DynaSched

A dynamic scheduler for HPC/AI/DA

Overview of the problem

- Utilization rates are falling
  - DOE has taken notice at facilities such as NERSC
  - Expected to see similar issues at Argonne, ORNL, LLNL
  - Caused by increasing use of HPC machines for AI/DA and workflows
- AI/DA are dynamic environments
  - Operation in HPC static scheduling requires allocating max probable resources
  - Many resources frequently lie idle during execution
- Custom workflows are dynamic at heart
  - Pursue DAG to solution
  - Uncertain which branches will be taken, what resources will be used
  - Must allocate max probable resources to avoid early termination
- Sole contact with scheduler is at time of submission
  - Obtain allocation envelope of maximum possibly required size

Requirements (I)

- Dynamic scheduler
  - Must support traditional HPC static operations
  - Treat non-traditional workloads as first-class citizens
    - No “islands” of allocation
    - Rapid response to dynamic requests (avoid idling applications)
  - Maximize utilization vs energy consumption
  - Anticipatory vs reactive to minimize dead time
- Application driven environment
  - App-system integration
  - Application is full partner in determining allocation, migration, pre-emption, delayed start, resources to pre-position
  - Extends from scheduling stage throughout app lifecycle

Requirements (II)

- Anticipatory launch
  - Pre-position to anticipated launch regions
    - Data, OS images, containers,…
  - Inter-job energy management
    - Power down, coast, power up,…?
- Learning to improve
  - Allow sys admin to identify desired metrics
  - Watch workloads vs metrics to improve scheduling algo and to better predict/preposition required resources

Requirements (III)

- Broaden range of allocated resources
  - Treat all resources used by applications as allocatable
  - CPU, GPU, fabric, power, total energy, memory
- Broaden range of scheduling
  - Allocation start/end
  - Sequencing of apps (DAG), data position
  - Static pipelines and dynamic branching

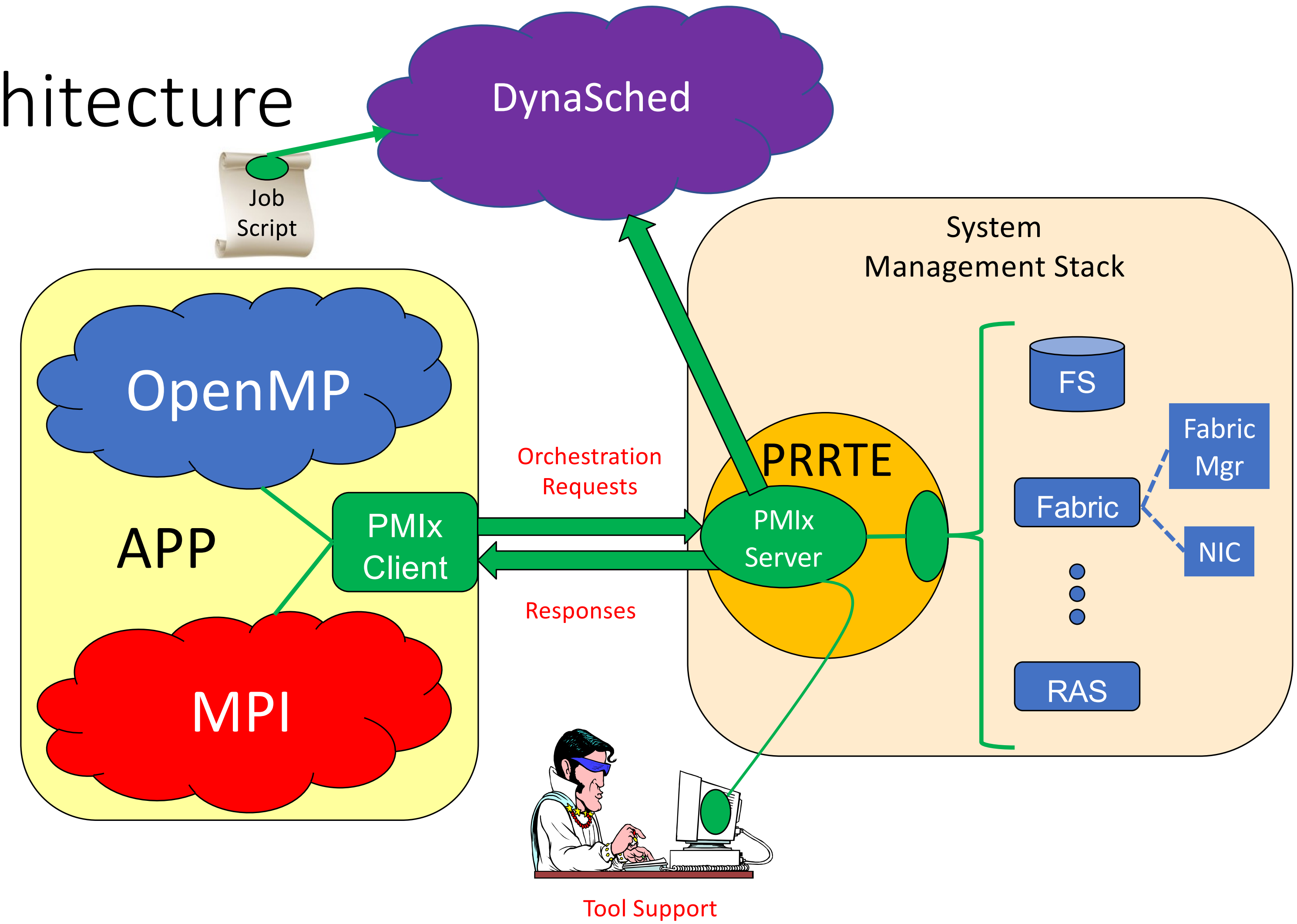
Requirements (IV)

