A job management system for utilization of idle supercomputer resources

Elena Fedotova, Stanislav Polyakov, Julia Dubenskaya, Minh Duc Nguyen
Skobeltsyn Institute of Nuclear Physics, Moscow State University

Idle supercomputer resources

In modern supercomputers, it is often problematic to utilize all of the available computational resources. On average, as much as 10% of the resources may be idle.

One approach to diminishing the amount of the idle computational resources is adding low-priority jobs that can be executed on the resources that would otherwise remain idle. This approach was used, for example, to execute ATLAS jobs on Titan-2 supercomputer.

Low-priority jobs

For this “opportunistic” approach to work, the additional jobs must be small, i.e. should not require many nodes or long execution time.

There are numerous problems that do not require parallel computations, such as Monte Carlo simulation in physics. However, many of them do require a lot of time to execute.

Container virtualization and container migration

Containers are isolated user space instances in many ways similar to virtual machines but always sharing OS kernel with the host machine. Some container platforms offer tools for container migration, allowing the users to freeze the container with all its contents and restore it on a different machine.

We propose to run the low-priority jobs inside containers and use container migration tools to divide the execution into relatively short periods.

A system for executing containerized low-priority jobs on idle supercomputer nodes

We propose a two-component system that executes low-priority non-parallel jobs on idle supercomputer nodes. The nodes must have the same container software installed on them.

The first component is an agent program. An instance of an agent program is launched on a computational node, requests one or several low-priority jobs and runs them inside containers. Before the end of the allotted time it saves the containers in which the jobs are still running.

The second component is a control program that maintains the queue of low-priority jobs, keeps track of their status, and interacts with the supercomputer scheduler.

The agent program

The agent program interacts with the node’s container software. Some details may vary depending on the software being used.

In our prototype we used Docker. In this implementation, the agent program launches new jobs inside containers created from specified images. Before the end of the allotted time the checkpoints are created for the containers that are still running, and moved to storage. For the saved jobs the agent program creates containers, moves checkpoint files from storage to the container directory, and restarts the process from the checkpoint.

Creating a checkpoint may require significant time so the process of saving the jobs must start in advance.

The control program

The control program maintains its own queue of low-priority jobs, distributes them between instances of the agent program, and keeps track of their status (which may be “New”, “Running”, “Saved”, or “Complete”). “New” and “Saved” jobs are given to agent program instances on their requests.

The control program also submits jobs consisting of agent program instances to the supercomputer scheduler.

Agent execution time

Typically the execution time of a job cannot be predicted precisely. Predicting when a user will submit a new job is also usually impossible. Therefore we assume that the time when a node running low-priority jobs will be required by a regular job is impossible to know in advance.

Because of the time required to save containers it is also impractical to stop low-priority jobs after the node is required by a regular job.

Thus we can make the agent execution time uniform and choose it based on tradeoffs between efficiency of idle nodes utilization (increases with agent execution time) and supercomputer performance with respect to regular jobs (decreases with agent execution time).

Agent synchronization

Choosing uniform agent execution time can result in a “sheaf” problem: the number of nodes running low-priority jobs may be sufficient to start some of the regular jobs waiting in the queue, but this does not happen because their termination time varies, and if only a few of the nodes are freed at any given time, the available nodes might only be sufficient to run more agent instances. Since the low-priority jobs are always availble, the problem perpetuates. Reservation of nodes by the scheduler does not prevent it completely.

To address this problem, we synchronize the termination time between the running instances of an agent program, rather than the execution time.

