A Strawman for an HPC PowerStack

Table of Contents

Organizers and Early Contributors (in alphabetical order):

- Christopher Cantalupo, Intel Corporation, USA
- Jonathan Eastep, Intel Corporation, USA
- Siddhartha Jana, Energy Efficient HPC Working Group, Global
- Masaaki Kondo, University of Tokyo, Japan
- Matthias Maiterth, Ludwig-Maximilians University, Germany
- Aniruddha Marathe, Lawrence Livermore National Laboratory, USA
- Tapasya Patki, Lawrence Livermore National Laboratory, USA
- Barry Rountree, Lawrence Livermore National Laboratory, USA
- Ryuichi Sakamoto, University of Tokyo, Japan
- Martin Schulz, Technical University Munich, Germany
- Carsten Trinitis, Technical University Munich, Germany

Note:
Our intent in publishing this document is to propose a strawman architecture for an HPC PowerStack as a starting point for discussions. We expect significant changes of its design and even approach over time, and we expect the design and refinements to be a collaborative effort across a broad community reaching well-beyond the organizers and early contributors.

Part of this work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-TR-756268.

Motivation for Designing an HPC PowerStack

The landscape of High-Performance Computing (HPC) is changing as we enter the exascale era and power and energy management are key design points for any next generation of supercomputers. Efficiently utilizing procured power and optimizing performance of scientific applications under power and energy constraints is challenging due to several reasons including dynamic phase behavior, processor manufacturing variability and increasing heterogeneity of node-level components. Extending the scope from the node- and application-level up to the system-level introduces further challenges on top of power-unaware job scheduling, which is known to be NP-hard on its own.

While there exists several individual efforts across the community to research automatic techniques for managing power and energy better, the majority of these techniques have been specialized to meet the needs of a specific HPC center or specific optimization goals and provide little support to connect them to each other. Some projects, most notably the PowerAPI efforts [1], discuss interfaces that form a good starting point for full stack integration, but these interfaces still need to be hooked up to the wide range of software components offered by academic partners, developers or vendors. Furthermore, a recent survey conducted by the EE HPC WG concluded that the majority of such techniques have lacked the application-awareness required to achieve the best system performance and throughput. Other observations were that each technique tended to target management of power and energy for a different subset of the site or system hardware and that the different techniques tended to perform management at different (and often conflicting) granularities. Unfortunately, the existing techniques have not been designed to coexist simultaneously on one site and cooperate on management in an integrated manner [2]. The lack of application-awareness, lack of coordinated management across different granularities, and the lack of widely accepted interfaces and consequent limited connectivity between modules, can result in substantially underutilized Watts and FLOPS.

To address these gaps, the HPC community needs a *holistic* stack for power and energy management and none currently exists. In our view, a holistic stack is one that is extensible enough to support the present and anticipated future needs of various different HPC centers, one that achieves best system performance and throughput through application-awareness, one that is designed to coordinate management at different granularities, and one that enables the seamless integration of software components from different developers and vendors.

In this seminar, our goal is bring together experts from academia, research laboratories, and industry in order to take stock of existing approaches, design points and power management concepts and requirements at the various participant's institutions, and to design concepts for a holistic power and energy management stack, which we refer to as the HPC PowerStack. Further, we hope to take steps toward defining the interfaces necessary for providing a complete prototype shared among many groups. The intention is to align development and research efforts across the community so that we may share development resources, avoid duplicating effort, agree on common interfaces and reap the rewards together as a community.

The intended final outcome of this collaboration will be a holistic, flexible and extensible concept of a software stack ecosystem that allows us to combine product-grade open-source software components to enable runtime optimization of system power, energy, and performance.

A Strawman Design

We envision a holistic power and energy management stack concept that is extensible, application-aware, has well defined interfaces and is capable of coordinating power and energy optimizations across many granularities of site or system hardware.

Key Design Principle: A Hierarchical Approach

Based on the state-of-the-art of the components available in the community for power and energy management, we propose a hierarchical HPC PowerStack concept to manage power and energy at three levels of granularity: the system level, the job level, and the node level. This implies the need for at least a power-aware system resource manager / scheduler, a power-aware job manager, and a power-aware node manager, but the discussions at the seminar may identify additional important actors that need to be integrated in the hierarchy.

