Surrogate Models in HPC Applications

A Lock-Free approach for an MPI-based Distributed Hash Table

Max Lübke, Marco De Lucia, Stefan Petri, Bettina Schnor

31st PARS Workshop, Hagen

September 18, 2025

Previous Work

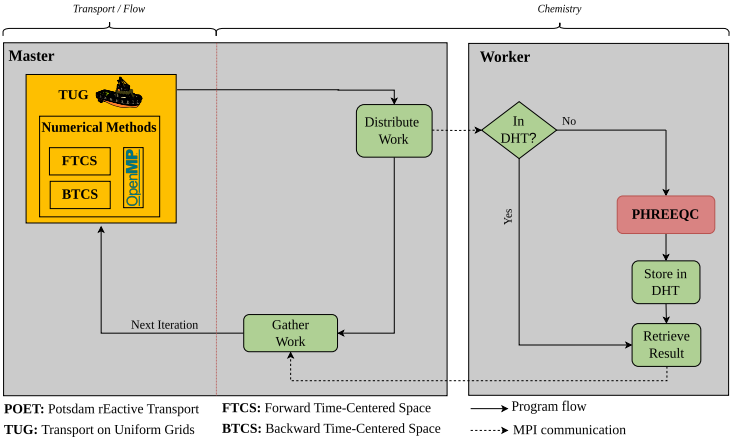
Reactive transport simulator POET[1]:

- MPI-parallelized simulator for coupled reactive transport in porous media
- Coupling of transport with complex geochemistry
- Integration of MPI-based Distributed Hash Table (MPI-DHT) as surrogate model
- Based on MPI's one-sided communication (RDMA)
- Caching and reusing results of previous simulations
- Speedup of up to factor 10 for simulations with acceptable accuracy loss

[1] M. De Lucia et al. "POET (v0.1): Speedup of Many-Cores Parallel Reactive Transport Simulations with Fast DHT-Lookups". In: *Geoscientific Model Development* 14.12 (2021), pp. 7391–7409. DOI: 10.5194/gmd-14-7391-2021.

POET

Potsdam rEactive Transport Simulator



POET

Rounding and Caching

Input:

Species	Concentration [mol/kgw]
Calcium (Ca^{2+})	$\mathbf{1.23} \cdot 10^{-4}$
Carbon (C^{4+})	$\mathbf{2.48} \cdot 10^{-4}$
Magnesium (Mg^{2+})	$\mathbf{3.69} \cdot 10^{-7}$
Chloride (Cl^{-})	$\mathbf{4.81} \cdot 10^{-6}$
...	...

Chemistry $\longrightarrow$

Output:

Species	Concentration [mol/kgw]
Calcium (Ca^{2+})	$2.34567814 \cdot 10^{-4}$
Carbon (C^{4+})	$5.48163331 \cdot 10^{-3}$
Magnesium (Mg^{2+})	$6.38471932 \cdot 10^{-8}$
Chloride (Cl^{-})	$5.81216202 \cdot 10^{-6}$
...	...

$\rightarrow$ **Use rounded input as key, exact output as value for caching in DHT**

MPI-DHT

Original Design:

- API with small function set: `DHT_create`, `DHT_read`, `DHT_write`, `DHT_free`
- Each process provides part of its memory as memory windows (`MPI_Win`) to store key-value pairs
- All memory operations are done via `MPI_Put` and `MPI_Get` (RDMA)
- Data handling is done by requesting process

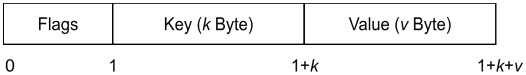
Key Requirements

for Distributed Key-Value Stores

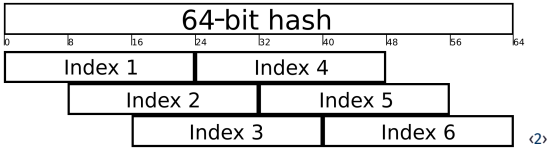
1. Design of **Addressing**: How to get the (memory) address of a key-value pair?
2. Design of **Collision Handling**: How to deal with hash collisions?
3. Design of **Data Consistency**: How to ensure data consistency in a concurrent environment?

