Dandelion: Small Clusters, Massive Throughput—The Future of Distributed Transactions

Vasilis Gavrielatos*, Antonios Katsarakis*, Chris Jensen, Nikos Ntarmos
Huawei Research

ABSTRACT

We present an in-memory, RDMA-enabled, highly-available, transactional Key-Value Store (KVS), dubbed Dandelion, focusing on significantly improving performance in small deployments (e.g., 5 machines). The focus on small deployments is motivated by the emerging memory expansion (e.g., via CXL), which enables the deployment of KVSes with few machines but lots of memory.

A small deployment presents locality opportunities that have not been examined by related work. Specifically, it is more likely that at any given time, we must send multiple messages to the same recipient. We leverage this by batching multiple requests in the same network packet. Similarly, it is more likely that at any given time, we have multiple requests that can be served by the local hashtable without going through the network. Sending all requests to the hashtable as a batch allows it to overlap their memory latencies through software prefetching. Finally, it is more likely that the node that requests a key, is itself a backup of that key. We leverage this by allowing local reads from backups.

Our evaluation shows that these optimizations and careful engineering result in hundreds of millions of distributed, strongly consistent, and 3-way replicated transactions per second with very few machines. This also translates to 3.3 - 6.5x throughput improvement over a state-of-the-art system, FaSST, in OLTP workloads in a 5-machine deployment. We characterize the impact and scalability of each of these optimizations with up to 10 machines – where Dandelion still offers up to 3.5× higher throughput than FaSST.

1 INTRODUCTION

This work focuses on reliable distributed Key-Value Stores (KVSes). Modern KVSes shard and replicate the data in-memory of multiple servers and provide strongly consistent transactions with high availability. They leverage RDMA for efficient networking to deliver high throughput while scaling into big deployments (e.g., 90 machines). FaRM [4, 5, 11] was the first such work, which sparked a multitude of subsequent works [2, 7, 10, 12–17]. Unlike these works, we focus on small deployments (e.g., 3-10 machines).

Smaller deployments exhibit various forms of locality. For example, it is more likely that at any given time, multiple messages from different transactions must be sent to the same recipient. Similarly, it is more likely that a key-value pair is stored in the machine that is searching for it. Such locality opportunities have, for the most part, not been exploited in the context of rdma-enabled transactional KVSes, because of the assumption that the deployment must always be large. Instead, related work has focused mostly on debating the correct usage of the RDMA primitives [3, 6, 10, 13].

We focus on smaller deployments, partially in anticipation of CXL [1] memory expansion. In its first version, CXL-1 will enable scaling up a few servers by adding more memory at a significantly lower cost (than buying more new servers). Further down the line, it is expected that CXL-2 will enable the pooling of memory, entirely removing the coupling between compute and memory. In either case, we will no longer need a large number of servers simply to fit the dataset in-memory. *We are faced then with the following challenge: if we can fit our dataset in a few machines, are those few machines sufficient to also achieve the target throughput?*

This work tackles this challenge by exploiting the locality opportunities presented in small deployments. Specifically, we build *Dandelion (DNL)*, a distributed, in-memory, highly-available, RDMA-enabled Key-Value Store, that achieves up to 6.5x (3.5x) higher throughput than a state-of-the-art system (FaSST [7]) with 5 (10) machines in popular OLTP workloads. Below, we introduce each of DNL's main components – networking, hashtable, and protocol – discussing the relevant locality opportunities we exploit.

Networking. The main locality opportunity that arises in small deployments is that there is a higher probability that multiple messages must be sent to the same node at the same time. We leverage this opportunity by batching multiple requests and responses in the same network packet. Batching multiple messages in the same packet amortizes the per-packet overheads incurred in CPU (software stack needed to transmit/receive), PCIe, and network (per-packet metadata in NIC caches, packet headers, routing, etc.).

While network batching is by no means a new idea, this work is the first to highlight its importance for in-memory, RDMA-enabled, transactional KVSes and characterize its performance benefits. Batching can yield up to a 10x throughput improvement.

Hashtable. In DNL, each server uses a hashtable to store and index key-value pairs. Again, locality facilitates batching, as it is more likely that at any given time, there are multiple requests that must be propagated to the local hashtable (e.g., , after receiving a batch of requests through the network). Batching in the hashtable has been shown to significantly increase throughput by overlapping the memory latencies of the different requests [8, 9].

