PennState
High Performance
Computing Lab

vmware®

Multiverse: Dynamic VM Provisioning for Virtualized High Performance Computing Clusters

JASHWANT RAJ GUNASEKARAN

MICHAEL CUI† , PRASHANTH THINAKARAN* , JOSH SIMONS† , MAHMUT T. KANDEMIR* , CHITA R. DAS*
† VMWARE, * PENN STATE

CCGrid 2021

21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

AGENDA

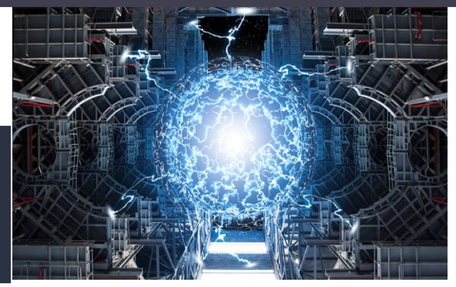
- Introduction
- Challenges & Motivation
- Design
- Implementation
- Evaluation & Experimentation
- Results and Discussion
- Conclusion

High Performance Computing

Virtualization for HPC

U.S. DEPARTMENT OF ENERGY

Galaxy PROJECT



Virtualized HPC

How traditional HPC schedulers handle virtualization?

Challenges with HPC schedulers

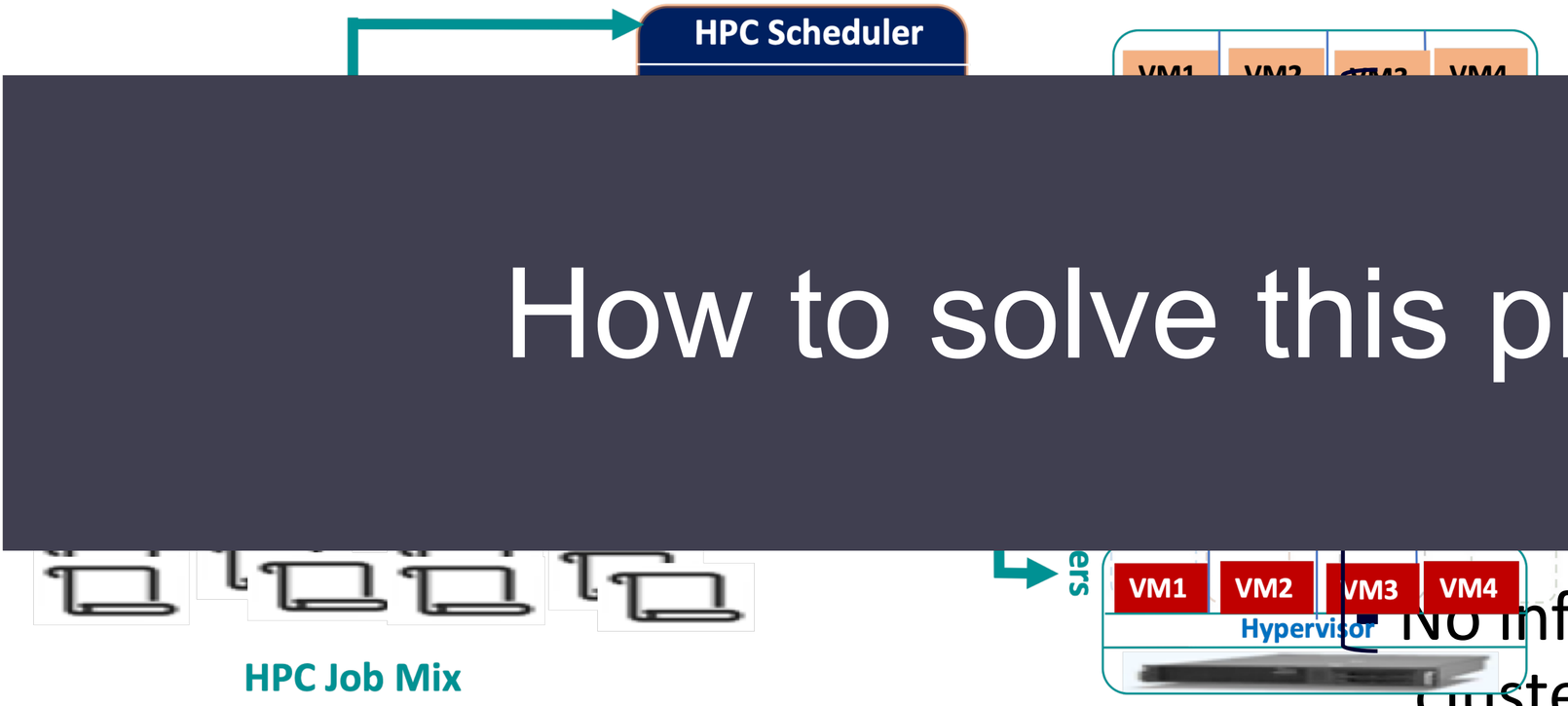
HPC Schedulers

- Focus on throughput and utilization

Poor interaction with VM Orchestrators. Results in Underutilization.

