MATRIX and Distributed Job Launch Overview

Ke Wang
Data-Intensive Laboratory
Illinois Institute of Technology
Sept. 6^{th}, 2013, CS554 Teaching

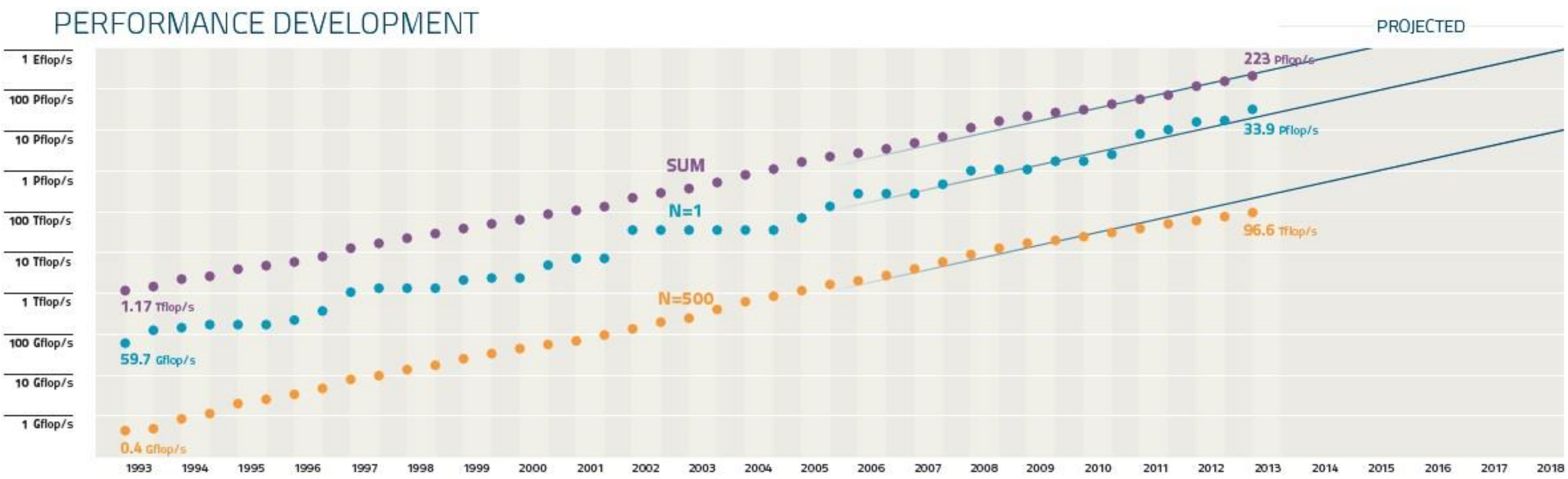
Outline

- Introduction & Motivation
- Related Work
- MATRIX
- Distributed Job Launch (DJL)
- Project Overview

Outline

- **Introduction & Motivation**
- Related Work
- MATRIX
- Distributed Job Launch (DJL)
- Project Overview

Exascale Computing



- **Today (June, 2013): 34 Petaflop (10^15 ops/sec)**
  - **O(100K) nodes**
  - **O(1M) cores**
- **Near future (~2022): Exaflop Computing (10^18 ops/sec)**
  - **~1M nodes**
  - **~1B processor-cores/threads**