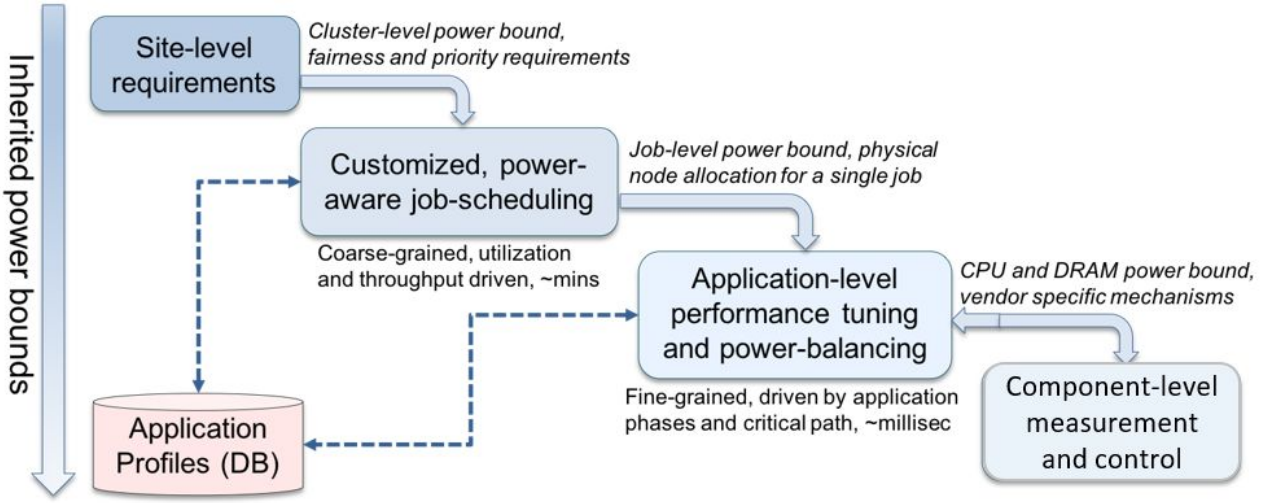

Figure 1: Conceptual diagram of the three levels of hierarchy needed for power and energy management (system, job and node level) driven by site-level requirements.

As shown in Figure 1, at each level in the hierarchy, a HPC PowerStack should provide options for adaptive management depending on requirements of the supercomputing site under consideration. Site-specific requirements such as system-level power bounds, user fairness, or job priorities will be translated as inputs to the job scheduler. The system-level job scheduler will choose power-aware scheduling plugins to ensure site-level compliance, with the primary responsibility of allocating nodes and job-level power constraints across multiple users and diverse workloads. Such allocations of physical nodes and job-level power bounds will serve as inputs to a fine-grained, job-level runtime system whose responsibility is to coordinate optimizations of compute node hardware control settings across the compute nodes in a job. The job-level runtime, in turn, relies on node-level measurement and control mechanisms. Continuous monitoring and analysis of application behavior may be required for decision-making.

Desired Features for an HPC PowerStack

Based on the high-level goals laid out above, we formulate the following (unsorted) general requirements or desired features, which he hope serve as a starting point for refinement at the seminar:

- Holistically coordinated power optimizations across the whole system in a scalable manner
- Capabilities for ensuring safe operation within electrical operating parameters, including protective layers to enable scenarios where system power caps are hard limits that must be enforced at all times
- Integration into the security concepts of existing and future HPC solutions, which is likely to be site specific
- Open source implementations with a flexible (not-sticky) software license to enable commercial as well as research uses
- Cross-platform and vendor neutral support to avoid locking users to a specific vendor
- Production-grade quality and easy deployment through standard package management interfaces, for both user and system level components
- Extensibility (e.g., through plugins or exchangeable components) to support diverse preferences at different HPC centers and facilitate rapid prototyping of new power, energy or performance optimization techniques
- Integration of software components from multiple vendors and developers using a set of well-defined and possibly standardized interfaces
- Real-time monitoring and control in order to adapt to dynamic scenarios. These can be *system-level* (e.g.: adapting the power cap at the system level due to power availability or node failures), or *application-level* (e.g.: adapting to critical path, CPU/memory boundedness, manufacturing variation in the allocation, load imbalance, etc.)

