Optimizing Diverse Workloads and System Resource Usage with Slurm

Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Daniel Burrill, Vijay Gadepally, Michael Houle, Matthew Hubbell, Hayden Jananthan, Michael Jones, Anna Klein, Peter Michaleas, Lauren Milechin, Julie Mullen, Guillermo Morales, Andrew Prout, Albert Reuther, Antonio Rosa, Siddharth Samsi, Charles Yee, Jeremy Kepner

Massachusetts Institute of Technology

MIT LINCOLN LABORATORY
SUPERCOMPUTING CENTER

Agenda

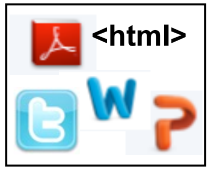
- Overview on LLSC
- Innovative allocation and scheduling technologies
  - LLSC developed tools
  - Slurm
- Performance
- Maximization of Resource Utilization
- Other Enhancements
- Summary

Lincoln Laboratory Supercomputing Center (LLSC) Role

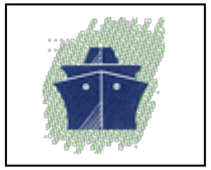
Mission Areas

- Air, Missile, and Maritime Defense Technology
- Homeland Protection/Bio
- Air Traffic Control
- Communication Systems
- Cyber Security
- Advanced Technology
- Space Systems and Technology
- ISR Systems and Technology
- Tactical Systems
- Engineering

Vast Data & Computation



OSINT


Weather

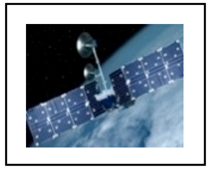

HUMINT


C2


Ground


Maritime


Air


Space


Cyber

LLSC develops & deploys unique, energy-efficient supercomputing that provides cross-mission

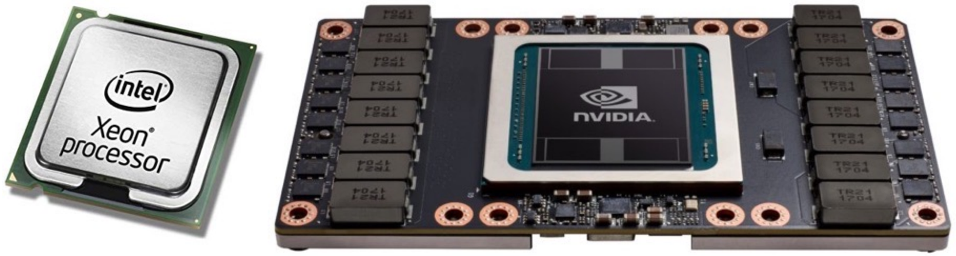
- Data centers, hardware, software, user support, and pioneering research
- 100x more productive than standard supercomputing[1]
- 100x more performance than standard cloud[2]

[1] *Interactive Grid Computing at Lincoln Laboratory*, Bliss et al., LL Journal, 2006
[2] *Scalability! But at what COST?*, McSherry et al., HotOS XV, 2015

TX-GAIA: Green AI Accelerator
- World Leader in Interactive AI Supercomputing -

Low Carbon Emission

- **Significant increase in computing power for simulation, data analysis, and machine learning**
- **Leverages power of 900 Nvidia Volta GPUs**