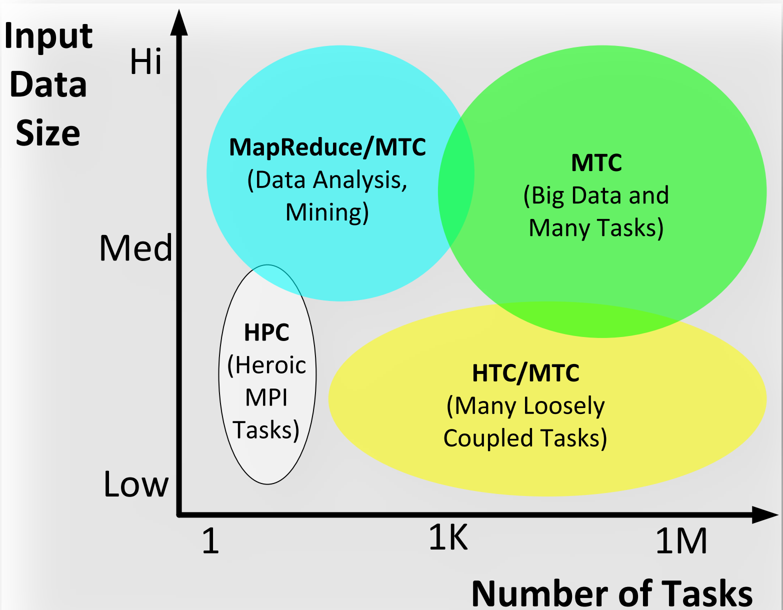
Top500 Performance Development,

http://s.top500.org/static/lists/2013/06/TOP500_201306_Poster.pdf

Major Challenges of Exascale Computing

- **Energy and Power**
  - 17.8MW (Top 1 Supercomputer)
  - 20MW limitation
- **Memory and Storage**
  - Retain data at high enough capacities
  - Access data at high enough rates
  - Support the desired computational rate
  - Fit within acceptable power envelope
- **Concurrency and Locality**
  - Accelerators, GPUs, MIC
  - Programmability
  - Minimizing data movement
- **Resiliency**
  - MTTF decreases, MPI suffers

Many-Task Computing (MTC)



Field	Description	Characteristics	Status
Astronomy	Creation of montages from many digital images	Many 1-core tasks, much communication, complex dependencies	Experimental
Astronomy	Stacking of cutouts from digital sky surveys	Many 1-core tasks, much communication	Experimental
Biochemistry*	Analysis of mass-spectrometer data for post-translational protein modifications	10,000-100 million jobs for proteomic searches using custom serial codes	In development
Biochemistry*	Protein structure prediction using iterative fixing algorithm; exploring other biomolecular interactions	Hundreds to thousands of 1- to 1,000-core simulations and data analysis	Operational
Biochemistry*	Identification of drug targets via computational docking/screening	Up to 1 million 1-core docking operations	Operational
Bioinformatics*	Metagenome modeling	Thousands of 1-core integer programming problems	In development
Business economics	Mining of large text corpora to study media bias	Analysis and comparison of over 70 million text files of news articles	In development
Climate science	Ensemble climate model runs and analysis of output data	Tens to hundreds of 100- to 1,000-core simulations	Experimental
Economics*	Generation of response surfaces for various economic models	1,000 to 1 million 1-core runs (10,000 typical), then data analysis	Operational
Neuroscience*	Analysis of functional MRI datasets	Comparison of images; connectivity analysis with structural equation modeling, 100,000+ tasks	Operational
Radiology	Training of computer-aided diagnosis algorithms	Comparison of images; many tasks, much communication	In development
Radiology	Image processing and brain mapping for neuro-surgical planning research	Execution of MPI application in parallel	In development

Note: Asterisks indicate applications being run on Argonne National Laboratory's Blue Gene/P (Intrepid) and/or the TeraGrid Sun Constellation at the University of Texas at Austin (Ranger).

- Bridge the gap between HPC and HTC
- Applications structured as DAGs
- Data dependencies will be files that are written to and read from a file system
- Loosely coupled apps with HPC orientations

- Falkon
  - Fast and Lightweight Task Execution Framework
  - http://datasys.cs.iit.edu/projects/Falkon/index.html
- Swift
  - Parallel Programming System
  - http://www.ci.uchicago.edu/swift/index.php

Job Management/Scheduling Systems

- Current
  - Centralized design with Master/Slaves architecture
  - Scalability issues at petascale and beyond
  - Single-point-of-failure
- Need to be
  - Fully distributed architecture with high concurrency
  - High throughput and system utilization
  - Reliability
- Problem
  - Distributed Load Balancing

Work Stealing

- A distributed load balancing technique
- Triggered due to uneven distribution of load (tasks in a workload) or presence of idle nodes in the system
- Idle node tries to steal tasks from busy nodes
  - Where to steal tasks?
  - How much tasks to steal?
  - How often to steal tasks?

