A fast MPI-based Distributed Hash-Table as Surrogate Model for HPC Applications

Max Lübke[1][0009−0008−9773−3038], Marco De Lucia[2][0000−0002−1186−4491], Stefan Petri[3][0000−0002−4379−4643], and Bettina Schnor[1][0000−0001−7369−8057]

[1] University of Potsdam, Institute of Computer Science, An der Bahn 2, 14476 Potsdam, Germany
{mluebke,schnor}@uni-potsdam.de
[2] GFZ Helmholtz Centre for Geosciences, Telegrafenberg, 14473 Potsdam, Germany
delucia@gfz.de
[3] Potsdam Institute for Climate Impact Research, Member of the Leibniz Association, Telegrafenberg, 14473 Potsdam, Germany
petri@pik-potsdam.de

Abstract. Surrogate models can enhance performance in HPC applications. Cache-based surrogates use pre-calculated simulation results for interpolation or extrapolation. However, this is only effective if retrieval is faster than simulation. This paper proposes an MPI-based distributed architecture where parallel processes share memory to build a distributed hash table. Three DHT approaches for HPC are presented. A lock-free design outperforms both coarse-grained and fine-grained locking DHTs, demonstrating good scaling for read and write performance. The lock-free DHT improved the runtime of a reactive transport simulation up to 42%.

Keywords: Distributed Hash Table · Key-Value Store · Surrogate Model · RDMA

1 Introduction

During the last years, several research groups have demonstrated the benefits of surrogate models for accelerating parallel multi-physics simulations. While some propose physics-informed neural networks, others use fast caches within their surrogate approach [15, 8, 2, 3]. For example, Reaktoro employs On-Demand Machine Learning (ODML) to speed up chemical calculations by extrapolating new chemical states [8]. POET utilizes a distributed hash table (DHT) as fast storage for simulation results, which are reused to approximate subsequent results [2]. However, the efficiency of such cache-based surrogates hinges on query and retrieval times being significantly faster than full physics simulations. Additionally, using a surrogate model results in a trade-off between runtime and accuracy.

The performance of distributed key-value stores has been a focus of much research, including *addressing*, *data consistency*, and *collision handling*. Another

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

key advancement has been the adoption of RDMA-capable networks to accelerate communication [6, 14, 4, 7, 1, 5, 12, 2]. While most key-value stores are server-based [4, 14, 7, 6, 9], this architecture is suboptimal for HPC due to additional hardware and setup requirements. Server-based approaches offer data consistency, but distributed approaches can enable direct data access from remote storage using RDMA, at the cost of requiring additional synchronization protocols. Due to space restrictions, we refer to a deeper disucssion of server-based key-value stores and related work in [10].

MPI, the de facto standard in HPC, offers an RMA API [13, pp. 547], making it a natural choice for implementing key-value stores to leverage HPC advancements [5, 12]. An MPI-based DHT was previously integrated into the POET simulator [2].

This paper explores fully distributed key-value store architectures based on MPI for seamless integration into scientific applications. The contributions of this work are:

- Presenting three distributed hash-table architectures using the MPI one-sided communication API: two synchronized and one lock-free approach.
- Evaluating the three approaches, demonstrating the performance advantages of the lock-free approach by up to 1,400 times in a synthetic benchmark.
- The lock-free DHT's excellent scaling for read and write requests, achieving 16 million read and 15 million write operations per second with 640 processes.
- Integrating the lock-free DHT as a fast data cache into a reactive transport simulation, results in runtime improvements of up to 42%.

2 Improving Synchronization Methods for Distributed Hash Tables with MPI

The MPI-DHT API [2] consists of four operations: `DHT_create`, `DHT_read`, `DHT_write`, and `DHT_free`. `DHT_read` and `DHT_write` use MPI's one-sided communication operations `MPI_Put` and `MPI_Get` to access remote memory. This section presents three approaches to solve the *data consistency* problem:

- *Coarse-grained locking*: The original MPI-DHT, employing MPI's passive target synchronization.
- *Fine-grained locking*: An MPI-DHT variant utilizing MPI's atomic operations for explicit synchronization.
- *Lock-free*: MPI-DHT with optimistic concurrency control, using checksums for conflict detection.