The system prototype

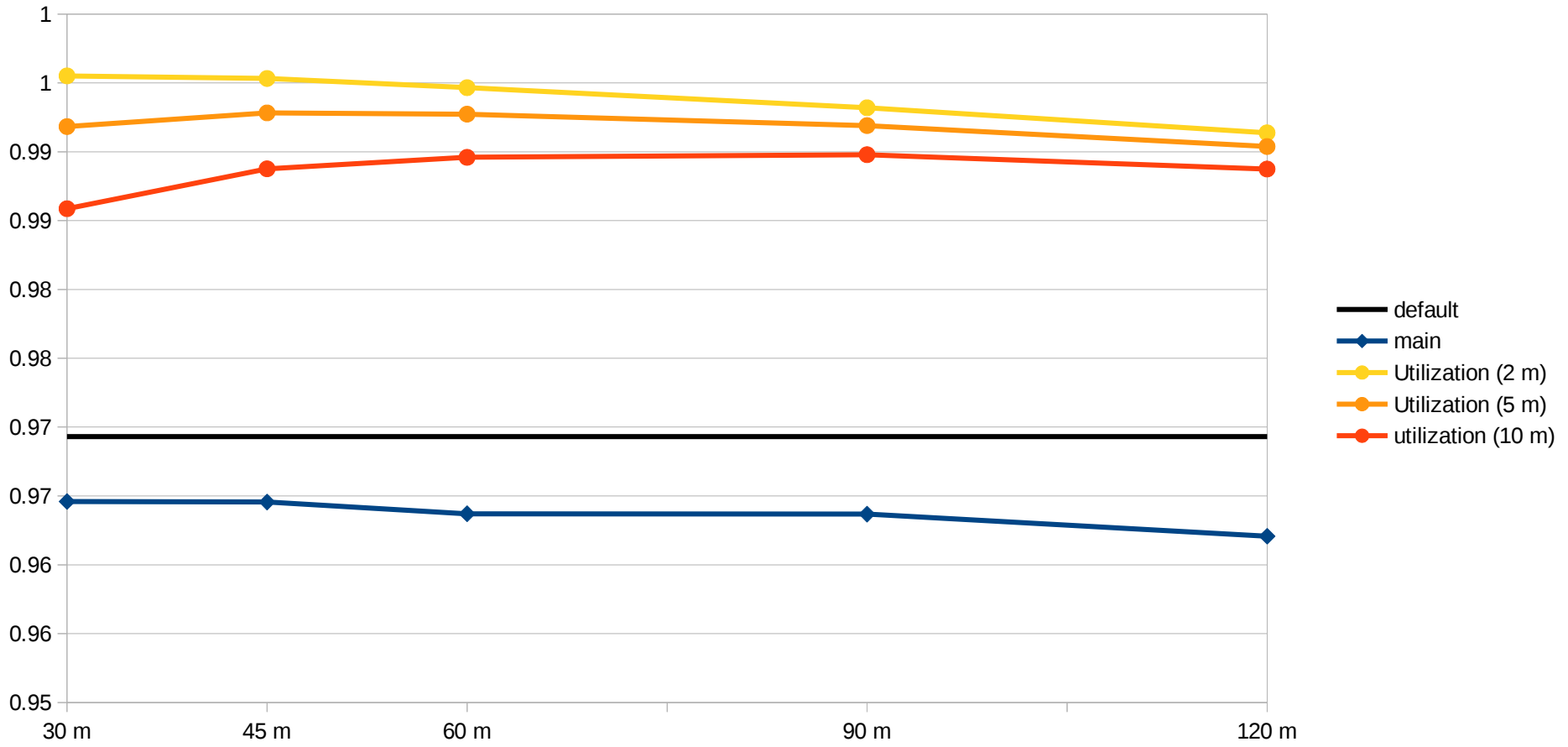
We implemented a prototype of the proposed system. The prototype works with container software Docker. Our tests demonstrate that it is functioning properly. The time to create checkpoints and restart containers from checkpoints is quite high, but we believe that increasing the network speed and using high-performance RAIDs should reduce it.

Simulation results

We conducted a series of experiments that simulated the scheduler with the proposed system, using the jobs with parameter distribution based on historical data from Lomonosov supercomputers.

In the experiments where we assumed that the regular job queue was always full, the average load of supercomputer was already quite high without our system (95.5% to 99.2%). Adding the system increased the effective load by 0.3% to 3.6%, reducing the average number of idle nodes (including nodes performing container operations rather than executing the jobs) by 40% to 78%.