Proposal to Limit the Initial Scope

For the first design revision of HPC PowerStack strawman and our first seminar, we are deliberately limiting the scope of the HPC PowerStack design and discussion and then expanding the discussions as needed from this initial set. In particular, for now we expect to exclude advanced scheduling, management of power in non-compute-node hardware or infrastructure, and energy efficiency metrics, although this could and should be discussed and adjusted during the seminar. Examples of advanced scheduling parameters include job priorities, user fairness, accounting/charging, cross system scheduling and co-scheduling jobs on nodes of a system. Examples of non-compute hardware, which we consider to keep out of scope for now, include the network fabric, IO nodes, and cooling infrastructure.

In order to have a concrete starting point for the discussion rather than a blue-sky concept, we propose to define the Roles and Responsibilities (R&Rs) for each actor as well as actor-actor interactions pragmatically, basing the R&Rs and interactions on experience with existenting open source software projects. This will provide the necessary starting point for the definition of the conceptual glue between actors, along with agreements on common interfaces between them, to ensure they coordinate management across granularities and interoperate properly. For any actors where suitable open source code is not available, we must undertake development of that actor ourselves as part of our envisioned HPC PowerStack collaboration, but for the remaining actors we will leverage existent product-grade open source code from the participants. The HPC PowerStack must leverage open source software to the extent possible to limit development effort and avoid duplication of past efforts.

Node Management Modes

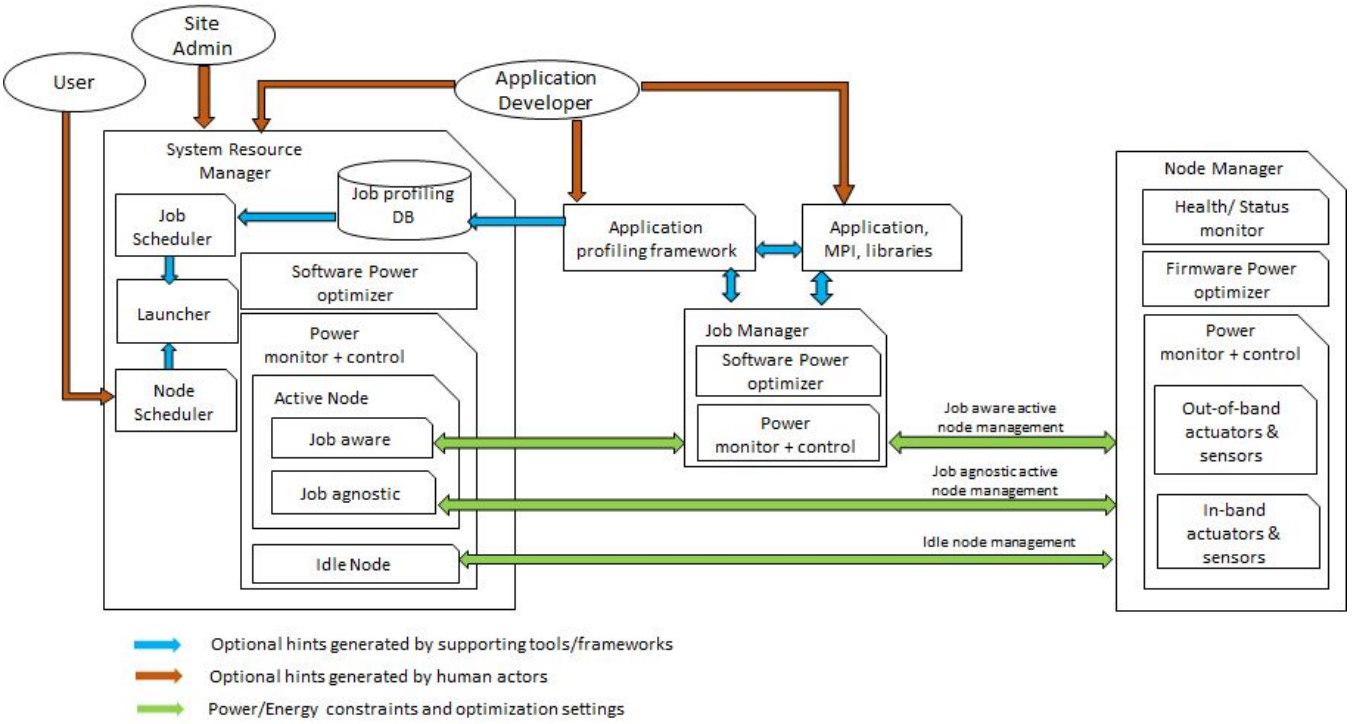

Figure 2: Interactions between the three layers of hierarchy.