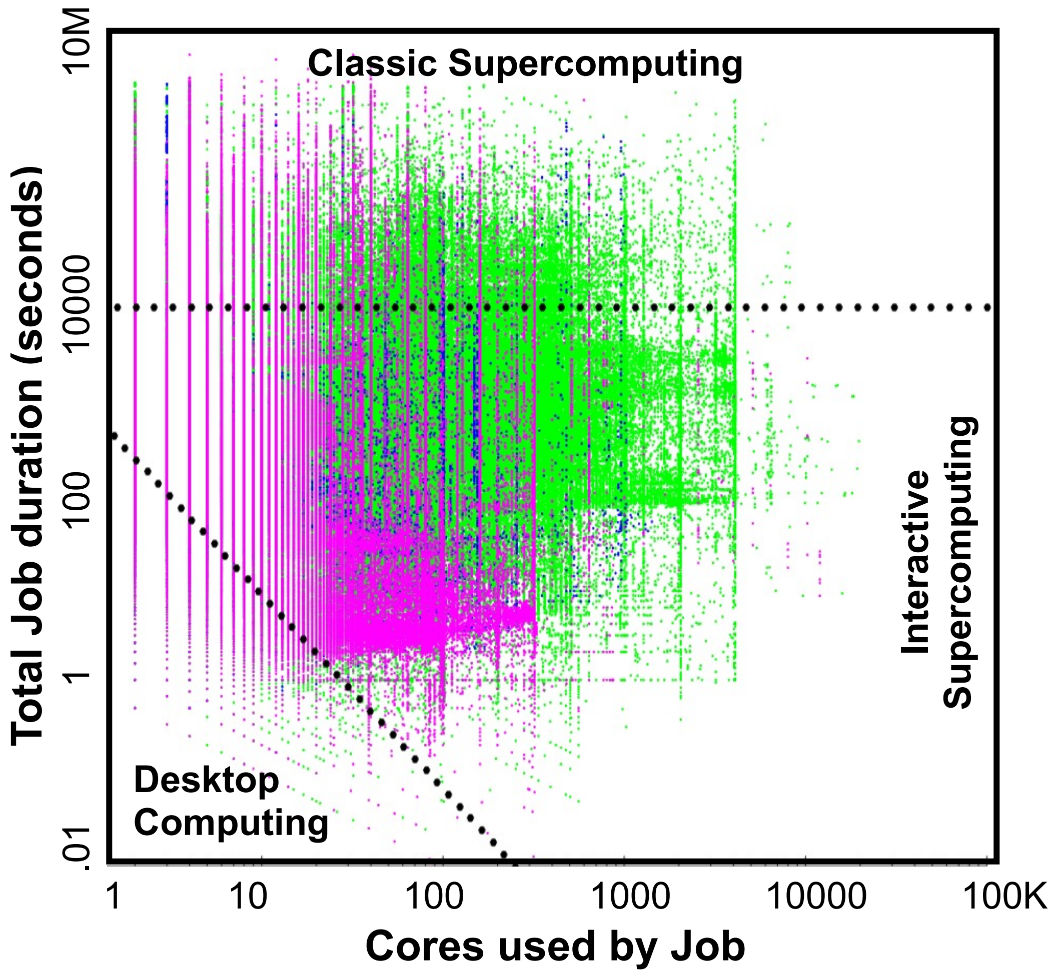


- **Largest AI Research System at a University**

	Capability
Processor	Intel Xeon & Nvidia Volta
Total Cores	737,000
Peak	7.4 Petaflops
Top500	5.2 Petaflops (#57 in World*)
Memory	172 Terabytes
Peak AI Flops	100+ Petaflops (#17 in World*)
Network Link	Intel OmniPath, 25 GB/s

*Based on 2020 Top500.org
AI Flops = 4x4 matrix multiply half precision in, single precision out (mixed precision training)

Interactive Supercomputing



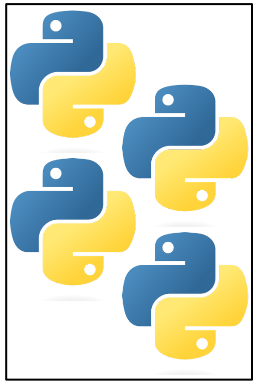
- **Desktop Computing**
  - **CPU-time <20 minutes**
- **Classic Supercomputing**
  - **Wall-clock time >3 hours**
- **Interactive Supercomputing**
  - **Between desktop and classic supercomputing**
  - **Shortens the "time to insight"**
  - **Ten development turns/day instead of one turn/week**

Large and Growing Need for Supercomputing

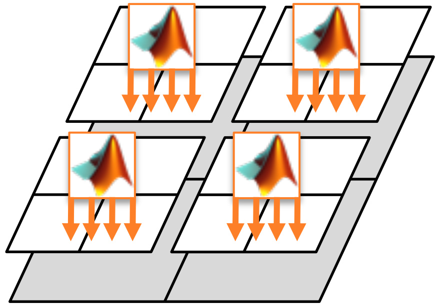
- requires world-class research to optimize diverse applications on complex hardware -

- **Challenge**: maximizing the efficiency of thousands of users distinct software
- **Approach**: LLSC developed innovative programming, *allocation*, and *scheduling* technologies

pPython Parallel Python

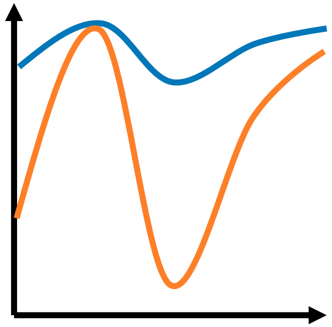


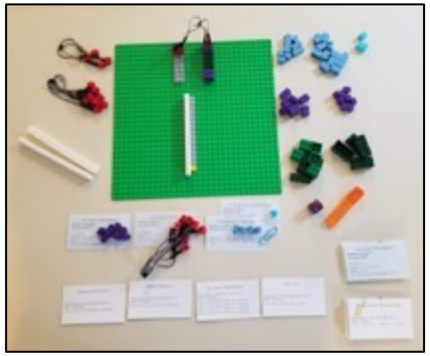
Hierarchical Allocation



3D Monitoring & Management

Spot Scheduling





HPC Training via Serious Games

Challenges with Shared Systems

- Challenges
  - Unintentional usages
    - Users often do not know whether their applications are aggressively multi-threaded or not.
    - Users often do not know how much memory their jobs require.
  - Unexpected failures
    - Node failed due to an out-of-memory error by other jobs on the same node
    - Take time to identify who caused OOM error

