HPC centers and their users significant productivity gains. Our response to this critical need is Flux, a RM software framework that addresses the key emerging challenges in a simple, extensible, distributed and autonomous fashion. It aims at managing the whole computing facility as one common pool of diverse resources. Hence, scheduling decisions will be far more efficient as well as extendible to accommodate emerging constraints such as power capping. Further, the FLUX design will integrate system monitoring and administration, lightweight virtualization, and distributed tool communication capabilities that are currently provided by disjoint and often overlapping software. Integration of these facilities within the common framework designed from the ground up for scalability, security, and fault tolerance will result in a more efficient and capable system.

In this paper we introduce our concept of the Flux next-generation resource management framework and present two core elements that we have prototyped: a communication framework including the Comms Message Broker (CMB), and workload run-time tools for efficient execution of transactions within a job. In the latter category we have developed a distributed Key Value Store (KVS) and scalable process management services, as well as other components and a model using the concept of lightweight jobs (LWJ). Here we present a detailed description of our CMB and KVS prototypes as well as some preliminary results and a performance model that demonstrates scaling properties of these core Flux components. The results have supported and will continue to guide our design choices as we build additional parts of our RM infrastructure. Flux will enable developers at the operating system and run-time levels to leverage the RM data stores and services in unprecedented ways, and it will position HPC centers to cope with diverse, extreme-scale resources.

2. A New Conceptual Model for HPC Resource Management

The vision of FLUX is to create a scalable RM software system that significantly improves operational efficiency and user productivity for workloads that span the entire computing facility. Our challenges include having to provide extreme scalability, low noise, fault tolerance, and heterogeneity management while under a strict power budget. Worse, the workloads themselves are also becoming increasingly diverse, dynamic, and large. Thus, fully realizing our vision through these challenges requires a major shift in our conceptual model of how the RM must manage, model, schedule, and allocate its resources.

2.1 Design Challenges

The first requirement in designing Flux on this new conceptual model is to take a center-wide, or global, resource view. The RM must be capable of imposing highly complex resource bounds to guarantee the highest operational efficiency at any level across the computing facility, while at the same time enabling the most efficient execution and scheduling of the workloads within these bounds. This requires the RM to have purview over the entire computing facility. The RM must manage the resources at the center as one common pool of resources, and the ability to see a broader spectrum of resources and their various constraints can then lead to more efficient scheduling strategies and execution environments.

We characterize this design challenge as the *multidimensional scale challenge*. Simply stated, the challenges include supporting extreme scalability, addressing noise as concurrency increases, and managing a drastically increased amount of run-time information that must be monitored, traced, and stored. Our requirement is that our RM shall handle increased scale in numbers of resources as well as jobs and other dimensions of RM data.

A second requirement for Flux is to include a rich resource model. This includes a generalized model of diverse types of resources, such as file systems, visualization systems, and serial batch systems, in addition to traditional clusters. With a richer resource model, the RM will be capable of imposing complex resource bounds, as opposed to the traditional view of CPUs, memory, and time limits. With a Flux RM that has the ability to model general resource relationships, as opposed to flat lists of nodes, we will be able to allocate computing resources tailored to the disparate limiting factors of our applications. For example, an application may be compute bound while others are I/O bound or power bound. This approach will enable stronger efforts to diagnose errors for both end users and support staff by associating jobs with other facility-wide events.

We chacterize this design challenge as the *diverse workload challenge*. As one specific example of emerging resource types, power is becoming a critical factor. When the computing facility becomes power bound instead of compute-node bound, the RM design must enable the scheduling of workloads based upon the maximum power limit at any level of the facility. Thus the resource representation must be generalized enough to model consumable resources like power, as well as the diversity of hardware.

A third requirement is that resource allocations must also be elastic. An application may have different phases with disparate performance-limiting factors; it must be able to grow and shrink its resource allocation dynamically. This is in contrast to traditional methods with static time-limit-bounded allocations.

We characterize this design challenge as the *dynamic workload challenge*. differentdifferentat different

Finally, the new conceptual model must meet the greater difficulties in code development by facilitating the integration of other key relevant software that can ease the difficulties. These software components should include system monitoring and administration, lightweight virtualization, and scalable tool communication. The integration will facilitate a higher level of leverage among these essential computing elements, and this will lead to significantly higher productivity for both end users and system administrators. These capabilities are currently provided through disjoint and often overlapping software, the integration of which can provide a richer, stronger environment and may reduce development costs through the ability to leverage the Flux framework.

Another challenge that we considered during the design included the risk of higher downtime costs in a more global model. If the RM is not designed adequately, a downtime could negatively impact the availability of a large portion of the HPC center’s resources. Thus our RM must be tolerant of hardware and software faults and failures with no single point of failure and must also support live software upgrades. This and other challenges, including security, integration risk, and backwards compatibility will be

In summary, the global resource view, rich resource model, elasticity, and seamless integration of other software represent the fundamental characteristics of the new resource management paradigm.

2.2 Conceptual Software Design

We describe some of the primary elements of this new conceptual model while addressing the multitude of design challenges described above. These models form the basis for the software design of Flux.

Unified Job Model: Traditionally, a job is simply defined to be a resource allocation, a concept too weak to support the new paradigm. Rather, we unify the traditional job notion with the notion of a resource manager instance—an independent set of resource manager services. The RM instance must be delegated the main responsibility of managing the resources allocated to the job. Then, the unified job model becomes the foundation on which to build a hierarchical, resource-management scheme to address the multidimensional scale challenge. In addition, an RM instance can implement compatibility mode with a particular traditional paradigm only over its own allocation, providing a straightforward path to address the backward compatibility challenge.

