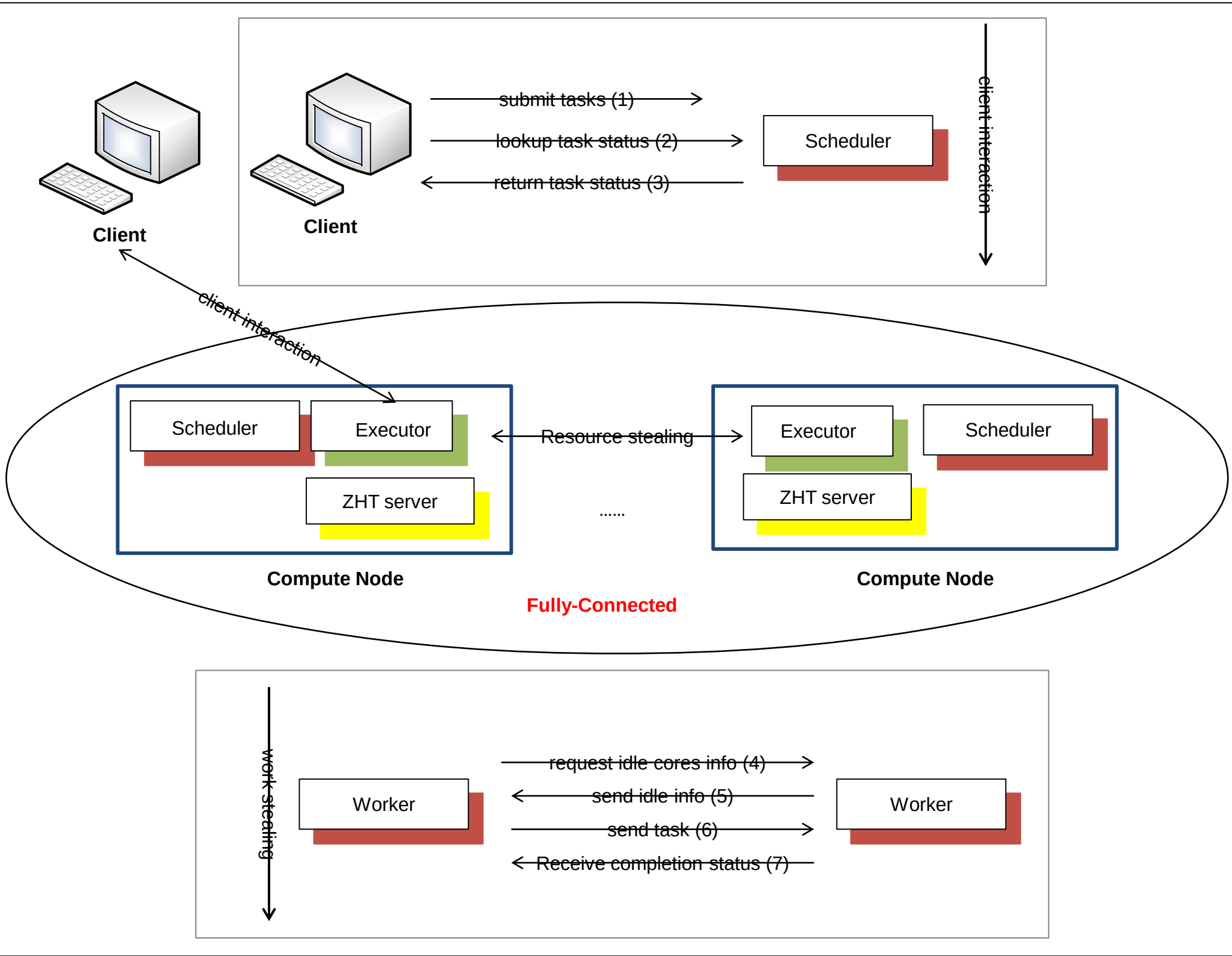
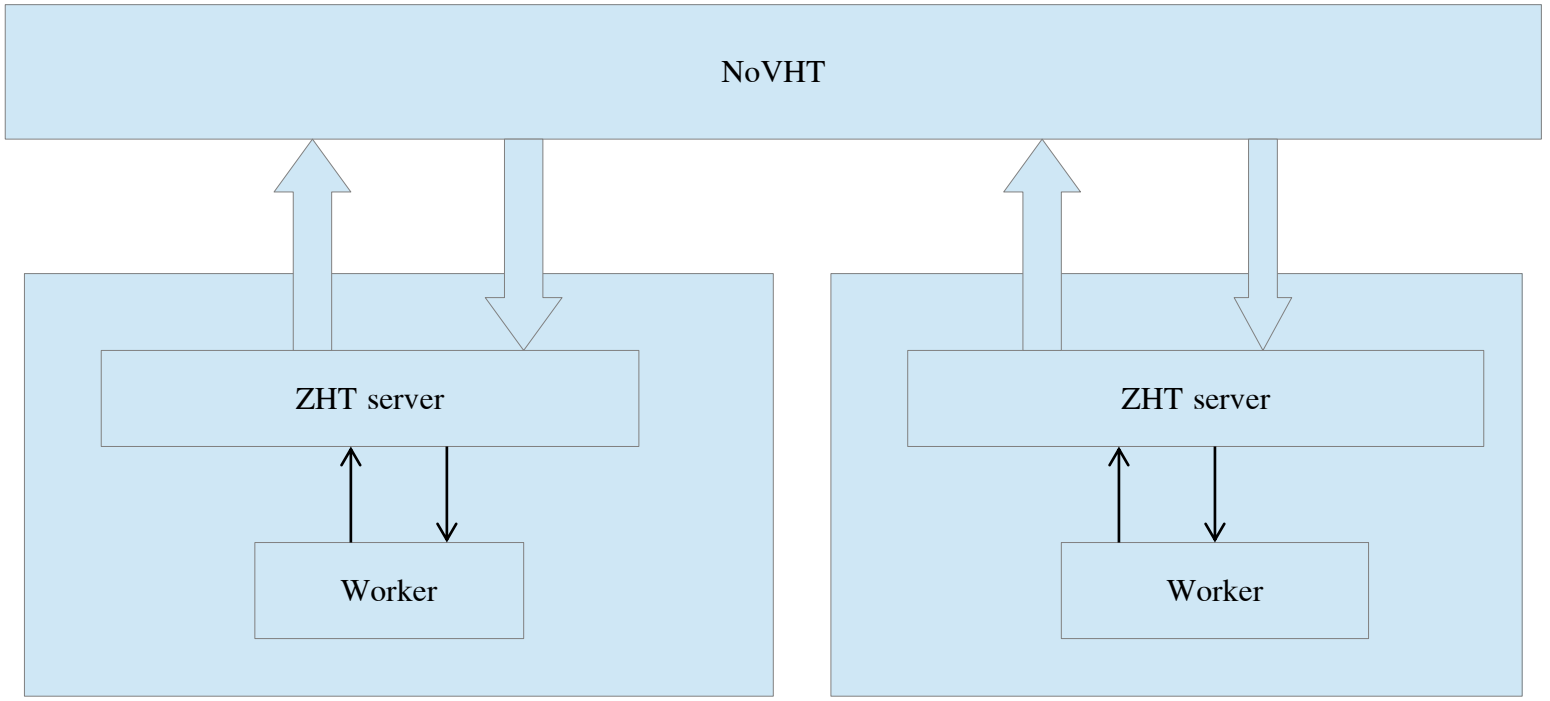
MATRIX-HPC

