Towards Realizing the Potential of Malleable Parallel Jobs

Bilge Acun
acun2@illinois.edu
Department of Computer Science,
University of Illinois at Urbana Champaign, Urbana, IL

Abhishek Gupta | Bilge Acun | Osman Sarood | Laxmikant Kale
IEEE International Conference on High Performance Computing (HiPC) 2014

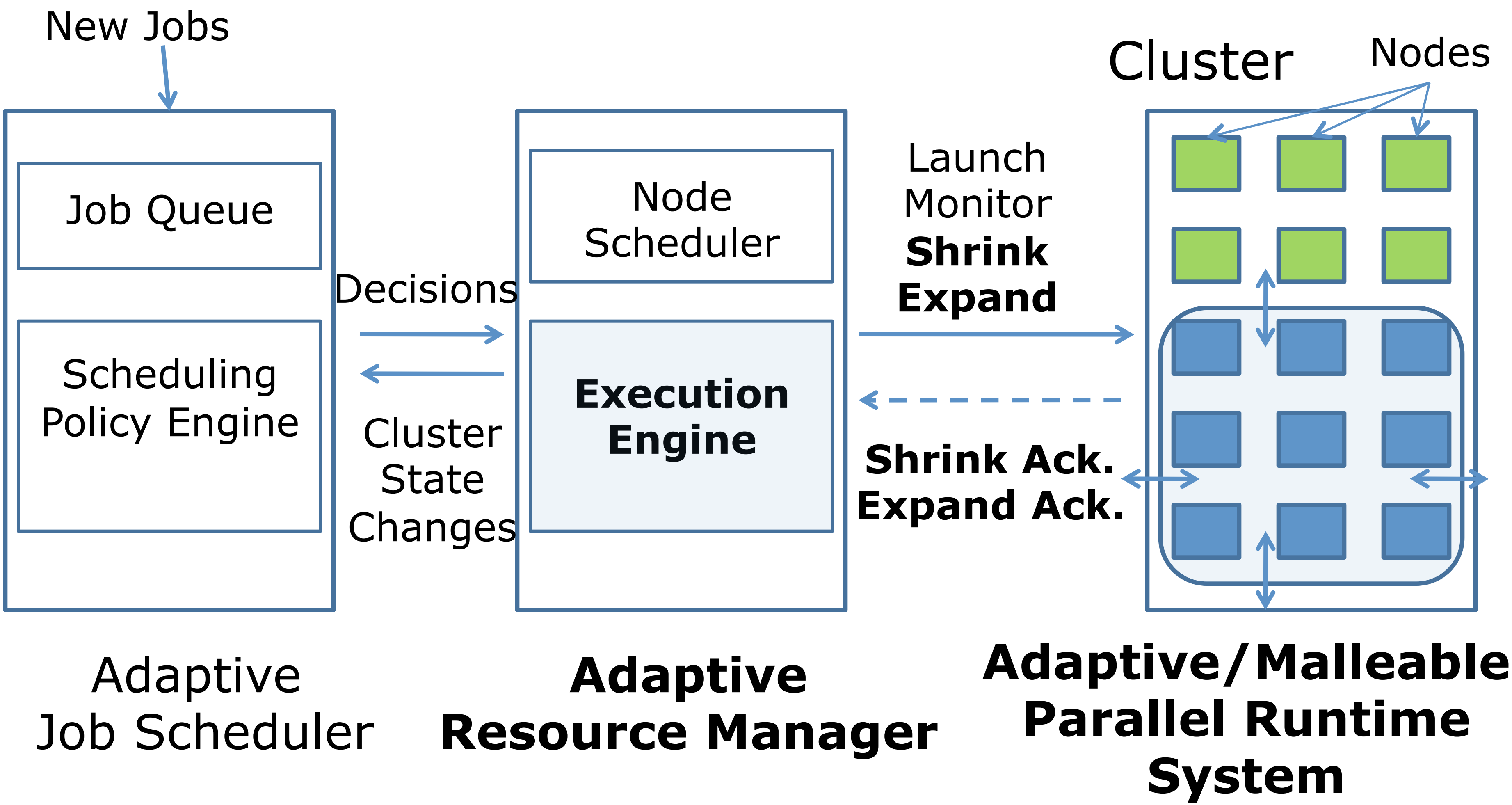
Malleable Parallel Jobs

- Dynamic shrink/expand number of processors
  - *Shrink*: A parallel application running on nodes of set A is resized to run on nodes of set B where $B \subset A$
  - *Expand*: A parallel application running on nodes of set A is resized to run on nodes of set B, where $B \supset A$
  - *Rescale*: Shrink or expand

- Twofold merit
  - Provider perspective
    - Better system utilization, throughput
    - Honor job priorities
  - User perspective:
    - Early response time
    - Dynamic pricing offered by cloud providers, such as Amazon EC2
      - Better value for the money spent based on priorities and deadlines

Malleable jobs have tremendous but unrealized potential,
What do we need to enable malleable HPC jobs?