- Fair sharing and fixed node reservations

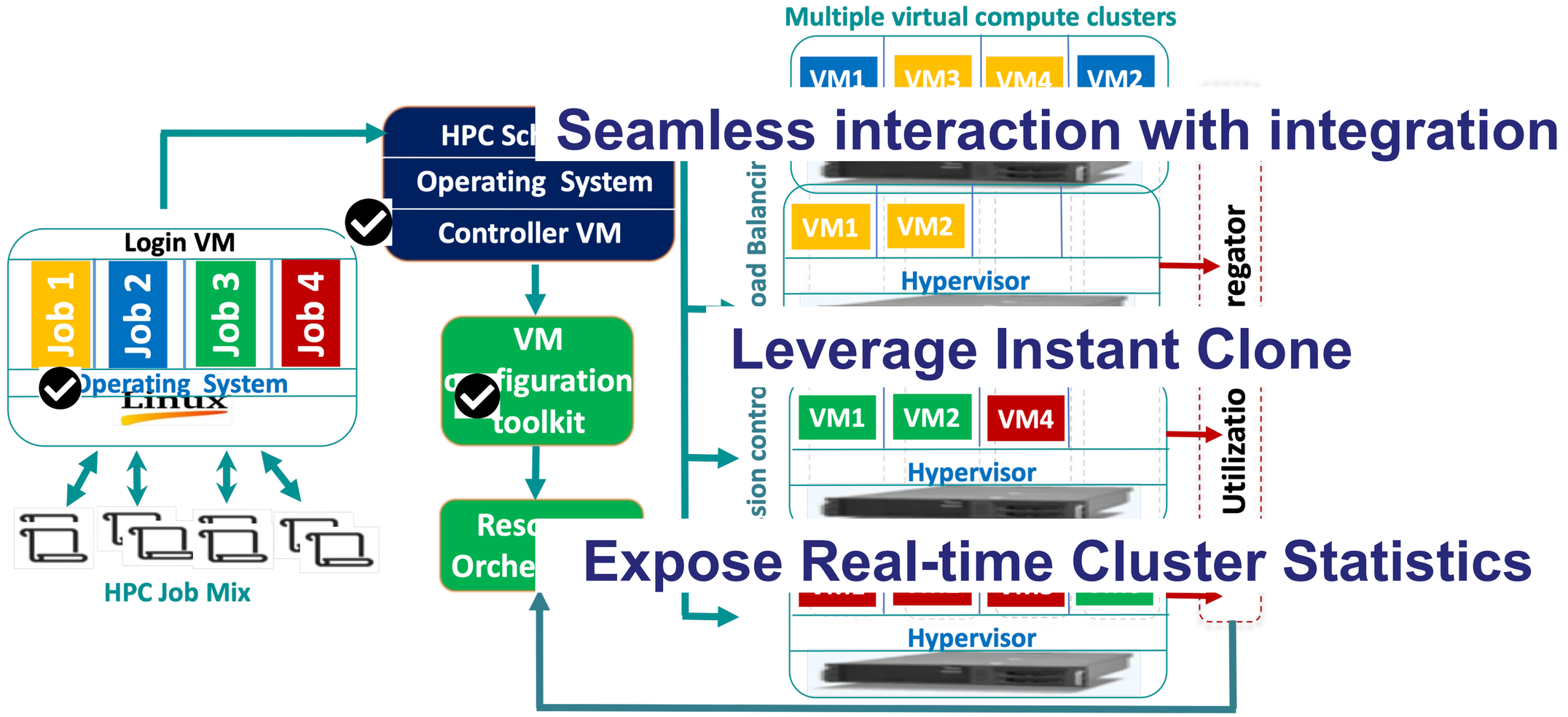
Why Underutilization?



How to solve this problem?

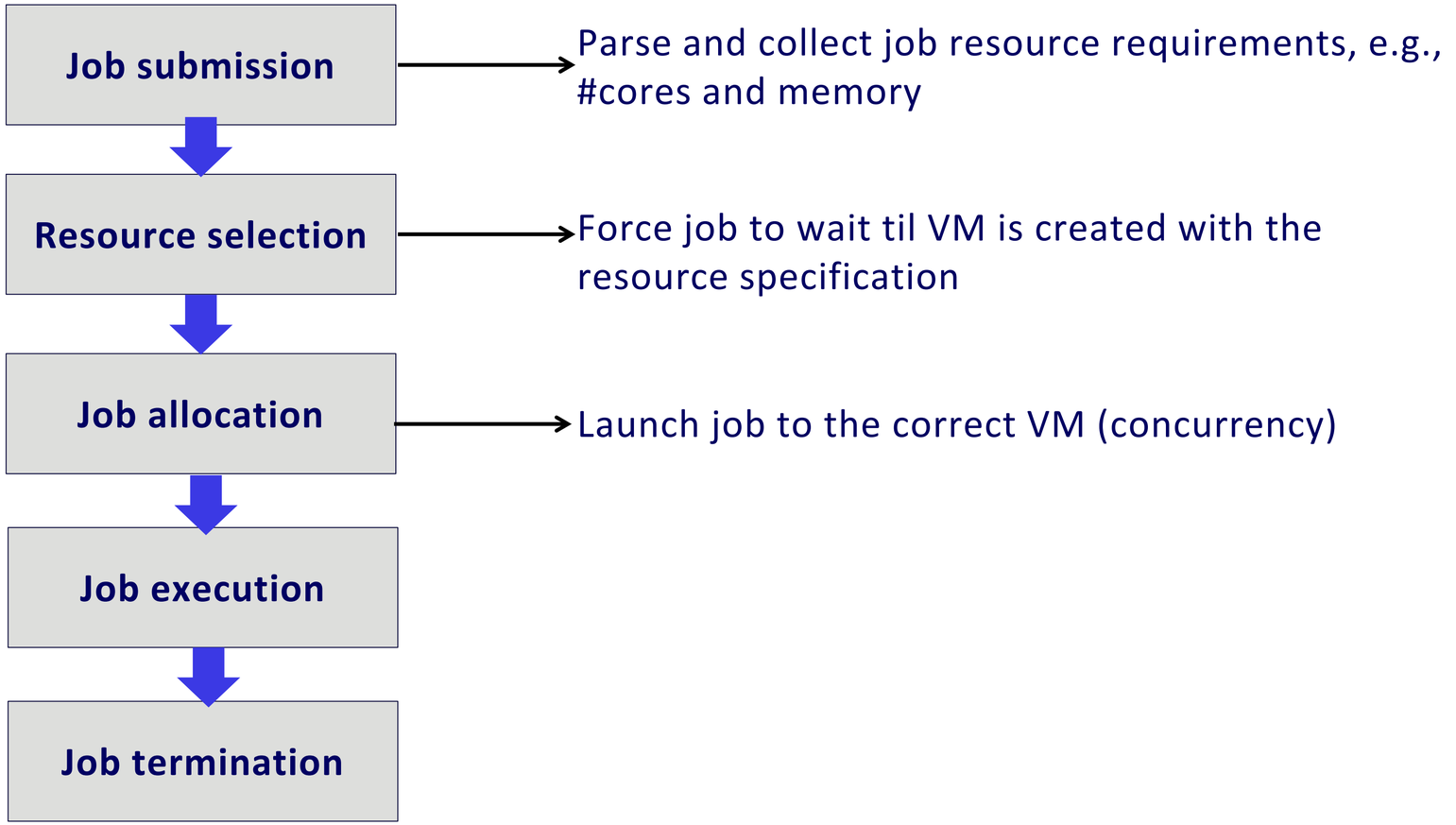
- No information about physical cluster resources