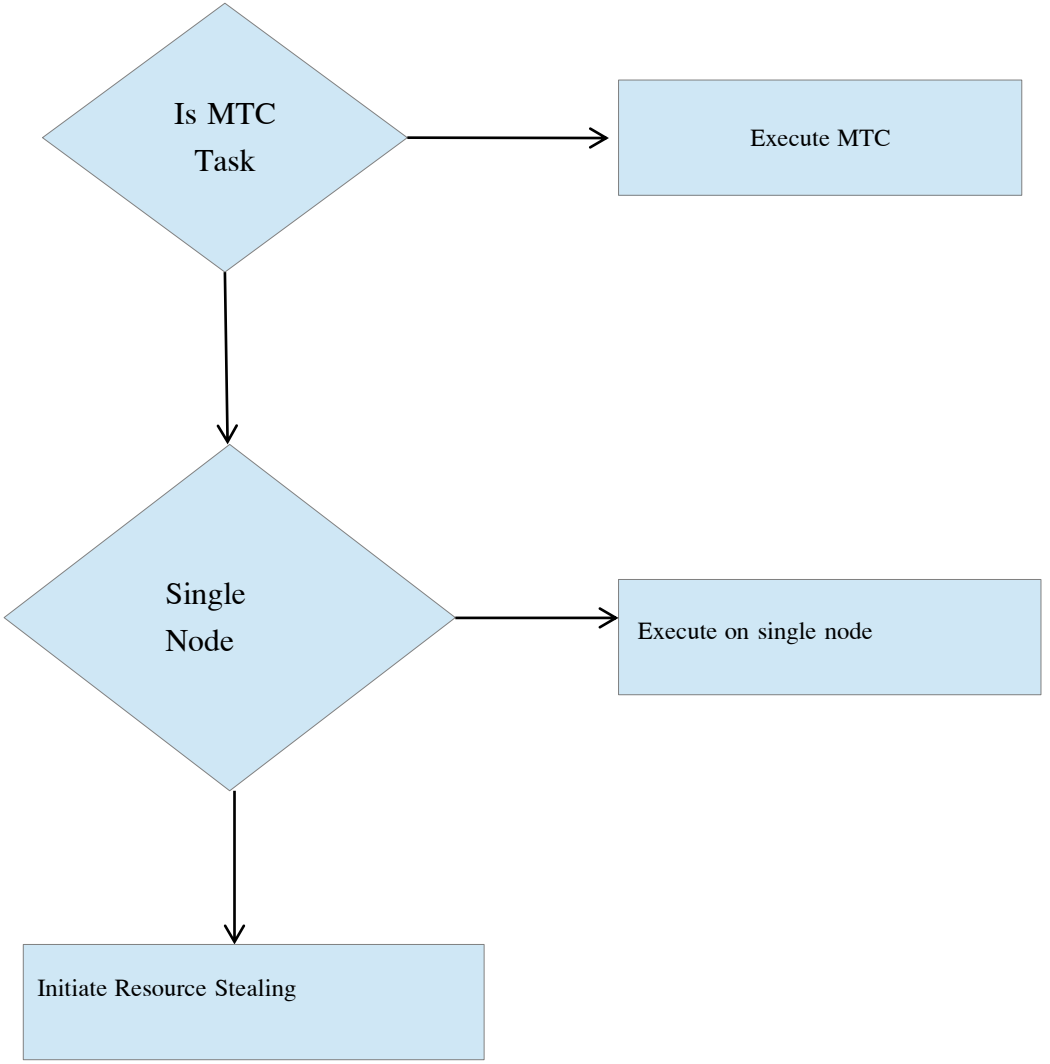
Many-Task computing execution fabRIc at eXascale – HPC implementation


