
ORACLE®

ORACLE®

Avneesh Pant

OFIWG – 17th June 2014

3rd party logo

Oracle IPC Communication Model

- Endpoint based communication model
  - No connections exposed
- Support for both BCOPY and ZCOPY messaging
  - Max message size currently limited to 1MB. May extend later
- Three distinct traffic class sizes
    - Control messages/requests: 300-600 Bytes
    - Cache Fusion transfers: 8K-32K
    - Storage block transfers: 1MB
- Synchronous and asynchronous operations
  - Use both signaled and silent completions

Oracle IPC Communication Model

- Utilize RDMA Read/Writes
  - Discourage these in favor of ZCOPY messaging if possible
- Starting to leverage atomics
  - Need extended atomics beyond currently specified by standard
  - Detect capability of HCA/API at runtime to use these extensions
- Primarily access IB fabric using RDS due to connection scaling issues
  - Moved to RC/XRC for some critical components in latest release
  - Looking at UD/DCT to address connection scaling

Oracle IPC Communication Model

- Message based communications
  - No stream semantics
- Communication channels are simplex
  - Sending and receiving endpoints. Can do away with QPs conserving hardware resources
- Communication channels provide ordered, reliable, non-duplicating messages between <send endpoint, target endpoint>
- Endpoint addressed using tuple <IP, Port, Timestamp>
  - 32 bit timestamp to distinguish different incarnation on same IP:Port

Oracle IPC Connection Management

- Utilize RDMA CM extensively for connection setup
  - Currently use address change for failover/failback
  - Connect request does not have timeout
  - Extend RDMA CM to support
    - APM
    - Additional Transports: XRC/DCT
- Have run into scaling issues with current implementation
  - IBACM needs enterprise class support (keep cache consistent across failover/failback)
  - FD per process to ACM server does not scale with 30K processes on a node. Memory mapped interface?

Oracle IPC Communication Model – Fault Tolerance

- Channels are resilient to network failures
  - Oracle IPC requires IP addresses to never fail
  - IPs are migrated to surviving ports/adapters
  - Support both bonded interfaces as well as active/active
  - Use RDMA CM address change events to track IP to HCA/Port association
  - Re-establish connection across address change events
  - For active/active support both failover and failback to maximize network throughput
  - Looking to add support for APM in next release
    - Still need address change event for failover across HCAs

Oracle IPC Communication Model – QoS/Partitioning

- Even with RC can get duplicate message across failover
  - Maintain separate sequence numbers at our layer
- Provide differentiated services for communication flows
  - Distributed Lock Manager messages are latency sensitive
  - Database Log Writes are on dedicated VLs (100s to 4K byte)
  - Storage IO uses RDMA Read/Writes on separate VLs
- Pkeys are used for partitioning/isolation between databases on the fabric
  - Increasing use in virtualized deployments

Oracle IPC Memory Management

- Oracle instance is a collection of processes and shared memory
  - All processes map the shared memory symmetrically
  - ~90% of memory is used for caching DB blocks
  - Remaining used for various types of memory pools
  - Memory can be dynamically moved between pools and buffer cache under pressure
  - Want to perform ZCOPY/RDMA transfers from any process to any shared memory address
  - Large number of processes per node – around 75 processes per hardware thread (15-20K processes on a 8 socket server)

Oracle IPC Memory Management

- Attempting to register each entire memory region in each process is not scalable
  - Oracle uses shared PD support.
  - Single process (usually a fatal process) allocates a shared PD and registers all memory with it
  - Other processes use the same shared PD with their context
  - Allow sharing of memory registrations across all processes
  - Each process still creates separate QP/SQ/RQ/CQ etc.
  - Would be useful if support is available in mainline code

Oracle IPC Memory Management

- Oracle discourages uses of RDMA if not needed
  - Hard to “revoke” keys to local memory that is source or target of RDMA. This can be frequent enough during txn aborts.
  - Memory registered in granules. May have multiple outstanding operations within granule across many processes at a time
  - Revoking key requires deregistering entire granule which impacts other ongoing transfers.
  - Need a way to generate efficiently memory keys from a previously registered region
    - Preferably avoid call into OS i.e. generate keys asynchronously on HCA via a new request

Oracle IPC Memory Management

- Memory keys can be considered overlays on currently registered regions
  - Underlying mapping is never changed. Registered shared memory will never change during lifetime of instance
  - Most times these are use once keys (request/response) so ability to specify lifetime of key can avoid an explicit call to de-allocate key
  - Txn aborts can then perform an explicit de-allocate of overlay key without impacting other operations

OFED Verbs Extensions: NUMA Support

- Most machines have deep NUMA hierarchies
- Would be good to have NUMA aware extensions
  - Oracle attempts to use NUMA local HCAs for each process
  - Extend device enumeration to be NUMA aware
  - IB resource allocations can benefit from this as well
    - Request CQ/QP memory to come from specific NUMA domain
    - NUMA affined allocation of interrupt vectors to CQs

OFED Verbs Extensions: NUMA Optimizations for Event Mode

- Polling is not feasible with large number of processes
  - Wait on OS fd for completion notification
  - Latency under load is non-deterministic i.e. once yield CPU no idea when process is scheduled back
  - Interrupt dispatch latency across sockets can add significant latency
  - Minimize cross process interference due to interrupts
    - Send and receive CQs. Request/response paradigm can have both CQs use the same vector
    - Selected vector affined to core process is running on.

OFED Verbs Extensions: Cancelling Receives

- For request/response patterns process queues receive buffer to SRQ before sending request out
  - In some cases response(s) may not arrive. Can be determined by requestor
  - Need ability to cancel these queued receives as subsequent requests may use different response buffer.
  - If this was possible we don’t require RDMA for most of these block fetches as buffers can be queued for zero copy receives
- Cancel can be opportunistic i.e. either completes successfully with data or with cancelled status

OFED Verbs Extensions: Triggered operation

- Some data transfers have inherent dependencies
  - Could benefit from triggered operations that can highlight these
  - Generalized mechanism to specify these operations
    - Start WR when some currently submitted WR has completed successfully (or not)?
    - Dependent operation specified via <QPN, WRID> tuple. Client responsible for guaranteeing WRID uniqueness
    - QPs must be within same PD (also support shared PD allowing inter-process triggered operation)
- Oracle has data dependence between serving dirty blocks across networks and Log Write Completions

OFED Verbs Extensions: Completion Semantics

- May require richer completion semantics in future
  - RDMA Write completions indicate nothing wrt remote memory
  - Must issue a RDMA Read following write to detect completion
  - Persistent Memory complicates things further – require writes to percolate to persistent domain
  - Allow specification of various completion semantics (HCA, Host, Persist etc.)
- Allow specification of consistent RDMA Read operations
  - Will require hardware support but got to start the discussion now

Questions?