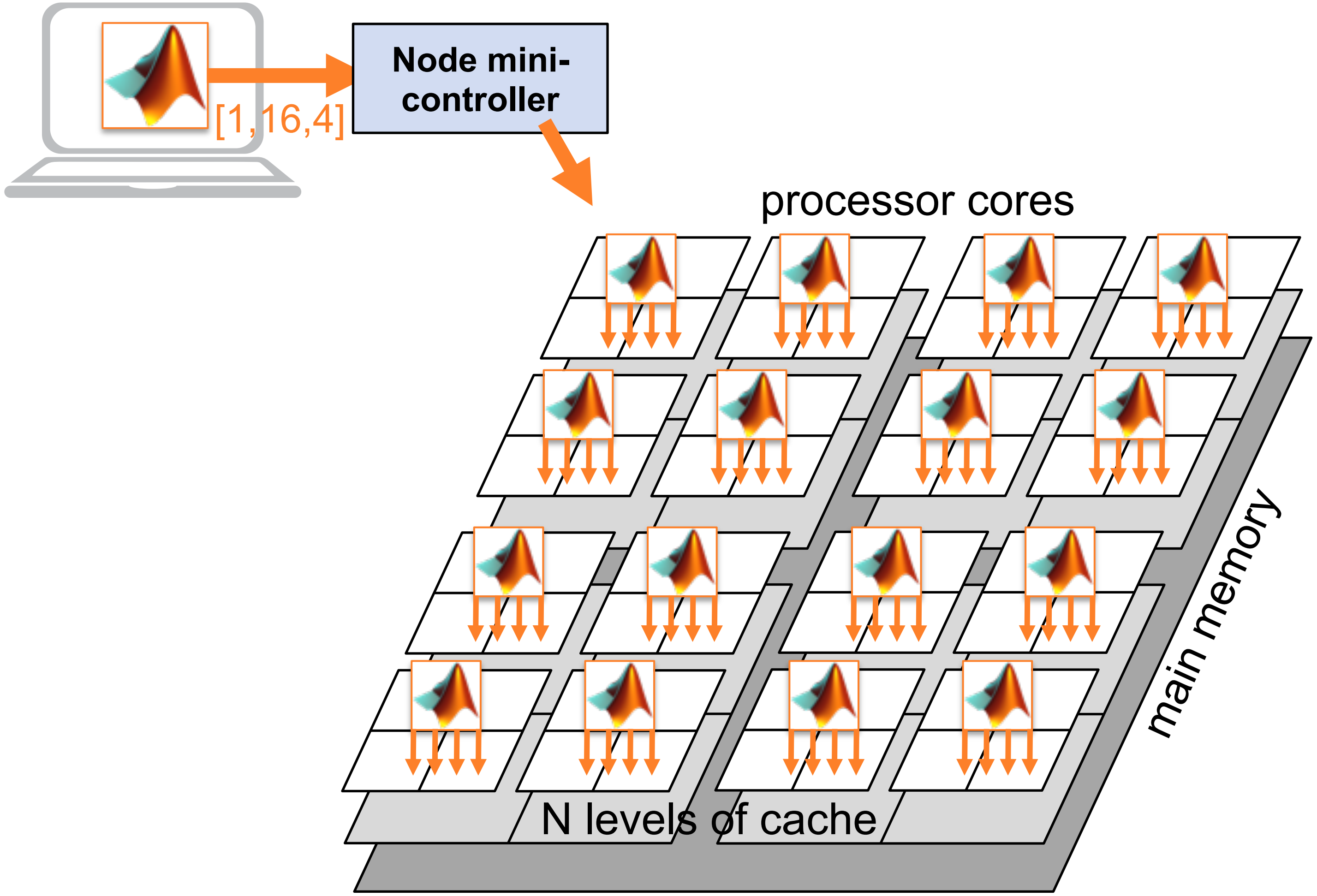
- Mitigation
  - Exclusive runtime environment on each node
  - Limit the impact by unintentional usage and/or failures to a single user
  - Innovative allocation with LLSC developed tools: LLsub, LLMapReduce, pMatlab and pPython
  - Slurm: ExclusiveUser=YES at partition level

Innovative *Allocation* Technologies

- **Traditional allocation**
  - Cores are slots to be filled with jobs
  - 10,000 cores = 10,000 jobs to be allocated
    - No ability to optimize for memory/core architecture

- **Hierarchical allocation**
  - User specifies: nodes, cores, and threads
  - Allocator divides up jobs among nodes
  - Dynamically writes <u>hardware aware</u> mini-controller on each node to independently start, monitor, and stop user processes on node

User requests: 1 node with 16 processes with 4 threads per process



Node-Based Job Scheduling for Large Scale Simulations of Short Running Jobs, Byun et al., IEEE HPEC 2021

