HyperQueue: Overcoming Limitations of HPC Job Managers

Stanislav Böhm
stanislav.bohm@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Jakub Beránek
jakub.beranek@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Vojtěch Cima
vojtech.cima@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Roman Macháček
roman.machacek@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Vyomkesh Jha
jha0007@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Alfréd Kočí
alfred.koci@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Branislav Jansík
branislav.jansik@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

Jan Martinovič
jan.martinovic@vsb.cz
IT4Innovations, VSB – Technical University of Ostrava
Czech Republic

ABSTRACT

In recent years, HPC workloads and communities have undergone substantial paradigm shifts. There is an increasing amount of users that want to leverage HPC clusters to execute many simple and embarrassingly parallel tasks as easily as possible. However, due to the limitations of traditional HPC job managers, these users must often resort to manual aggregation of tasks into a smaller number of jobs to reduce job manager overhead. This approach is both labour-intensive and inefficient, as it lacks dynamic load balancing required to fully utilize computational nodes with tens or hundreds of cores. We introduce *HyperQueue* - a task scheduling runtime that can execute a large amount of tasks on top of an HPC job manager by automatically aggregating tasks into jobs and dynamically load balancing them across all allocated nodes and CPU cores. *HyperQueue* is an open-source tool that is designed for ease of use and deployment.

KEYWORDS

scheduling, task management, resource management, cluster

1 INTRODUCTION

Modern HPC systems contain a large number of computational nodes, with each node having tens or hundreds of CPU cores. There is an increasing number of HPC users that want to leverage all this computational power with quite simple workloads, for example by running a large number of single-node, relatively short-lived tasks in an embarrassingly parallel manner [1].

While HPC systems are prepared for almost arbitrarily complex distributed computations, it can be surprisingly difficult to execute a large number of such simple tasks on them efficiently. HPC job managers such as Slurm [2] or PBS are almost ubiquitously used to manage computational resources of HPC clusters. The most straightforward approach in this scenario is thus to simply map each task directly to a single job.

However, this is usually infeasible, because the manager introduces nontrivial overhead and thus severely limits the amount of jobs that can be submitted by a single user. The manager is also usually configured in a way that does not allow allocating sub-node resources (running multiple jobs on a single node concurrently), either for security or performance stability reasons. Therefore, if the tasks only use a few cores, they cannot fully utilize the provided computational nodes. These limitations can pose a barrier to entry for new HPC users, since they cannot submit tasks to the cluster in a straightforward way.

To overcome these limitations, we are introducing *HyperQueue*, a task execution runtime that can efficiently execute many single-node tasks on top of a Slurm/PBS cluster. Users can submit a large amount of tasks into *HyperQueue* in a simple way; it will then automatically aggregate tasks into a smaller amount of jobs. Once the jobs are started, it will distribute and dynamically load balance the tasks across all allocated jobs, nodes and cores to efficiently utilize available resources.

HyperQueue is an open-source Rust project released under the MIT licence: https://github.com/It4innovations/hyperqueue.

2 RELATED WORK

There are several other approaches that can be used instead of creating a separate job for each task. One alternative is to run an additional scheduler on top of ("outside") the internal job manager. Snakemake [3] can aggregate multiple tasks into groups. This approach reduces the number of submitted jobs and thus decreases the overhead introduced by the job manager. Nevertheless, this aggregation is performed eagerly, before the tasks even start to execute. Therefore, the tasks cannot be load balanced dynamically in response to actual node and core utilization.

Another option is to start a scheduling runtime within ("inside") a Slurm/PBS job to enable load balancing of tasks independently of the outer job manager. The main disadvantage of this approach is

that the user has to manually aggregate tasks into jobs to avoid creating too many of them. Such aggregation is not always trivial and it does not enable task load balancing across different jobs, which may cause inefficient node utilization. As an example, Dask [5] can be used in this way.

To achieve both load balancing and automatic task aggregation, a scheduling system has to run both outside the job manager as well as inside the submitted jobs. Merlin [4] uses this approach, but introduces several other limitations. Its load balancing is limited, because it requires users to specify queues with predetermined concurrency and also does not allow specifying task resource requirements. It is also nontrivial to deploy, since it requires setting up several complex dependencies.