Components of a Malleable Jobs System



We will focus on Malleable Parallel Runtime

Related Work

- Prior works focus on job scheduling strategies
- Parallel runtime for malleable HPC jobs open problem
- Existing approaches
  - Residual processes when shrinking
    - Charm++ malleable jobs (Kale et al.)
    - Dynamic MPI (Cera et al.)
  - Too much application specific programmer effort on resize
    - Dynamic malleability of iterative MPI applications using PCM

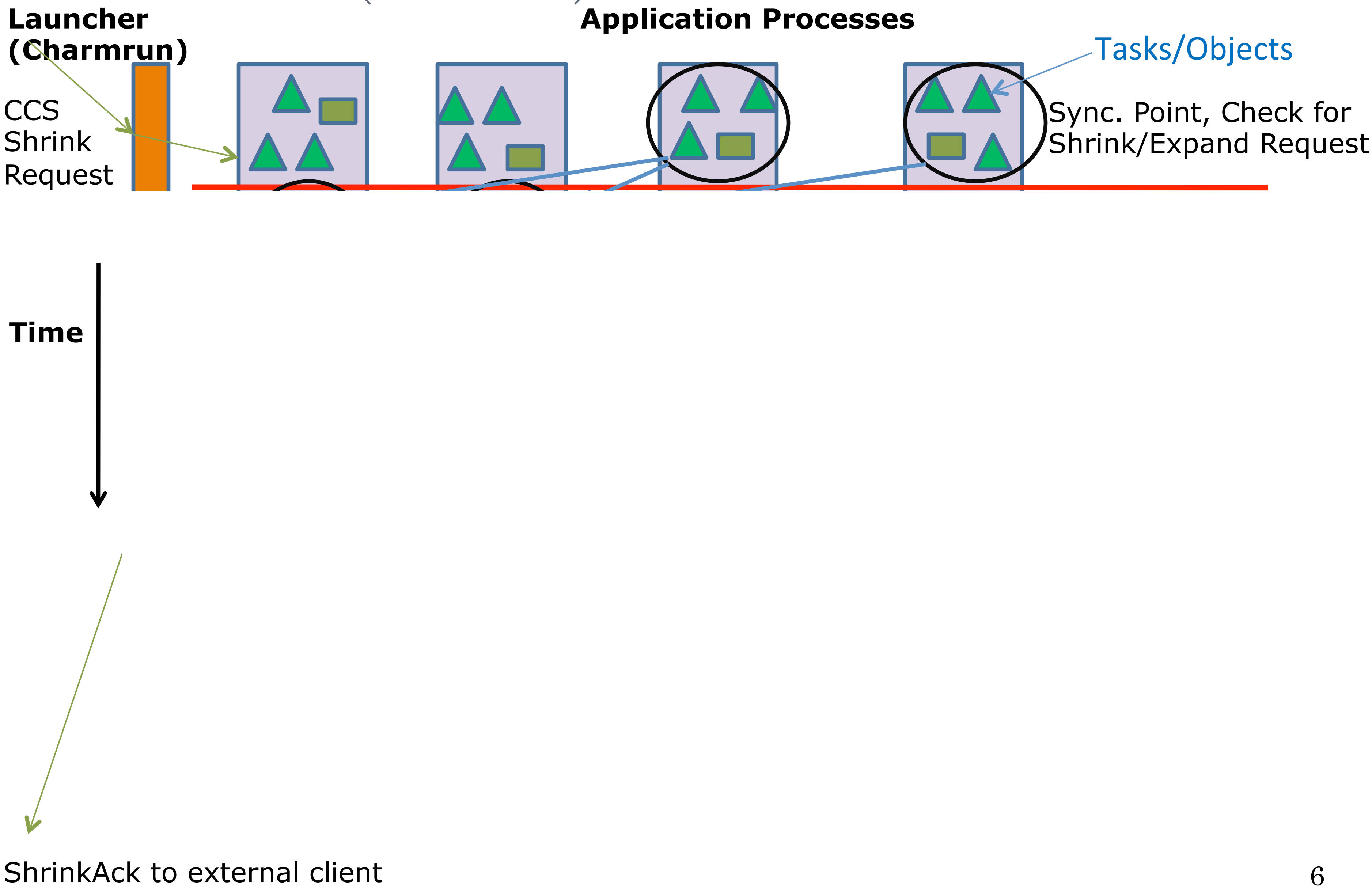
Our focus: parallel runtime to render a job malleable
•No residual processes
•Little application-specific programming effort
•Goals: Efficient, Fast, Scalable, Generic, Practical, Low-effort!

