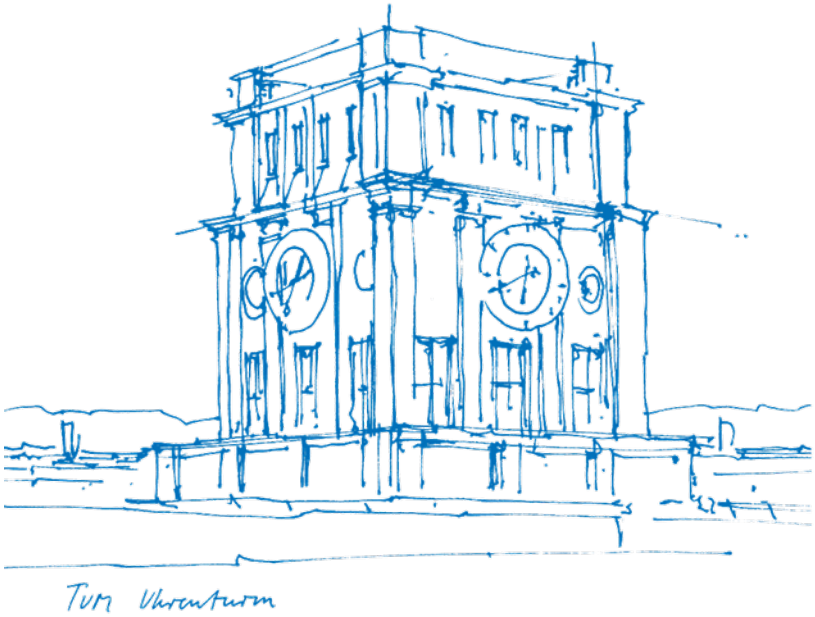
Chair III: Database Systems
Chair XXV: Data Science and Engineering
Department of Informatics
Technical University of Munich

One Buffer Manager to Rule Them All: Using Distributed Memory with Cache Coherence over RDMA

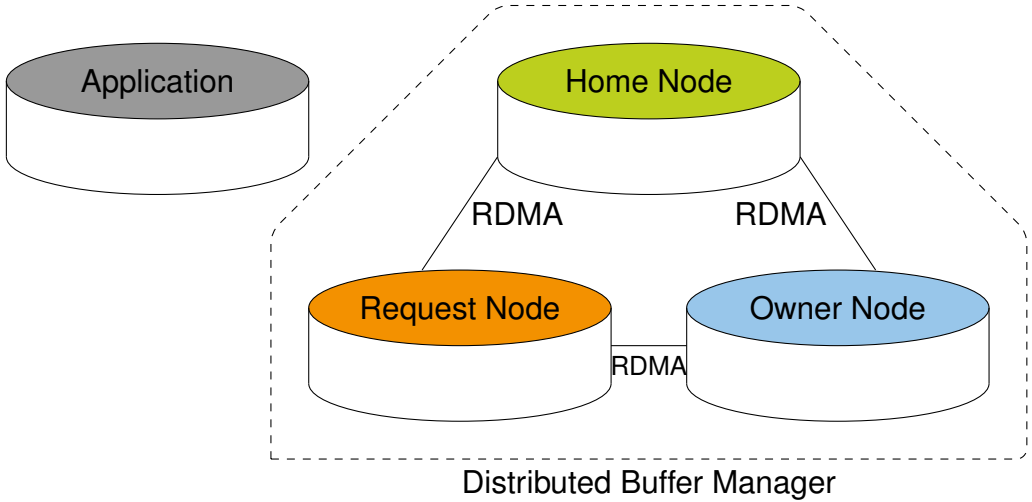
Magdalena Pröbstl, Philipp Fent, Maximilian E. Schüle,
Moritz Sichert, Thomas Neumann, Alfons Kemper

Copenhagen, Denmark, August 16, 2021



Distributed Buffer Manager

- **Goal:** combine traditional database management systems to efficient distributed data processing
- **Solution:** combine a canonical buffer pool with distributed memory access



Related Work: DBMS and RDMA

Database Systems and RDMA (Remote Direct Memory Access)

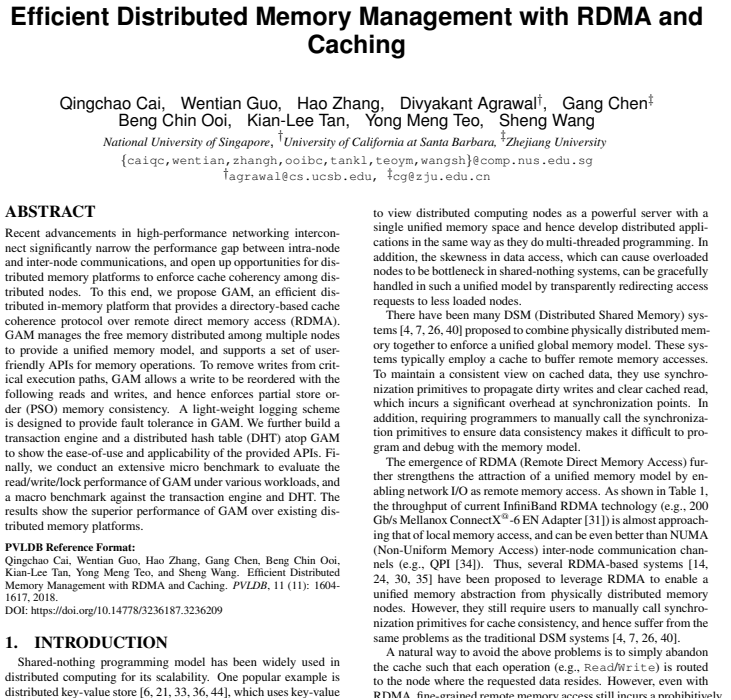
- in-memory join processing (Barthels et al., SIGMOD 2015)
- distributed index structures (Zieger et al., SIGMOD 2019)
- decentralized lock management (Yoon et al., SIGMOD 2018)

GAM (VLDB 2018)

- Cache coherence protocol over RDMA
- Unified memory model over multiple nodes
- Interface abstracts lock management
- Limited to main memory only

This work

- Extension of cache coherence to background storage (converging performance characteristics of SSDs and RDMA)
- Distributed buffer manager to abstract the memory layout from the application's perspective



Efficient Distributed Memory Management with RDMA and Caching

Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal[‡], Gang Chen[‡]
Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, Sheng Wang
National University of Singapore, [†]*University of California at Santa Barbara*, [‡]*Zhejiang University*
{caiqc,wentian,zhangh,ooibc,tankl,teoym,wangsh}@comp.nus.edu.sg
[†]agrawal@cs.ucsb.edu, [‡]cg@zju.edu.cn

ABSTRACT

Recent advancements in high-performance networking interconnect significantly narrow the performance gap between intra-node and inter-node communications, and open up opportunities for distributed memory platforms to enforce cache coherency among distributed nodes. To this end, we propose GAM, an efficient distributed in-memory platform that provides a directory-based cache coherence protocol over remote direct memory access (RDMA). GAM manages the free memory distributed among multiple nodes to provide a unified memory model, and supports a set of user-friendly APIs for memory operations. To remove writes from critical execution paths, GAM allows a write to be reordered with the following reads and writes, and hence enforces partial store order (PSO) memory consistency. A light-weight logging scheme is designed to provide fault tolerance in GAM. We further build a transaction engine and a distributed hash table (DHT) atop GAM to show the ease-of-use and applicability of the provided APIs. Finally, we conduct an extensive micro benchmark to evaluate the read/write/lock performance of GAM under various workloads, and a macro benchmark against the transaction engine and DHT. The results show the superior performance of GAM over existing distributed memory platforms.