Outline

- Introduction & Motivation
- **Related Work**
- MATRIX
- Distributed Job Launch (DJL)
- Project Overview

Related Work

- HPC resource manager
  - SLURM: LLNL
  - Condor: UW-Madison
  - SGE: Sun Microsystems
  - PBS: OpenPBS in NASA, TORQUE in Adaptive Computing Enterprises, and PBS Pro in Altair Engineering
  - Cobalt: ANL
- MTC task execution framework
  - Falkon: UChicago and ANL
  - Turbine: Apache
  - Sparrow: UC Berkeley
  - Charm++: UIUC

Outline

- Introduction & Motivation
- Related Work
- **MATRIX**
- Distributed Job Launch (DJL)
- Project Overview

MATRIX

- MAny-Task computing execution fabRIc at eXascale
- Dynamic job scheduling system at the granularity of node/core levels for extreme scale applications
- Work stealing is applied to achieve distributed load balancing
- Support of various workloads: HPC jobs, and MTC task with/without dependency

Adaptive Work Stealing

ALGORITHM 1. Dynamic Multi-Random Neighbor Selection (DYN-MUL-SEL)

Input: Node id (*node_id*), number of neighbors (*num_neigh*), and number of nodes (*num_node*), and the node array (*nodes*).
Output: A collection of neighbors (*neigh*).

```
selected[num_node];
for each i in 0 to num_node do
    if (i != node_id) then
        selected[i] = FALSE;
    else
        selected[i] = TRUE;
    end
end
neigh[num_neigh];
index = −1;
for each i in 0 to num_neigh−1 do
    repeat
        index = Random( ) % num_node;
    until !selected[index];
    selected[index] = TRUE;
    neigh[i] = nodes[index];
end
return neigh;
```

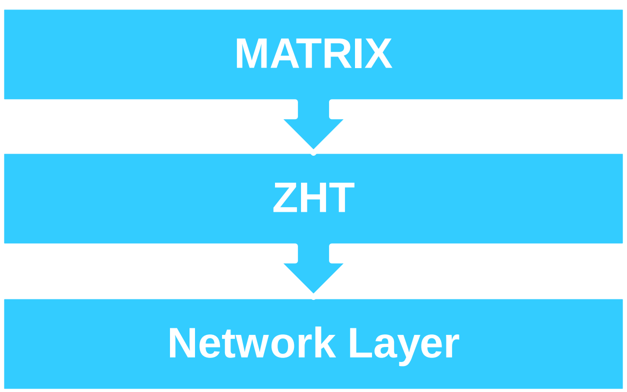
Work Stealing (continued)