We currently envision that the HPC PowerStack needs to support three compute node management modes. Each compute node must be managed according to a single mode at a given time, but all three management modes may be employed across the nodes of the system simultaneously. The modes involve similar but slightly different interactions and communications between actors (as illustrated in Figure 2). The three modes are as follows:

1. ***Job-aware active node management:*** in this mode, the nodes allocated for executing a job are managed as a unit by the job manager. The job manager actively optimizes job power, energy, or performance by coordinating optimizations of hardware controls in the compute nodes across all compute nodes in the job. In this mode, the system resource manager is responsible for conveying power or energy constraints for the job and the choice of optimization strategy to the job manager. The job manager employs the optimization strategy instructed by the system resource manager while enforcing the given constraints. Some or all of the supported optimization strategies leverage application-awareness (and optional application profiling data) to obtain the best power, energy, or performance improvements.

2. ***Job-agnostic active node management:*** in this mode, the nodes allocated for executing a job are managed as a unit. However, the job manager actor is not present or is inactive. Instead, the system resource manager is responsible for optimizing job power, energy, or performance. To do so, it talks directly to the node manager on each compute node to configure that node's hardware controls such that job power or energy constraints are enforced. In this mode, optimizations for power, energy, and performance are neither performed on each node nor coordinated across nodes. Since this approach is not job-aware, it leads to missed opportunity for optimizing power, energy, or performance but does enforce constraints on power or energy.

3. ***Idle node management:*** in this mode, nodes are managed as individual units. This mode is used on nodes that are not actively assigned any jobs (i.e., idle nodes). In this mode, the system resource manager talks directly to the node manager on each compute node to configure that node's hardware controls and enforce power or energy constraints on that node. These constraints are decided based on system constraints and power or energy resources allocated to other running jobs.

Communication Model Between Resource and Job Manager

The concepts of HPC PowerStack are intentionally designed for both present-day centralized resource manager / scheduler designs and future decentralized designs. The model for communication between the resource manager / scheduler and the job manager is different in each case. What the two cases have in common is that the job scheduler propagates power or energy constraints and optimization settings to the job power manager through scheduler daemons running on the compute nodes. Similarly, the job power manager propagates power consumption measurements and other feedback through the scheduler daemons to the rest of the scheduler.

In the centralized case (Figure 3), the scheduler propagates power or energy constraints and optimization settings to the job manager through the scheduler daemon on one compute node that is deemed the master. Similarly, the job manager provides its feedback to the scheduler through the scheduler daemon on the master compute node. The job manager is a scalable distributed runtime that is expected to have agents running on all job compute nodes. The job manager is responsible for ensuring that inputs from the scheduler daemon on the master compute note are properly propagated to its agents on all other compute nodes. Similarly, the scheduler daemon on the master compute node is responsible for ensuring that feedback from the job manager is propagated to the rest of the scheduler daemons and scheduler as needed.