PVLDB Reference Format:
Qingchao Cai, Wentian Guo, Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient Distributed Memory Management with RDMA and Caching. *PVLDB*, 11 (11): 1604-1617, 2018.
DOI: https://doi.org/10.14778/3236187.3236209

1. INTRODUCTION

Shared-nothing programming model has been widely used in distributed computing for its scalability. One popular example is distributed key-value store [6, 21, 33, 36, 44], which uses key-value to view distributed computing nodes as a powerful server with a single unified memory space and hence develop distributed applications in the same way as they do multi-threaded programming. In addition, the skewness in data access, which can cause overloaded nodes to be bottleneck in shared-nothing systems, can be gracefully handled in such a unified model by transparently redirecting access requests to less loaded nodes.

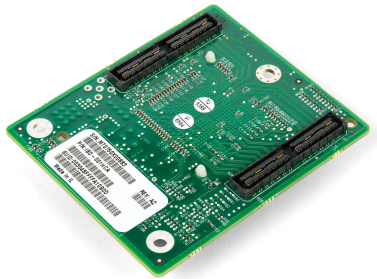
There have been many DSM (Distributed Shared Memory) systems [4, 7, 26, 40] proposed to combine physically distributed memory together to enforce a unified global memory model. These systems typically employ a cache to buffer remote memory accesses. To maintain a consistent view on cached data, they use synchronization primitives to propagate dirty writes and clear cached read, which incurs a significant overhead at synchronization points. In addition, requiring programmers to manually call the synchronization primitives to ensure data consistency makes it difficult to program and debug with the memory model.

The emergence of RDMA (Remote Direct Memory Access) further strengthens the attraction of a unified memory model by enabling network I/O as remote memory access. As shown in Table 1, the throughput of current InfiniBand RDMA technology (e.g., 200 Gb/s Mellanox ConnectX$^{®}$-6 EN Adapter [31]) is almost approaching that of local memory access, and can be even better than NUMA (Non-Uniform Memory Access) inter-node communication channels (e.g., QPI [34]). Thus, several RDMA-based systems [14, 24, 30, 35] have been proposed to leverage RDMA to enable a unified memory abstraction from physically distributed memory nodes. However, they still require users to manually call synchronization primitives for cache consistency, and hence suffer from the same problems as the traditional DSM systems [4, 7, 26, 40].

A natural way to avoid the above problems is to simply abandon the cache such that each operation (e.g., `Read`/`Write`) is routed to the node where the requested data resides. However, even with RDMA, fine-grained remote memory access still incurs a prohibitively

Structure

How to combine distributed memory with traditional page caching in a buffer manager?



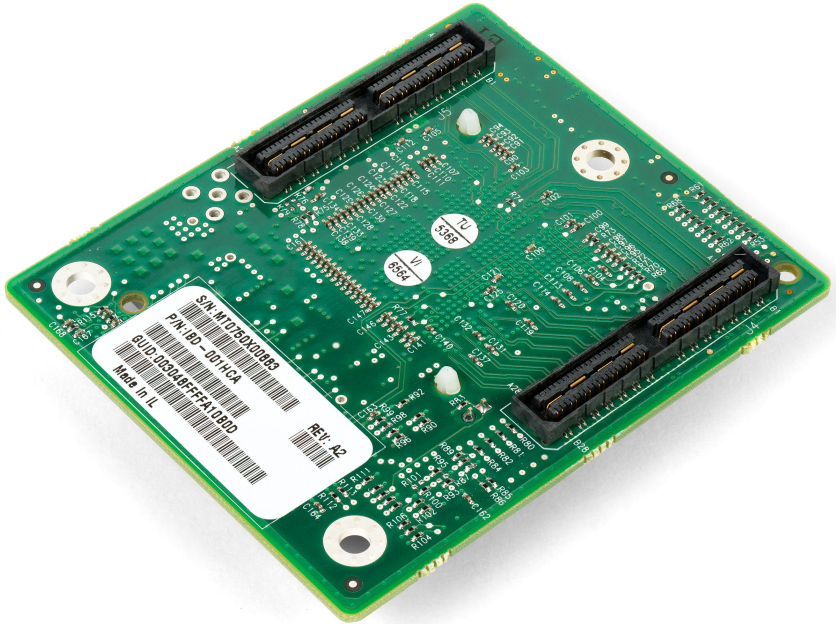
CC BY-SA 4.0 Dmitry Nosachev

Cache Coherence Protocol
Interface for I/O primitives
Transition states

Distributed Buffer Manager
Traditional Approach
Global Buffer Pool

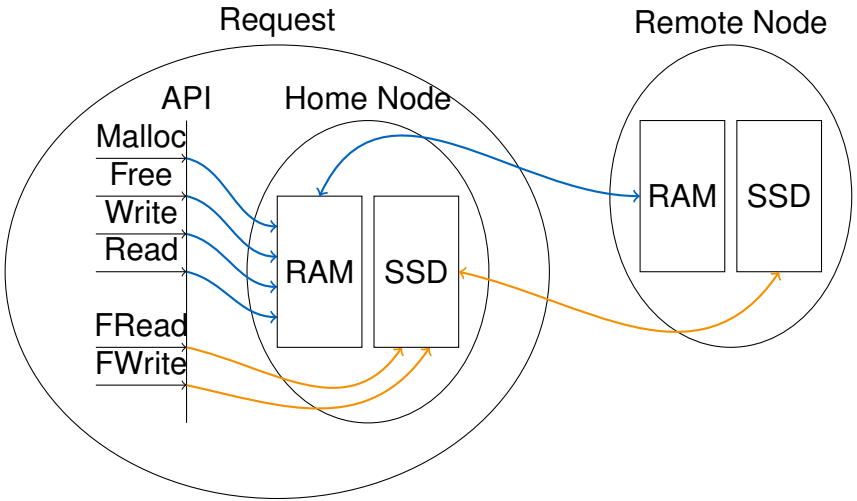
Evaluation
Micro-Benchmarks
Hash Table and Buffer Manager

Cache Coherence Protocol



Cache Coherence Protocol: Interface

- hide the memory layout from the application's perspective
- each entity: node
  - interface for remote reads and writes
  - node ID: for location, locks and responsibilities for data
- partitioned global address space (PGAS): unified view
  - size,
  - the pointer to the node (hosts the data),
  - node ID,
  - flag: main memory/background storage.