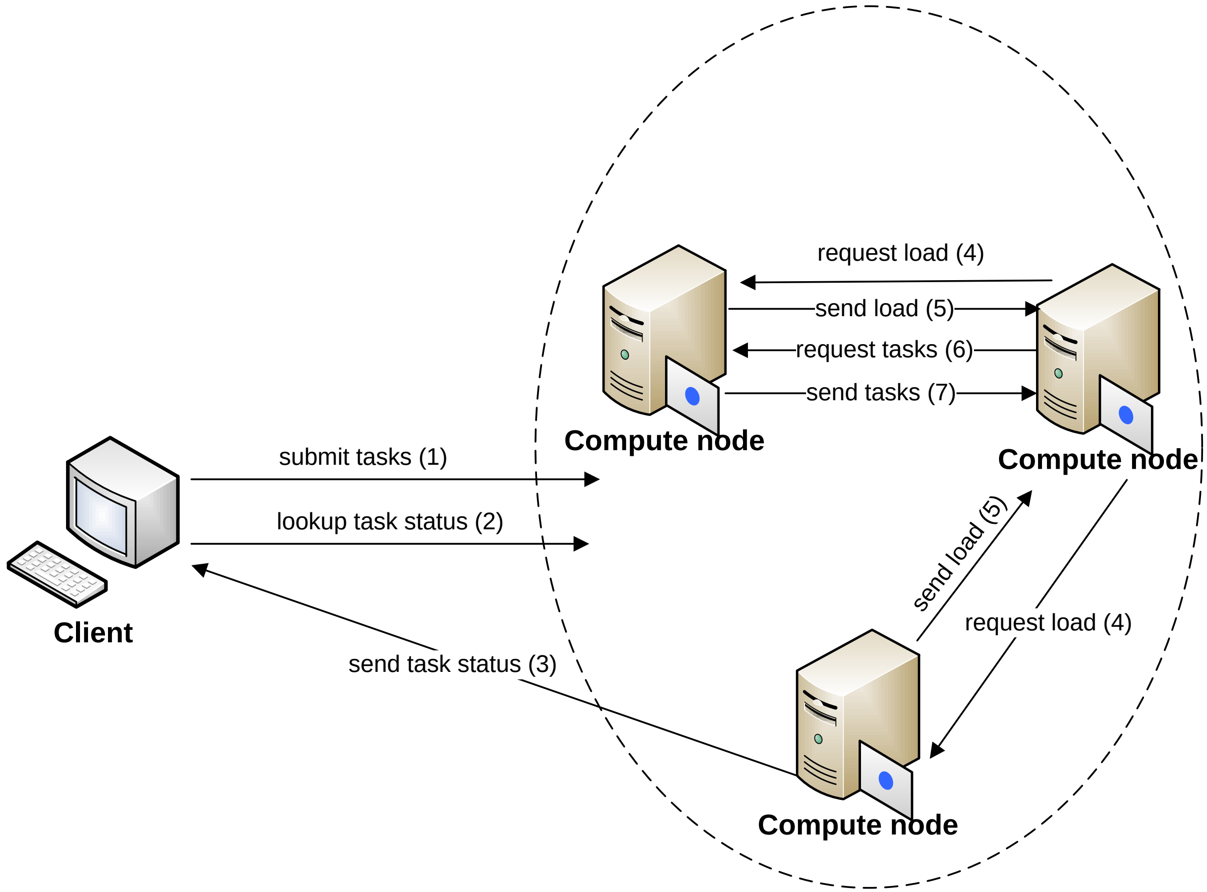
ALGORITHM 2. Adaptive Work Stealing Algorithm (ADA-WORK-STEALING)

Input: Node id (*node_id*), number of neighbors (*num_neigh*), number of nodes (*num_node*), the node array (*nodes*), and the initial poll interval (*poll_interval*).
Output: NULL

```
neigh = DYN-MUL-SEL(node_id, num_neigh, num_node, nodes);
most_load_node = neigh[0];
for each i in 1 to num_node−1 do
    if (most_load_node < neigh[i].load) then
        most_load_node = neigh[i];
    end
end
if (most_load_node.load == 0) then
    Sleep(poll_interval);
    poll_interval = poll_interval × 2;
    ADA-WORK-STEALING(node_id, num_neigh, num_node, nodes, poll_interval);
else
    num_task_steal = number of tasks stolen from most_load_node;
    if (num_task_steal == 0) then
        Sleep(poll_interval);
        poll_interval = poll_interval × 2;
        ADA-WORK-STEALING(node_id, num_neigh, num_node, nodes, poll_interval);
    else
        poll_interval = 1;
    end
end
```