Multiverse Design



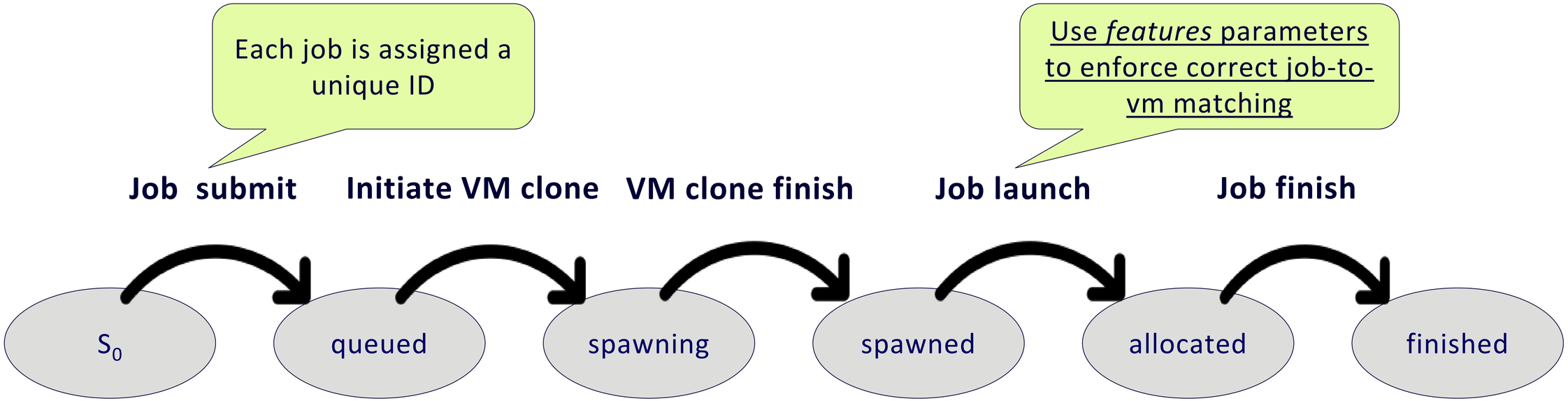
Design Specifications

- Customize VM size
- Customize software/libs
- Pre-create a set of VM templates for different users/jobs



Design Specification

- Need to be thread-safe
- Schedulers are multi-threaded and are thread-safe
- Finite state machine



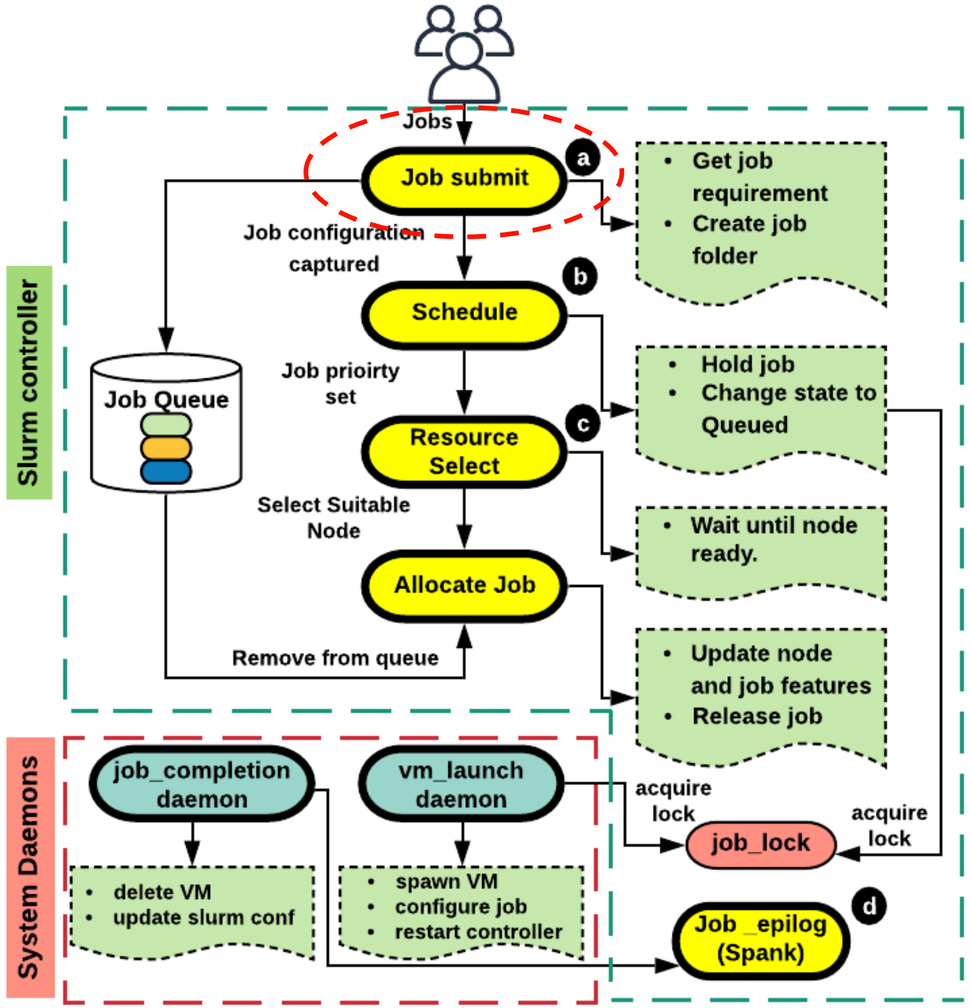
Admission Control & Load Balancing

- A light-weight DB can be used to maintain physical resource utilization info
  - Node capacity can be populated by querying VM orchestrator
  - Metrics include CPU/memory/#VMs, etc
- DB is queried before spawning new VMs for AC and LB
  - When resources are not available, jobs remain in queued state
- DB updates are triggered by VM clone/destroy events
- AC can be customized by specifying an admin parameter – over-commitment ratio
- LB policies are customizable with different algorithms
  - Assumes Dynamic Resource Selection is not available (e.g., Scale-out vSphere Edition)

