• Course Project round 3 meetings signup!
• Final class on Dec 6th
• No class on Dec 11th
• Poster session Dec 13th – More details very soon!
RDMA: REMOTE DIRECT MEMORY ACCESS
MOTIVATION

Need to access remote data fast
- Increasing NIC speeds (up to 100Gbps)
- OS/CPU bottlenecks

RDMA
- Perform direct memory access (DMA) from NIC!
- Bypass remote CPU, OS etc.

RDMA cost / availability
FARM

Approach
- Model distributed memory as shared address space
- Communication primitives over RDMA

Features
- Memory Management
- Transactions
- Datastructures
Key idea: One sided RDMA read/writes

How to implement writes?
- Circular buffer on receiver
-Recv polls at“Head”
- Sender writes at“Tail”
- Ensure sender doesn’t overwrite
RDMA CHALLENGES

Page Table Size
- Doing DMA requires NIC to cache page tables
- Need for larger pages to make page table smaller
- PhyCo – kernel driver that allocates 2GB pages!

Caching queue pair data
- Need a queue pair (connection) between every sender-receiver
- \(2m^2 t^2\) for \(m\) machines, \(t\) threads per machine
- Solution: Share queue pair among threads – \(\frac{2m^2 t}{q}\)
CONNECTION MULTIPLEXING

- $q=1$
- $q=2$
- $q=4$
- $q=8$
- $q=16$

Requests / μs vs. Servers
FARM API

Tx* txCreate();
void txAlloc(Tx *t, int size, Addr a, Cont *c);
void txFree(Tx *t, Addr a, Cont *c);
void txRead(Tx *t, Addr a, int size, Cont *c);
void txWrite(Tx *t, ObjBuf *old, ObjBuf *new);
void txCommit(Tx *t, Cont *c);

Lf* lockFreeStart();
void lockFreeRead(Lf* op, Addr a, int size, Cont *c);
void lockFreeEnd(Lf *op);
Incarnation objGetIncarnation(ObjBuf *o);
void objIncrementIncarnation(ObjBuf *o);

void msgRegisterHandler(MsgId i, Cont *c);
void msgSend(Addr a, MsgId i, Msg *m, Cont *c);
Every 2GB alloc is region
32-bit id, 32-bit offset

Map regions in hash ring
Why multiple rings?
Parallel recovery
Load balancing
MEMORY ALLOCATION

Hierarchy
- Slabs, regions, blocks
  - Thread-level, private slab allocators
  - Blocks multiples of size 1MB
  - Regions on size 2GB

Hints
- Applications request allocation “close”
- Same block as hint or same region or nearby position
TRANSACTIONS

Transaction components
- Reuse standard protocols from DB (2-phase commit, OCC)
- Components: Read set, write set
- Coordinator that runs transaction

Process
- Prepare message to lock write set
- Validate messages to check read set
- Commit messages: first to replicas then to primaries
Locks are still expensive! → Design lock-free read operations

Version numbers stored per-cache line – Why do we need this?

Use memory barriers to update one line at a time

![Diagram showing object and cache lines with version numbers](image-url)
HASHTABLE CHALLENGES

Goals
- Perform most operations using single RDMA read
- Achieve good utilization (avoid resizing hash table)

Challenges
- Chaining / Cuckoo hashing: Key could be in many disjoint locations
- Hopscotch hashing: Each bucket has a neighborhood of H-1 buckets
- But large H → more reads and small H → poor utilization
HASHTABLE SOLUTIONS

Soln: Chained associative hopscotch

Maintain overflow chain per-bucket
- Add key to overflow if reqd
- Small chains limit overhead
- Inline values next to key

Other optimizations
- Lookups use lock-free read
- Combine updates in 1 transaction
New networking hardware enables fast systems

Insights
  Avoid CPU overheads using RDMA read
  Design higher-level primitives based on that

Drawbacks
  Need to do multiple round trips?
  Hardware dependent wins?