The MPI library was chosen for its wide use in HPC, providing a "plug-and-play" solution for MPI-parallelized applications.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

2.1 Coarse-Grained Lock-Based MPI-DHT

The original implementation uses MPI's passive target synchronization with `MPI_Win_lock` and `MPI_Win_unlock` [2]. Each process allocates memory, which is divided into buckets containing a key-value pair and metadata, and which is shared with the other processes using a `MPI_Window`. A bucket's address is a process rank and an index of the bucket within the memory window. A 64-bit hash sum of the key determines the target rank, and a set of bucket indices is derived from the hash sum.

For write operations, the target rank and possible bucket indices are calculated. The first bucket is checked; if unoccupied, data is written. If occupied, the next bucket is checked until an empty bucket or a matching key is found. Read operations similarly traverse buckets to find a matching key.

The MPI-DHT implements a Readers&Writers semantic. `DHT_read` or `DHT_write` operations lock the *entire* target rank's memory window in shared or exclusive mode, respectively.

2.2 Fine-Grained Lock-Based MPI-DHT

Here, adressing and collision handling are identical to the coarse-grained approach. However, instead of locking the entire memory window, the fine-grained mechanism locks only the bucket being accessed. To avoid excessive memory overhead from using MPI windows for each bucket, the implementation uses self-implemented locking on 8-byte integers with `MPI_Compare_and_swap` and `MPI_Fetch_and_op` operations, inspired by Open MPI's passive-target synchronization.

A lock value of $0x10000000$ indicates an active writer (exclusive lock). Write processes atomically set the lock to this value if it is currently zero. Readers increment the lock atomically until the value is less than $0x10000000$. If the value is greater or equal to $0x10000000$, the read request is revoked, and the process tries to acquire the lock again. This allows concurrent reads but exclusive writes. Locks are released by decrementing the lock value. An 8-byte lock is included in each bucket, requiring up to 7 bytes of padding, resulting in a maximum overhead of 15 bytes per bucket compared to the coarse-grained approach.

All windows are pre-locked with `MPI_Win_lock_all` during setup to ensure that the MPI standard is not violated by performing RMA operations outside an epoch [13, p. 588].

2.3 Lock-Free MPI-DHT

This approach uses the same addressing and collision handling but replaces locking with checksum calculation. Each bucket includes a 32-bit value for storing a checksum. The approach is inspired by the work of Pilaf [14].

For `DHT_write`, the origin process calculates a checksum of the key-value pair and appends it to the bucket data. A reading process retrieves the bucket data, recalculates the checksum, and returns the key-value pair if both checksums

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

match. Mismatches trigger a retry, and persistent mismatches invalidate the bucket. Write operations can overwrite invalid buckets. This checksum-based approach is a lock-free mechanism, avoiding atomic operations or locks. It adds only 4 bytes of memory overhead per bucket for the checksum compared to the coarse-grained approach.

Similar to the fine-grained approach, all windows are locked by all processes with `MPI_Win_lock_all` prior to any RMA operation.

3 Evaluation

3.1 Test Bed

The benchmarks were conducted on the cluster of the Potsdam Institute for Climate Impact Research (PIK). Each node has two AMD EPYC 9554 CPUs with 64 cores each and a base clock speed of 3.1 GHz. All 128 cores per node were used. The nodes share 768 GB of DDR5 memory and are interconnected via NVIDIA Mellanox ConnectX-7 NDR Infiniband (400 Gbps per port). The GNU Compiler Collection 14.1 and OpenMPI 5.0.6 with UCX 1.17.0 were used for the experiments. The code was compiled with `-O3 -DNDEBUG`. Open MPI was configured to use single atomic remote memory operations[4].