Implementation Overview

- We chose Slurm which is popular and open-source
- Slurm plug-in APIs allow to customize Slurm dynamically without changing Slurm code
  - A Slurm plug-in is a dynamically linked code object which is loaded explicitly at run time by the Slurm libraries
- Our choice of Slurm plug-ins are inspired by existing works
  - Integrate Slurm with Singularity containers: https://git.biohpc.swmed.edu/biohpc/singularity/tree/utsw/src/slurm
  - Extend Slurm functionality: https://github.com/grondo/slurm-spank-plugins/blob/master/slurm-spank-plugins.spec
  - Elasticluster: https://github.com/elasticluster/elasticluster
- Main challenges lie in the need to restart Slurm controller to add new VMs to compute nodes while preserving existing job states
  - 5 plug-ins, 2 daemons, and locks for synchronization

Implementation on Slurm

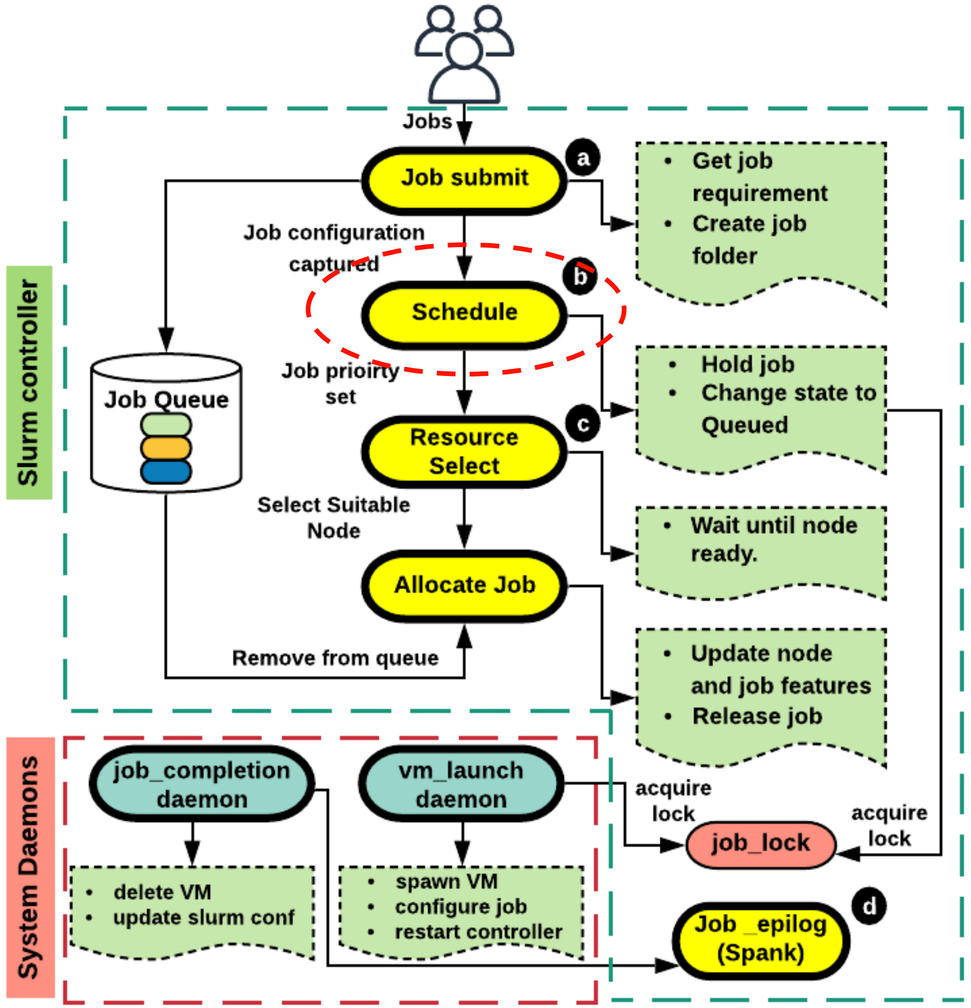


Job submit plug-in

- Called by Slurm controller right after job submission
- Log or modify job configuration parameters

- Job name, #cpus, memory, #nodes, submit time
- Establish a 1:1 mapping between each job and a temporary directory