Hierarchical Allocation Performance Benchmark

Workloads Configuration

Configuration	Rapid Tasks	Fast Tasks	Medium Tasks	Long Tasks
Task time, t	1s	5s	30s	60s
Job time per processor, Tjob	240s	240s	240s	240s
Tasks per processor, n	240	48	8	4

Reference: Scalable system scheduling for HPC and big data, https://www.sciencedirect.com/science/article/pii/S0743731517301983

C. Byun et al., "Node-Based Job Scheduling for Large Scale Simulations of Short Running Jobs," 2021 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2021,

Benchmark Size Configuration

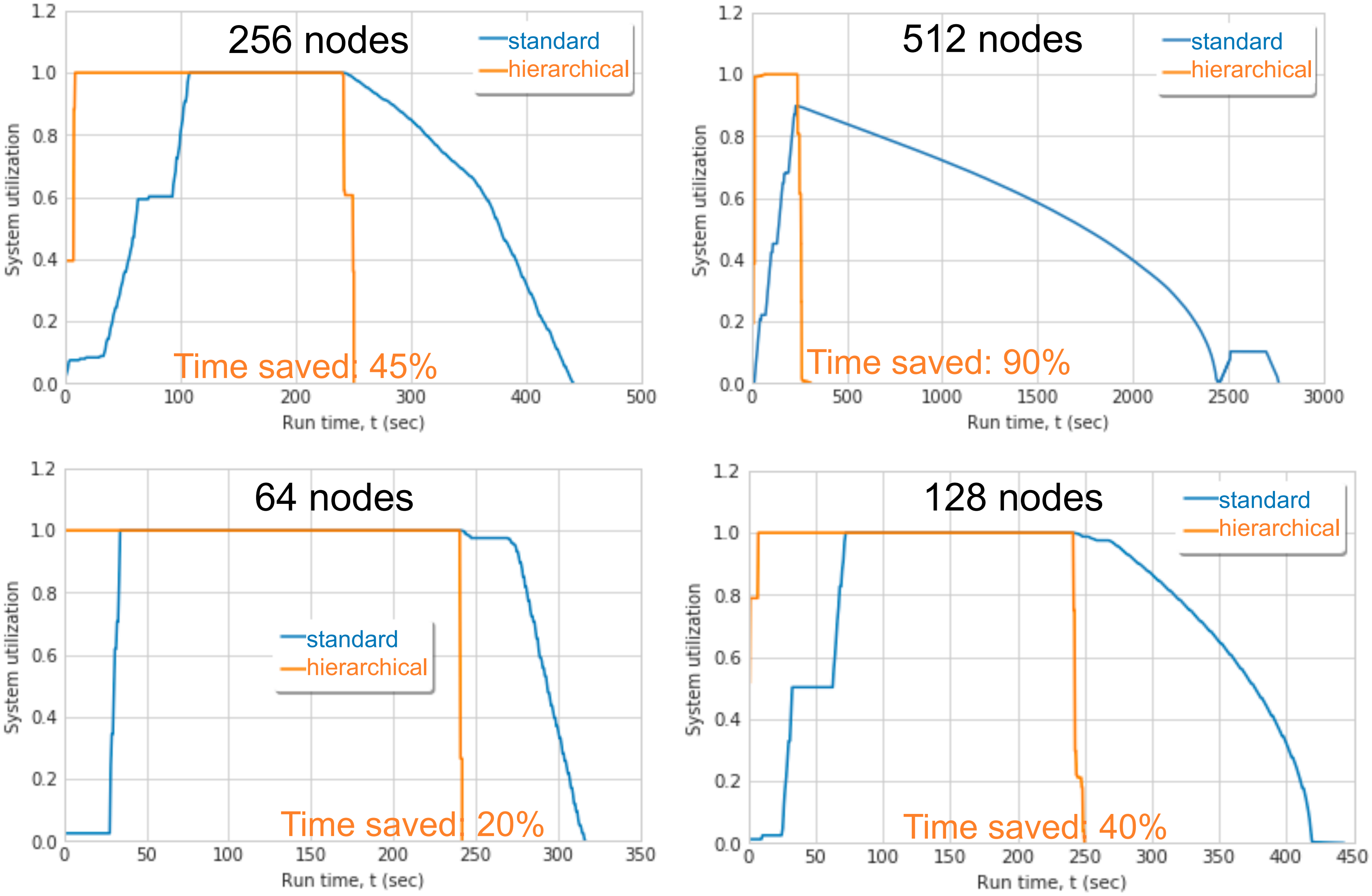
Nodes	32	64	128	256	512
Cores per node	64	64	64	64	64
Processors, P (cores)	2048	4096	8192	16384	32768

- **Multi-level scheduling (LLMapReduce MIMO)* creates an array job of P scheduler tasks**
- **Hierarchical allocation (LLMapReduce MIMO with the triples mode) creates an array job of scheduler tasks equivalent to the number of nodes**
- **Total number of compute tasks (N) varies to maintain the total job time per process (Tjob) constant**