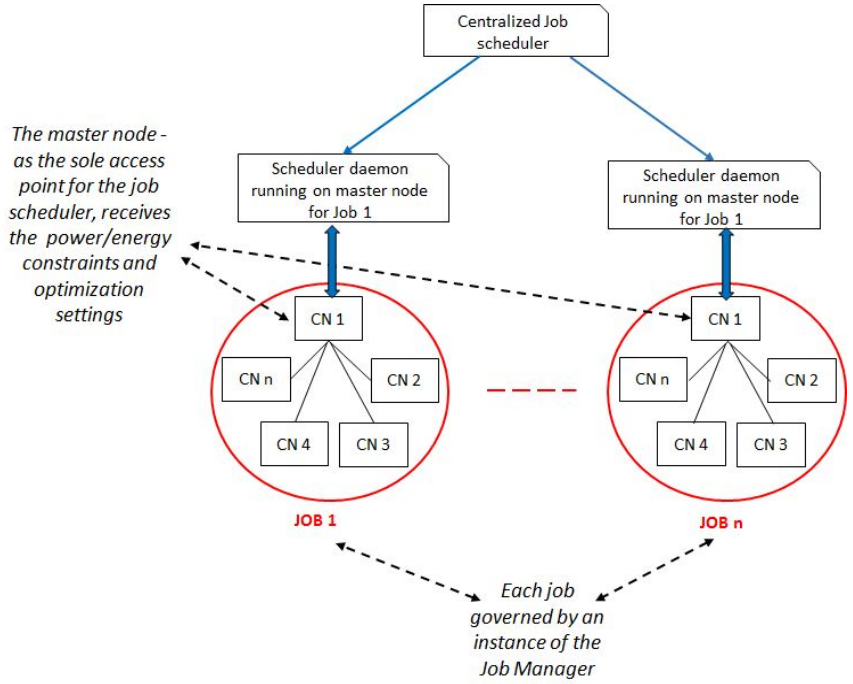

Figure 3: Centralized Case: Master Node Representing all the Compute Nodes Within a Job

In the decentralized case (Figure 4), the scheduler daemons can propagate power or energy constraints and optimization settings to the job manager on any or all compute nodes. Similarly, the job manager can propagate its feedback to the scheduler to the scheduler daemon on any or all compute nodes. The job manager is responsible for ensuring that inputs from the scheduler daemons are properly propagated to its agents on all compute nodes, and it is also responsible for resolving conflicting inputs. Similarly, the scheduler daemons are responsible for ensuring that feedback from the job manager is propagated to the rest of the scheduler daemons and scheduler as needed.

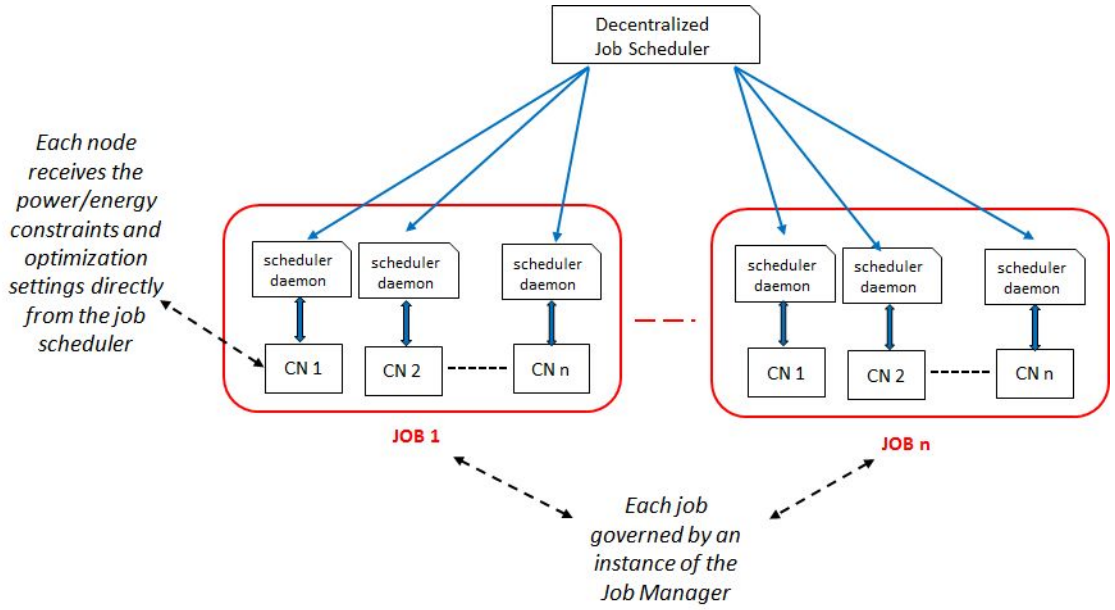

Figure 4: Decentralized Case: No Specific Master Node - All Nodes Receive Job Constraints and Optimization Settings

Open Questions

The following is a list of open questions to help drive the discussions at the seminar. The list is not intended to be sorted or exhaustive. Further open questions are to be identified and discussed at the HPC PowerStack seminar.