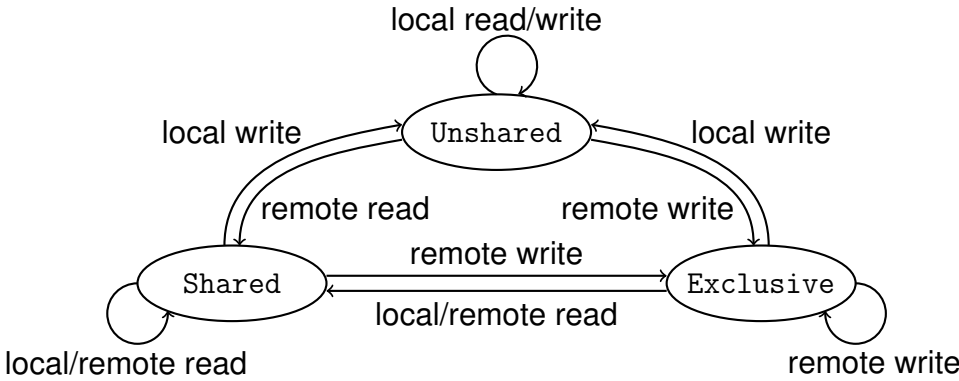


Listing 1: The struct `GlobalAddress` to identify distributed main memory and files globally.

```
struct __attribute__ ((packed)) GlobalAddress {
  size_t size; char *ptr; uint16_t id; bool isFile; }
```

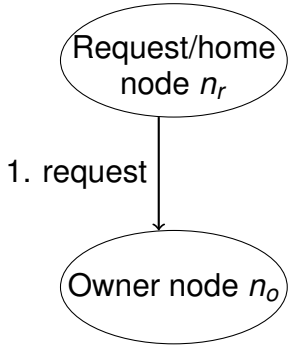
Cache Coherence Protocol: Cache

- remote memory access: 10x times slower (vs. local memory) access
- LRU (least recently used) software cache aims to reduce the remote memory accesses
- hash-table indexed by its global address
- 5 roles for a node
  - *home node*: where data is stored
  - *remote nodes*: want data (other nodes)
  - *request node*: requests shared/exclusive access to data
  - *sharing node*: request node with shared access to data
  - *owner node*: request node with exclusive access
- 3 states for data
  - *unshared*: data resides in the home node
  - *shared*: one or more nodes have read permission
  - *exclusive*: only one node



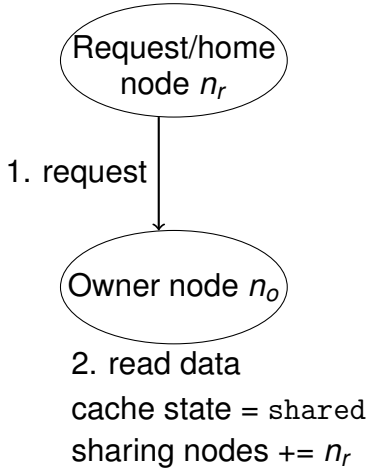
Cache Coherence Protocol: Read

- **Local Read:** data can be accessed right away. The status remains unchanged (no additional sharing node).
- **Remote Read:** request to the associated remote node, status set to shared, data gets cached



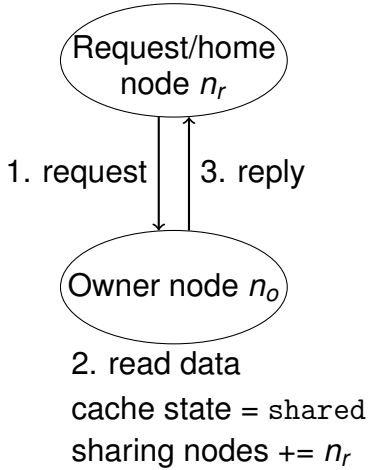
Cache Coherence Protocol: Read

- **Local Read:** data can be accessed right away. The status remains unchanged (no additional sharing node).
- **Remote Read:** request to the associated remote node, status set to shared, data gets cached



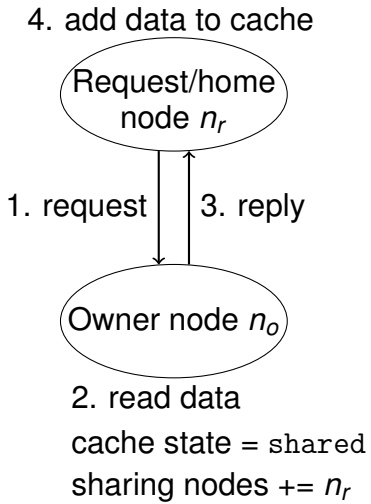
Cache Coherence Protocol: Read

- **Local Read:** data can be accessed right away. The status remains unchanged (no additional sharing node).
- **Remote Read:** request to the associated remote node, status set to shared, data gets cached



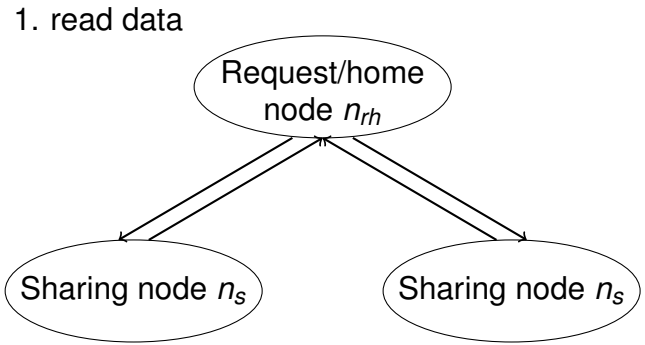
Cache Coherence Protocol: Read

- **Local Read:** data can be accessed right away. The status remains unchanged (no additional sharing node).
- **Remote Read:** request to the associated remote node, status set to shared, data gets cached



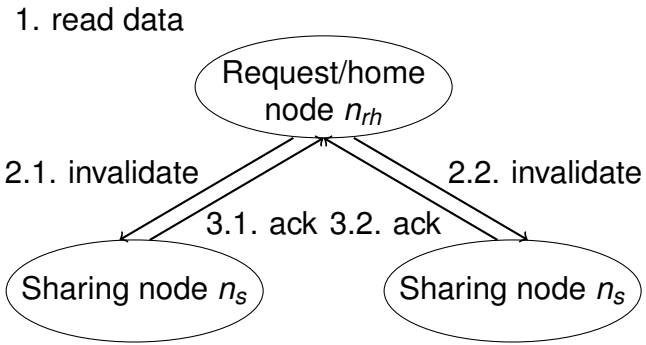
Cache Coherence Protocol: Local Write

- memory is placed locally
- check if the data is already cached by another node
- send invalidate requests to all sharing nodes