We are not aware of any existing tool that automatically aggregates tasks and also fully dynamically load balances them while being easy to deploy and use. We consider these properties crucial for enabling user-friendly execution of workflows containing many embarrassingly parallel tasks on HPC clusters.

3 HYPERQUEUE

HyperQueue is a task scheduling runtime designed to efficiently execute a large number of embarrassingly parallel tasks on a Slurm/PBS cluster in a simple way. We list the key features of HyperQueue in the following list.

Task Aggregation HyperQueue does not require any manual task aggregation. Users simply submit tasks into HyperQueue and it takes care of distributing them to computational resources (nodes and cores) allocated by Slurm/PBS. HyperQueue can either use computational resources (jobs) allocated externally, or it can submit jobs automatically, as needed by the current workload.

Load balancing HyperQueue uses a work-stealing scheduler that supports task resource requirements (how many cores does a task need), task priorities, core pinning and is NUMA-aware. The scheduler is dynamic with respect to computational resources; new workers may be added or removed during workload computation and new tasks may be submitted at any time.

Scaling HyperQueue supports task arrays and also output streaming, which reduces pressure on network filesystems commonly present on HPC clusters. The average runtime overhead is around $100\mu s$ per task, which allows processing of fine-grained tasks.

Deployment HyperQueue is distributed as a single binary that depends only on `libc`, does not need any third-party services, and does not require any elevated privileges; it is thus trivial to deploy.

3.1 Architecture

The architecture of HyperQueue is depicted in Figure 1. The central component is the server, which manages workers and schedules tasks onto them. It is designed to be a long-lived process which can be deployed e.g. on a login node. Users can connect to it in order to submit tasks or observe the status of computation.

The worker component runs on the computational nodes of a cluster, inside a Slurm or a PBS job. Its main purpose is to execute tasks assigned to it by the server. The workers communicate with the server over the network; they must be able to initiate a TCP/IP connection to the server.

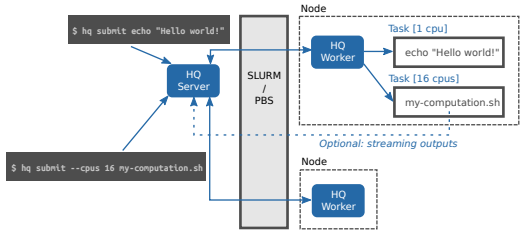


Figure 1: HyperQueue architecture

3.2 Usage example

(1) Start server on a login node:

```
$ hq server start &
```

(2) Ask for a Slurm/PBS job and start HyperQueue worker
- Ask for nodes in PBS:
  ```
  $ qsub <qsub-args> -- hq worker start
  ```
- Ask for nodes in SLURM:
  ```
  $ sbatch <sbatch-args> --wrap "hq worker start"
  ```

(3) Submit a task:

```
$ hq submit my-computation.sh
```

(4) Check the status of the computation:

```
$ hq jobs
+----+--------------------+----------+-------+
| Id | Name               | State    | Tasks |
+----+--------------------+----------+-------+
|  1 | my-computation.sh | FINISHED | 1     |
+----+--------------------+----------+-------+
```

ACKNOWLEDGEMENT

This work was supported by the LIGATE project. This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956137. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, Sweden, Austria, the Czech Republic, Switzerland.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).

REFERENCES

[1] Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Joseph Koning, Tapasya Patki, Thomas R. W. Scogland, Becky Springmeyer, and Michela Taufer. 2018. Flux: Overcoming Scheduling Challenges for Exascale Workflows. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. 10–19. https://doi.org/10.1109/WORKS.2018.00007

[2] Morris A. Jette, Andy B. Yoo, and Mark Grondona. 2002. SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 44–60.