MPI-DHT

Addressing: Calculating 64-bit hash of the key $\rightarrow$ Derive target process rank and index within target memory window by modulo operation to determine a bucket



Collision Handling: Each process has B buckets $\rightarrow$ Find smallest i, such that $B \leq 2^i$.
Round i up to next multiple of 8.
Derive multiple indices by moving hash pointer.



[2]

[2] De Lucia et al., "POET (v0.1): Speedup of Many-Cores Parallel Reactive Transport Simulations with Fast DHT-Lookups".

MPI-DHT

Data Consistency:

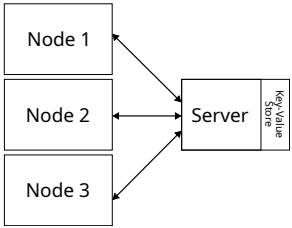
- Readers & Writers semantic
- Before access: Locking of the whole remote memory window via `MPI_Win_lock`
- For *read access* $\rightarrow$ shared lock, for *write access* $\rightarrow$ exclusive lock
  $\hookrightarrow$ Let the MPI implementation handle synchronization of concurrent accesses
- After access: Unlocking via `MPI_Win_unlock`

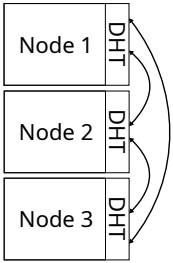
$\rightarrow$ Refer as coarsed-grained locking MPI-DHT

Compare MPI-DHT with other implementations

Choosing a Key-Value Store

- There are plenty of RDMA-accelerated Key-Value stores out there
  ↪ HERD, FaRM, Pilaf, Memcached, Redis, DAOS ...
- Most of them are not publicly available or not maintained anymore
- Also a question of use case: **server-based** vs. **fully distributed**





→ We selected **DAOS** for comparison

DAOS

- Distributed Asynchronous Object Storage system[3]
  $\hookrightarrow$ Usage of Key-Value store
- Client-Server architecture with dedicated server processes
- RDMA via `libfabric` or UCX
- *Read*: Client sends RPC to server, server responds via RDMA `Put`
- *Write*: Client sends RPC to server, server reads data via RDMA `Get` and writes to local storage
  $\rightarrow$ Messages $<$ 18 KByte are sent eagerly within RPC

[3] Zhen Liang et al. "DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory". In: *Supercomputing Frontiers*. Ed. by Dhabaleswar K. Panda. Lecture Notes in Computer Science. 2020, pp. 40–54. ISBN: 978-3-030-48842-0. DOI: 10.1007/978-3-030-48842-0_3.

MPI-DHT vs. DAOS

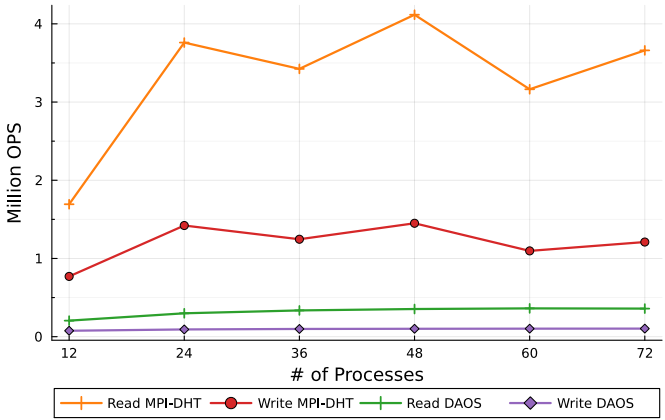
Test bed and Methodology

Minimal Test bed:

- 4 Nodes, each equipped with 2 $\times$ Intel Xeon 5-26540v4 (12 cores, 2.4 GHz)
- 64 GByte RAM
- RoCE (ConnectX-6 Dx 100 Gbit/s)
- Rocky Linux 8.8, Mellanox OFED 1.0.1, Open MPI 4.1.5a1

- Separate benchmarking of 100,000 `Write` and `Read` operations per process
- Scaling Clients up to 72 processes (3 Nodes)
- 80 Bytes Key, 104 Bytes Value
- Random generation of keys (uniform distribution)
- 5 repetitions, median values of throughput

MPI-DHT vs. DAOS



→ **Can we achieve better scaling with MPI-DHT?**

Improving Synchronization

Implement two new approaches to reduce synchronization overhead while maintaining **data consistency**:

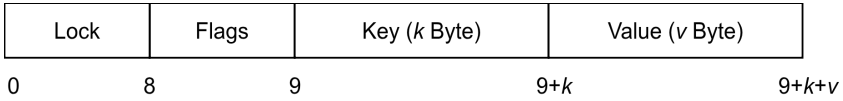
- Fine-grained locking: Use MPI's atomic operations to implement locking of buckets
- Lock-free: Optimistic concurrency control with checksums, adopted from Pilaf[4]

For **addressing** and **collision handling** we keep the original design.