3.2 Synthetic Benchmarks

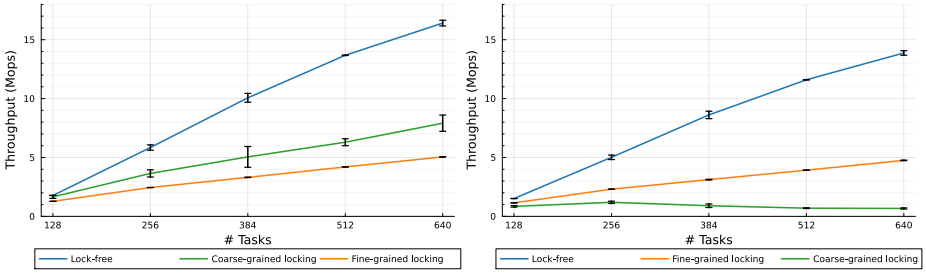
- **First experiment:** This benchmark evaluates maximum read/write throughput. It generates a random number to derive an 80-byte key, with a value size of 104 bytes (modeling POET's key-value pair size) [2]. Uniform and Zipfian distributions were used for random number generation. The Zipfian distribution (skew of 99, range 1 to 712,500) models POET's access requests. The benchmark writes and then reads 500,000 key-value pairs.
- **Second experiment:** This benchmark evaluates a more read-intensive mixed workload (95% reads, 5% writes), also reflecting POET's access pattern [2].

The experiments used 1 to 5 nodes (up to 640 processes), with each process providing 1 GB of memory for the DHT. The benchmark was replicated five times, and median throughput values, calculated as operations per second, with standard deviations are shown in Figures 1a and 1b (uniform distribution) and Figures 2a and 2b (Zipfian).

In the first experiment, the lock-free DHT outperformed both synchronized approaches in read-only and write-only benchmarks with both distributions. For example, with 640 processes, the lock-free DHT achieved over 16 million read operations per second, about 3 times higher than fine-grained locking and 2 times higher than coarse-grained locking. Write throughput was lower than read throughput for all approaches, as expected.

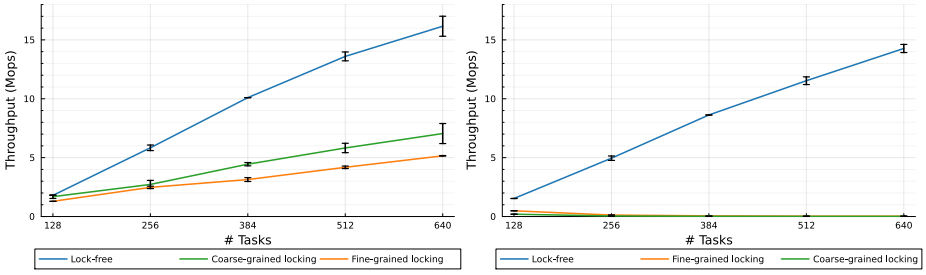
[4] See `osc_ucx_acc_single_intrinsic` of Open MPI's MCA configuration: `https://docs.open-mpi.org/en/main/mca.html`

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28



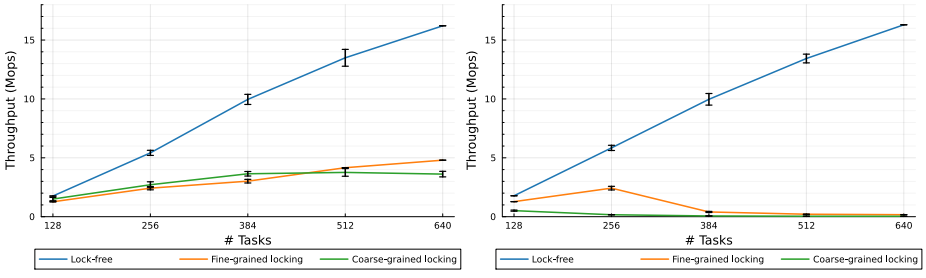
(a) Read only (b) Write only

Fig. 1. Throughput of read and write operations with *uniform* distributed keys.



(a) Read only (b) Write only

Fig. 2. Throughput of read and write operations with *zipfian* distributed keys.



(a) Uniform Distribution (b) Zipfian Distribution

Fig. 3. Throughput of mixed workload with a 95% Read/5% Write ratio for uniform and zipfian distributed keys.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

Table 1 shows write-only performance for 640 processes, demonstrating the overhead of locking. The lock-free DHT performed significantly better than the other approaches.