Definitions and Goals

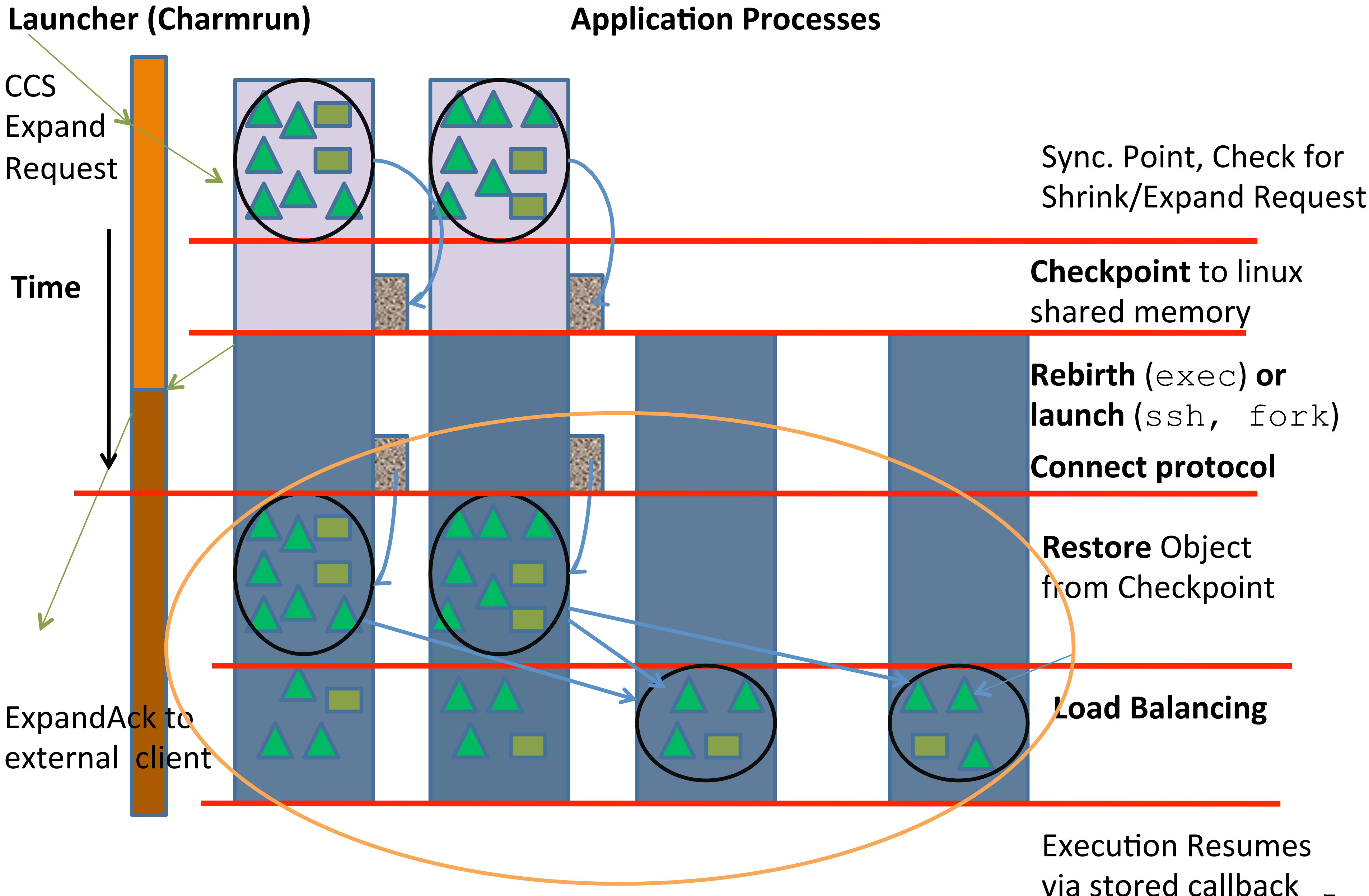
- *Shrink*: A parallel application running on nodes of set A is resized to run on nodes of set B where B ⊂ A
- *Expand*: A parallel application running on nodes of set A is resized to run on nodes of set B, where B ⊃ A
- *Rescale*: Shrink or expand

- Goals:
  - Efficient
  - Fast
  - Scalable
  - Generic
  - Practical
  - Low-effort

APPROACH (SHRINK)



Approach (Expand)