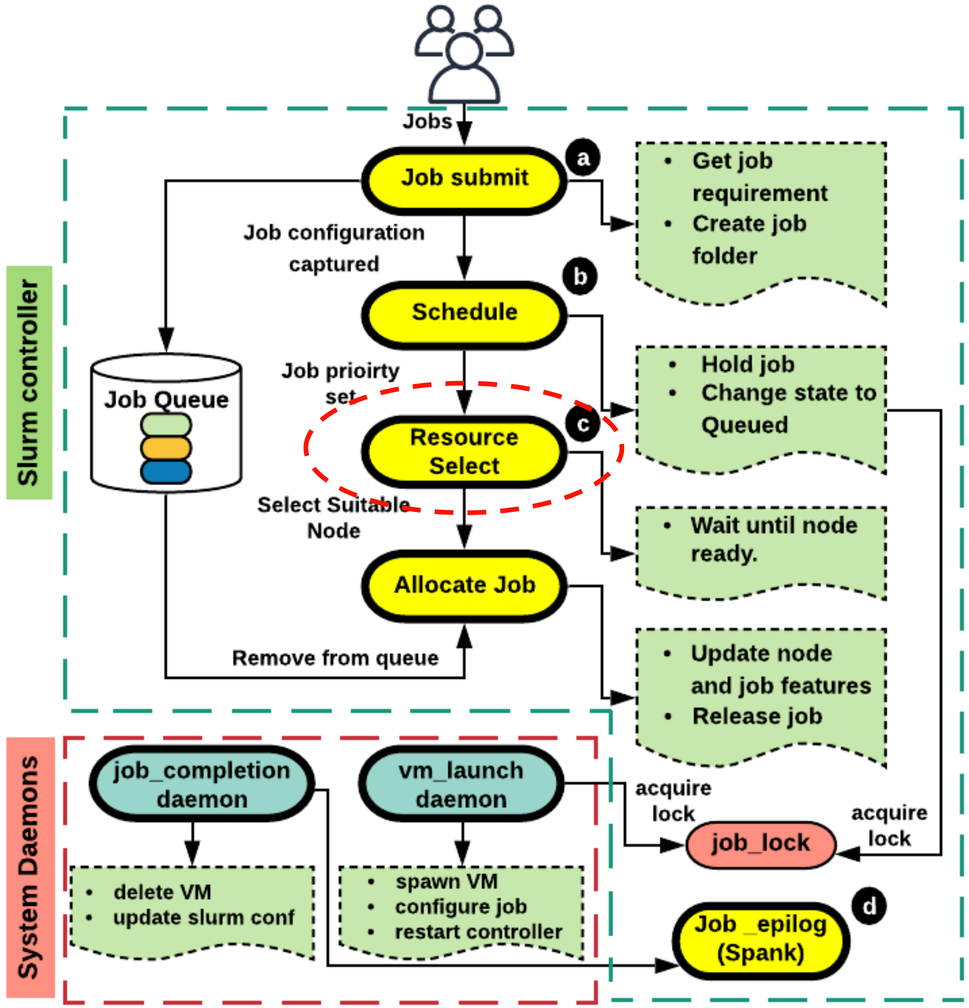
Implementation on Slurm



Scheduler plug-in

- Called after the job submit plug-in
- Change job priority to change order in job queue
- override the *slurm_sched_p_initial_priority* function
- Add the job to an internal queue

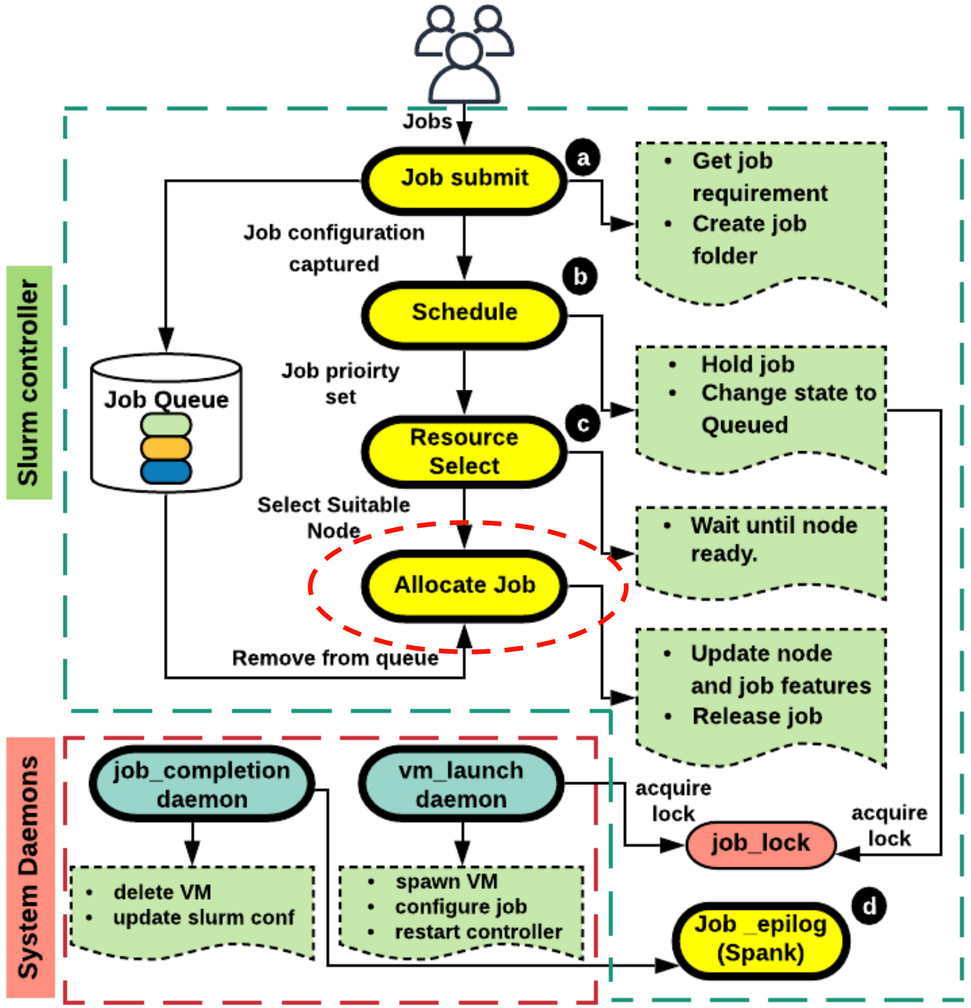
Implementation on Slurm



Resource selection plug-in

- Called after the scheduler plug-in
- Test resource availability and select resources
- Support various resource selection algorithms
  - cons_res, cray_aries, linear

