An Infrastructure for Dynamic Resource Arbitration on Heterogeneous Nodes

Swann Perarnau

Argonne National Laboratory

Introduction

U.S. D.O.E Exascale Computing Project

The DOE-led Exascale Computing Initiative (ECI), a partnership between two DOE organizations, the Office of Science (SC) and the National Nuclear Security Administration (NNSA), was formed in 2016 to accelerate research, development, acquisition, and deployment projects to deliver exascale computing capability to the DOE laboratories by the early to mid-2020s.

ECP, a 7 year project, is focused on delivering specific applications, software products, and outcomes on DOE computing facilities.

See: **https://www.exascaleproject.org**

Hardware Context: Energy, Performance Variability

Power/Energy efficiency matters:

- Production systems are starting to draw a significant amount of power
- One of the reason so many systems turn to accelerators
- Operational issues can arise from significant power draw

Performance variability creates opportunities

- Components vary on perf/watt: rise in SKU variants (binning)
- Application are not always perfectly balanced

-> move power around, avoid idle time

Software Context: Applications, Workflows

Exascale applications are significantly more complex:

- In-situ analysis
- Checkpointing
- Load balancing between node-level components

Rise of new scientific workloads:

- Scientific machine learning
- Workflows
- Human-in-the-loop
- Facility integration (instrument to compute)

Where Can Systems Research Help ?

Can we design an infrastructure to optimize/arbitrate the resource usage of scientific workflows, more comprehensive and easier to deploy ?

-> Application-level awareness: better understanding of inner behavior of applications, runtimes.

-> Node-level management: address topology-wide issues

-> Reconfigurable: adaptivity to users, platforms, new usages

Argo, an ECP project

Argo is a collection of interoperating low-level software components that provide application-directed management of resources such as power and memory.

Four major software products:

- AML
  - Memory library targeting deep DRAM hierarchy
  - Application-facing abstractions enabling explicit high-level memory management
- Node Resource Manager (Argo NRM)
  - Centralized node resource management infrastructure
  - Resource partitioning, application progress tracking, enforcement of power limits
- UMap
  - User-level library mapping NVRAM/SSD into the memory hierarchy
  - Uses transparent DRAM cache with policies customizable to applications
- PowerStack
  - Power management infrastructure improving energy usage, cost-to-solution, and performance
  - Comprehensive, hierarchical, application-driven

Outline

Power Management At Exascale

HPC Powerstack

Bring together experts from academia, research laboratories and industry in order to design a holistic and extensible power management framework, which we refer to as the PowerStack. The PowerStack explores hierarchical interfaces for power management at three specific levels: batch job schedulers, job-level runtime systems, and node-level managers.

See: **https://powerstack.caps.in.tum.de/**

Node-level Power Management: NRM

Open-source framework for node-level resource management for HPC.

Overarching goal: **Provide a user-facing, configurable resource controller daemon**

Single point of control for:

- Tracking live processes on a compute node
- Monitor resource and performance of application components
- Provide access to existing resource control knobs
- Implement configurable control loops

Impact Goals:

- Allow users to tune on-the-fly critical resource allocations between components
- Provide easy access to the many new resource control knobs available

Overall Architecture

Inspired by control theory abstractions:

- Sensors: means to monitor a resource or application
- Actuators: means to perform an action w.r.t. a resource or an application
- Goals: control objectives

Framework around a user-space daemon embedding a control infrastructure:

- Communication-layer to report, retrieve information about the system
- Tracking logic to discover sensors/actuators on the system
- Event-based information collection
- Flexible control loop synthesis/configuration

Sensors

Means to monitor a resource or an application

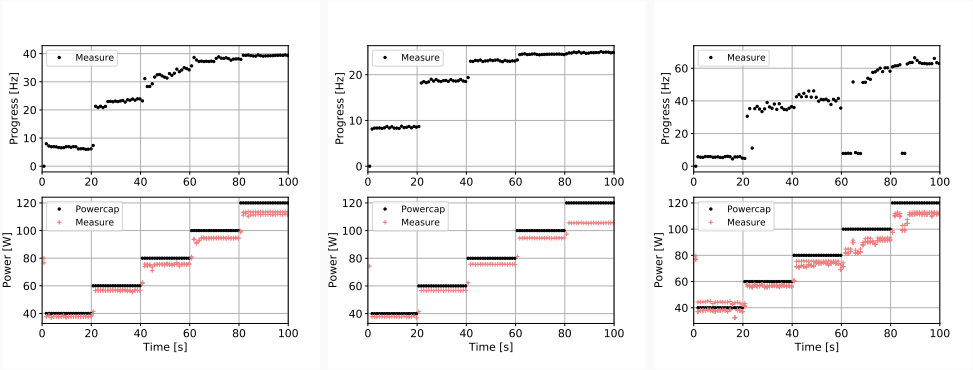
Multiple types:

- Progress: application reports its own performance
- Profiling: PMPI, OMPT events about runtime behavior
- Hardware sensors: msr/sysfs (machine-specific) information about hardware
- Hardware counters: indirect application performance metrics

Filtering/Aggregation of events might also be needed.

Ongoing work to also support statistics on top of raw values.