*** C. Byun et al., "LLMapReduce: Multi-level map-reduce for high performance data analysis," 2016 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2016**

Hierarchical Allocation Performance

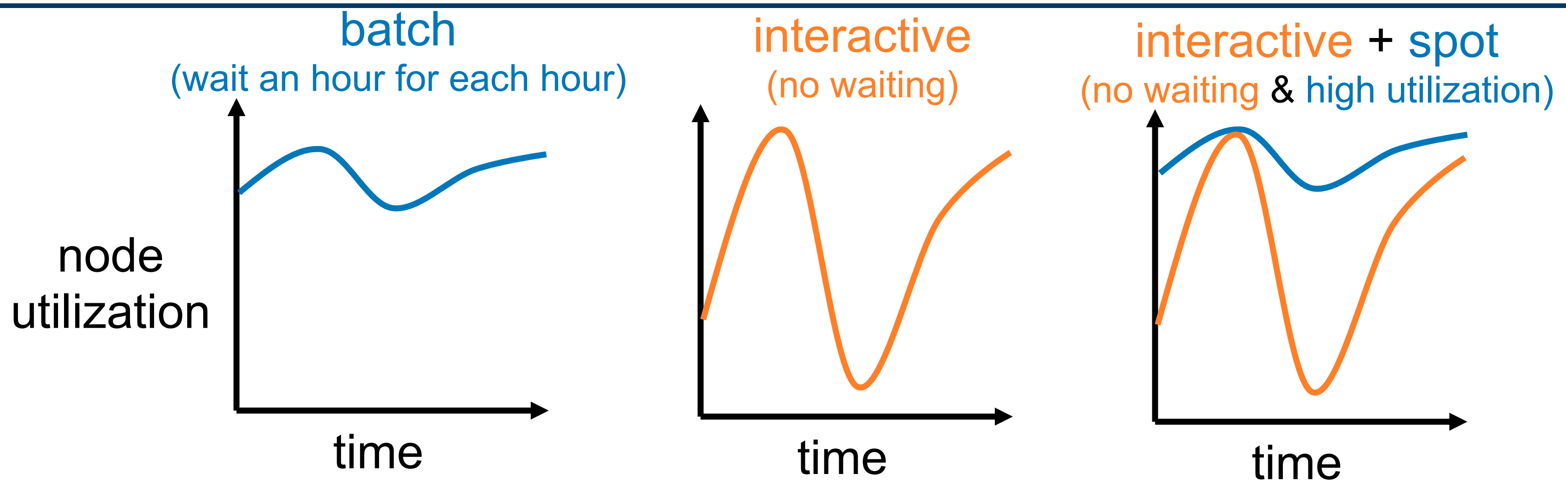
- **Dramatically accelerates large interactive jobs**
- **Transitioned to default LLSC approach**



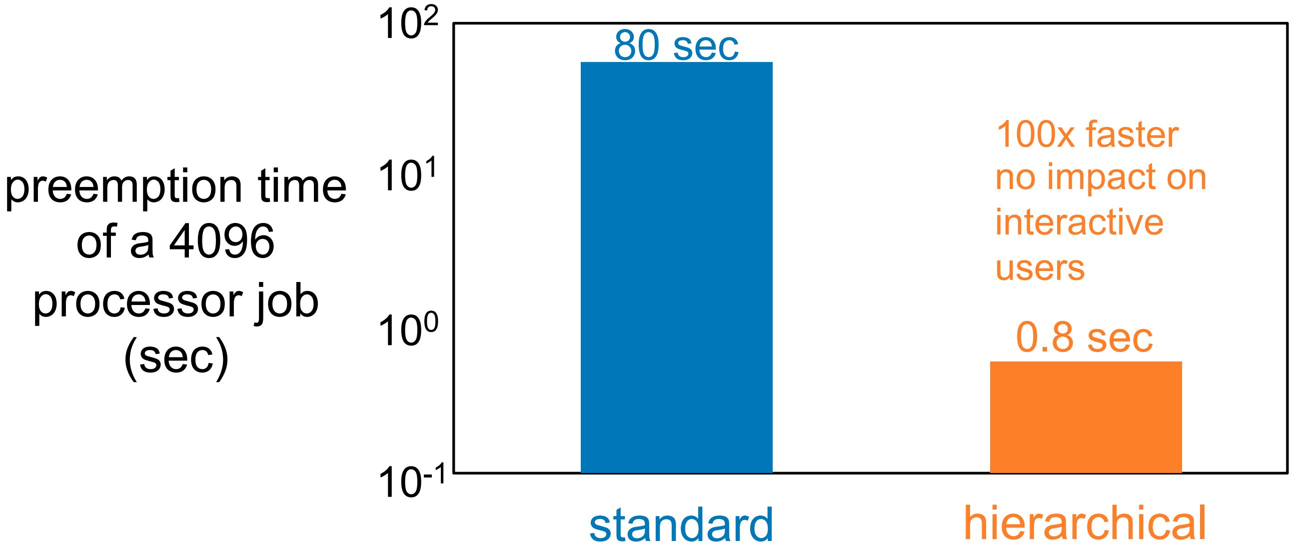
Node-Based Job Scheduling for Large Scale Simulations of Short Running Jobs, Byun et al., IEEE HPEC 2021