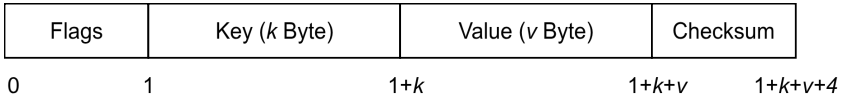
[4] Christopher Mitchell, Yifeng Geng, and Jinyang Li. "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store". In: *Proceedings of the 2013 USENIX Conference on Annual Technical Conference.* USENIX ATC'13. San Jose, CA, 2013, pp. 103–114.

Different Synchronization Approaches

Fine-grained locking:



Lock-free:



Evaluation

Test bed

HPC2024 Cluster at PIK:

- 2 $\times$ AMD EPYC 9554 (64 Cores, 3.1 GHz)
- 768 GByte RAM
- InfiniBand NDR (ConnectX-7 400 Gbit/s per port)
- GCC 14.1, Open MPI 5.0.6., UCX 1.17.0
- Using `osc_ucx_acc_single_intrinsic` for atomic operations

Evaluation

Methodology

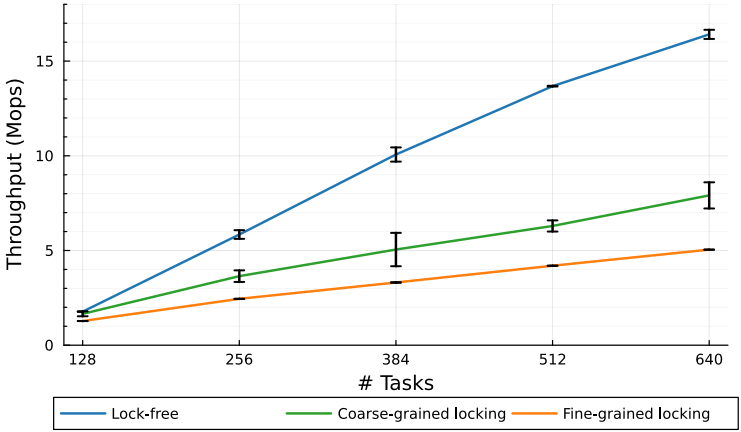
Two experiments, scaling up to 640 processes (5 Nodes):

1. Separate benchmarking of 500,000 `Write` and `Read` operations per process
2. Benchmarking of mixed workload with 95% `Read` and 5% `Write` operations; 1,000,000 operations in total

- 80 Bytes Key, 104 Bytes Value
- Random generation of keys (uniform **and** zipfian distribution)
- 5 repetitions, median values of throughput