Client
Client
submit tasks (1)
lookup task status (2)
return task status (3)
Scheduler
client interaction
client interaction
Scheduler
Executor
ZHT server
Resource stealing
.....
Executor
Scheduler
ZHT server
Compute Node
Compute Node
Fully-Connected
work stealing
Worker
request idle cores info (4)
send idle info (5)
send task (6)
Receive completion status (7)
Worker

MATRIX-HPC ZHT and NoVHT



Simple Flowchart



Resource Stealing

- Task is identified MTC or HPC with the number of nodes requested
- If HPC task, check if the number of cores are sufficient on a single node
- If yes, waits till the node is available for execution
- If No, resource stealing is initiated
- Neighbors are selected in the same way as chosen for work stealing
- The idle cores information is collected from each node
- Out of the selected nodes, again it is traversed to select the actual nodes sufficient for task execution

Resource Stealing

- Nodes are selected along with their indexes and number of cores available.
- Tasks ID and description are updated to ZHT and NoVHT by building package.
- ZHT_Insert is directly called by the worker.
- After updating ZHT, worker migrates the tasks
- The task migration is only adding the task id directly into the ready queue of selected nodes.
- The task description updated on NoVHT consists of number of cores required divided accordingly to the available on all nodes

Resource Stealing

- After Migrating, it works the same with taking the task from ready queue.
- Check for MTC or HPC.
- HPC tasks migrated might be again subjected to work stealing.
- States are not maintained yet for work stealing after migration of HPC tasks
- As the tasks are completed, the information will be updated at the source which initiated the resource stealing.
- Once the task is completed execution on all nodes, it is pushed to complete queue.

Back off implementation

- Two types of Back off time
- When a task has sufficient cores on the same node, resource stealing is not initiated.
- The ready_queue is locked until it is allocated to the task.
- Idle_cores information is polled frequently.
- If the sufficient cores are not available on the same node, resource stealing is initiated.
- After receiving number of idle cores with each node, if sufficient cores are not available, the node implements a back off by releasing all the resources.
- Once the back off time reaches threshold, the task is pushed at the back of ready queue.

Resource Release

- Select available core information from all nodes
- While getting the information, ready queue and idle_core variable is locked.
- After selecting the nodes on which task will be launched, the resources locked on the nodes which will not be used are released
- If sufficient cores are not available, back off is implemented and all the resources are released.