Innovative *Scheduling*

- **Traditional dilemma**
  - **Batch: 90% of nodes used, but user waits an hour per hour of run time**
  - **Interactive: launches immediately, but 50% of nodes used**
  - **Same wall-clock performance but interactive is preferred and more efficient**



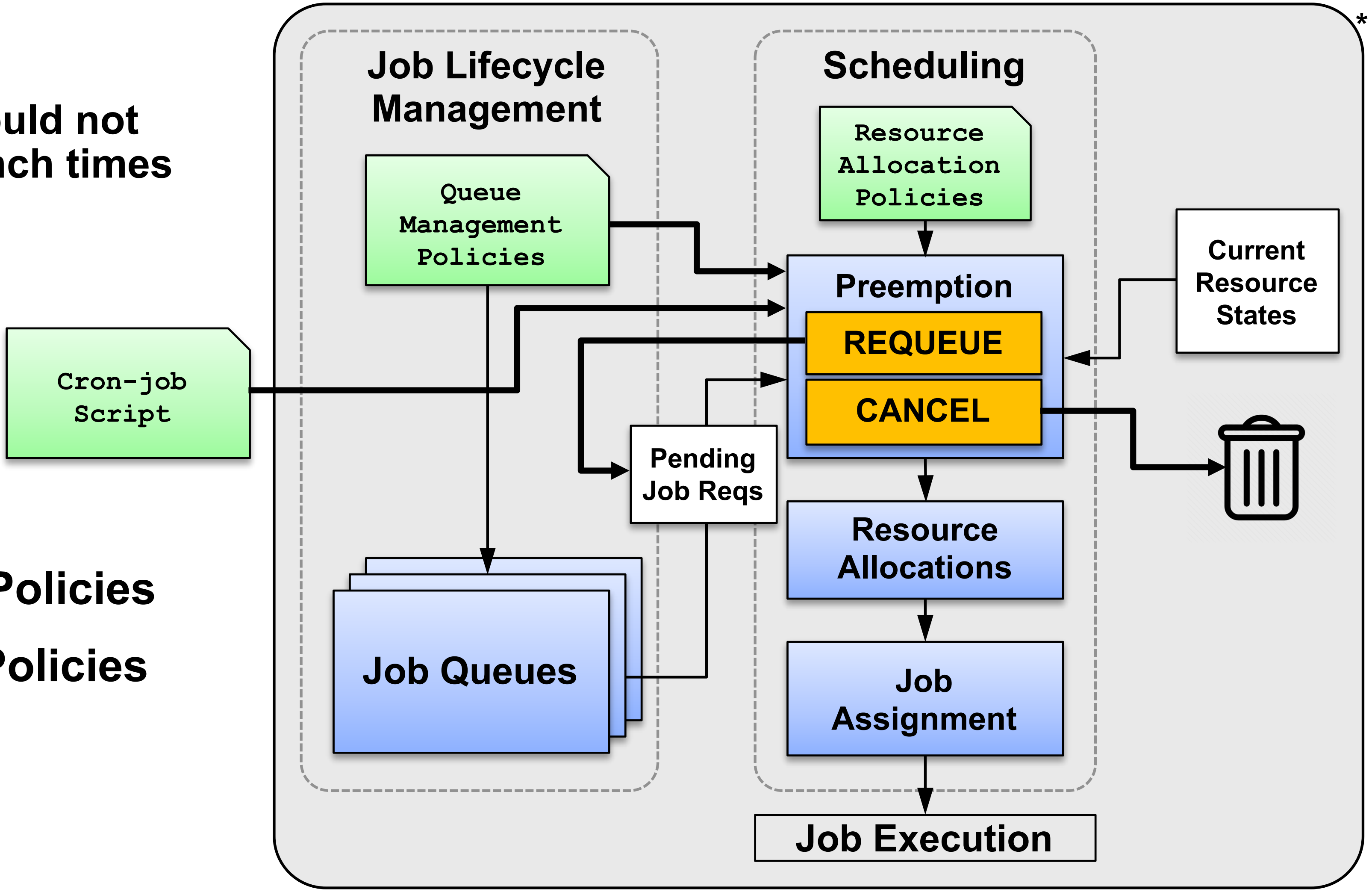
- **Best of both worlds**
  - **Spot jobs that can be instantly preempted by interactive jobs**
  - **Achievable with Hierarchical Allocation fast preemption**



Best of Both Worlds: High Performance Interactive and Batch Launching, Byun et al., IEEE HPEC 2020