Table 1. First experiment: Write-only Performance for 640 Processes in Million Operations per Second.

Workload	Coarse-Grained	Fine-Grained	Lock-Free
uniform	0.67	4.75	13.9
zipfian	0.01	0.03	14.3

For the second mixed workload benchmark (Figure 3a and 3b), the lock-free DHT's throughput was close to its read-only performance. Fine-grained locking showed some improvement over coarse-grained locking with the uniform distribution, but both synchronized approaches were challenged by the Zipfian distribution. Table 2 shows checksum mismatches in the lock-free DHT. Mismatches occurred only with the Zipfian distribution, indicating concurrent writes, but with negligible results.

Table 2. Second experiment: Checksum mismatches for the lock-free DHT.

Workload	# of Tasks	# of Mismatches	Percentage [%]
mixed - Zipfian	128	13	$1.1 \cdot 10^{-5}$
mixed - Zipfian	256	16	$6.5 \cdot 10^{-6}$
mixed - Zipfian	384	25	$6.8 \cdot 10^{-6}$
mixed - Zipfian	512	31	$6.3 \cdot 10^{-6}$
mixed - Zipfian	640	64	$1.1 \cdot 10^{-5}$
Others	Any	0	0

3.3 HPC Use-Case: POET

POET is an MPI-parallelized reactive transport simulator that combines solute flow and transport in porous media with geochemical reactions. The simulation used an explicit upwind advection scheme on a 500×1500 grid with homogeneous species concentrations and the same chemistry setup as in previous work [2]. Due to advective transport, a sharp reaction front occurs, allowing caching of previously simulated results in the DHT. Input parameters for the geochemical simulation are rounded to serve as keys for the DHT, and stored values consist of the exact results. The simulation ran for 500 time steps, simulating flux, transport, and geochemical reaction.[5]

The simulation was scaled from 1 to 5 nodes, with one MPI task per CPU core. Experiments were repeated three times for each DHT approach and a reference without DHT. Median runtimes are shown in Figure 4. The average hit rate over all DHT runs was 91.8%.

The lock-free DHT was the only approach that improved simulation runtime. The best result was a 41.9% reduction in simulation time with 128 processes.

[5] The code is available: [11].

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

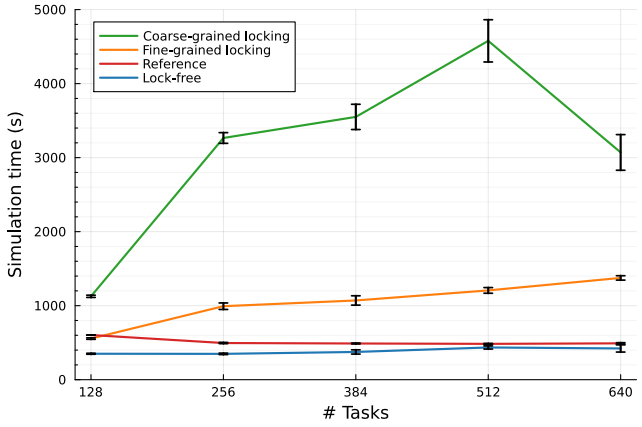


Fig. 4. Runtime of the chemical simulation of POET w/ and w/o DHT.

Table 3 shows the checksum mismatches observed during simulations with the lock-free DHT, which were minimal. Additionally, the relative performance gain is shown.

Table 3. Checksum mismatches and performance gain of the POET simulation with lock-free MPI-DHT compared to the Reference Run without DHT.

# of Tasks	# of Mismatches	Percentage [%]	Performance Gain [%]
128	1507	$4.4 \cdot 10^{-4}$	41.9%
256	3049	$8.9 \cdot 10^{-4}$	29.5%
384	4315	$1.3 \cdot 10^{-3}$	23.3%
512	2884	$8.4 \cdot 10^{-4}$	10.1%
640	4421	$1.3 \cdot 10^{-3}$	14.1%

4 Conclusion and Future Work

This paper compared MPI-based DHT implementations with coarse-grained locking, fine-grained locking, and lock-free approaches. The lock-free DHT outperformed the synchronized approaches in synthetic benchmarks, with read throughput of 16 million operations per second and write throughput of 15 million operations per second. In the POET simulation, the lock-free DHT improved runtime, reducing it up to 42%.