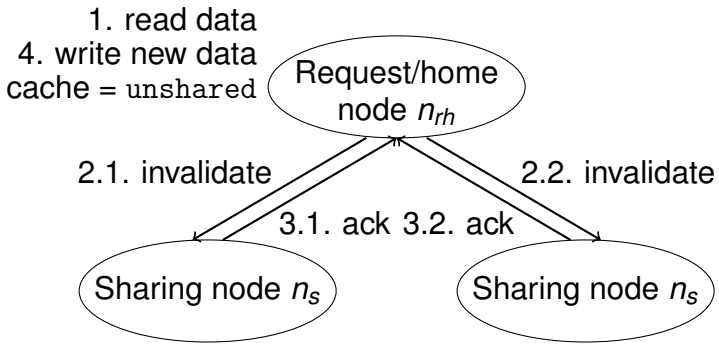
Cache Coherence Protocol: Local Write

- memory is placed locally
- check if the data is already cached by another node
- send invalidate requests to all sharing nodes



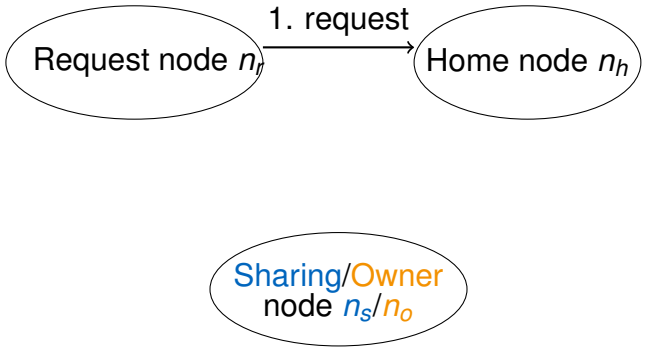
Cache Coherence Protocol: Local Write

- memory is placed locally
- check if the data is already cached by another node
- send invalidate requests to all sharing nodes



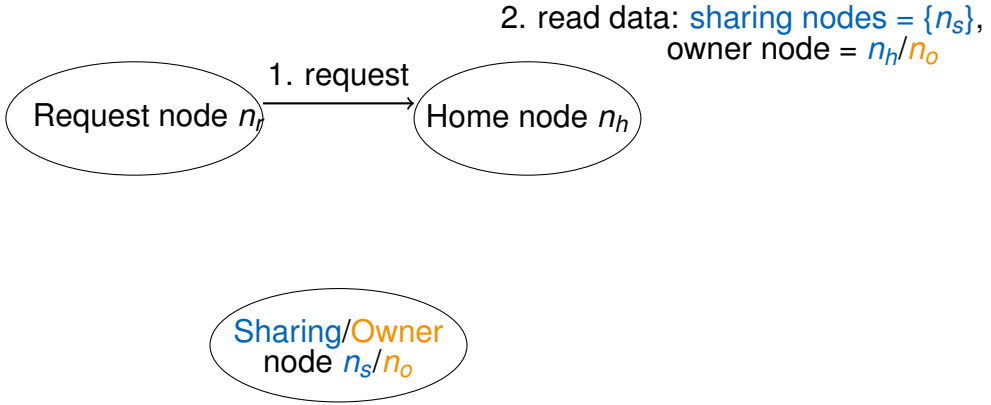
Cache Coherence Protocol: Remote Write

- memory is placed in a different node
- request node connects to the home node and sends a write request
- 3 cases
  - *unshared*: data is stored; state set to exclusive
  - *shared*: invalidate cached item on each sharing node
  - *exclusive*: the data cannot be changed; the request will be denied



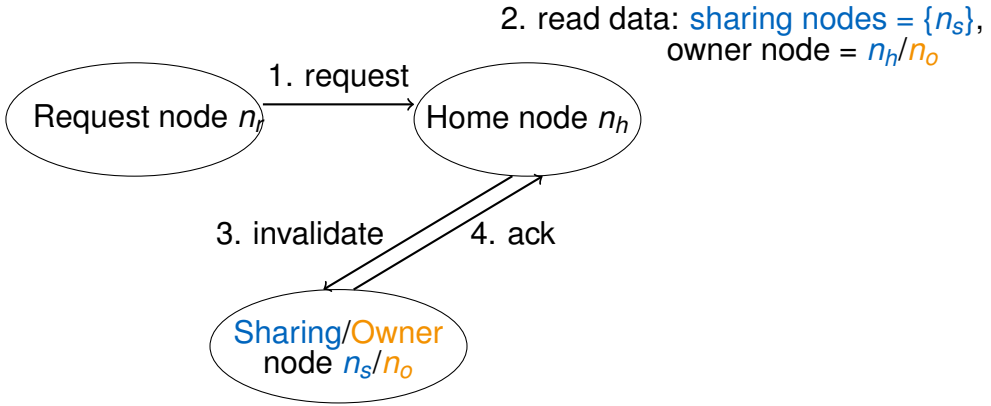
Cache Coherence Protocol: Remote Write

- memory is placed in a different node
- request node connects to the home node and sends a write request
- 3 cases
  - *unshared*: data is stored; state set to exclusive
  - *shared*: invalidate cached item on each sharing node
  - *exclusive*: the data cannot be changed; the request will be denied



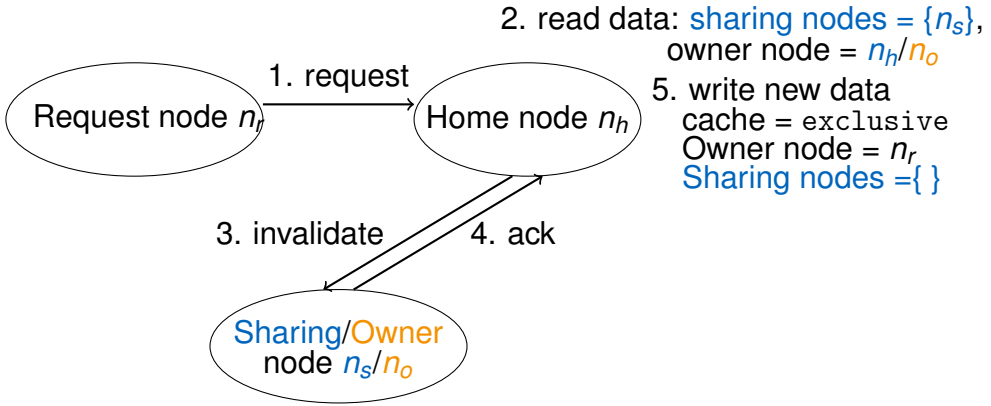
Cache Coherence Protocol: Remote Write

- memory is placed in a different node
- request node connects to the home node and sends a write request
- 3 cases
  - *unshared*: data is stored; state set to exclusive
  - *shared*: invalidate cached item on each sharing node
  - *exclusive*: the data cannot be changed; the request will be denied