Various Approaches of Spot Job Implementation

- **Goal**
  - Low priority spot jobs would not adversely impact the launch times of the normal jobs
- **Via Resource Allocation Policies**
- **Vis Queue Management Policies**
- **Customized Script**
  - Cron-job script



* Figure adapted from "Interactive Supercomputing on 40,000 Cores for Machine Learning and Data Analysis", HPEC 2018

Summary of Experiments

Preemption Approaches	Preemption Mode	Partitions	Job Types	Job Sizes
Automatic by scheduler	REQUEUE CANCEL	Single, Dual	Individual, Array, Triple-mode	Small, Medium, Large
Lua job submission script	REQUEUE	Dual	N/A	N/A
Manual	REQUEUE	Dual	Individual, Array, Triple-mode	Large
Cron-job script	REQUEUE	Dual	Individual, Array, Triple-mode	Large

Monitoring Resource Utilization

- **System usage at a glance**
  - **LLload displays summary of all jobs running on the system**

```
$ LLload -g --all
Cluster name: txgreen
Jupyter notebook jobs:
----------------------
    NodeName  Users(GPU)
    ---------------------
[J]c-12-15-2: sa12345
[J]c-15-13-3: br67890
[J]c-15-14-4: ma12345
  [J]c-4-1-1: tr67890
  [J]c-5-9-4: ca12345
 [J]c-6-15-4: ma67890

Node information for each user:
-------------------------------
Username: ab12345, Nodes used: 5
    HOSTNAME CORES -  USED =  FREE    LOAD MEMORY -   USED =   FREE GPUS -  USED = FREE LOAD GPUMEM -  USED =  FREE
   c-14-10-2    40 -    40 =     0    0.00  384GB -   14GB =  370GB    2 -     2 =    0 0.00   64GB -   0GB =  63GB
   c-14-11-1    40 -    40 =     0    0.00  384GB -   13GB =  371GB    2 -     2 =    0 0.00   64GB -   0GB =  63GB
   c-14-12-1    40 -    40 =     0    0.03  384GB -   14GB =  370GB    2 -     2 =    0 0.00   64GB -   0GB =  63GB
   c-14-12-2    40 -    40 =     0    0.01  384GB -   23GB =  361GB    2 -     2 =    0 0.00   64GB -   8GB =  56GB
   c-14-13-1    40 -    40 =     0    1.08  384GB -   20GB =  364GB    2 -     2 =    0 0.11   64GB -   4GB =  59GB
```

- **Challenge to obtain GPU usage on each node**
- **Current implementation uses a ssh remote execution of “nvidia-smi command” on each GPU node**
- **Ideal if Slurm provides GPU usage information.**

User Feedback and Coaching

- **Before**

```
$ LLload -g -u WI12345
Cluster name: txgreen
Username: wi12345, Nodes used: 5
    HOSTNAME CORES -  USED =  FREE    LOAD MEMORY -   USED =   FREE  GPUS -  USED = FREE LOAD GPUMEM -  USED =  FREE
    c-8-16-2    40 -    40 =     0    1.95  384GB -  261GB =  123GB     2 -     2 =    0 0.37   63GB -   3GB =  60GB
   c-13-14-1    40 -    40 =     0    2.33  384GB -   51GB =  333GB     2 -     2 =    0 0.30   63GB -   3GB =  61GB
    c-14-2-1    40 -    40 =     0    2.31  384GB -   51GB =  333GB     2 -     2 =    0 0.37   63GB -   3GB =  61GB
     c-8-7-1    40 -    40 =     0    2.38  384GB -  138GB =  246GB     2 -     2 =    0 0.37   63GB -   3GB =  60GB
    c-7-15-1    40 -    40 =     0    2.56  384GB -   44GB =  340GB     2 -     2 =    0 0.36   63GB -   3GB =  60GB
```

- **After**

```
$ LLload -g -u WI12345
Cluster name: txgreen
Username: wi12345, Nodes used: 3
    HOSTNAME CORES -  USED =  FREE    LOAD MEMORY -   USED =   FREE  GPUS -  USED = FREE LOAD GPUMEM -  USED =  FREE
     c-8-6-1    40 -    20 =    20    1.92  384GB -   47GB =  337GB     2 -     1 =    1 0.24   63GB -   1GB =  62GB
    c-12-3-1    40 -    40 =     0    4.78  384GB -   72GB =  312GB     2 -     2 =    0 0.52   63GB -   2GB =  61GB
    c-8-12-2    40 -    40 =     0    4.53  384GB -   49GB =  335GB     2 -     2 =    0 0.48   63GB -   2GB =  61GB
```

Other Enhancements

- Securing Slurm environment
- Jupyter Notebook Portal Service
- Limiting number of interactive jobs per user