Job Hierarchy Model: To scale the new conceptual model in the scaling limit of the en- tire computing facility, we must avoid a centralized approach: the new paradigm requires a hierarchical management scheme with a well-balanced, multilevel delegation structure. For this purpose, we use a tree-based job hierarchy model that has many proven advantages for extreme scalability. In this model, a job is only required to manage its children jobs, which would be only a small fraction of the total number of jobs that are run across the entire computing facility. Further, several guiding principles throughout the job hierarchy strike a balance between the management responsibility of a parent job and delegation and empowerment of a child job:

1. Parent bounding rule: the parent job grants and confines the resource allo- cation of all of its children.
2. Child empowerment rule: within the bound set by the parent, the child job is delegated the ownership of the allocation and becomes solely responsible for most ecient uses of the resources.
3. Parental consent rule: the child job must ask its parent job when it wants to grow or shrink the resource allocation, and it is up to the parent to grant the request.

In general, these rules enforce the first principle of the new model: imposing highly complex resource bounds to guarantee the highest operational efficiency at any level across the computing facility, while enabling most ecient execution and scheduling of the workloads within these bounds. At the same time, this model is the most fundamental design concept, which forms the basis to address many of the design challenges including the *multidimensional scale*, *rich resource model* and *dynamic workload* challenges previously described.

Generalized Resource Model: In the traditional paradigm, compute resources are modeled primarily as a collection of compute nodes, a simplistic perspective ill- suited for the new paradigm. Today’s applications are diverse with disparate limiting performance factors beyond floating point computation.

Further, computing centers are increasingly concerned about managing new resource types such as power and shared persistent storage. The generalized resource model is our concept to represent various resource types and their re- lationships that can impact how well applications perform and the computing facility operates. Our generalized resource model also includes a unified resource specification and description language. Speaking the same resource description language for request specification provides transparency and fine-grained expressibility. Our generalized resource model addresses the *diverse workload* challenge.

More specifically, the unified language approach allows users to express their resource requests more flexibly, e.g., using ranges or boolean expressions instead of hard amounts to allow requests to be fulfilled from several equivalent resource types. This makes the scheduling granularity of jobs finer and more malleable.

Resource Allocation Elasticity Model: As our applications and their programming models are becoming increasingly dynamic, the new paradigm must support an elasticity model where an existing resource allocation can grow and shrink, depending on the current needs of applications and/or the computing facility.

We support the elasticity model within our job hierarchy framework above: a child job sends a grow or shrink request to its parent, which can go up the job hierarchy until all requisite constraints are known for this request. Also, combining this with the generalized resource model, the elasticity can be expressed for any resource including power consumption. Our elasticity model addresses the *dynamic workload* challenge.

Common Scalable Persistent Communication Infrastructure Model: Our scalability strategy with respect to a large number of compute nodes is to provide a common scalable communication framework within each job. When a job is created, a secure, scalable overlay network with common communication service is established across its allocated nodes. Except for the root-level job, the existing communication session of the parent job assists the child job with rapid creation of its own session.

A communication session is only aware of its parent and child and passes the limited set of control information through this communication channel. Thus, this model enables highly scalable communication within a job, while limiting communications between jobs, addressing the *multidimensional scale* challenge as well as security issues.

Further, this backbone per-job communication network supports many well- known bootstrap interfaces for distributed programs including many MPI implementations as well as run-time tools. This provides tightly integrated support for the development and use of scalable code development run-time tools and research, which can have a large impact on user productivity.

Lightweight Virtualization Model: The lightweight virtualization model is our response to challenges posed by higher downtime costs, separation-of-concerns and security. Full-fledged virtualization techniques like Xen and Kernel-based Virtual Machine (KVM) have many advantages for these design challenges, but that approach has proven to be ineffective for HPC due in large part its high overhead [1]. Instead, our virtualization strategy exploits OS-enforced resource management and isolation mechanisms to launch applications in containers with virtually no impact on performance [2]. Within a container, private file system namespaces allow the system and applications to have divergent file system views, and to access file systems with different constraints.

Jim and Mark: Do we need this explained here? Is this the right place and explanation?

4 FLUX

* 4.1  Approach
* 4.2   Communication Message Broker
* 4.5  Scalable Key-Value Store

5 Preliminary Results

* 1. 5.1  Performance Model
  2. 5.2  Experimental Validation
  3. 5.3  Case study: Easy Integration of Tools and Middleware
  4. 6  Related work
  5. 7  Discussions
  6. 8  Concluding remarks
* Acknowledgments
* The authors acknowledge the contributions of Matthieu, etc. of CEA and Jon Bringhurst of LANL, who all provided substantial and insightful feedback on the Flux vision document and in subsequent discussions. Or something like that.
* This article has been authored by Lawrence Livermore Na- tional Security, LLC under Contract No. DE-AC52-07NA27344 with the U.S. De- partment of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide li- cense to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. (LLNL-CONF-xxxxxx).

References

* 1. Regola, N., Ducom, J.C.: Recommendations for virtualization technologies in high performance computing. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science. CLOUDCOM ’10, Wash- ington, DC, USA, IEEE Computer Society (2010) 409–416
  2. Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., Rose, C.A.F.D.: Performance evaluation of container-based virtualization for high performance com- puting environments. 21st Euromicro International Conference on Parallel, Dis- tributed, and Network-Based Processing (PDP) (2013)