Implementation on Slurm

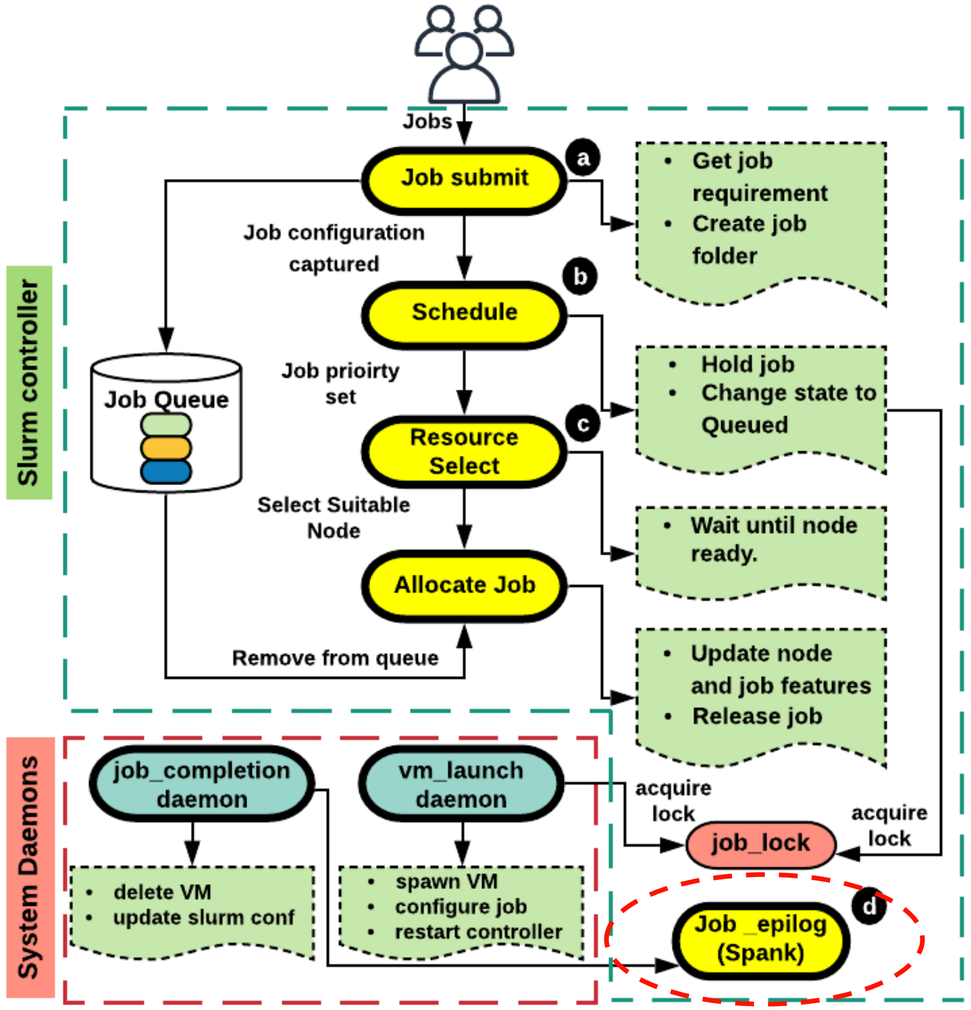


Allocate job plug-in

- Called after the resource select plug-in
- Allocate job to selected resources

- We use it to allocate jobs to VMs
- Change both job and node features so that Slurm can uniquely match them
- Release job from pending/hold state

Implementation on Slurm

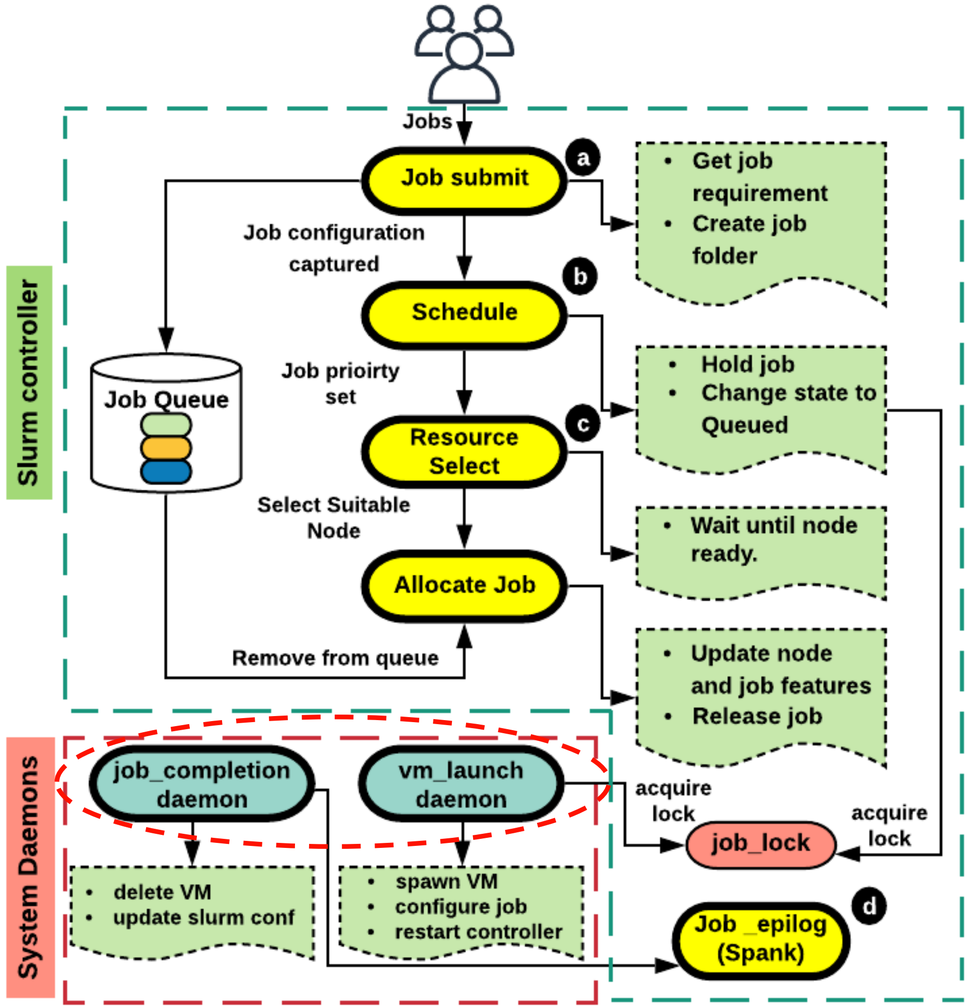


SPANK plug-ins

- Slurm Plug-in Architecture for Node and job (K)control
- Invoked in five “contexts”
- Dynamically modify job runtime behavior

- Mark VM as “down” to prevent future use
- Copy job output and error logs to master and login nodes
- Notify job_completion daemon of job completion

Implementation on Slurm



VM launch daemon

- Start VM cloning according to job config in a background process (non-blocking)
- Periodically check VM cloning progress
- Re-start or cancel VM cloning if failure
- Add new VMs to Slurm config file and restart Slurm controller

Job completion daemon

- Periodically check for completed jobs
- Remove VM from Slurm config file
- Delete job configuration details
- Delete VM

Admission Control & Load Balancing

- Implemented a python API to query vCenter for host utilization metrics
- Used an sqlite DataBase to store current system status in the Slurm controller
- If resource is not available for a job, it will be delayed and stored in a queue