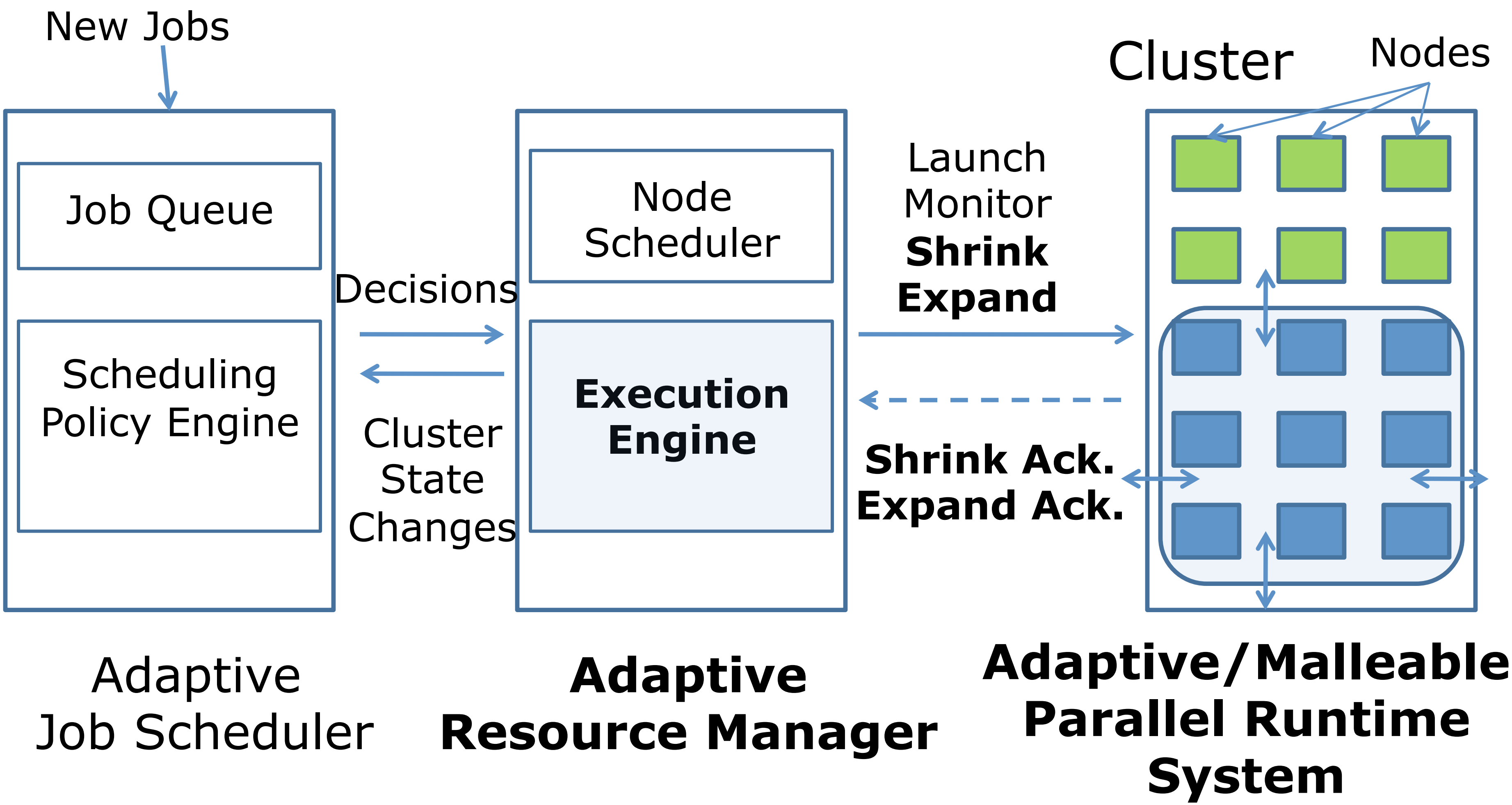


Malleable RTS Approach Summary

- Task/object migration
  - Application-transparent redistribution
- Checkpoint-restart
  - Clean restart (rebirth)
- Load balancing
  - Efficient execution after rescale
- Linux shared memory
  - Fast and persistent checkpoint

- Implementation atop Charm++

Components of a Malleable Jobs System



Adaptivity in Resource Manager

- *How* and *when* to
  - Communicate scheduling decisions to parallel application
  - Detect success or failure of those actions
- Resource manager to RTS communication channel (*how*)
- Split phase execution of scheduling decisions (*when*)

Algorithm 1 *Shrink-Expand* Split-phase Execution

```
while true do
    jobDecisions = ScheduleJobs(jobQueue, clusterFreeNodes,
    runningJobs, T_rescale, optionalArgs)
    nodeDecisions = ScheduleNodes(jobDecisions,
    clusterNodeState, optionalArgs)
    UpdateSchedNodeMap(nodeDecisions)
    postponedActions = ExecuteDecisions(jobDecisions)
    repeat
        ProcessBufferedShrinkAcks()
        ExecutePostponedDecisions(postponedActions)
    until (jobQueue != empty or a job finished)
end while
```

Experimental Evaluation

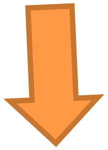
- Four HPC mini-applications with Charm++:
  - **Stencil2D:** 5-point stencil on a 2D grid using Jacobi relaxation
  - **LeanMD:** Mini-app version of NAMD molecular dynamics app
  - **Wave2D**: 2D mesh based mini-app for simulating wave propagation
  - **Lulesh:** Charm++ version of LULESH hydrodynamics mini-app

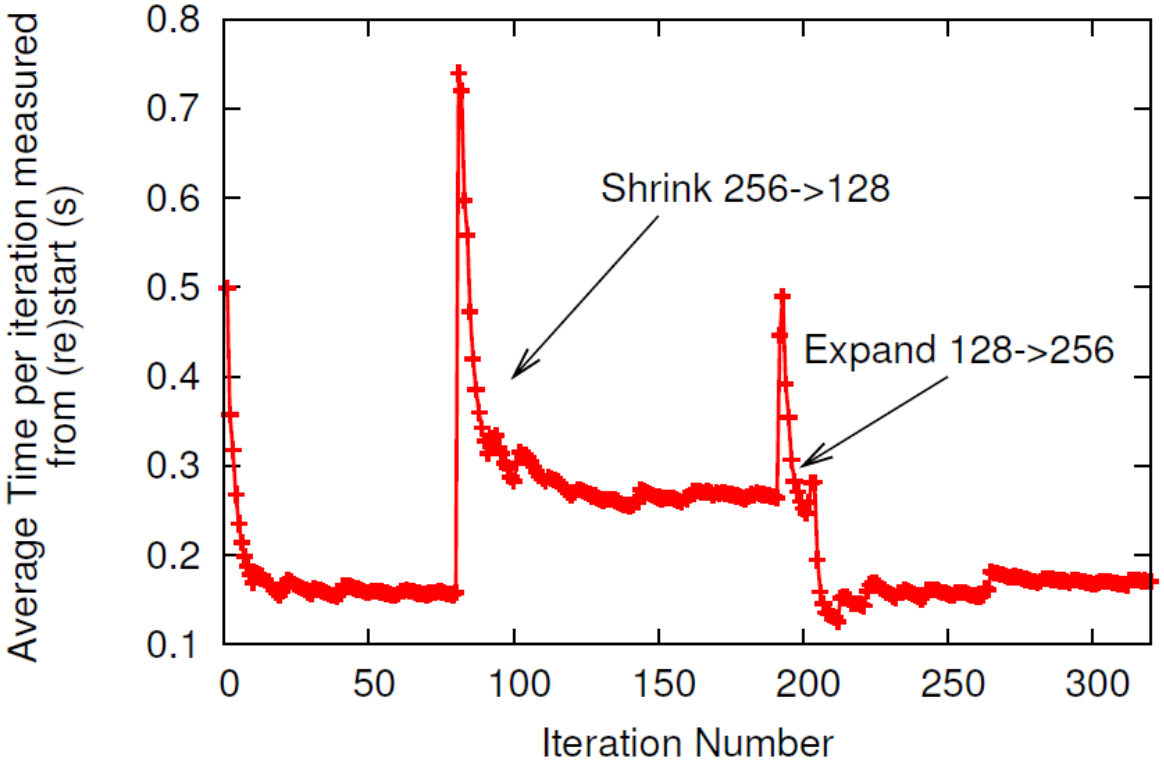
  - All experimental results are done on **Stampede**