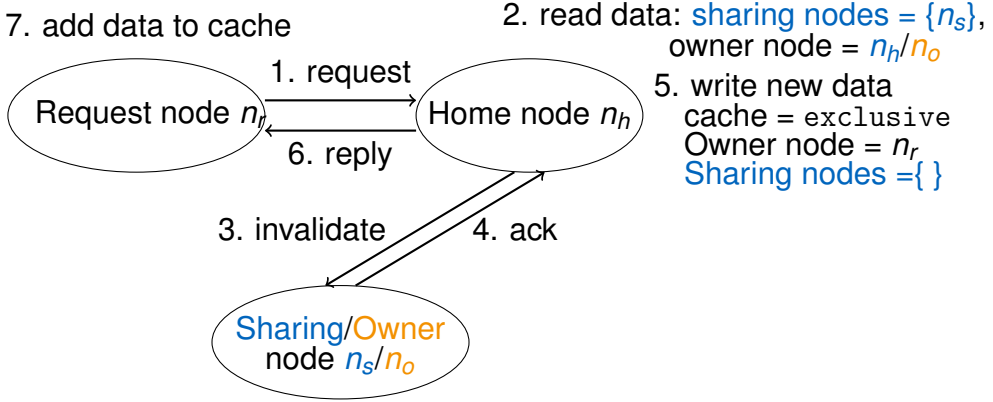
Cache Coherence Protocol: Remote Write

- memory is placed in a different node
- request node connects to the home node and sends a write request
- 3 cases
  - *unshared*: data is stored; state set to exclusive
  - *shared*: invalidate cached item on each sharing node
  - *exclusive*: the data cannot be changed; the request will be denied



Cache Coherence Protocol: Remote Write

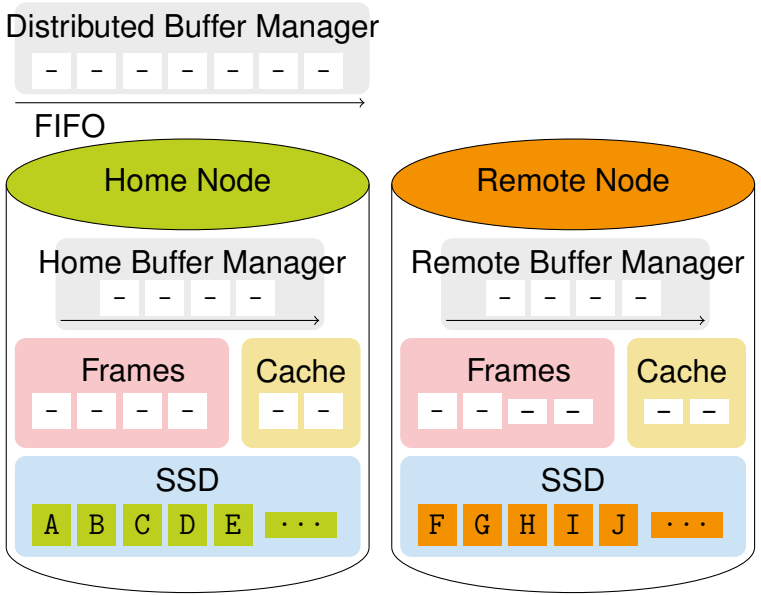
- memory is placed in a different node
- request node connects to the home node and sends a write request
- 3 cases
  - *unshared*: data is stored; state set to exclusive
  - *shared*: invalidate cached item on each sharing node
  - *exclusive*: the data cannot be changed; the request will be denied



Distributed Buffer Manager

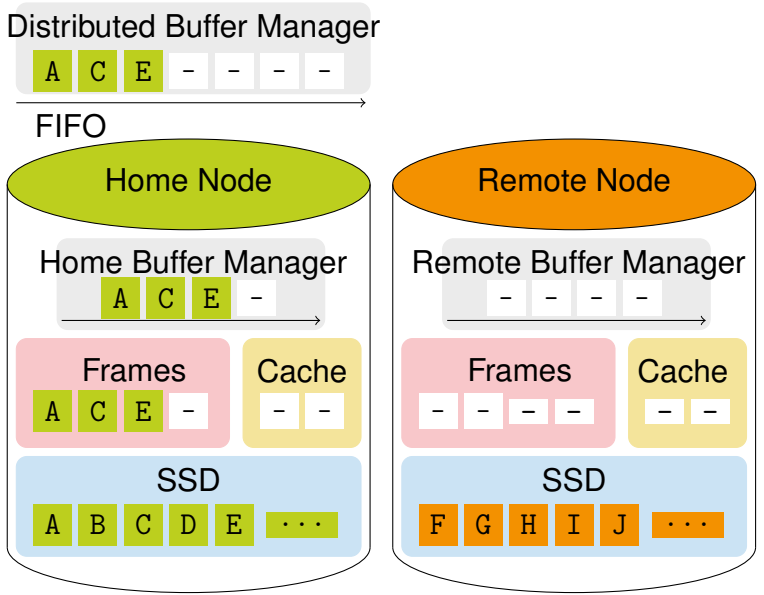
Distributed Buffer Manager: Traditional Approach

- every node provides a local buffer manager
- evicts pages to the local background storage
- global buffer manager = amount of local buffer managers



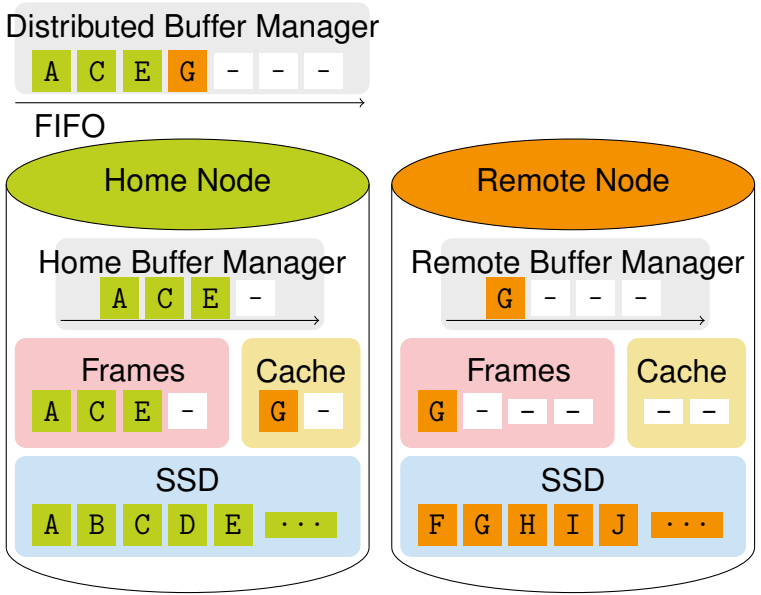
Distributed Buffer Manager: Traditional Approach

- every node provides a local buffer manager
- evicts pages to the local background storage
- global buffer manager = amount of local buffer managers