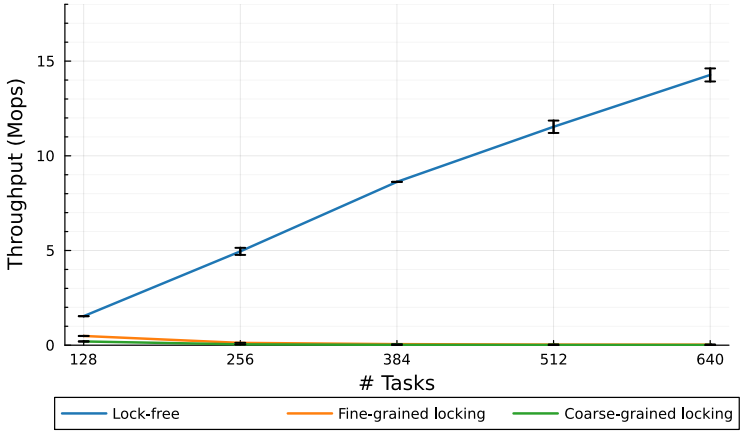
Uniform Distribution

Read Only



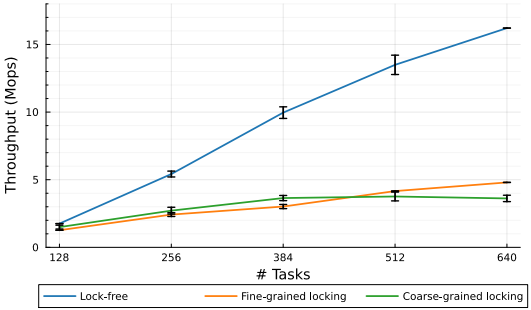
Zipfian Distribution

Write Only

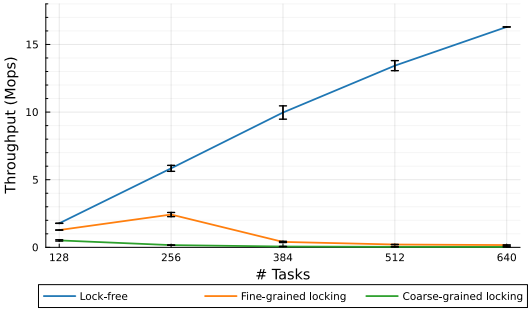


Mixed Workload (95% Read, 5% Write)

Uniform vs. Zipfian Distribution



Uniform Distribution



Zipfian Distribution

$\rightarrow$ **In maximum $1.1 \cdot 10^{-5}\%$ checksum mismatches**

POET

Performance Evaluation

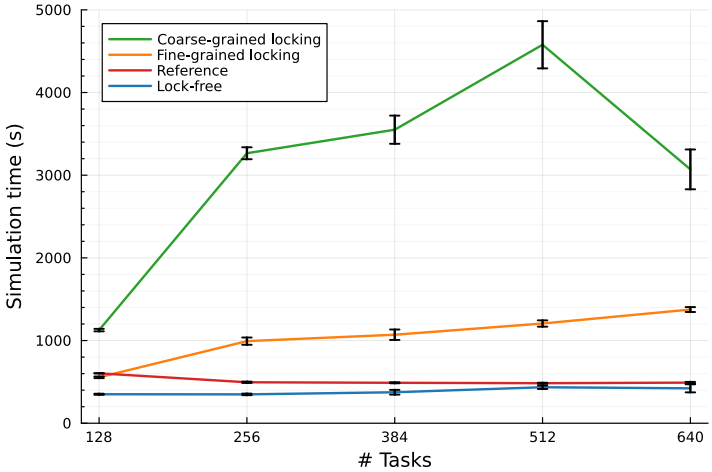
Benchmarking again on the HPC2024 Cluster at PIK with up to 5 Nodes (640 processes):

- 500 $\times$ 1500 Grid
- *Transport*: explicit upwind advection, constant flux
- *Chemistry*: homogeneous equilibrium of Calcite
- Injection of magnesium chloride $\rightarrow$ Dissolution of Calcite, Precipitation of Dolomite
- 500 Time Steps
- 3 repetitions, median values of chemistry runtime

$\rightarrow$ Compare implementations of MPI-DHT with POET's runtime without surrogate model

POET

Performance Evaluation



Further details:

- 375,000,000 DHT_read operations
- Hit rate of 91.8 %
- Maximum of $1.3 \cdot 10^{-3}\%$ checksum mismatches

Conclusion and Future Work

- MPI-DHT outperformed DAOS in our benchmarks
- Coarse-grained locking does not scale
- Introducing fine-grained locking and lock-free approach
- Lock-free approach shows throughput improvements up to factor 3 for read operations, and up to factor 1,400 for write operations
- Runtime of POET reduced by up to 42% using lock-free MPI-DHT as surrogate model

Future Work:

- Improving collision handling
- Compare against other distributed key-value stores
- Benchmark other benchmark scenarios for POET

Thank you for your attention!

The work was published in the Proceedings of the 25th International Conference on Computational Science (ICCS) as "*A fast MPI-based Distributed Hash-Table as Surrogate Model for HPC Applications*"

Max Lübke (mluebke@uni-potsdam.de)

Hagen, September 18, 2025