- Evaluate against design goals

RESULTS: ADAPTIVITY

Low is better

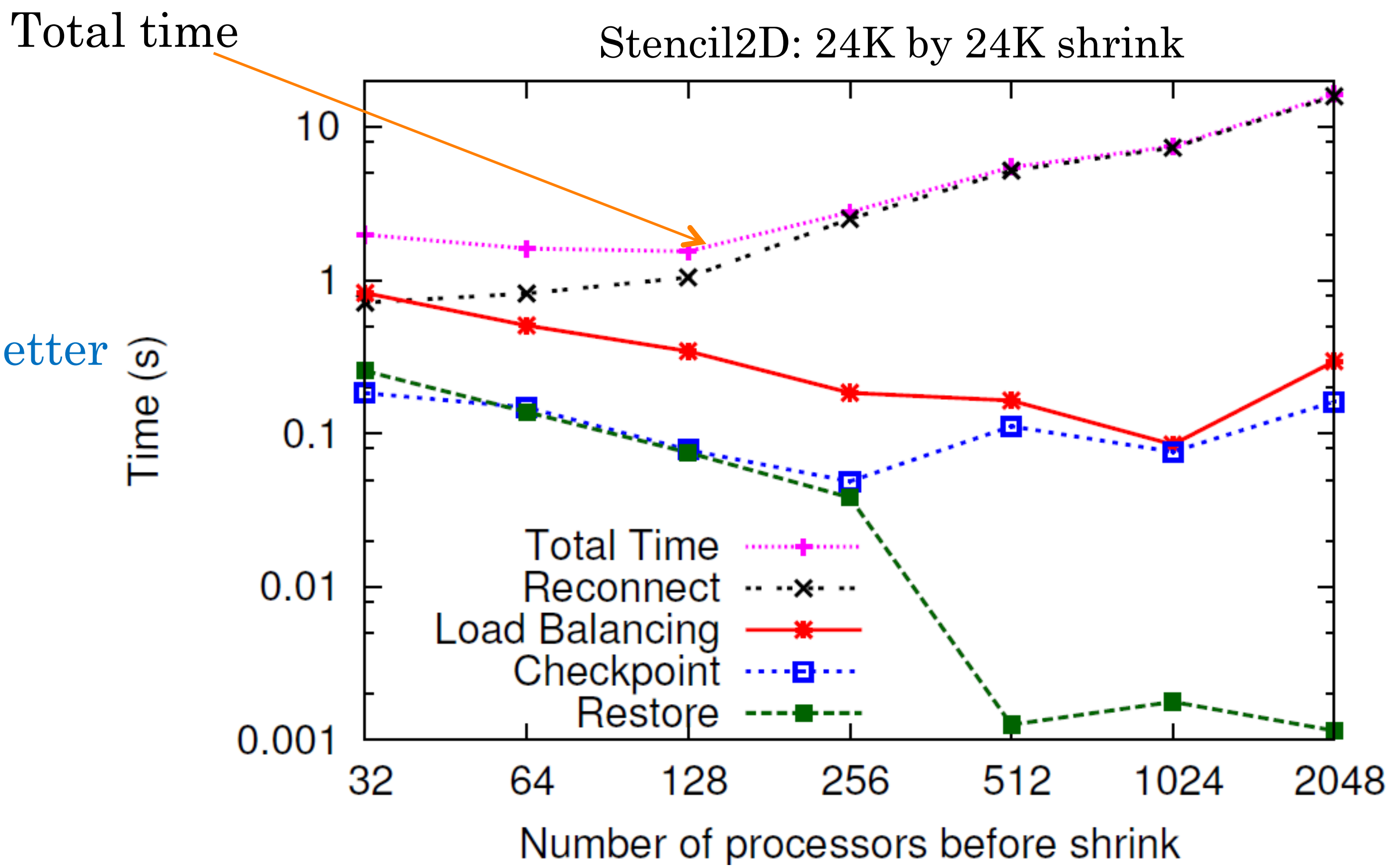




LeanMD: Adapting load distribution on rescale, showing that our approach is efficient