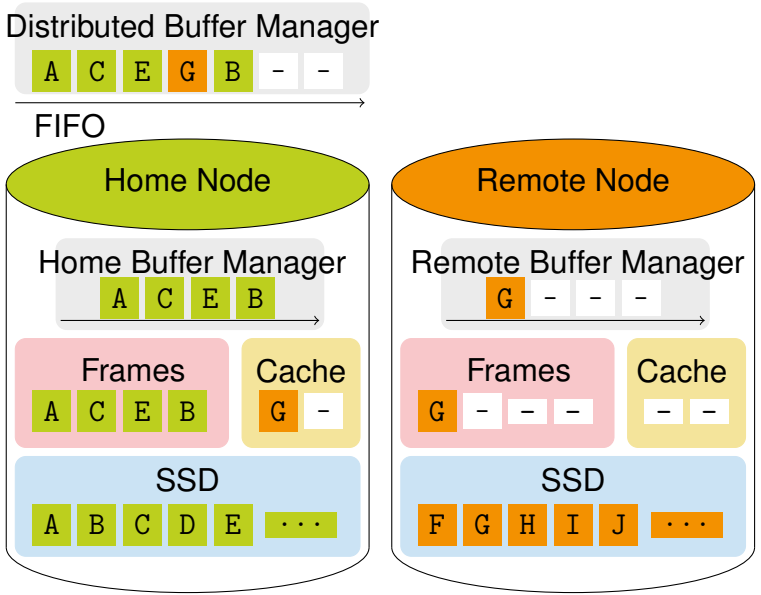
Distributed Buffer Manager: Traditional Approach

- every node provides a local buffer manager
- evicts pages to the local background storage
- global buffer manager = amount of local buffer managers



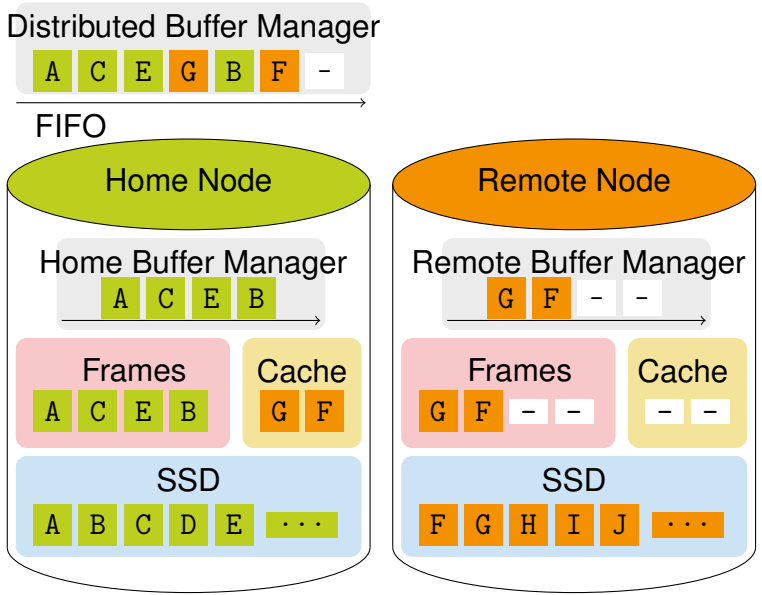
Distributed Buffer Manager: Traditional Approach

- every node provides a local buffer manager
- evicts pages to the local background storage
- global buffer manager = amount of local buffer managers



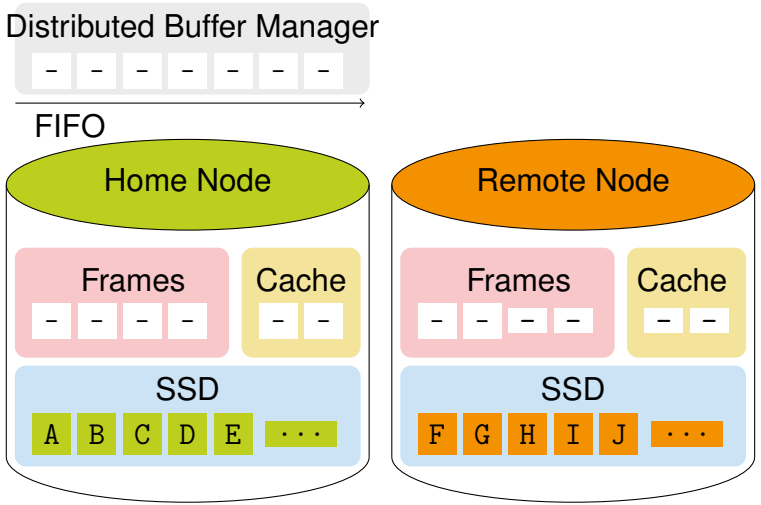
Distributed Buffer Manager: Traditional Approach

- every node provides a local buffer manager
- evicts pages to the local background storage
- global buffer manager = amount of local buffer managers



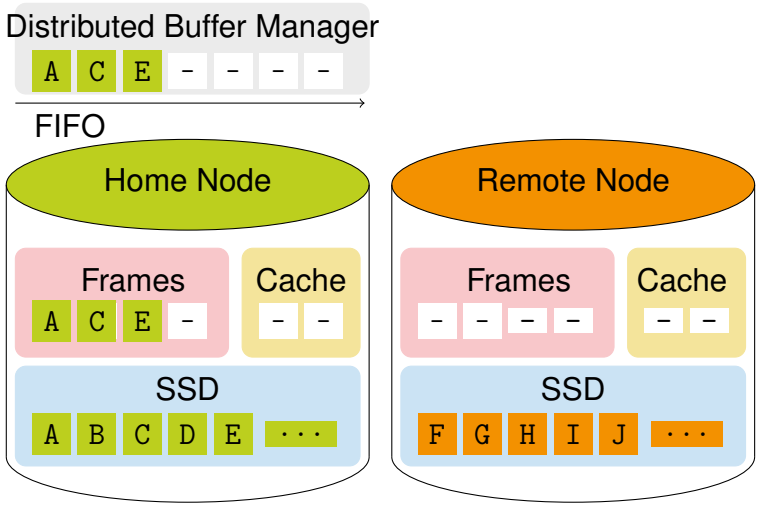
Distributed Buffer Manager: Global Buffer Pool

- RDMA: bypass CPU to access remote memory
- load pages either from a home or remote SSD first into frames of the home node
- If frames run out, frames of the remote node are taken and the page is cached locally according to the cache coherence protocol.
- global buffer manager = amount of loaded frames
- two queue eviction strategy (FIFO and LRU)
- strategy: fully utilizing the distributed buffer pool before swapping out pages
- coherence protocol responsible for caching and locking



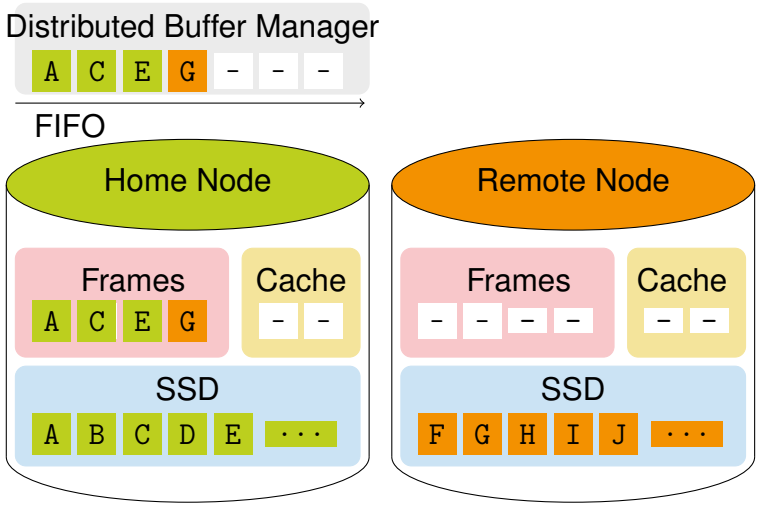
Distributed Buffer Manager: Global Buffer Pool

