Dynamic Self-Aware Runtime Software for Exascale Systems

Christian Engelmann, Geoffroy Vallée, and Thomas Naughton
Oak Ridge National Laboratory

Prof. Frank Mueller
North Carolina State University

Why Do wee Need A Dynamic Self-Aware Runtime Software for Exascale Systems?

Background and motivation

• Power consumption is a major component of operating costs (1MW/year = $1,000,000/year)
• With lower component reliability and higher component counts faults will occur frequently
• At 1 billion cores, even a tiny amount of load imbalance can severely affect performance
• System power consumption, resilience, and performance properties change dynamically
• Application performance and resilience needs change as well

Objective

• Offer awareness about system properties and application needs
• Provide autonomous adaptation to dynamically changing system properties and application needs
• Optimize power consumption, resilience, and performance

Preliminary accomplishments

• Energy-aware job scheduling
• Energy-aware computing using and voltage/frequency scaling
• Proactive fault tolerance using process migration
• Load balancing via process migration or data repartitioning

A Dynamic Self-Aware Runtime for Energy Efficiency, Resilience, and Performance

Proposed concept

• A runtime that is aware of dynamic changes and able to autonomously respond
• Employs a control loop with system monitoring (observe), decision models (decide), and corrective actions (react)
• Awareness APIs offer a holistic view of the current system state
• A controller processes the system state and application QoS requests using models
• Feedback interfaces enable corrective actions

Proposed Research In A Dynamic Self-Aware Runtime for Exascale Systems

System monitoring:
- **Load/Power awareness:**
  - Core/node utilization
- **Reliability awareness:**
  - Early failure indications
  - Core/node temperatures
- **Progress awareness:**
  - Comm. patterns/wait times
  - Application epochs

Decision models:
- Energy-, load-, and progress-aware power management
- Load-, progress-, and reliability-aware scheduling and migration
- Single and across-node models
- Feedback/feed-forward control

Corrective actions:
- Power management
- Task scheduling
- Process pinning and migration

Self-aware runtime framework:
- Monitoring via OS, IPMI, SMART
- Data dissemination using Gossip
- Event-based framework using out-of-band and/or piggybacking
- Integrated with vendor RAS system, OS, programming runtime, and application

Task scheduling to improve:
- **Energy efficiency** and **load balance** by matching system resources with application needs

Single-node power management to improve a node’s:
- **Energy efficiency** by slowing down underutilized cores
- **Load balance** by slowing down cores that are ahead and speeding up those behind

Single-node process pinning and migration to improve a node’s:
- **Reliability** by wear-leveling cores
- **Reliability** by distributing heat

Across-node power management to improve a system’s:
- **Energy efficiency** by slowing down all cores

Across-node process migration to improve a system’s:
- **Load balance** by moving processes from over- to underutilized nodes
- **Reliability** by wear-leveling node usage
- **Reliability** by distributing hot spots across nodes
- **Reliability** by anticipating imminent node failures (proactive fault tolerance)