Results: Scalability

Low is better

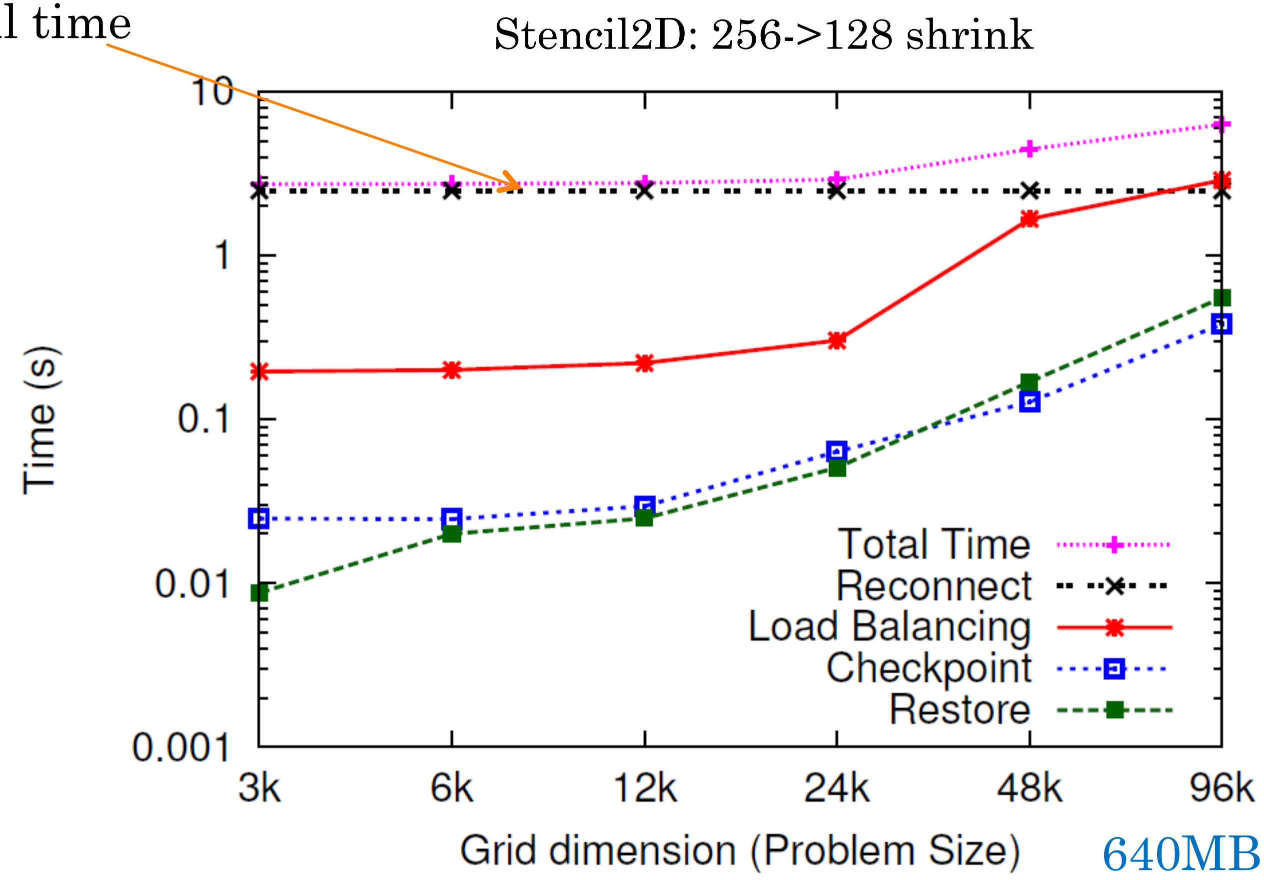


Scales well with increasing number of processors

Results: Scalability

Total time

Low is better



640MB per process at 96K

Scales well with increasing problem size

Results Summary

- Adapts load distribution well on rescale (Efficient)
- 2k->1k in 13s, 1k->2k in 40s (Fast)
- Scales well with core count and problem size (Scalable)
- Little application programmer effort (Low-effort)
  - 4 mini-applications: Stencil2D, LeanMD, Wave2D, Lulesh
  - 15-37 SLOC, For Lulesh, 0.4% of original SLOC
- Can be used in most supercomputers (Practical)

What are the benefits of malleability?

Applicability and Benefits

- Provider perspective
  - Improve utilization: malleable jobs + adaptive job scheduling
  - Stampede interactive mode as cluster for demonstration

- Non-traditional use cases
  - Clouds: Price-sensitive rescale in spot markets
  - Proactive fault tolerance