MATRIX Architecture





MATRIX Components

- Client/Benchmarking tool
  - has a task dispatcher generating a workload of tasks
  - assigns the tasks to the system
- Compute Nodes
  - each one has an execution unit that is responsible for executing the tasks, and for load balancing through work stealing
  - each one has a ZHT server for metadata management
  - at booting time, each one pushes identity and location info (ip addr + port no.) to a shared file system for allowing N-N communication

Types of Messages

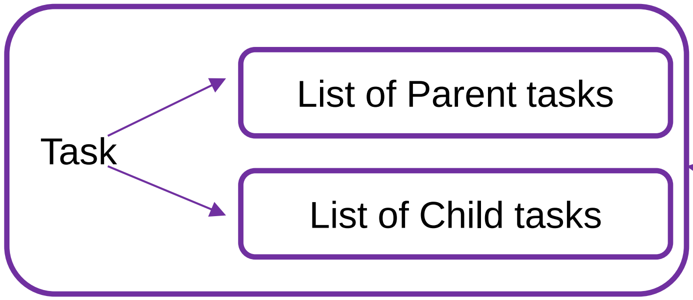
- **ZHT Insert:** write the metadata of tasks
- **ZHT Lookup:** retrieve the existing information from ZHT, e.g. task dependency
- **ZHT Update:** modify the information based on state change
- **MATRIX Insert:** submit jobs to compute nodes
- **Load Information:** idle nodes query the load information
- **Work Stealing:** steal tasks from the most heavy loaded neighbor
- **Client Monitoring:** periodically monitor system utilization and progress