- To avoid starvation, we ensure newly incoming jobs are queued behind delayed jobs
- For LoadBalancer, we implemented two simple policies:
  - First available
  - Random selection among compatible hosts

Experimentation & Evaluation

Experiment Setup

- *220 core* HPC cluster.
- *1TB* Memory
- *72TB* shared datastore

Workload

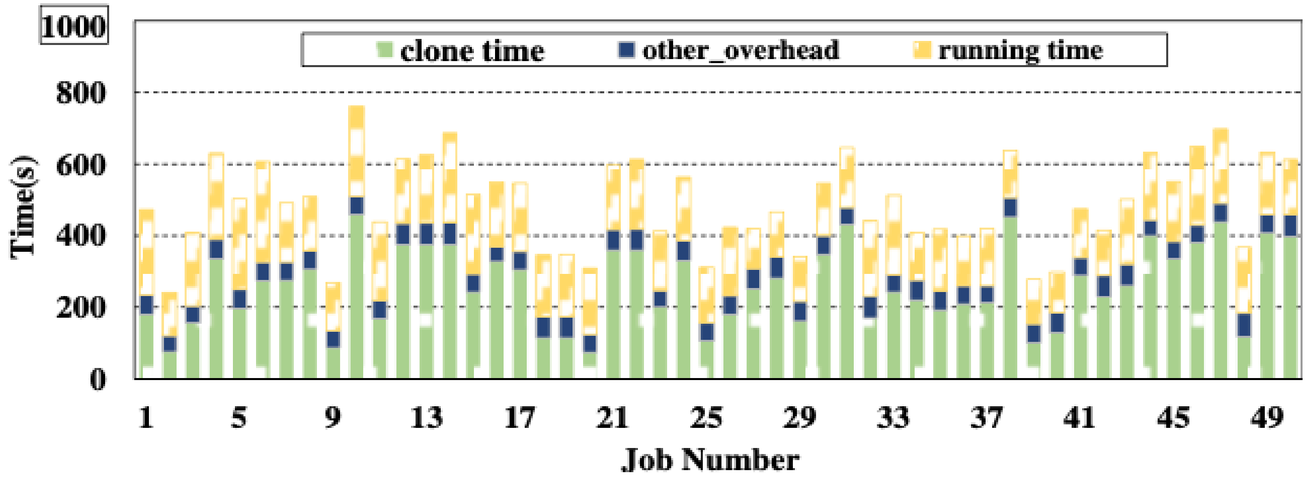
- *HPCC, HPL, RandomAccess*.
- *Small (2vCPU, 4GB), Large (8vCPU, 16GB)*
- *50 job/s, 100jobs/s*

Evaluation

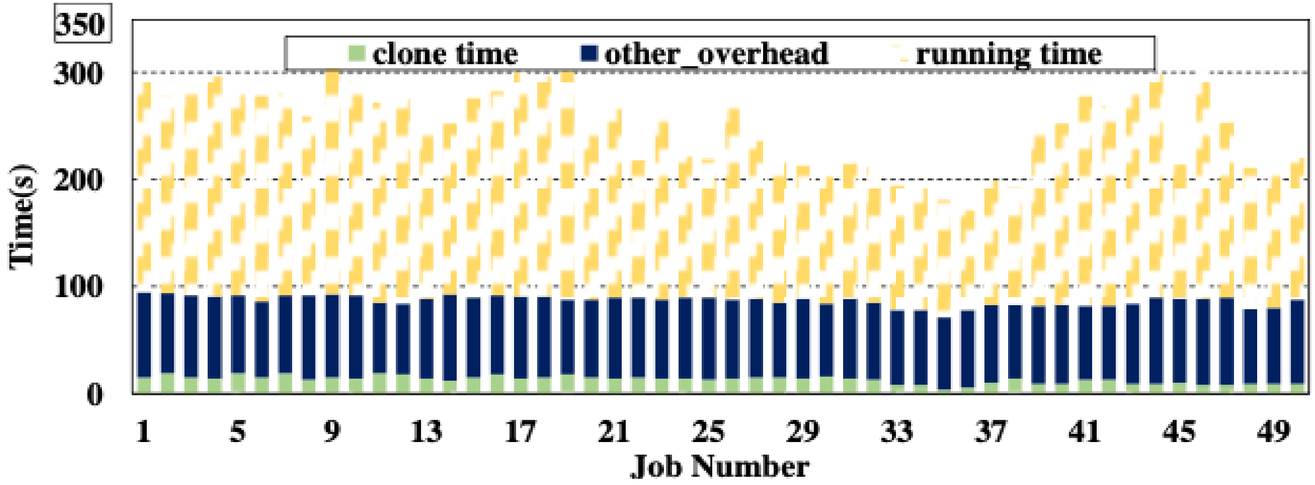
- Compare between full clone and instant clone
- Job run time
- Overhead, e.g., cloning time, VM config time, Slurm restarting time, etc