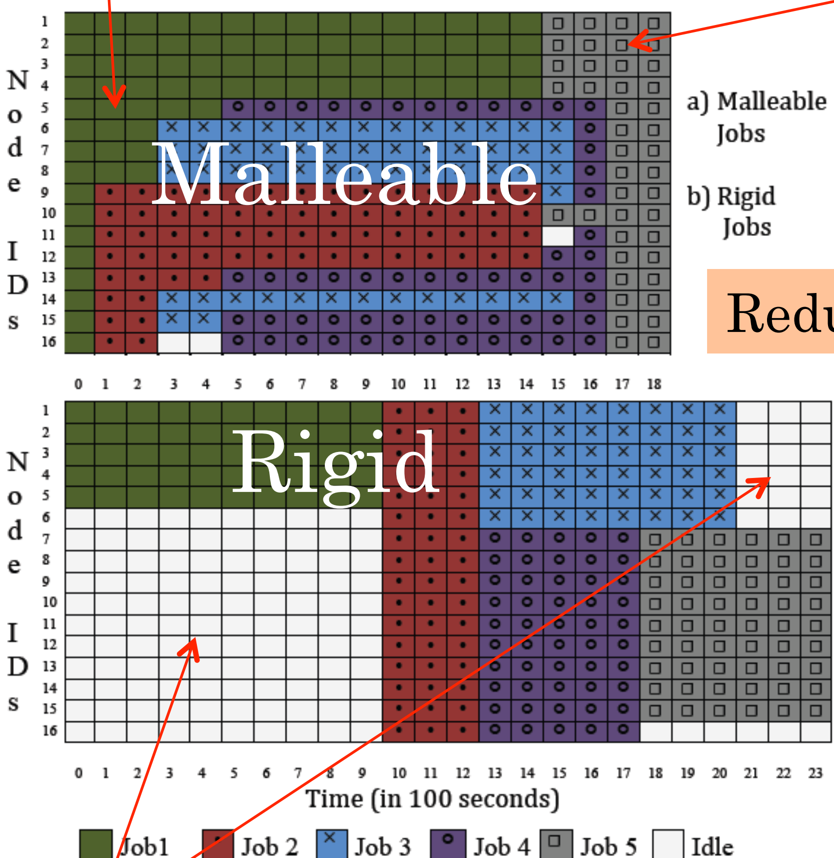
PROVIDER PERSPECTIVE: CASE STUDY

Job 1 shrinks

Reduced response time

Job 5 expands

Improved utilization



Cluster State

Reduced makespan

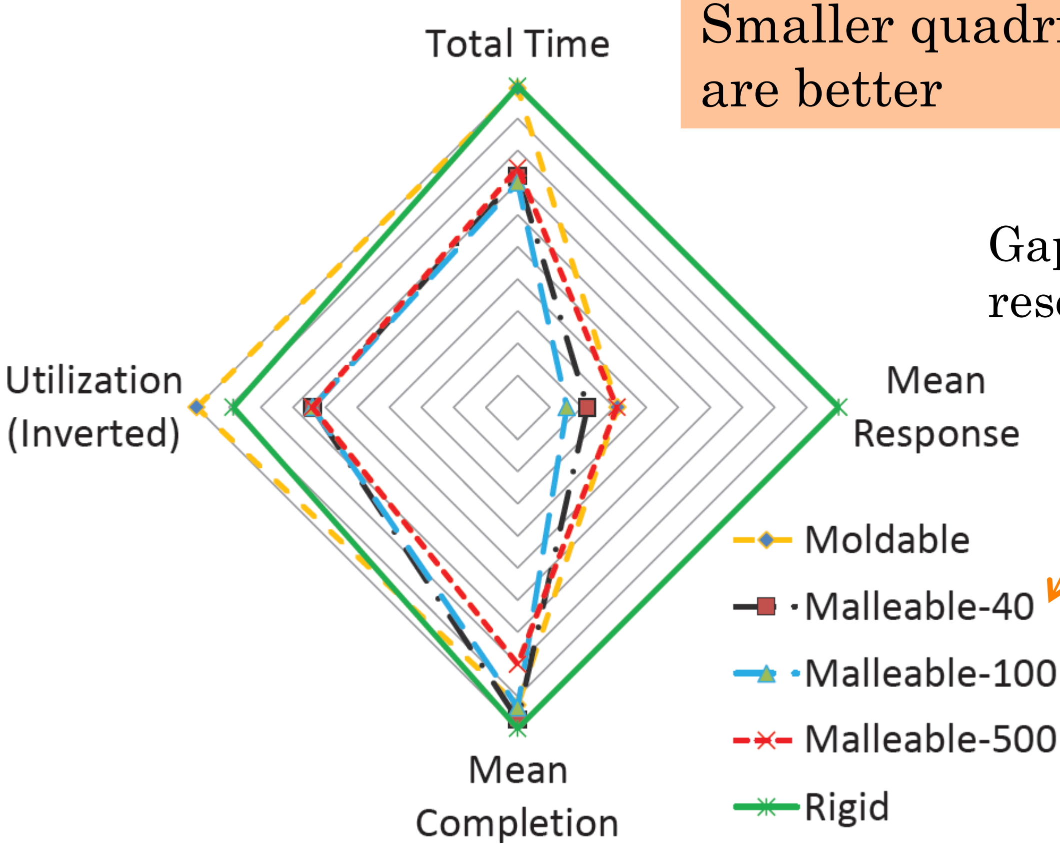
- 5 jobs
- Stencil2D, 1000 iterations each
- 4-16 nodes, 16 cores per node
- 16 nodes total in cluster

- Dynamic Equipartitioning for malleable jobs
- FCFS for rigid jobs

Idle nodes

Time

Provider Perspective: Case Study



Smaller quadrilaterals are better

Gap (s) between 2 rescale for same job

Significant improvement in mean response time and utilization

Benefits: Non-traditional use cases

- Clouds spot markets
  - Price-sensitive rescale over the spot instance pool
    - Expand when the spot price falls below a threshold
    - Shrink when it exceeds the threshold.
- Proactive fault tolerance
  - Shrink on failure imminent notice from resource manager
  - Expand when failed node comes back