- Are the three layers of hierarchy (system, job, node) sufficient or is a further refinement needed? Are the communication and node management modes adequate?
- Which pieces of the software stack should be pluggable and/or interchangeable between developers, users, partners, vendors, … and hence need a set of rigid APIs and API approaches, for interoperability between components?
- What elements of the HPC PowerStack need to run in protected mode and should they interact with the user and/or the system administrators?
- Are there any actors beyond job manager and node manager that should have direct access to HW controls and monitors?
    - Monitoring tools? Applications?
    - Read only access? Or read/write?
    - In what setting does it make sense to allow it? Research settings? Production settings?
    - Would allowing write access introduce open-loop control challenges? If applications or tools could modify controls and not just the job manager or resource manager, how can a job manager or resource manager ensure power or energy constraints are enforced?

- How to handle software or hardware failures or unexpected job terminations?
    - What happens if user software crashes, what if the HPC PowerStack crashes?
    - What protective mechanisms are needed to ensure operation within power or energy constraints in spite of failures or job terminations? Mechanisms in the resource manager? Mechanisms in the hardware (e.g., in the BMC)?
    - How to detect failure of job manager? Heartbeat interfaces to protective mechanisms?
    - How to restore node hardware controls to sane states in the event of a crash?
    - How to protect the system from malicious power attacks? (Intended / Unintended)
- What role should Linux power governors / OS components play in the HPC PowerStack?
    - Can coordinated optimization across nodes be accomplished with the governor model? Too node-local?
    - Would including governors introduce open-loop control challenges? If both the governors and job manager or resource manager could modify controls, how can a job manager or resource manager ensure job or system power or energy constraints are enforced?
    - What interfaces would be needed to coordinate control between governors and job manager or resource manager? Heartbeat interface? Use existent interfaces to configure the governor to perform no control?
- Are the two communication models (fully central and fully decentralized) sufficient or what other consideration for resource/job management interactions should be taken into account?

Conclusions and Seminar Goals

This document proposes a strawman architecture for the HPC PowerStack as a starting point for discussions at the upcoming HPC PowerStack seminar. We expect significant changes of the design over time, and we expect the design and refinements to be a collaborative effort across a broad community reaching well-beyond the organizers and early contributors.

At the seminar we hope to have achieved the following goals:

- Starting with this strawman, a consensus of a mutually agreed upon component and interface model for an HPC PowerStack matching the experiences and requirements from the attendees.
- Mapping of existing environments and constraints onto the strawman model to ensure its validity across multiple real-world scenarios.
- A gap analysis of missing components and interfaces compared to currently available hardware and software solutions, and plans for covering these gaps
- Roadmap from PowerStack-conforming prototypes towards production-ready PowerStack implementations
- A continued collaboration between the attendees (and possible further community partners) to continue driving the design and realization of an HPC PowerStack for present systems as well as exascale systems and beyond.

We are excited about the seminar and the opportunity to work with you towards these goals and towards a homogenization or, where useful and applicable, standardization of an HPC PowerStack.

References

[1] *“High Performance Computing - Power Application Programming Interface Specification”,* https://github.com/pwrapi/powerapi_spec
[2] *“First Global Survey of Energy and Power Aware Job Scheduling and Resource Management”,* Energy Efficient HPC Working Group, insideHPC, https://insidehpc.com/2017/12/first-global-survey-energy-power-aware-job-scheduling-resource-management/

Appendix: Participating Actors and Their Roles

Survey of actors and the various levels in the HPC software stack with community examples for each. The list of examples is certainly not complete, so please add other projects as needed.