Results: Overall time breakdown

Full clone

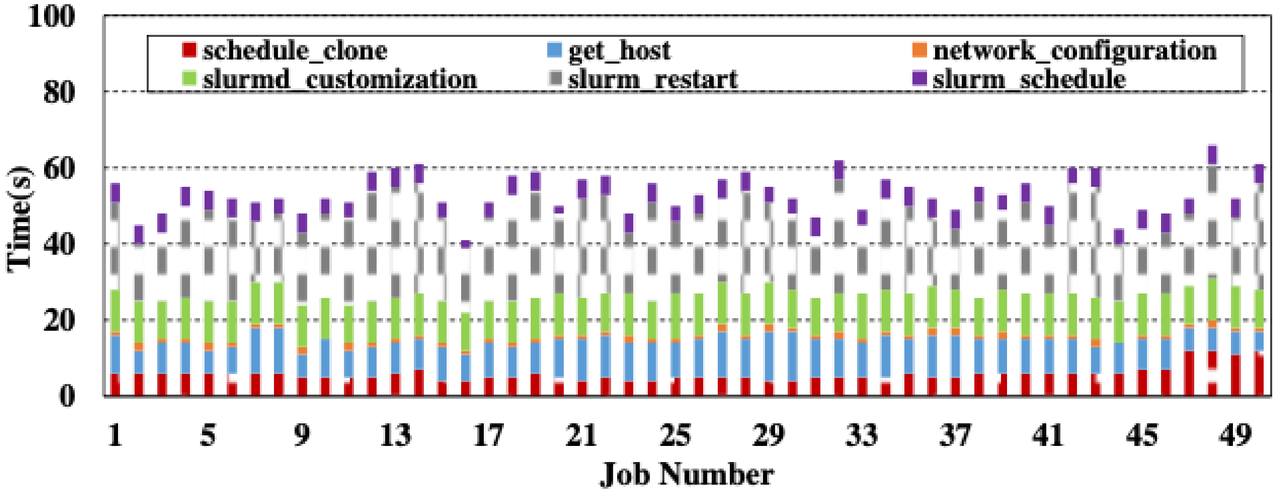


Instant clone

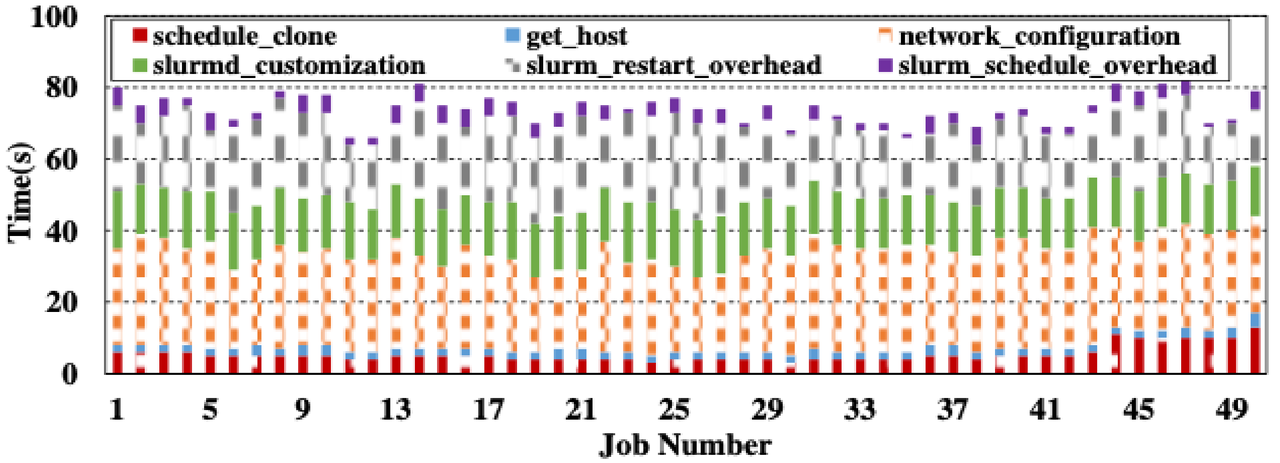


Results: Overheads breakdown

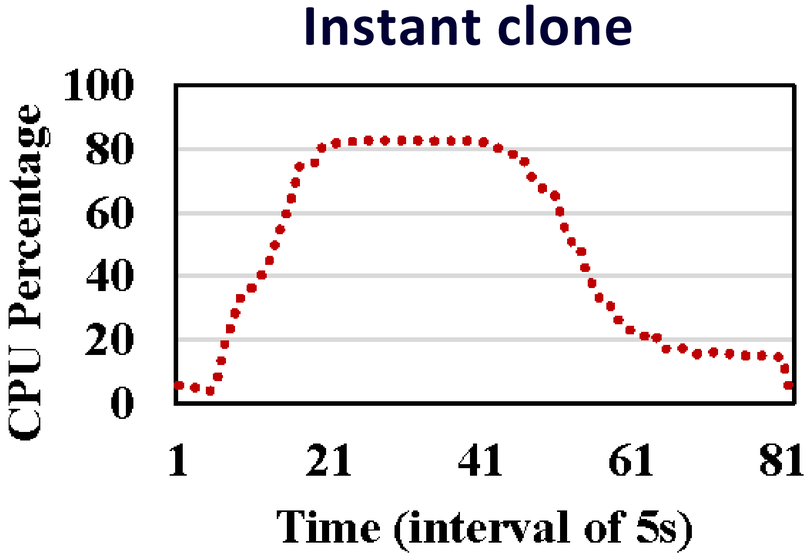
Full clone

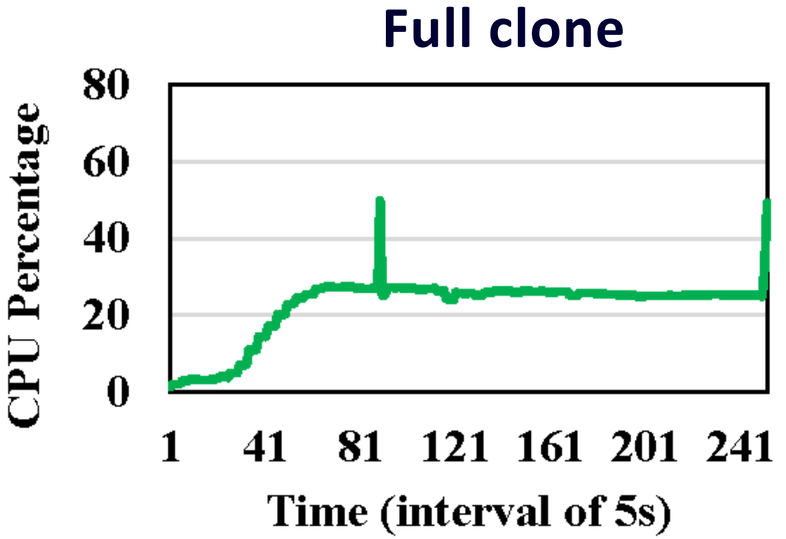


Instant clone



Results: Utilization and Throughput





- **1.5x** more throughput.
- **40%** higher CPU utilization.

Concluding Summary

- Design a generic VM-per-job model to integrate HPC scheduler with VM orchestrator.
- Develop policies to negotiate physical resources between scheduler and orchestrator.
- Expose system state to develop an admission control system and a dynamic load balancer.

Thank you !

…. Questions ?