Progress Example



Actuators

Means to perform an action on the system

Multiple types:

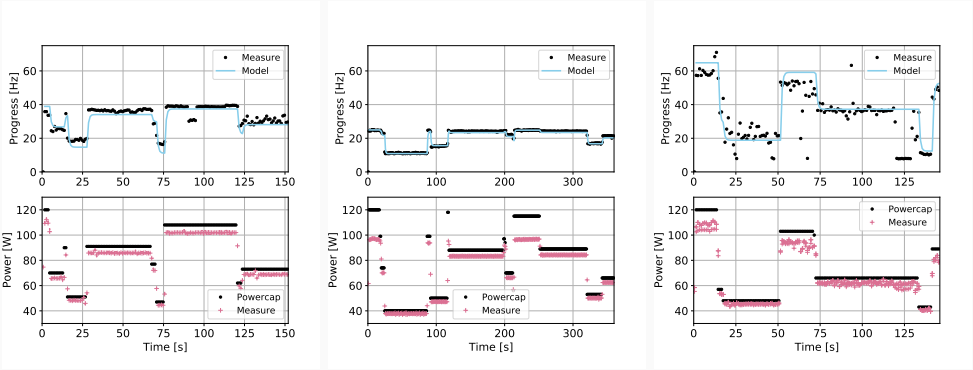
- Hardware actuators: power control, fans, rdt
- Software: priorities, arbitrary code execution

Interacts with the system, can be difficult depending on permissions.

Need to build an understanding of the relationship between domain of control and domain of monitoring:

- Hardware knobs limited to specific socket/core
- Which processes are interacting with which resource (cgroups)

RAPL Example



Goals

Express optimization objective of this specific setup

Traditionally:

- Minimize power under a performance bound
- Maximize performance under a power budged

Need to keep track and adapt control depending on:

- Frequency of events (both sensors and actions)
- Active components (change in workflow)
- Major changes in behavior (phases)

-> Control synthesis

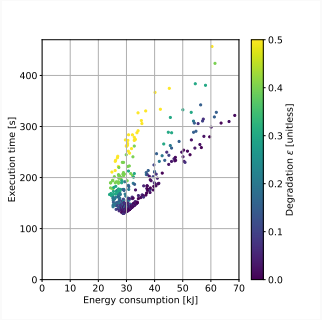
Collaboration with control theory experts from Grenoble University/INRIA.

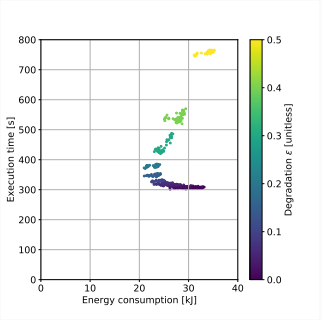
- Sensor: application progress, RAPL
- Actuator: RAPL
- Goal: minimize power under performance bound
- Control: classic PID controller, configured through offline measures

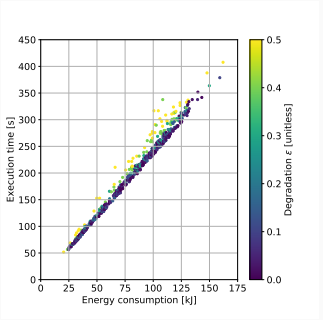
Characterization/Validation on Grid5000 clusters.

Sensors smoothing based on median value over control window.

Results


Execution time [s]
Energy consumption [kJ]
Degradation ε [unitless]


Execution time [s]
Energy consumption [kJ]
Degradation ε [unitless]


Execution time [s]
Energy consumption [kJ]
Degradation ε [unitless]

Framework for Resource Management

NRM is designed to be flexible, could be used to perform management of other resources.

-> How generic can we make the event data collection ?

-> More complex control strategies, or allow for reconfiguration on-the-fly

-> Permissions, user-access, configuration management are difficult to deal with.

Technical Issues: Monitoring

Typical problems:

- Need root to allow some sensors/actuator
- Not all events are being sent all the time
- PMPI, OMPT require `LD_PRELOAD`
- Containers are difficult to inject on the fly
- Containers are not using cgroups on HPC systems
- How much data should be kept/collected ?
- What about support for `another sensor` ?

Technical Issues: Scoping

Problem: **the granularity of your monitoring can be different than the granularity of your control**

Typical example: RAPL is socket-wide, application uses multiple sockets.

-> scopes: on top of measurements, NRM reports the list of physical resources in the scope of this measurements.

E.g.: a progress for the whole node, or two progress signals (one for each socket).

- Makes the timeseries database at the core of event management a tad weird
- You don't want the user to deal with topology complexity in their code
- Tends to make the control logic brittle to topology changes
- Choice of how to aggregate measurements across scope less than trivial

Technical Issues: Exporting Control

How to allow design/prototyping of new control methods, without having to dive into the core of the framework ?

-> Control synthesis helps: provide a description of the control problem independent of the complexity of the implementation

-> Use the NRM as a pass-through: disable internal control, and allow control actions to be sent from a client

-> Allow integration with the powerstack: a system-wide or job-level resource controller can provide distributed control on top.

Conclusion

On Power

Energy efficiency will matter more and more in the future, and it's not just about reducing power consumption.

Most of the existing works focus on job or system level power management, but there is work that can be done at the node level.

Also:

- How to avoid startup parameters ?
- How to deal with phases, control reconfiguration ?
- AI/ML methods, instead of control theory ?

On Reconfigurable Resource Management

The system software stack is full of configuration parameters and heuristics that cannot be touched by the users (scheduling, MPI algorithms).

Improving the flexibility of the stack, and the amount of data available at all layers to understand application behavior will lead to better resource utilization across the stack.

Also:

If you allow for more dynamism at the node-level, how does it impacts performance across nodes ?

Thanks!

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.