[3] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (08 2012), 2520–2522. https://doi.org/10.1093/bioinformatics/bts480 arXiv:https://academic.oup.com/bioinformatics/article-pdf/28/19/2520/819790/bts480.pdf

[4] Luc Peterson, Rushil Anirudh, Kevin Athey, Benjamin Bay, Peer-Timo Bremer, Vic Castillo, Francesco Natale, David Fox, Jim Gaffney, David Hysom, Sam Jacobs, Bhavya Kailkhura, Joe Koning, Bogdan Kustowski, Steven Langer, Peter Robinson, Jessica Semler, Brian Spears, Jayaraman J. Thiagarajan, and Jae-Seung Yeom. 2019. Merlin: Enabling Machine Learning-Ready HPC Ensembles.

[5] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.

HyperQueue

Overcoming Limitations of HPC Job Managers

Böhm, Beránek, Cima, Macháček, Jha, Kočí, Jansík, Martinovič

IT4Innovations, VSB-TU Ostrava, Czech Republic

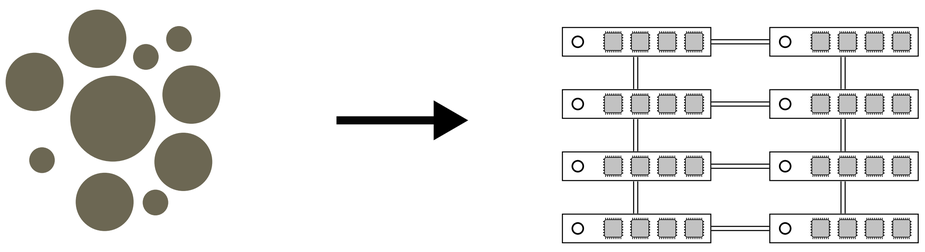
IT4Innovations/hyperqueue

https://def.it4i.io

HyperQueue lets you easily execute millions of embarrassingly parallel tasks on a Slurm/PBS cluster with dynamic load balancing and NUMA-aware scheduling

Motivation 1

Many simple tasks → **Slurm/PBS Cluster**



Execute a large number of simple, embarrassingly parallel tasks on an HPC cluster as easily as possible.

Challenges 2

Task duration x Job manager overhead
- Tasks can be very short (milliseconds)
- Workflows can contain a large number of tasks
- HPC job managers introduce a lot of overhead per each task

Challenge #1: **Keep overhead per task minimal**

Task core granularity x Node utilization
- Tasks use diverse (often small) amounts of threads/cores
- HPC nodes contain many (tens, hundreds) cores
- HPC job managers often allow only a single job per node

Challenge #2: **Achieve high node utilization**

Limitations of existing approaches 3

Submit tasks directly as Slurm/PBS jobs
- 👎 High overhead per task → limited task throughput
- 👎 Usually only a single job per node allowed → inefficient node usage
- 👎 Manual aggregation of tasks into jobs is required to reduce overhead

Run a meta-scheduler outside of Slurm/PBS
e.g. Snakemake
- 👍 More tasks may be submitted than what Slurm/PBS allows
- 👎 No dynamic task load balancing → inefficient node usage
- 👎 Manual aggregation of tasks into jobs is required to reduce overhead

Distribute tasks inside each Slurm/PBS job
e.g. Dask, GNU Parallel
- 👍 Load balancing across nodes inside a job
- 👎 No load balancing across jobs
- 👎 Manual aggregation of tasks into jobs is required to reduce overhead

Run both outside Slurm/PBS and inside its jobs
Merlin
- 👍 Automatically load balances tasks into jobs
- 👍 Load balances tasks inside jobs
- 👎 Queues with fixed concurrency → balancing not fully dynamic
- 👎 Does not allow specification of CPU task requirements
- 👎 Nontrivial to deploy

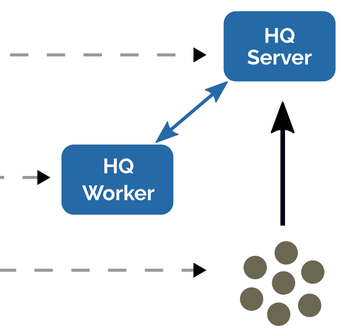
HyperQueue 4

Runtime designed for executing a large number of tasks.

