Resource Elasticity at Task-Level

Jonas Posner
Advisor: Prof. Dr. Claudia Fohry
Research Group Programming Languages / Methodologies
University of Kassel, Germany
{jonas.posner | fohry}@uni-kassel.de

UNI KASSEL VERSITÄT

https://www.uni-kassel.de/go/IPDPS21

Motivation

- Adaptive resource management of supercomputers highly improve throughput and decrease energy consumption.
- Adaptivity must be backed by at least three major layers:
  1. global job schedulers,
  2. programming systems, and
  3. algorithms/applications.
- Recent research addresses these layers, but no comprehensive solution has yet been established.
- *Problem:* Elastic algorithms cause a non-negligible additional development effort.

Contribution

- **This work proposes a novel resource elasticity scheme:**
  - Applications using our runtime automatically adapt to the addition and release of multiple compute nodes
  - *No* explicit synchronization points or additional programming effort are required.
  - Intermediate level of a task-based runtime system.
  - The load is automatically balanced dynamically.

Dynamic Independent Tasks

- Tasks are free of side effects.
- Task processing may generate a task result and new tasks.
- Task results are reducible.
- Each worker maintains a partial result.
- The final result is computed from the partial results.
- Each compute node runs a single process that maintains multiple workers.
- Local workers share tasks with other local workers.
- When a process runs out of tasks, the process attempts to steal tasks from random processes, followed by lifeline buddies, which are predetermined by a graph.

Resource Elasticity Scheme

- Addition and release of nodes is controlled by process 0, which can not be released.
- Resource changes can be triggered any time, but multiple requests are handled sequential.
- Actions are performed distributed and asynchronously to task processing, but *not* to work stealing.
- When releasing nodes:
  - The corresponding processes stop processing and send all remaining tasks and results to staying processes.
  - The lifeline-graph is recalculated to exclude the processes to be released from future work stealing.
- When adding nodes:
  - New processes are started on these nodes.
  - The lifeline-graph is recalculated, with the new processes automatically being given tasks.

Experiments

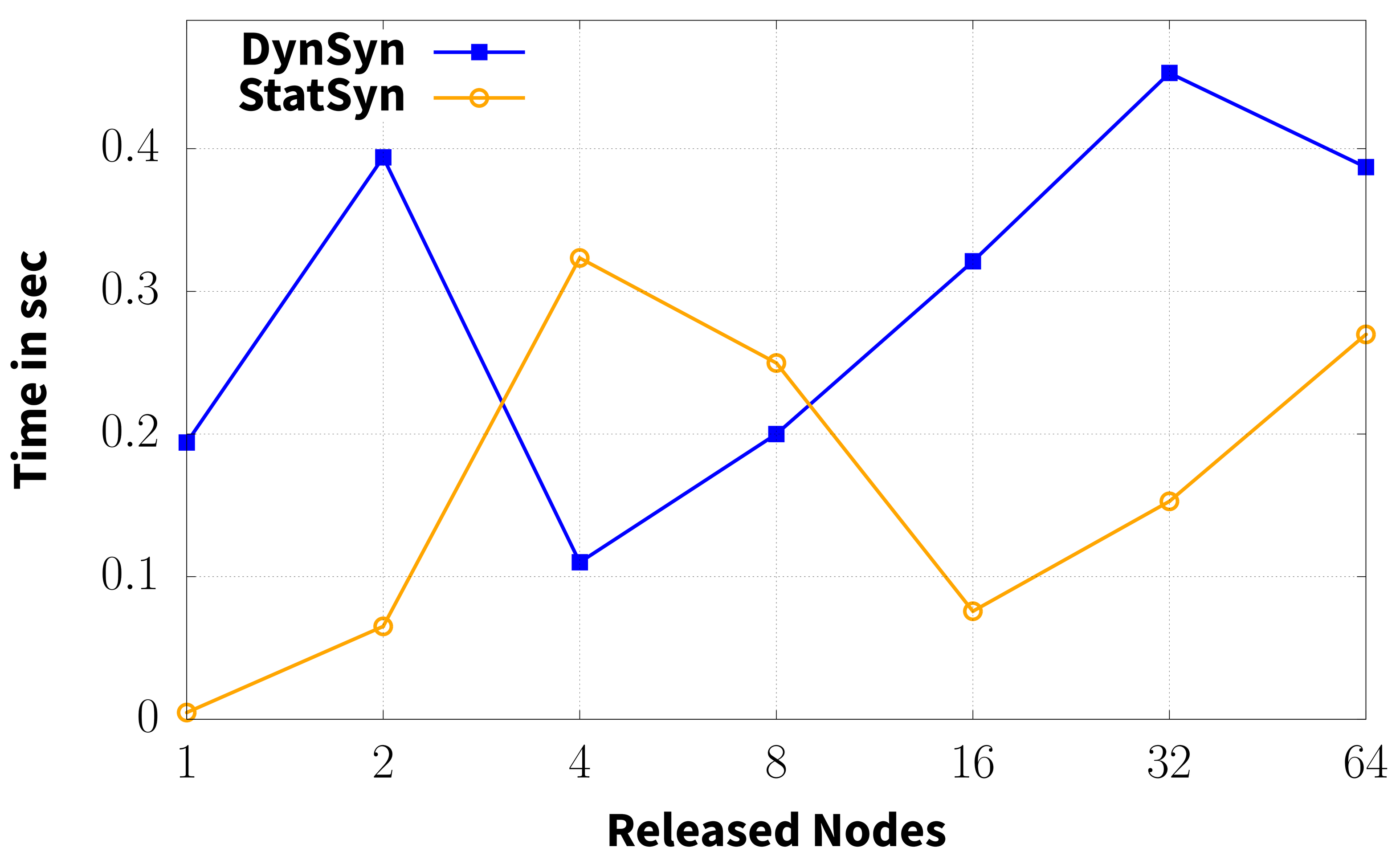
- We implemented the elasticity scheme and conducted experiments with up to 128 nodes.
- Two synthetic benchmarks (StatSyn and DynSyn) provide smooth weak scaling with a base calculation time of 100s.
- Adding and releasing compute nodes is triggered after 50s running time, and affects the total running time.

Conclusion

- Our novel task-level resource elasticity scheme enables the addition and release of nodes on-the-fly.
- No explicit synchronization points or additional programming effort is required.
- Experiments have shown low costs and good scalability for both adding and releasing nodes.
- Future work should integrate handling unexpected resource changes, such as fail-stop failures.

Performance

Costs for Releasing Nodes



Costs for Adding Nodes