Pseudo Code

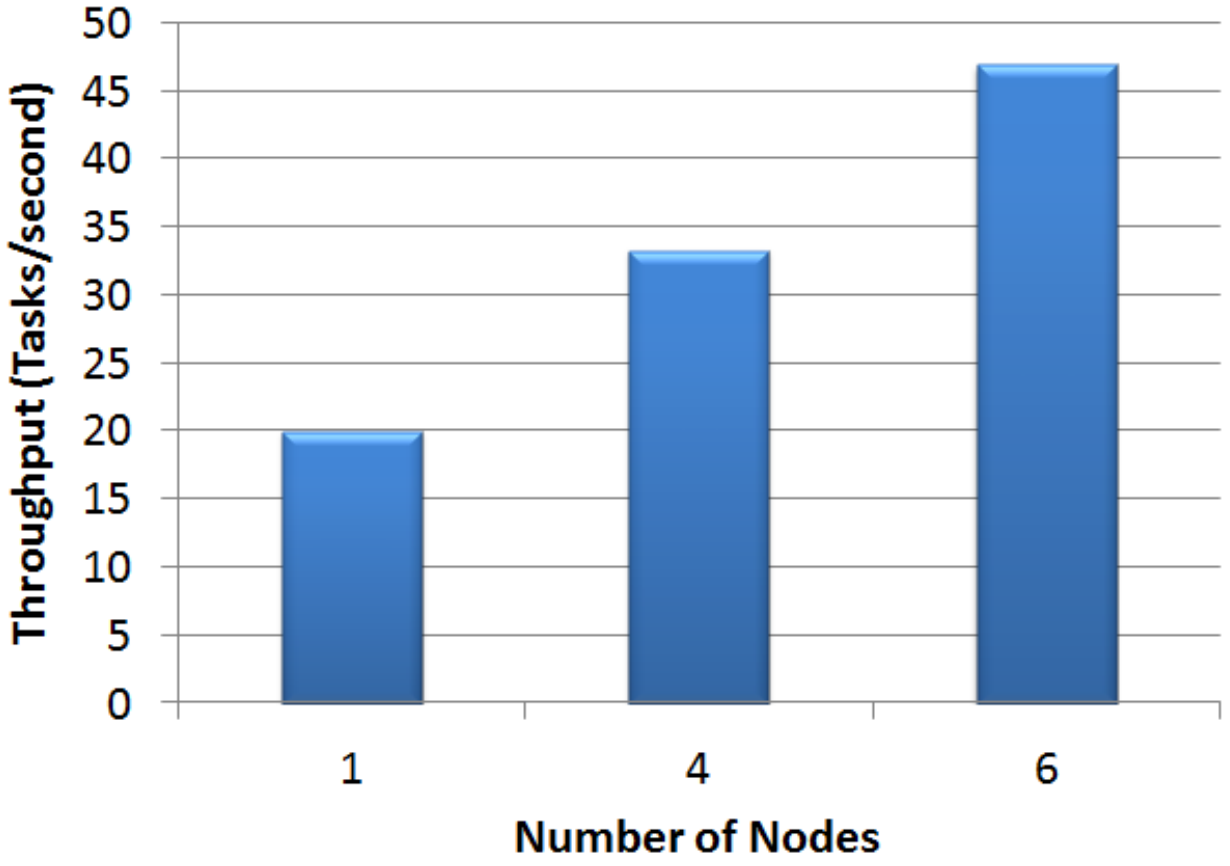
Resource_Stealing(num_of_cores,package)

- Selected_neigh = select_nodes_in_random()
- get_idle_core_information(idle_core_info)
- Success=check_for_sufficient_cores(idle_core_info,num_of_cores,selected_neigh)
- If(!success)
- release_resources(idle_core_info)
- else
- for(i=0;i<selected_neigh_count)
- package= build_package_with_self_index()
- Update_ZHT_and_NoVHT(package)
- Migrate_Tasks(selected_neigh[i].index)

Benchmarking

- Tested till 6 nodes.
- Each tasks needs at least 2 nodes to run
- 10 cores for each task
- Tested on Jarvis cluster.
- Each node in Jarvis has 8 cores.
- 1000 tasks of sleep 0 is passed to the schedulers.
- Single client

Benchmarking



Advantages and Disadvantages

- To start with, the advantage is that, all the cores on all systems will me mostly occupied since the resource stealing happens at the core level.
- Resource Release happens very quickly if the nodes are not required to run the tasks.
- The current implementation of random nodes might not work at higher scales,
- Network overhead might be higher since for each HPC task broken into sub tasks to migrate, ZHT and NoVHT update needs to be done and sent across network.
- If all the nodes are busy, resource stealing with high requirement for a task might lead to backing-off very often and hence increases turn around time.

Future work

- Complete system implementation
- Test the system up-to 64 node scales and collect results.
- The random selection needs to be modified in order increase performance.
- A new node selection policy to distinguish between free nodes and busy nodes can be implemented
- Try to decrease network overhead for HPC tasks.

Questions?