- Resilient
  - Assets can appear/disappear without warning
  - Degradation can occur
    - Example: communication link noise
- Application classes
  - Traditional HPC – static, bulk synchronous
  - Workflow HPC – execute DAG of apps
  - Data analytics – stream processing of extremely large data sets
  - Machine learning – training and inference, multiple algos (i.e., don't fixate on current DL approach!)
  - Dynamic analysis – population modeling, statistical analysis (dynamic resizing, non-synchronous)

Assumptions

- Preemption is acceptable
  - When needed for higher priority/more urgent applications
  - Register preemption abilities
    - Handshake requirements (if any)
    - Ability/willingness to accept preemption
  - App participates in preemption
    - Handshake timing, recovery requirements
- Node sharing may be acceptable
  - If we can provide adequate resource allocation and security walls between users

Architecture



DynaSched LRH Scheduler

- Lagrangian objective function
  - Combine constraints into objective function using time-dependent parameters (Lagrangian multipliers)
  - Maximize number of sessions completed
    - Within specified time and energy constraints
    - Must complete entire DAG of dependencies
- Receding horizon
  - Optimal control method used in job shops
  - Predict evolution of system for limited time into future
  - Control based on prediction until next measurement of system state

Objective Function

$$ObjFn(\alpha, \beta, \gamma) = \alpha \frac{T_{100}}{T} - \beta \frac{TEC}{TSE} - \gamma \frac{AET}{\tau}$$

TEC = Total Energy Consumed
TSE = Total System Energy
AET = Application Execution Time
α,β,γ = Lagrangian multipliers [0,1], $\alpha+\beta+\gamma=1$
τ = time constraint
T_{100} = number of sessions fully completed
T = number of sessions submitted

Term for each constraint (will need to extend)
Initial multiplier values set by experiment
- Adjust on the fly

Determine when to preempt

Methodology

- At each time step
  - Collect set of all submissions that
    - Meet all precedence constraints
    - Adequate energy to execute at least minimum work
    - Meet minimum resource constraints
  - Evaluate ObjFn for each submission
  - Order submissions based on ObjFn
  - Find first submission that can be scheduled within time horizon - map it
  - Continue until no additional submissions can be started within time horizon
- Increment time
  - Recreate submission set
    - New submissions, dynamic requests, session completions
  - Proceed as above

Multiplier Adjustment

- Simple feedback controller
  - Inputs
    - Deviation from full utilization
    - Current multipliers
    - Some metric on sessions in queue and/or in execution?
    - Historical behaviors?
  - Outputs
    - Updated multipliers
- Neural network controller
  - Continuously train using queue metric, utilization?
  - Output multipliers

My Role: Consultant

- Transfer knowledge
  - LRH scheduler, possibly discuss adding NN integration
  - PRRTE, PMIx code bases
- Setup platform for investigations
  - Stable, robust PRRTE environment (pre-production ready)
  - Stable, robust PMIx environment (production ready)
  - DynaSched prototype
- Guide investigations
  - Advise application team
  - Actively participate in early investigations

Step 1: Enhance PRRTE for RM role

- Bootstrap backbone daemon network during cluster boot
- Security enhancements for privileged operation
  - Deal with privileged-nonprivileged interactions
- Restore PRRTE daemon resilience
- Implement session instantiation procedure
  - Fork/exec local daemon at user level
  - Instant on wireup of session daemons
- General code cleanup
  - Picky compiler, Coverity reports

Step 2: PRRTE scheduler "hooks"

- Reuse existing PMIx APIs where possible
  - PMIx_Allocate_resources
  - PMIx_Job_control
- Upward communication
  - Relay allocation requests from apps
  - Preemption registration from apps
  - Resource inventory and changes in availability
  - Changes in session status (terminate, etc)
- Downward communication
  - Session instantiation/modification orders
  - Pre-emption orders

Step 3: Simple “greedy” scheduler

- Priority based queueing system
- Take highest priority request that fits within available resources
  - Tie: take largest one, then longest
- Adjust priorities based on waiting time
  - Bump priority each time step you don’t get allocated
- Explore preemption strategies and their impact
  - When to preempt vs wait, which session(s) to preempt first
- Explore strategies for responding to dynamic requests
  - Allocation changes (extend, release, elongate)
  - Running vs waiting priority balance

Step 3a: Application Integration

- Select candidate applications
  - Mix of ML, DA, workflow – dynamic
  - Traditional HPC – static
  - Hybrid – OpenMP/MPI resource coordination via PMIx
- Integrate with PMIx to DynaSched
  - Dynamic examples
    - Replace static max possibly needed assumption
    - Add resource allocation step(s)
  - Static examples
    - Add preemption hooks
- Explore impact
  - Total time to solution
  - System utilization
  - Coding complexity (user adoption obstacles)

Estimated Timeline

- Step 1: Enhance PRRTE for RM role
  - Two calendar months (Ralph, mentee)
- Step 2: PRRTE scheduler “hooks”
  - One calendar month (Ralph, mentee)
- Step 3: Simple “greedy” scheduler
  - Two calendar months (Ralph, mentee)
- Step 3a: Application Integration
  - ?? (App person(s))
- Exploration and report
  - ?? (All)