Actors	Roles and Responsibilities	Interoperability with other actors	Examples within the community (edit as needed)
System Resource Manager + Job Scheduler	Apart from the traditional role of monitoring resource usage and allocating resources to jobs, this actor is also responsible for one/more of the following power-aware tasks: 1. Assign power budgets for idle nodes (and potentially other resources) 2. Assign a job power budget for the active nodes running a job 3. Monitor and control power budgets for each job and/or user 4. Perform analysis to determine an efficient energy/power budget for a given job and/or node. 5. Account for facility-level and cooling constraints that are external to the system. 6. Record energy / power based telemetry data from each node (and potentially other resources)	This actor interacts with the Job Manager to convey settings such as the job's power budget, desired operating frequency, specific job-aware power management algorithms, etc. These settings can be configured either by the end user during job launch or by the system admin. This actor may also have the capability of receiving feedback from job profiling tools and databases.	Slurm, ALPS, PBSPro, Cobalt
Power-aware Job manager	In the job-aware active node management mode of PowerStack, this actor owns management of the active nodes running a job. Management of idle nodes is outside its scope. This actor performs the following tasks: 1. Manages the control knobs in all compute nodes of the job and optimizes them at runtime to achieve the desired power consumption, efficiency, or other settings 2. Scalably aggregates application profile/telemetry data from each node servicing the given job. This actor should run with different Linux group permissions from the application: that group should have sufficient privileges to access node power and performance monitors and controls.	This actor interacts with the following: 1. System Job Scheduler: The job manager provides aggregated data to the job scheduler during the execution of the application. In turn from the scheduler, it receives configuration settings like job budgets and selection of desired optimization algorithms 2. Application profiling framework: The job manager receives the per-node application profiling data from this framework and then proceeds to aggregate this data to create a summary of the dynamic application behavior 3. Application layer: The job manager may receive additional optional hints from the actual user application 4. Node manager: The job manager interacts with the node manager to drive per-node monitoring and control of hardware knobs.	GEOPM, Conductor

Power-aware Node manager	This actor is responsible for: 1. Providing telemetry data to system monitoring tools or visualization tools external to the PowerStack. 2. Providing access to node-level hardware controls and monitors. It should provide this access to userspace for a subset of controls and monitors specified in a whitelist. Whitelists should contain the controls and monitors that are designated “safe” to access by a privileged Linux group. Alternatively, OS kernel modules may be designed and loaded to enable access to specific hardware knobs from the userspace.	This actor interacts with: 1. External system monitoring tools: the node manager interacts with system monitoring tools to provide node-level power, energy, temperature, and other telemetry. 2. System resource manager: has direct access to the hardware control knobs via the node manager. This access is only used in cases where the nodes are either idle or the system is not equipped with a job manager 3. Job manager: interacts with the node manager to monitor job nodes and control job node HW knobs to enforce job-wide settings like power consumption limits, efficiency targets, etc.	msr-safe, PAPI, PowerAPI, libvariorum, NVML, Redfish, HDEEM
Applications	This actor corresponds to the application layer that runs in userspace. This includes all linked libraries including distributed memory communication libraries.	This actor interacts with the following: 1. Job manager: applications may provide optional hints to the job manager to demarcate specific regions of code (aka phases) as optimization targets for the job manager, or the applications or libraries may coordinate with the job manager to optimize tunable parameters within their code 2. Application profiling framework: if running, this framework may optionally collect profiling information from the application	HPC apps, MPI, SHMEM
Application profiling framework	This actor has the role of collecting application profiling data at runtime on each node of the job.	This actor interacts with the following: 1. Job manager: The per-node profiling data may be fed to the job manager 2. System resource manager: In the absence of the job manager, the per-node profiling data may also be provided as feedback to the system-wide resource manager/job scheduler.	Caliper, Score-P, PowerAPI
Site admin	The site admin has the role of configuring system power/energy policies and setting constraints.	This actor interacts with: 1. Electricity service provider: through a demand/response interface with provider, site admin configures power caps for systems at the site. 2. Job manager: either be a one-time interaction during the setup of the job manager or an infrequent interaction after setup, where the site admin selects the desired power optimization algorithm 3. Node manager: This is also a one-time or infrequent interaction where the site-admin configures the whitelist to adjust what node hardware controls and monitors the job and the system power manager will have access to.	Human interface

Application developer	This actor has the role of developing, debugging, tuning, instrumenting applications, and modifying applications to enable tuning of parameters	This actor interacts with: 1. Application profiling framework: the application developer instruments their code with profiling APIs provided by the application profiling framework. 2. Job manager: the application developer may optionally modify application or library code in order to expose and support tunable parameters and extend the job manager's optimization algorithms to coordinate optimization of those tunable parameters	Human interface
User	This actor has the role of requesting resources to execute a job	The actor interacts with: 1. System resource manager / scheduler: user makes requests for resources to execute the job through job queue tools	Human interface