Securing Slurm environment

- **Cluster-level privacy enforcement** `PrivateData=jobs,reservations,usage,users`
  - **Controls what type of information is hidden from regular users.**
  - **Users can see only their own jobs, reservations, and usage**
- **Difficult to find system resource availability**
- **LLfree provides summary of system usage**
  - **A cron job by a privileged user generate the system resource usage in every 15 seconds**
  - **The system resource usage is saved in a file visible by all users**
  - **LLfree read the file and display as shown here**
  - **Side benefits: reduce the scheduler loads from users who issue Slurm commands too frequently to check resource availability**

```
$ LLfree
LLGrid: TXGREEN (running slurm-wlm 23.02.3)
==========================================================================
Partition      | Type                 | Cores       | Nodes       | GPUs
--------------------------------------------------------------------------
normal         | Xeon-e5              | 6272        | 224         | N/A
--------------------------------------------------------------------------
xeon-p8        | Xeon-p8              | 13392       | 279         | N/A
--------------------------------------------------------------------------
debug-cpu      | Xeon-p8              | 528         | 11          | N/A
--------------------------------------------------------------------------
manycore       | Xeon64c              | 37888       | 592         | N/A
--------------------------------------------------------------------------
gaia           | Xeon-g6:Volta        | 7880        | 197         | 394
--------------------------------------------------------------------------
debug-gpu      | Xeon-g6:Volta        | 360         | 9           | 18
--------------------------------------------------------------------------
```

Jupyter Notebook Portal Service

MIT LINCOLN LABORATORY
SUPERCOMPUTING CENTER

Menu: Return to the grid portal home page

Job#	Status	Link	Action
No Jupyter Notebook currently running.			

☐ Show job comments

Hide Advanced Launch Options

Partition: Jupyter CPU
CPU Type: Intel Xeon-P8
CPU Count: 4
GPU Resource Type:
GPU Resource Count:
Exclusive: ☐
Time limit:
Anaconda/Python Version: Anaconda 2022a - Python 3.8
Application: Jupyter Notebook
Reservation:

Launch Notebook

LLfree: SuperCloud (TX-Green)

Partition	Type	Cores	Nodes	GPUs
xeon-p8	Xeon-p8	13392	279	N/A
debug-cpu	Xeon-p8	528	11	N/A
gaia	Xeon-g6:Volta	7880	197	394
debug-gpu	Xeon-g6:Volta	360	9	18
normal	Xeon-e5	6356	227	N/A
manycore	Xeon64c	37888	592	N/A

2023-08-01 06:23:05 Auto refresh is: Off

Limiting Number of Interactive Jobs

- **Some users create a large number of interactive jobs, which result in wasting computing resources.**
- **Recipe:**
  - **Create GRES complex, ijob, and keep track of the resource usage**

    ```
    GresTypes=ijob
    AccountingStorageTRES=ijob
    ```
  - **Assign ijob=1 to all interactive jobs**
    - **Enforce by Lua job submission script**
  - **Attach "high" QoS to interactive jobs to increase their scheduling priority**
    - **"high" QoS limits the number of ijob instances to the desired number (decision by the LLSC team)**

      ```
      MaxTRESPU = gres/ijob=2
      ```
    - **Association-based enforcement to impose on job submission**

      ```
      AccountingStorageEnforce=limits,qos
      ```

MIT SuperCloud World Leading Interactive Performance

Enterprise

16,000+ MS Windows instances launched in 5 minutes[1]

Supercomputing

32,000+ AI processes launched in 2 seconds[2]

MIT SuperCloud

250,000+ analysis environments launched in 40 seconds[2]

Big Data

100,000,000+ database inserts/second[3]

Database

= world record

- **100x more productive than standard supercomputing[4]**
- **100x more performance than standard cloud[5]**
- **100% GOTS; 20x GovCloud performance/price[6]**

[1]*Interactive Launch of 16,000 Microsoft Windows*, HPEC 2018; [2]*Interactive Supercomputing on 40,000 Cores for Machine Learning and Data Analysis*, HPEC 2018; [3]*Achieving 100,000,000 database inserts per second*, HPEC 2014; [4]*Interactive Grid Computing at Lincoln Laboratory*, LL Journal, 2006; [5]*Scalability! But at what COST?*, McSherry et al., HotOS XV 2015; [6]*Scalable system scheduling for HPC and big data*, JPDC 2018

Summary

- LLSC is well positioned to server diverse work loads for large scale jobs with Slurm
  - Innovative allocation, and scheduling technologies
- LLSC has demonstrated a number of world-leading interactive performance
- LLSC is now able to monitor users' jobs better by separating users' jobs by node
  - This provides opportunities to improve system efficiency
  - Currently user feedback is based on manual process to monitor users' jobs using LLSC-developed tool, LLload
  - Future work is to automate some of these manual processes such as identifying inefficient jobs
- We are looking for a better way to collect GPU usage, possibly by Slurm, which we can make our tool work more efficiently
  - GPU loads averages in 1 min, 5 min, and 15 min?

Questions?

cbyun@ll.mit.edu

MIT LINCOLN LABORATORY
SUPERCOMPUTING CENTER