SUMMARY

- A novel technique to enable malleability in HPC jobs
- Salient features: task migration, load-balancing, checkpoint-restart, and Linux shared memory.
- Scheduler-RTS communication and split-phase scheduling
- Experimental evaluation: fast, scalable, and effective

- Related and ongoing work:
  - Malleable jobs with Charm++ integrated into Torque/MOAB
    - "A Batch System with Efficient Adaptive Scheduling for Malleable and Evolving Applications" Suraj Prabhakaran et al. IPDPS'15
  - Adaptive Computing
    - Standardize API for malleable and evolving jobs

Backup

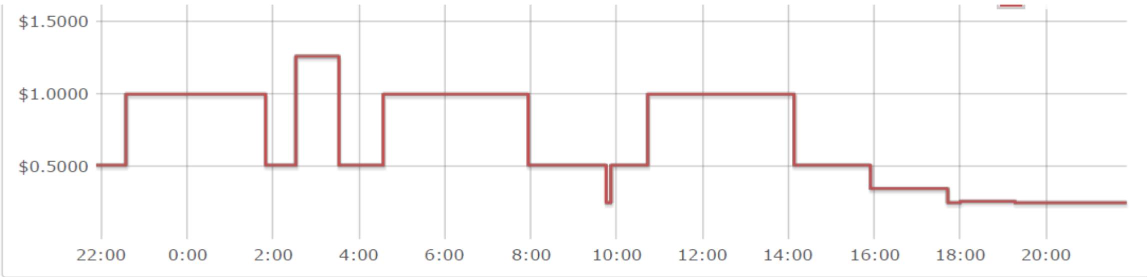
RESULTS

Table 2: *Shrink Expand* time breakup (in seconds) for different applications

Application	*Shrink*: 256 → 128					*Expand*: 128 → 256				
	LB	Checkpoint	Reconnect	Restore	Total	LB (Post)	Checkpoint	Reconnect	Restore	Total
LeanMD	0.515	0.039	2.102	0.003	2.658	0.056	0.016	7.079	0.003	7.154
Lulesh	0.560	0.531	2.533	0.432	4.056	0.458	0.520	7.083	0.436	8.496
Wave2D	1.219	0.243	2.542	0.336	4.340	1.046	0.244	7.067	0.337	8.695
Stencil2D	0.299	0.050	2.501	0.054	2.904	0.133	0.036	7.076	0.038	7.283
Stencil2D_Net	5.86	0.057	2.584	0.056	8.556	4.096	0.042	9.495	0.044	13.678

User Perspective: Price-sensitive Rescale in Spot Markets

- Spot markets
  - Bidding based
  - Dynamic price



Amazon EC2 spot price variation: cc2.8xlarge instance Jan 7, 2013

- Set high bid to avoid termination (e.g. $1.25)
- Pay whatever the spot price or no progress
- Can I control the price I pay, and still make progress?
- Our solution: keep two pools
  - Static: certain minimum number of reserved instances
  - Dynamic: price-sensitive rescale over the spot instance pool
    - Expand when the spot price falls below a threshold
    - Shrink when it exceeds the threshold.

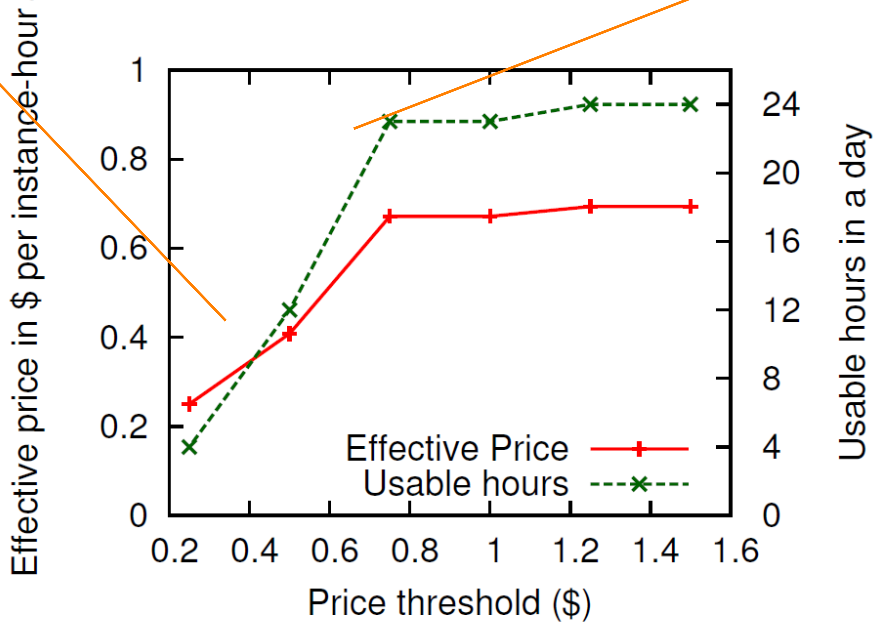
User Perspective: Price-sensitive Rescale in Spot Markets

Price Calculation

No rescale: $16.65 for 24 hours

With rescale: freedom to select price threshold

Usable hours may be reduced



Dynamic shrinking and expansion of HPC jobs can enable lower effective price in cloud spot markets

Proactive Fault Tolerance