Protocol. DNL features a customizable protocol skeleton, through which we implement three protocols. One of the protocols is very similar to FaRM's OCC protocol, while the other two have not yet been explored, as far as we know. We expect that the community can use the skeleton to explore more protocols. The protocol skeleton leverages locality, by allowing reads of local replicas, despite of whether the replica is a primary or a backup.

Summary. This work anticipates that memory expansion (i.e., via CXL) will enable transactional KVSes in a small number of machines with access to lots of memory. We build DNL, an in-memory, RDMA-enabled, transactional KVS to leverage three locality opportunities found in small deployments. Specifically, DNL batches in the network to amortize the fixed per-packet costs, batches in the hashtable to enable software prefetching and can read from local backups in the protocol. Our evaluation shows that DNL surpasses 250 Million distributed, strongly-consistent, and 3-way replicated OLTP (write-dominant) transactions per second with just 5 machines. This also translates into 3.3 - 6.5x throughput improvement over the state-of-the-art distributed in-memory KVS (FaSST).

*The two authors contributed equally to this work.

REFERENCES

[1] Compute express link (cxl). https://www.computeexpresslink.org/.

[2] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Aleksandar Dragojevic, Dushyanth Narayanan, and Miguel Castro. RDMA Reads: To Use or Not to Use? *IEEE Data Eng. Bull.*, 40(1):3–14, 2017.

[4] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, 2014. USENIX Association.

[5] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the Symposium on Operating Systems Principles*, SOSP '15, pages 54–70, New York, 2015. ACM.

[6] Anuj Kalia, Michael Kaminsky, and David Andersen. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, pages 437–450, Berkeley, CA, USA, 2016. USENIX Association.

[7] Anuj Kalia, Michael Kaminsky, and David Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-sided (RDMA) Datagram RPCs. In *Proceedings of the 12th Conference on Operating Systems Design and Implementation*, OSDI'16, pages 185–201, USA, 2016. USENIX.

[8] Hyeontaek Lim, Dongsu Han, David Andersen, and Michael Kaminsky. MICA: A Holistic Approach to Fast In-memory Key-value Storage. In *Proceedings of the 11th Networked Systems Design and Implementation*, NSDI'14, pages 429–444, USA, 2014. USENIX Association.

[9] Vikram Narayanan, David Detweiler, Tianjiao Huang, and Anton Burtsev. Dramhit: A hash table architected for the speed of dram. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 817–834, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, and Marcos Aguilera. Storm: A fast transactional dataplane for remote data structures. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, SYSTOR '19, page 97–108, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Alex Shamis, Matthew Renzelmann, Stanko Novakovic, Georgios Chatzopoulos, Aleksandar Dragojević, Dushyanth Narayanan, and Miguel Castro. Fast general distributed transactions with opacity. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 433–448, New York, NY, USA, 2019. ACM.

[12] Chao Wang and Xuehai Qian. Rdma-enabled concurrency control protocols for transactions in the cloud era. *IEEE Transactions on Cloud Computing*, 11(1):798–810, 2023.

[13] Xingda Wei, Zhiyuan Dong, Rong Chen, and Haibo Chen. Deconstructing RDMA-enabled distributed transactions: Hybrid is better! In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 233–251, Carlsbad, CA, October 2018. USENIX Association.

[14] Xingda Wei, Sijie Shen, Rong Chen, and Haibo Chen. Replication-driven live reconfiguration for fast distributed transaction processing. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 335–347, Santa Clara, CA, July 2017. USENIX Association.

[15] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 87–104, New York, NY, USA, 2015. Association for Computing Machinery.

[16] Erfan Zamanian, Carsten Binnig, Tim Harris, and Tim Kraska. The end of a myth: Distributed transactions can scale. *Proc. VLDB Endow.*, 10(6):685–696, feb 2017.

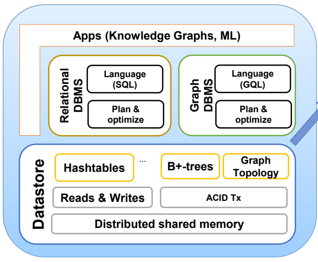
[17] Ming Zhang, Yu Hua, Pengfei Zuo, and Lurong Liu. FORD: Fast one-sided RDMA-based distributed transactions for disaggregated persistent memory. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 51–68, Santa Clara, CA, February 2022. USENIX Association.

Dandelion: Hundreds of Millions of Distributed Replicated Transactions with Few Machines

Antonios Katsarakis   Vasilis Gavrielatos   Chris Jensen   Nikos Ntarmos

Huawei Research

Datastores 101



Backbone of wide range of applications, services and systems (including DBMS)

Performance-critical intra-datacenter storage layer to access & manipulate data with

> *Efficiency*
high throughput, cost-effective

> *Programmability*
transactions, strong consistency

> *High availability*
replication and fault tolerance

Evolution of Datastores
(note: 10x leaps that need redesign)

Traditional Era

Traditional Datastores
Disk-based, slow network
➤ Efficiency: non-scalable, low performance

#1

#2 Exploit new HW for efficiency + programmability

NoSQL Datastores
Sacrifice programmability for performance
Ok for best-effort web-service but not critical app!
➤ Programmability: sacrifice transactions, consistency

New Era

Modern Datastores: e.g., Microsoft's FaRM [SOSP'15]
➤ Scalable, good perf., transactions, consistency, availability
- In-memory
- Fast (RDMA) network
- *FaRM* ➔ 140M TATP txs/sec (90 machines – 5TB)

#3

Next-gen Datastores: academia + our goal
➤ Modern datastores + better perf., less cost, higher capacity
- HW acceleration: FPGA, ASICs, smart NICs/switches
- CXL expansion, Disaggregated memory, PMEM

More memory with less compute ➔ **50TB** in **5-10 machines**

Challenge: Can we achieve same or better performance with 10x fewer machines (compute) ?

Stoppers!

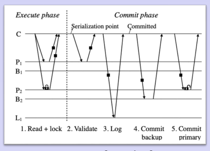
1. Fast **Networks**, but small packets ✗

eRPC [NSDI'19 – best paper]
Efficient general-purpose RPCs
DPDK, UDP, RoCE/IB (two-sided) RDMA
Retransmissions
Support packets > MTU
- General purpose, Reliable, callback API
- Great for large/medium messages … **BUT**

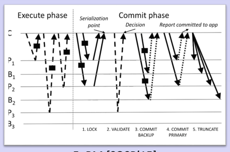
Issues with small (8-24B) requests
1. RoCE packets have 66B Header = 10-30% BW utilization
2. Too many packets, NIC/switch cannot keep up with packet rate

Datastores are dominated by
Small data messages → access/update small key-value pairs (e.g. 8-32B)
Tiny protocol logic messages → lock/unlock, commit/abort, truncate, etc.

2. Fault-tolerant strongly-consistent **Protocols** but complex and costly execution/commit phases ✗
- lock-based execution phase or
- complex multiple round-trip (RTT) commit
- protocols have tradeoffs, but datastores no protocol alternatives

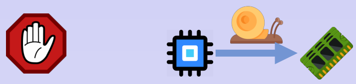


FaSST [OSDI'16]



FaRM [SOSP'15]

3. Concurrent in-memory **Data Structures** but practically blocking & not memory-aware ✗



4. **CPUs** with many cores but cycles are wasted ✗
on small packets, many protocol actions, stalled waiting on memory, one-at-a-time request execution

Dandelion: across-stack innovation + SW-HW co-design + batched execution (OLTP *like* OLAP)!

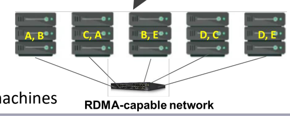
Dandelion

Context

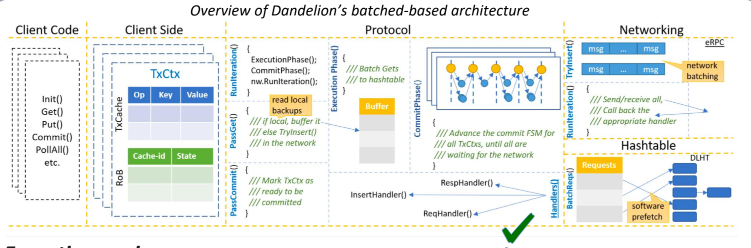
In-memory dataset: small/moderate sized **key-value** pairs
- **Distributed** across multiple machines for capacity
- **Replicated** for availability in the face of faults
- **Interactive Transactions**

Goal
1. **Reliable**: availability, strong consistency
2. **Programmable**: expressive, transactions
3. **Top performance/cost**: e.g., better performance with 10x fewer machines

```
tx.Init();
  auto val = tx.Get(A)
  if (val == 0) {
    tx.Put(B, 0);
  }
  tx.Insert(C, 1);
  tx.Delete (D)
tx.Commit();
```



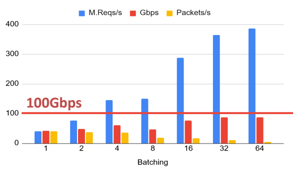
Overview of Dandelion's batched-based architecture



Execution engine: end-to-end **asynchronous & batched-based** → avoids wasting CPU cycles

Network

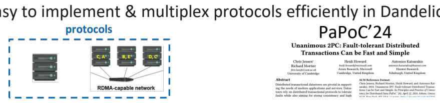
Datastore-optimized RPCs-over-RDMA ✓



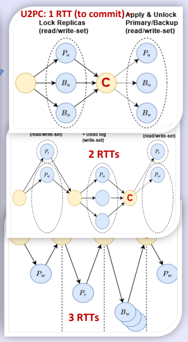
Application-level batching
- necessary to fully utilize modern (100s Gbit net.)
- dandelion can batch 10s requests in one packet

Protocols

Easy to implement & multiplex protocols efficiently in Dandelion

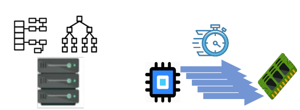


3 Novel Fast Reliable Protocols: latency, throughput, lock-window
- Efficient logging and decentralized recovery
- Strongest consistency, Correctness: verified in TLA+
- High performance
  - **Lock-free execution phase** (+ caching)
    - neither reads nor writes grab locks →
    - programmability, performance, simple recovery ✓
  - **Fast reliable commit phase**
    - read-only / write-only txs: 0-1RTT
    - read/write txs: 1RTT – 3RTTs (based on protocol) ✓



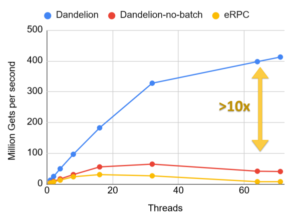
Data structures

Practically non-blocking and **memory-aware** ✓



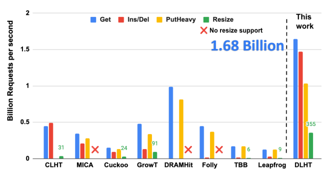
Dandelion Hashtable (DLHT) [Tree (DLTree)]
1. **Non-blocking requests [reads]**
2. **Most requests: 1 [few] memory access**
3. **Pipelined batched request processing**
   → overlap memory access + in-order completion
4. **Transaction-native and batched API** …

Networking



Dandelion networking **>10x** over eRPC

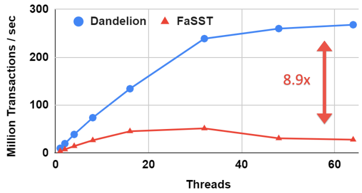
Data Structures

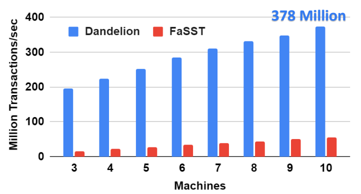


DLHT (Hashtable – single server)

Dandelion Hashtable **>1.6B ops/sec** on a single server

Dandelion: network + protocols + data structures + async. batched engine





Fault-tolerant Distributed Transactions (TATP, 5 servers – by default, 3-way replication)

Dandelion **>250M** distributed replicated txs (5 servers) | **12x-7x** faster than FaSST (3-10 servers)

Commodity servers: *18-core Intel Xeon Gold 6254 (2 sockets), 128GB DRAM*   Network: *100Gbit RoCE*

HUAWEI