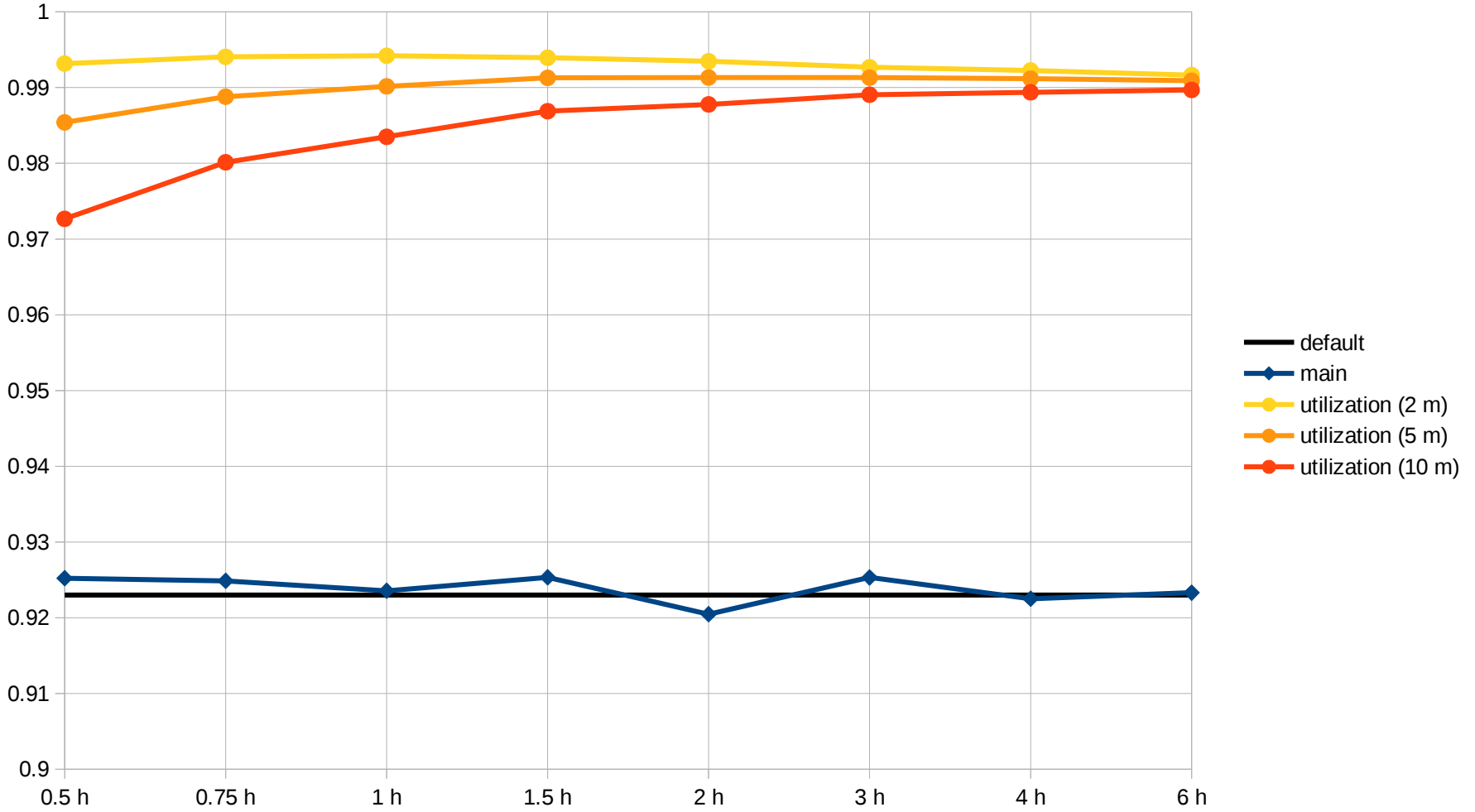
Effective load for 1024 nodes with a full queue based on Lomonosov-1 data



Simulation results (part 2)

In the experiments where we tried to match the average load (without our system) to the historical levels of Lomonosov supercomputers, we added regular jobs according to Poisson distribution. In these experiments, adding our system increased the effective load to 98.8% (from 92.3% for Lomonosov-1 and 88.8% for Lomonosov-2). In this case, there were also no significant changes in the average load by regular jobs.

Effective load for 4000 nodes with a queue and average load based on Lomonosov-1 data



Conclusion

- We proposed a system for utilization of idle resources of supercomputers by running additional low-priority jobs in containers.
- We created a prototype of the system working with Docker containers.
- Simulation of the supercomputer scheduler with the proposed system shows significant (up to 10%) increase in effective utilization of supercomputer resources.

Acknowledgements

The work was supported by the Russian Foundation for Basic Research grant 18-37-00502. We also thank Sergey Zhumatiy for provided information and useful discussions.