- RDMA: bypass CPU to access remote memory
- load pages either from a home or remote SSD first into frames of the home node
- If frames run out, frames of the remote node are taken and the page is cached locally according to the cache coherence protocol.
- global buffer manager = amount of loaded frames
- two queue eviction strategy (FIFO and LRU)
- strategy: fully utilizing the distributed buffer pool before swapping out pages
- coherence protocol responsible for caching and locking



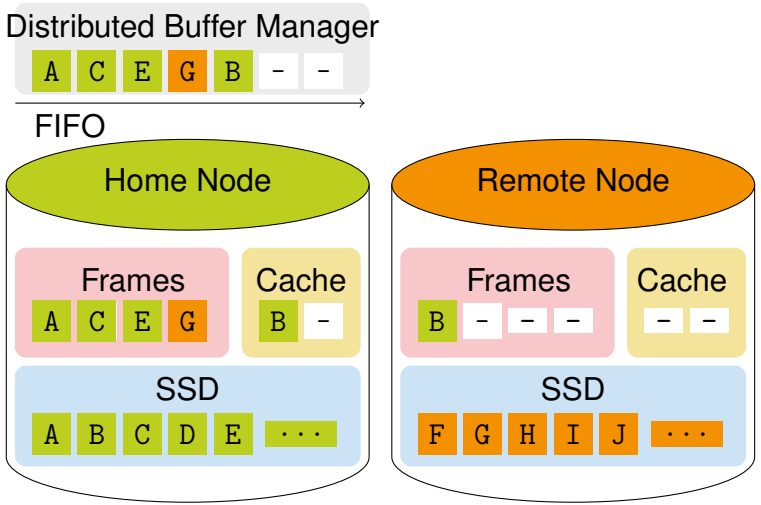
Distributed Buffer Manager: Global Buffer Pool

- RDMA: bypass CPU to access remote memory
- load pages either from a home or remote SSD first into frames of the home node
- If frames run out, frames of the remote node are taken and the page is cached locally according to the cache coherence protocol.
- global buffer manager = amount of loaded frames
- two queue eviction strategy (FIFO and LRU)
- strategy: fully utilizing the distributed buffer pool before swapping out pages
- coherence protocol responsible for caching and locking



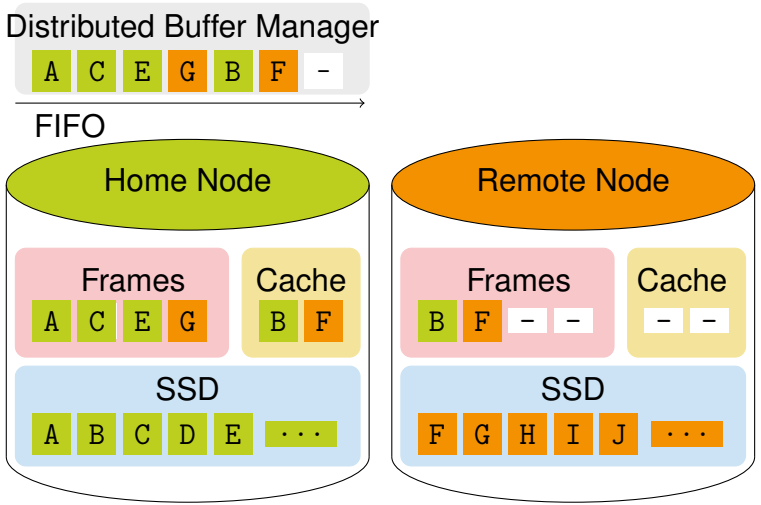
Distributed Buffer Manager: Global Buffer Pool

- RDMA: bypass CPU to access remote memory
- load pages either from a home or remote SSD first into frames of the home node
- If frames run out, frames of the remote node are taken and the page is cached locally according to the cache coherence protocol.
- global buffer manager = amount of loaded frames
- two queue eviction strategy (FIFO and LRU)
- strategy: fully utilizing the distributed buffer pool before swapping out pages
- coherence protocol responsible for caching and locking



Distributed Buffer Manager: Global Buffer Pool

- RDMA: bypass CPU to access remote memory
- load pages either from a home or remote SSD first into frames of the home node
- If frames run out, frames of the remote node are taken and the page is cached locally according to the cache coherence protocol.
- global buffer manager = amount of loaded frames
- two queue eviction strategy (FIFO and LRU)
- strategy: fully utilizing the distributed buffer pool before swapping out pages
- coherence protocol responsible for caching and locking



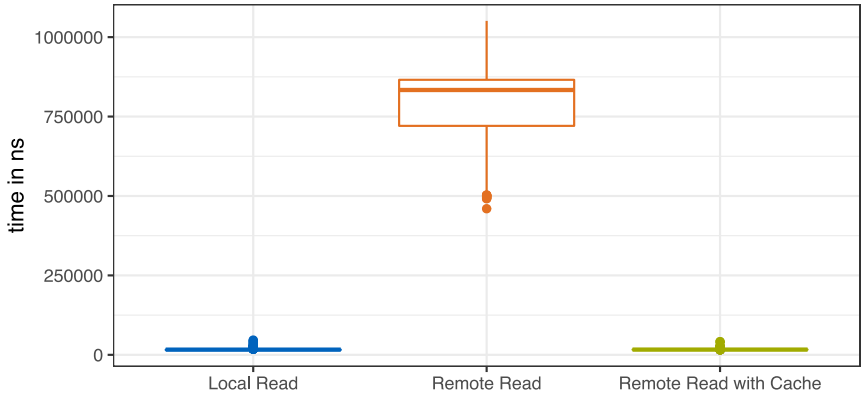
Evaluation

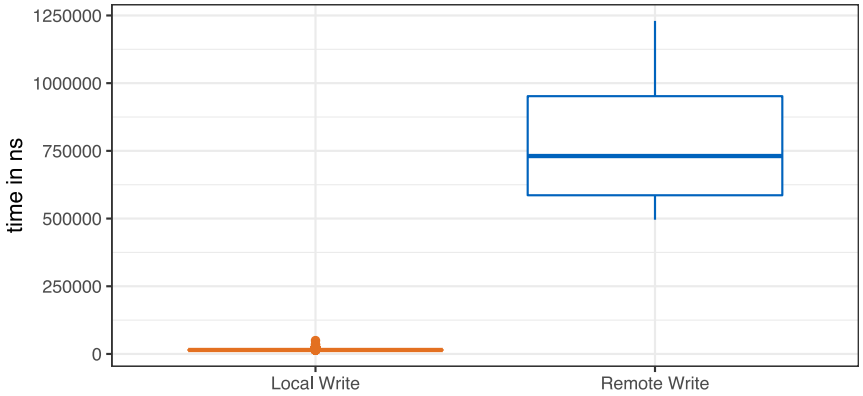

PowerEdge T620
PowerEdge T620
PowerEdge T620
DELL
DELL
DELL

Evaluation: Set-Up

- **System:** Two Ubuntu 18.04.03 LTS servers, 2x20 Intel Xeon E5-2660 v2 processors (supporting hyperthreading)
- **Memory:** 256 GiB main memory and 1TiB of SSD as background storage
- **Network Card:** Mellanox ConnectX–3 VPI NIC supporting FDR InfiniBand with 56 GBit/s, Mellanox SX6005 switch
- **Experiments:** micro-benchmarks on the different API functions, distributed hash table, buffer manager