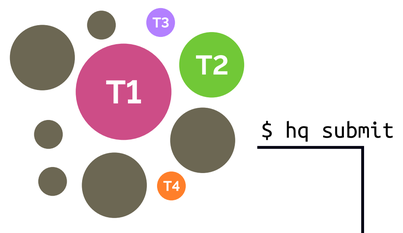
- **Transparent task execution on top of a Slurm/PBS cluster**
  - Tasks are distributed amongst jobs, nodes and cores
  - Slurm/PBS jobs with HQ workers are started automatically
- **Dynamic load balancing across jobs**
  - NUMA-aware work-stealing scheduler
  - Supports core pinning, task priorities, task arrays
  - Nodes and tasks may be added/removed on the fly
- **Scalability**
  - Low overhead per task (~100 μs)
  - Handles hundreds of nodes and millions of tasks
  - *Output streaming* avoids creating many files on network filesystems
- **Easy deployment**
  - Single binary, no installation, depends only on *libc*
  - No elevated privileges required
- **Open source**

Usage example 5

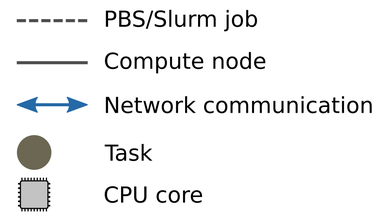
```
# Start the HyperQueue server (on a login node)
$ hq server start

# Ask for computational resources
(PBS)   $ qsub <qsub-params> -- hq worker start
(Slurm) $ sbatch <sbatch-params> --wrap "hq worker start"

# Submit a HyperQueue job with 100 tasks
$ hq submit --cpus=4 --array=1-100 my-computation.sh

# Check the state of submitted jobs
$ hq jobs
+----+-------------------+----------+-------+
| Id | Name              | State    | Tasks |
+----+-------------------+----------+-------+
|  1 | my-computation.sh | FINISHED | 1     |
+----+-------------------+----------+-------+
```

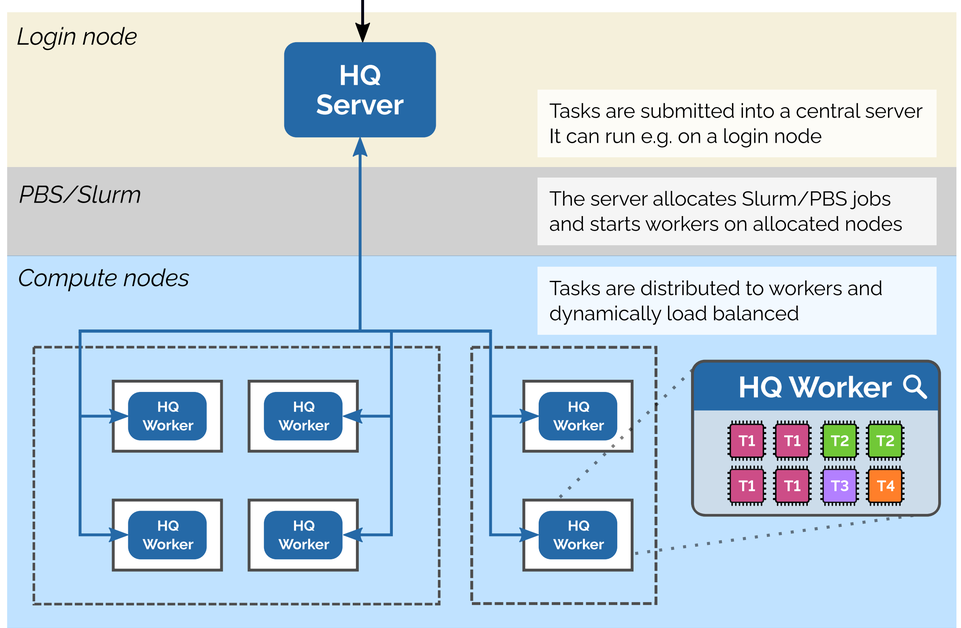


Architecture 6



$ hq submit





This work was supported by the LIGATE project. This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956137. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, Sweden, Austria, the Czech Republic, Switzerland.
This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER