Summary of "FaRM: Fast Remote Memory"

Pei-Xuan Xie (peixuanx), Tai-Ying Lin (taiyingl), Changfeng Liu (changfel)

Problem and Motivation

Due to the decreasing price of DRAM, a cluster with one hundred machines can hold tens of terabytes of main memory, which is sufficient to store all the data for many applications or at least to cache the applications' working sets. This removes the overhead of flash/disk so applications could perform random data access. However, the network communication remains a bottleneck. RDMA provides reliable user-level reads and writes of remotes memory that achieves low latency and high throughput. Today, RoCE supports RDMA over Ethernet with data center bridging at competitive prices so that RDMA could help solve the problem of traditional systems with TCP/IP networking.

Hypothesis

FaRM is a memory distributed platform that exploit RDMA to achieve low latency and high throughput at the same time. FaRM provides a shared address space for directly accessing data and general distributed transactions to simplifying programming. Two mechanisms are provided by FaRM to improve performance: 1. Lock-free reads over RDMA 2. Collocating objects and function shipping to enable the use of efficient single machine transactions. Moreover, a hash table algorithm is built on the top of FaRM to achieve high space efficiency while a small number of RDMA reads for lookup is needed.

Solution Overview

Communication primitives

FaRM uses one-sided RDMA reads to access data directly and uses RDMA writes to implement a fast message passing primitive. To achieve both low latency and high message rates at the same time, there are two main problems need to be solved.

Problem 1:

The performance of RDMA operations decreased significantly as the amount of memory registered for remote access increasing. This happens when the NIC runs out of space to cache all the page tables and thus keeps fetching page table entries from system memory across the PCI bus.

Solution:

In this paper, they implemented PhyCo, a kernel driver that allocates a large number of physically-contiguous and naturally-aligned 2 GB memory regions at boot time. PhyCo maps the regions into the virtual address space of the FaRM process aligned on a 2 GB boundary which allowed us to modify the NIC driver to use 2 GB pages and reduced the number of page table entries per region from more than half a million to one.

Problem 2:

A significant decrease in request rate happens when the cluster size increases because the NIC runs out of space to cache queue pair data.

Solution:

First, they reduce the size of required queue pairs between every pair of threads per machine from $2mt^2$ to $2 * m * t$ (m: the number of machines, t: the number of threads per machine) by using a single connection between a thread and each remote machine. Second, they futher reduce the size to total of $2 * m * t/q$ queue pairs per machine by using queue pair sharing among q threads in a NUMA-aware way. This trades off parallelism for a reduction in the amount of queue pair data on the NIC.

Transaction

Problem 3:

The implementation uses optimistic concurrency control and two-phase commit to ensure strict

serializability. Having very low latency, the two-phase commit protocol can reduce conflicts and improves performance by reducing the amount of time locks are held, however, it may be too expensive to implement common case operations.

Solution:

FaRM provides two mechanisms to achieve good performance in the common case:

1. Single machine transactions: Reducing the number of messages
   Application can collocate the objects accessed by a transaction on the same primary and on the same replicas, and shipping the transaction to the primary. In this way, write set locking and read set validation are local and hence the prepare and validate messages are not needed. The primary only needs to send a commit message with the buffered writes to the replicas before unlocking the modified objects.
2. Lock-free read-only operations: Reducing delays due to locks
   Objects are first locked in a mode that allows lock-free reads and the primary locks objects in exclusive mode just before updating them.

Hashtable

Problem 4:

In the original algorithm (hopscotch hashing), large neighbourhoods perform poorly with RDMA because they result in large reads and simply using small neighbourhoods does not work well as it requires frequent resizes which results in poor space efficiency.

Solution:

Introduce a new algorithm, chained associative hopscotch hashing which achieves a good balance between space efficiency and the size and number of RDMAs used to perform lookups by combining hopscotch hashing with chaining and associativity.

The new algorithm has the following features:

1. If an insert fails to move an empty bucket into the right neighbourhood, it adds the key-value pair to the overflow chain of the key's bucket instead of resizing the table which limits the length of linear probing during inserts.
2. Uses associativity to amortize the space overhead of chaining and of FaRM's object meta-data across several key-value pairs.
3. Guarantees that a key-value pair is stored in the key's bucket or the next one and overflow blocks store several key-value pairs to improve performance and space efficiency.

Limitations and Possible Improvements

1. This paper uses replicated logging to SSDs for durability and availability. However it uses an existing parallel recovery mechanism without ad-hoc design, which can decrease the performance of the system. Besides, a large number of messages logged to SSDs can also reduce the speed of whole system. These problems can be solved by optimize the current recovery mechanism and transaction and replication protocol.
2. This paper uses YCSB benchmark to evaluate the performance, which lacks failures in transactions. This can make the evaluation less convincing. Future work needs to evaluate the results over transactions with/without failures.
   FaRM key-value stores mainly focus on giving high performance of operations over small values and lack the mechanism for large data values.

Comparison to "No compromises: distributed transactions with consistency, availability, and performance"

Focus on the new protocols in transaction and recovery :

1. Distributed transactions and replication
   - Modified 2-phase commit protacal

- Steps: Lock, Validate, Replicate, Update and unlock
- Benefits:
  a.) Usage of primary-backup replication: Reduces the number of copies and also reduces the number of messages transmitted during a transaction.
  b.) Coordinators communicate directly with primaries and backups: Reduces latency and message counts
  c.) Usage of one-way RDMA writes records: Reduces waiting for remote CPUs and also allows the remote CPU work to be lazy and batched.

2. Failure Recovery
   - Steps: Failure detection, Reconfiguration, Transaction state recovery, Recovering data, Recovering allocator state
   - Recovery is complicated due to CPU does not process remote accesses and we cannot rely on CPU rejecting messages.
   - Benefits: By minimizing the use of CPU and ensuring the parallelism, FaRM provides durability and high availability.

Summary of Class Discussion

Q1: What is the difference between RDMA and RDMA message?
RDMA is one-sided and RDMA message is two-sided.

Q2: Specify the three memory allocators: region, block and slab.
- Region is cluster-wide allocator that uses PhyCo to allocate memory for the region and then it registers the region with the NIC to allow remote access
- Block is machine-wide allocator that allocates blocks from shared memory regions.
- Slab is thread-wide allocator that allocates small objects from large blocks.

Q3: The outline for design of FaRM:
- Minimize the usage of CPU and memory copy, and maximize the usage of RDMA
- Minimize communication
- Ensure that transactions happen parallelly

Q4: what is memory barrier? How to ensure that there is no modification during the checking process within cache-line?
memory barrier is a barrier instruction that enforces an ordering constraint on memory operations (operations issued prior to the barrier are guaranteed to be performed before operations issued after the barrier). In FaRM, we have memory barrier to enforce atomic order within single cache-line.

Q5: How to ensure the consistant object states under concurrent writes among multiple cache lines?
FaRM uses cache coherent DMA to ensure the consistency. It stores the version number of an object in the first word of its header and the beginning of each cache line after the first one.
An lockFreeRead will check if the version is unlocked and the version number in the object's header is matched with all other version numbers stored in cache lines. If they are matched, it will be a consistent read. If they are not matched, it will retry RDMA after a randomized backoff.