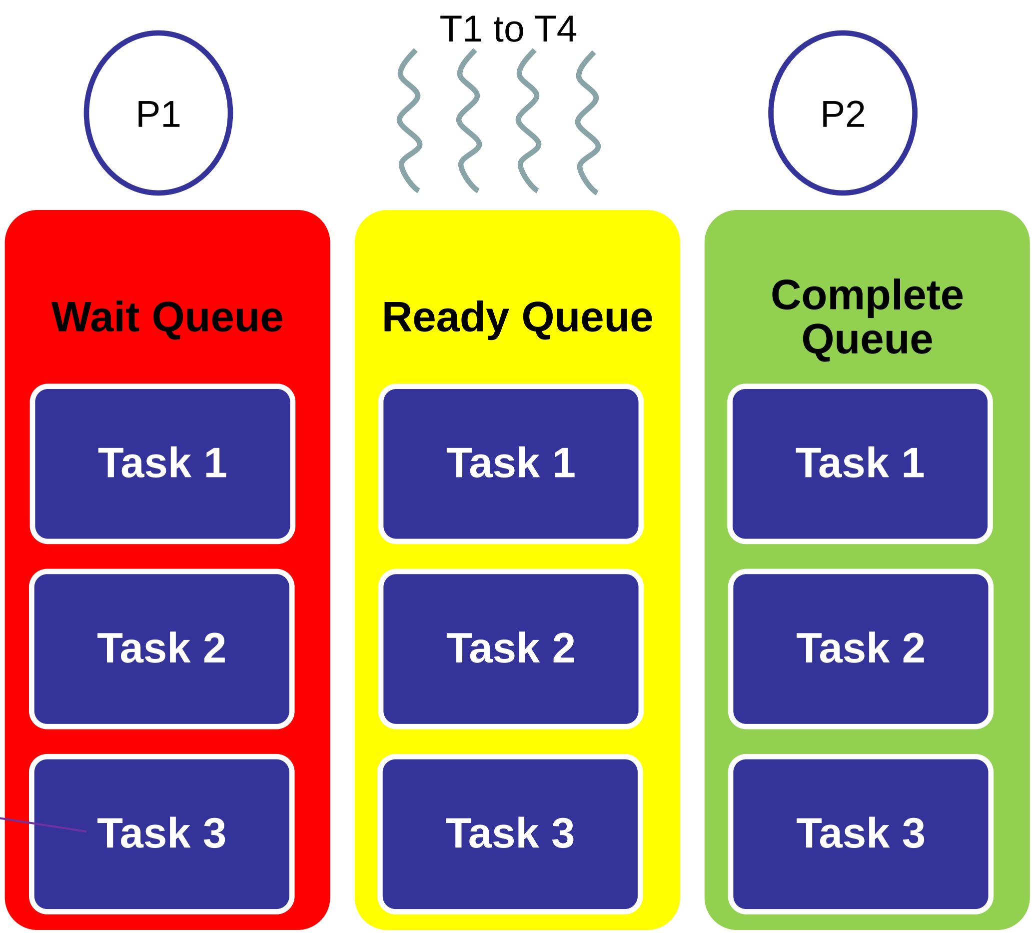
Task Submission

- Best case scenario
  - ➢ tasks are evenly distributed to compute nodes
  - ➢ work stealing happens at the end
  - ➢ one dispatcher does round-robin
  - ➢ multiple dispatchers through ZHT hashing
- Worst case scenario
  - ➢ all tasks are submitted to one arbitrary compute nodes
  - ➢ work stealing happens at the beginning

Execution Unit

- Wait Queue – Hold tasks that are waiting for other tasks to complete
- Ready Queue – Hold tasks that are ready to execute but waiting for CPU
- Complete Queue – Hold tasks that have completed execution
- T1-T4 – Four execution threads
- P2 – Sends notifications for every completed task
- P1 – Receives notification and moves tasks from wait queue to ready queue





Load Balancing and Monitoring

- Load Balancing
  - Background thread keeps checking the state of queues and performs work stealing
  - Tunable number of neighbors and tasks to steal
  - Regulating network traffic - Exponential back-off, number of consecutive failed attempts
- Client Monitoring
  - One task dispatcher periodically monitors the status of every compute node and submitted workload

Outline

- Introduction & Motivation
- Related Work
- MATRIX
- **Distributed Job Launch (DJL)**
- Project Overview

Distributed Job Launch

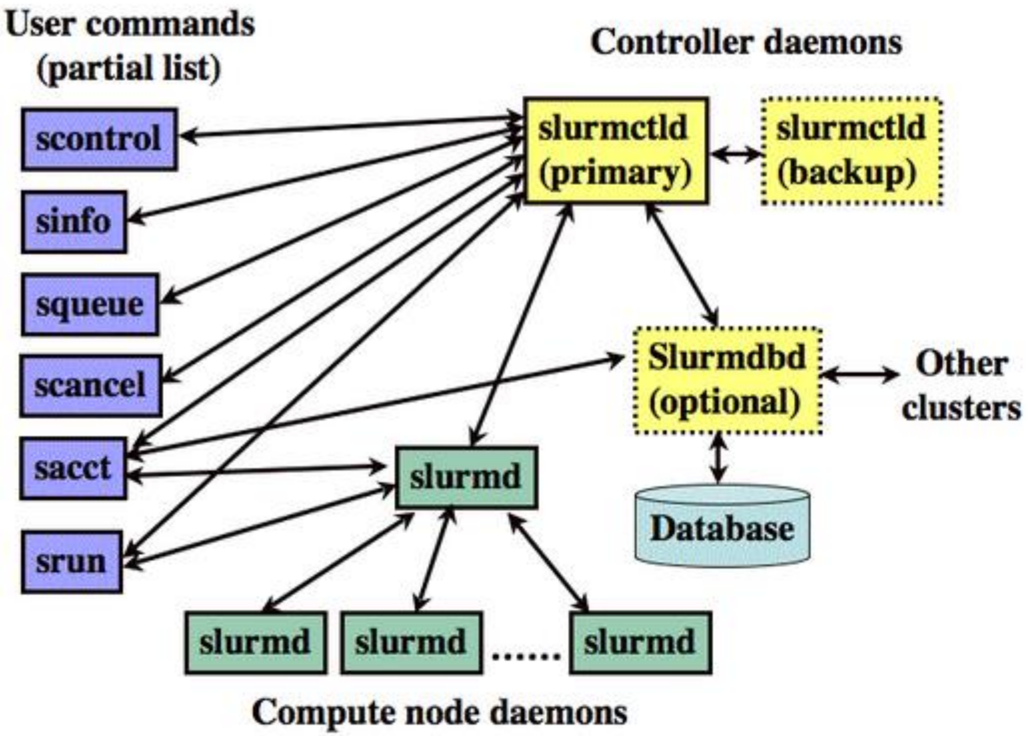
- launching jobs (usually HPC ones that require multiple nodes) to available resources as fast as possible for execution
- a core system service of resource managers
- traditional centralized paradigm with one controller managing all the compute daemons (e.g. SLURM job launch)
- need distributed controllers with each one managing a partition of compute daemons

Challenges and Solutions

- How the controllers maintains the job and resource information?
- How the controllers communicate with each other, and resolve resource contention to get free resources for jobs when the jobs could not be satisfied locally?
- A distributed key-value store (e. g. ZHT) can be used to
  - ➢ store job and resource metadata in a distributed way
  - ➢ resolve resource contention by the atomic “compare and swap” operation
  - ➢ hide the complexities of controllers communicating with each other for replication, failure and recovery, and consistency features
- Develop a distributed job launch based on SLURM and ZHT

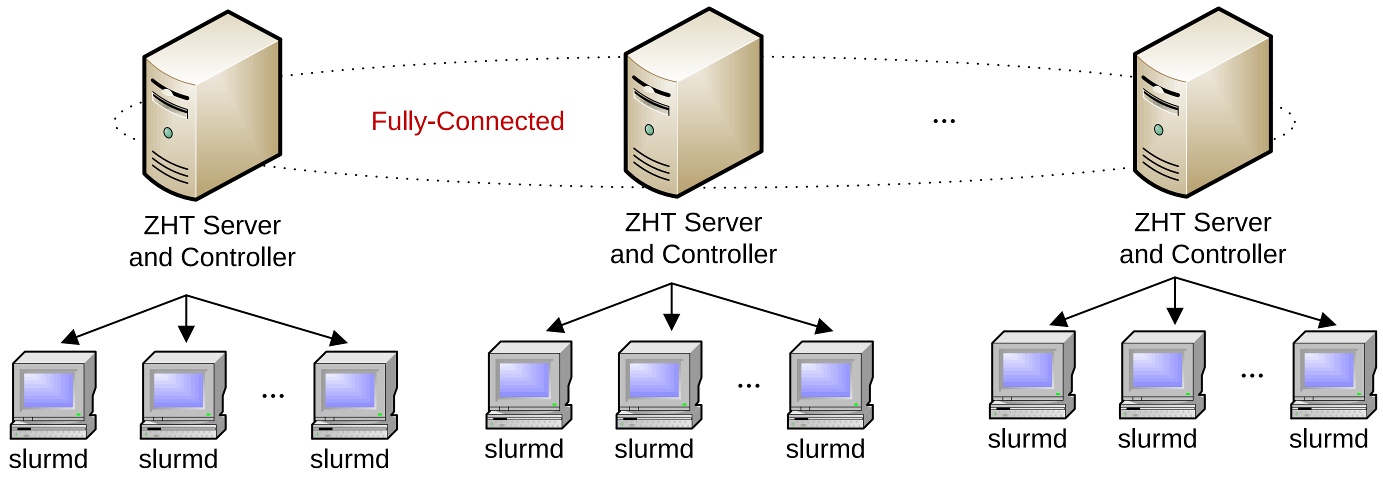
Distributed Job Launch based on SLURM and ZHT

- SLURM



- ZHT
  - ZHT project overview and tutorial

Architecture



Data stored in ZHT

Key	Value	Description
controller id	number of free node, free node list	The free (available) nodes in a partition managed by the corresponding controller
job id	original controller id	The original controller that is responsible for a submitted job
job id + original controller id	involved controller list	The controllers that participate in launching a job
job id + original controller id + involved controller id	participated node list	The nodes in a partition that are involved in launching a job

Resource Stealing

- When a partition cannot satisfy a job, the controller is stealing resources from other partitions

Compare and Swap operation

ALGORITHM 1. Compare and Swap

Input: key (*key*), value seen before (*seen_value*), new value intended to insert (*new_value*), and the storage hash map (*map*).
Output: A Boolean value indicates success (*TRUE*) or failure (*FALSE*).

```
current_value = map.get(key);
if (!strcmp(current_value, seen_value)) then
    map.put(key, new_value);
    return TRUE;
else
    return FALSE;
end
```

Resource Stealing

ALGORITHM 2. Resource Stealing

```
Input: number of nodes required (num_node_req), number of controllers (num_ctl), controller membership list (ctl_id[num_ctl]).
Output: involved controller ids (ctl_id_inv), participated nodes (par_node[]).
num_node_allocated = 0; num_try = 0; num_ctl_inv = 0;
ctl_id_inv = calloc(20 * 100, sizeof(char));
for each i in 0 to 19; do
    par_node[i] = calloc(100 * 100, sizeof(char));
end
while num_node_allocated < num_node_req do
    remote_ctl_idx = Random(num_ctl);
    remote_ctl_id = ctl_id[remote_ctl_idx];
    again:
    remote_free_resource = c_zht_lookup(remote_ctl_id);
    if (remote_free_reource == NULL) then
        continue;
    else
        remote_num_free_node = strtok(remote_free_source);
        if (remote_num_free_node > 0) then
            num_try = 0;
            remote_num_node_allocated =
                remote_num_free_node > (num_node_req –
                num_node_allocated) ? (num_node_req –
                num_node_allocated) : remote_num_free_node;
            if (allocate nodes succeeds)  then //compare and swap
                num_node_allocated +=
                    remote_num_node_allocated;
                par_node[num_ctl_inv++] = allocated node list
                strcat(ctl_id_inv, remote_ctl_id);
            else
                goto again;
            end
        else
            usleep(100000);
            num_try++;
            if (num_try > 2) do
                release all the allocated nodes;
                Resource Stealing again;
            end
        end
    end
end
return ctl_id_inv, par_node;
```

Important Parameters:

sleep length after a resource stealing failure

Number of tries before de-allocate resources

Project Overview

- 1. MATRIX: BenchJMS
  - benchmarking different HPC Job management systems (SLURM, Condor, SGE, PBS, Cobalt)
  - 1 student
  - no need to write code
- 2. MATRIX: BenchTEF
  - benchmarking different MTC task execution frameworks (Falkon, Sparrow, Turbine, CloudKon, MATRIX)
  - 1 student
  - no need to write code

Project Overview

- 3. MATRIX: DJLSys (3 students)
  - working directly with our distributed job launch code
  - study different resource stealing algorithms under high system utilization
  - need to read/write C/C++ programs
- 4. MATRIX: DJLSim (2 students)
  - simulating distributed job launch system
  - Study different resource stealing algorithms under high system utilization up to exascale
  - discrete event simulation, Java

Project Overview

- 5. MATRIX: Swift/M (3 students)
  - using Swift to run large-scale scientific applications to generate workloads used by MATRIX
  - working directly with MATRIX and Swift scripting language
  - need to read/write C/C++, and Swift programs
- 6. MATRIX: Mon/Sim (2 students)
  - simulating distributed monitoring systems with hierarchical tree based aggregation and reduction
  - Study optimal fan out, tree height to build the communication tree up to exascale
  - Study techniques to rebuild the tree after failure happens
  - discrete event simulation, Java

More Information

- More information:
  - http://datasys.cs.iit.edu/~kewang/
- Contact:
  - kwang22@hawk.iit.edu
- Questions?