Acknowledgments. The authors gratefully acknowledge the Ministry of Research, Science and Culture (MWFK) of Land Brandenburg for supporting this project by providing resources on the high performance computer system at the Potsdam Institute for Climate Impact Research.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28

References

1. Cassell, B., Szepesi, T., Wong, B., Brecht, T., Ma, J., Liu, X.: Nessie: A Decoupled, Client-Driven Key-Value Store Using RDMA. IEEE Transactions on Parallel and Distributed Systems **28**(12), 3537–3552 (2017). https://doi.org/10.1109/TPDS.2017.2729545
2. De Lucia, M., Kühn, M., Lindemann, A., Lübke, M., Schnor, B.: POET (v0.1): Speedup of many-cores parallel reactive transport simulations with fast DHT-Lookups. Geoscientific Model Development **14**(12), 7391–7409 (2021). https://doi.org/10.5194/gmd-14-7391-2021
3. De Lucia, M., Kühn, M.: DecTree v1.0 – chemistry speedup in reactive transport simulations: Purely data-driven and physics-based surrogates. Geoscientific Model Development **14**(7), 4713–4730 (2021). https://doi.org/10.5194/gmd-14-4713-2021
4. Dragojević, A., Narayanan, D., Castro, M., Hodson, O.: FaRM: Fast Remote Memory. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). pp. 401–414 (2014)
5. Gerstenberger, R., Besta, M., Hoefler, T.: Enabling highly-scalable remote memory access programming with MPI-3 One Sided. Scientific Programming **22**(2), 75–91 (2014). https://doi.org/10.3233/SPR-140383
6. Jose, J., Subramoni, H., Luo, M., Zhang, M., Huang, J., Wasi-ur-Rahman, Md., Islam, N.S., Ouyang, X., Wang, H., Sur, S., Panda, D.K.: Memcached Design on High Performance RDMA Capable Interconnects . In: 2011 International Conference on Parallel Processing. pp. 743–752 (2011). https://doi.org/10.1109/ICPP.2011.37
7. Kalia, A., Kaminsky, M., Andersen, D.G.: Using RDMA efficiently for key-value services. In: Proceedings of the 2014 ACM Conference on SIGCOMM. pp. 295–306. Chicago Illinois USA (2014). https://doi.org/10.1145/2619239.2626299
8. Leal, A., Matculevich, S., Kulik, D., Saar, M.: Accelerating Reactive Transport Modeling: On-Demand Machine Learning Algorithm for Chemical Equilibrium Calculations. Transport in Porous Media **133** (2020). https://doi.org/10.1007/s11242-020-01412-1
9. Liang, Z., Lombardi, J., Chaarawi, M., Hennecke, M.: DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory. In: Panda, D.K. (ed.) Supercomputing Frontiers. pp. 40–54. Lecture Notes in Computer Science (2020). https://doi.org/10.1007/978-3-030-48842-0_3
10. Lübke, M., De Lucia, M., Petri, S., Schnor, B.: A fast MPI-based Distributed Hash-Table as Surrogate Model demonstrated in a coupled reactive transport HPC simulation (2025). https://doi.org/10.48550/arXiv.2504.14374
11. Lübke, M., De Lucia, M., Petri, S., Schnor, B.: MPI-DHT Source Code and Benchmarks (ICCS25). [Data set] (Apr 2025). https://doi.org/10.5281/zenodo.15235555
12. Maynard, C.M.: Comparing UPC and one-sided MPI: A distributed hash table for GAP. Tech. rep., Cray User Group (GUG) (2012), `https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap195.pdf`
13. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Vers. 4.1 (2023), `https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf`
14. Mitchell, C., Geng, Y., Li, J.: 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. pp. 103–114. USENIX ATC'13, USA (2013)
15. Raissi, M., Perdikaris, P., Karniadakis, G.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_28