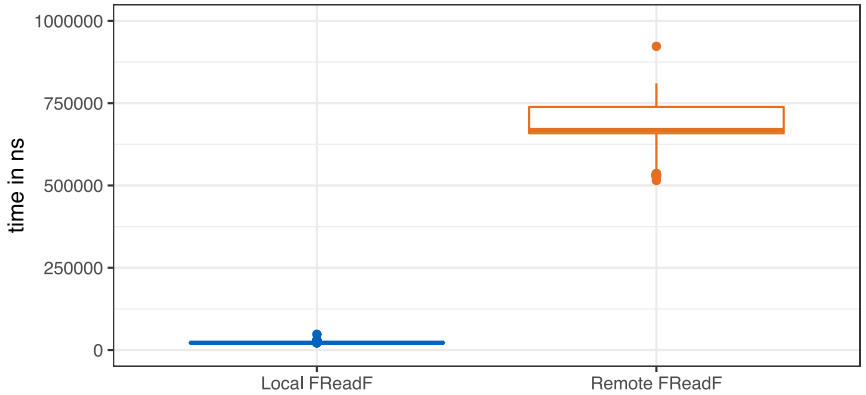
Evaluation: Micro-Benchmarks read/write

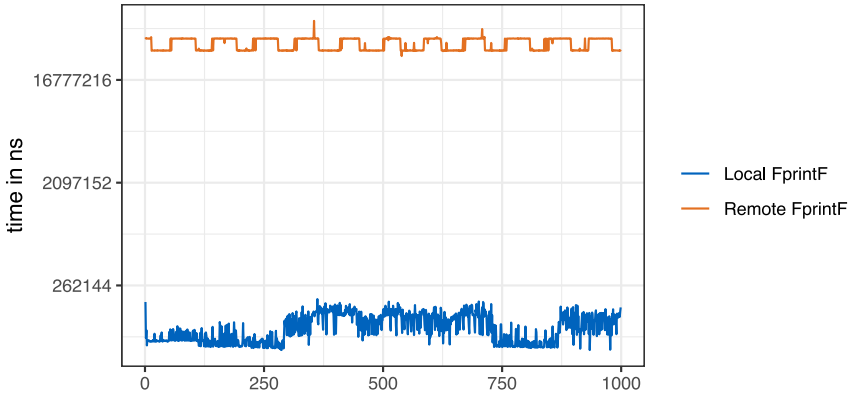




- remote read 30x slower than local read
- remote write 30x slower than local write
- remote read with caching nearly as fast as local read

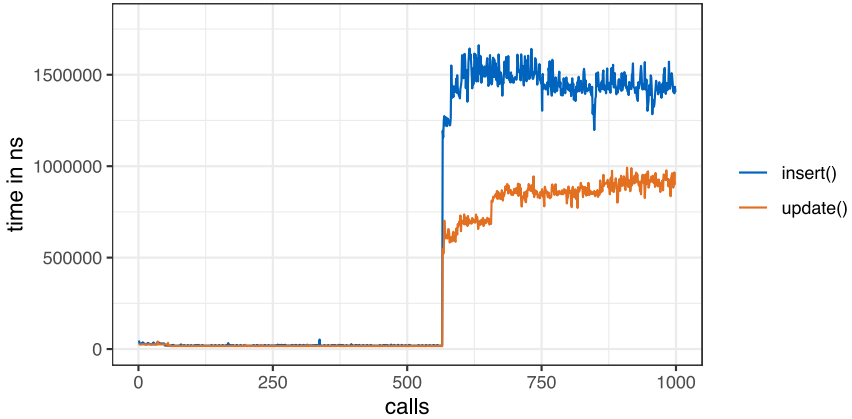
Evaluation: Micro-Benchmarks fread/fprint





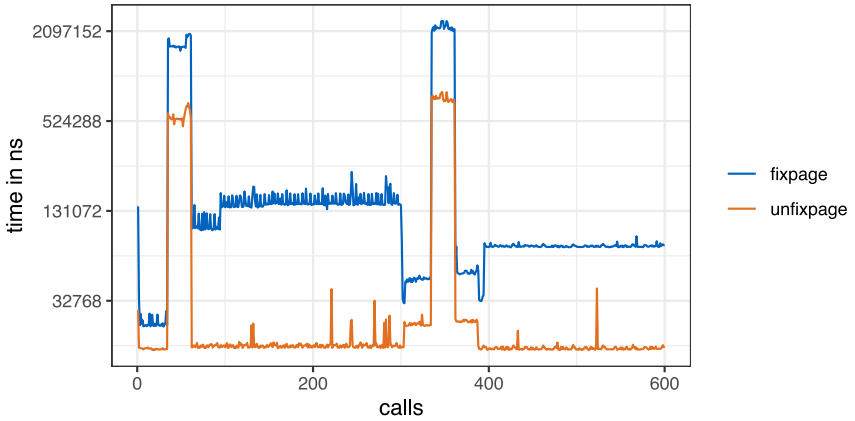
- remote file read as fast as remote read
- remote file read 20x slower than local file read
- remote file write 300x slower than local file write

Evaluation: Hash Table and Buffer Manager



Distributed Hash Table

- insert/update 1000 key-values-pairs
- performance drop when storage of home node exceeded
- insert needs twice the time of update
- two calls for insert: memory allocation and writing



Buffer Manager

- 30 frames on each node (page size 912B), 300x fix/unfix
- peak after 30 pages: pages are fixed on the remote node
- after 60 calls: pages get swapped out locally
- after hard reset (300 calls): same observation

Conclusion

- extended cache-coherence protocol: background storage
- distributed buffer manager, distributed buffer pool: pinning and evicting pages to frames of remote nodes
- two-stage caching with almost no overhead due to the quickly converging performance characteristics of SSDs and RDMA capable networks like InfiniBand
- eviction strategy: fully utilizing the distributed buffer pool before swapping out pages
- Future work:
  - benchmark TPC-C
  - implement load balancer for distributed buffer pool

Thank you for your attention!

https://github.com/tum-db/cachecoherence