OPENFABRICS ALLIANCE

Exadata in Enterprise Data Centers

Author: Sumanta Chatterjee
Richard Frank
Date: April 5, 2011

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Outline

- Evolution of Exadata since last report
- Current models
- New opportunities

Exadata Database Machine X2-8 Full Rack

Extreme Performance for Consolidation, Large OLTP and DW

- 2 x64 Eight-processor Database servers (Sun Fire 4800)
  - High Core, High Memory Database Servers
  - 128 CPU cores (64 per server)
  - 2 TB (1 TB per server)
  - 10 GigE connectivity to Data Center
    - 16 x 10GbE ports (8 per server)
- 14 Exadata Storage Servers X2-2
  - All with High Performance 600GB SAS disks

  OR
  - All with High Capacity 2 TB SAS disks
- 3 Sun Datacenter InfiniBand Switch 36
  - 36-port Managed QDR (40Gb/s) switch
- 1 “Admin” Cisco Ethernet switch
- Redundant Power Distributions Units (PDUs)

Add more racks for additional scalability

Exadata Database Machine X2-2 Full Rack

Pre-Configured for Extreme Performance

- 8 x64 Dual-procesor Database Servers (Sun Fire X4170 M2)
  - 96 cores (12 per server)
  - 768 GB memory (96GB per server)
  - 10 GigE connectivity to Data Center
    - 16 x 10GbE ports (2 per server)
- 14 Exadata Storage Servers X2-2
  - All with High Performance 600GB SAS disks

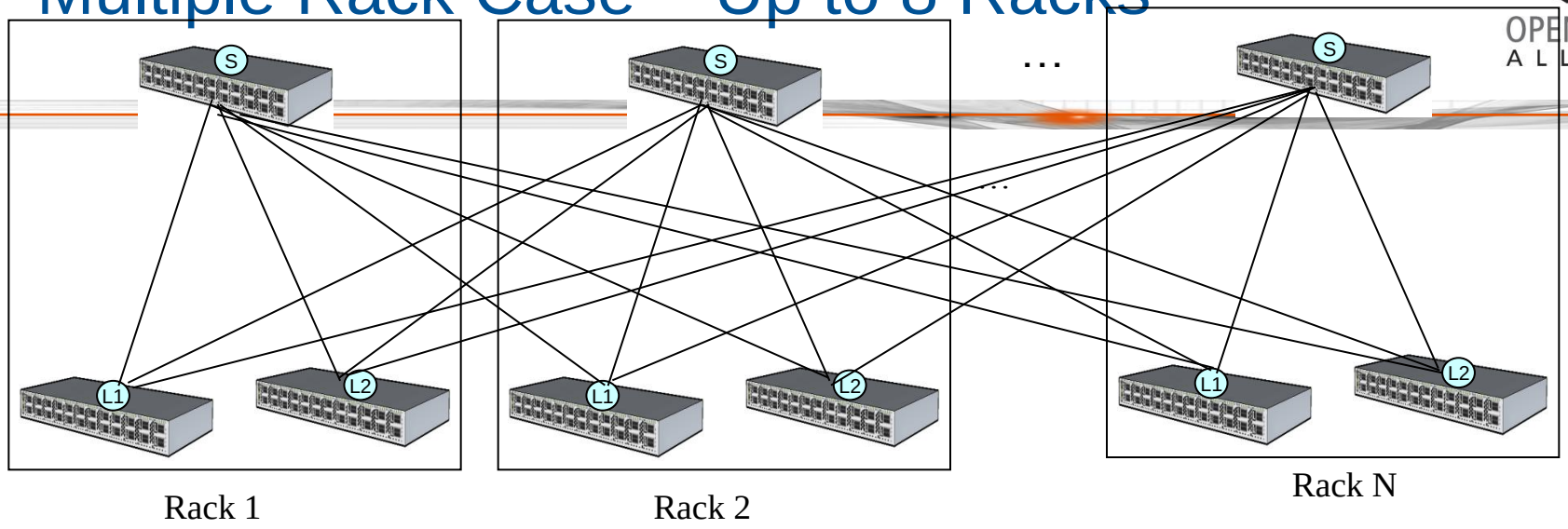
  OR
  - All with High Capacity 2 TB SAS disks
- 3 Sun Datacenter InfiniBand Switch 36
  - 36-port Managed QDR (40Gb/s) switch
- 1 “Admin” Cisco Ethernet switch
- Keyboard, Video, Mouse (KVM) hardware
- Redundant Power Distributions Units (PDUs)

Add more racks for additional scalability

Scaling Out to Multiple Full Racks

- Single InfiniBand Network
- Switch to a “Fat Tree” Topology
  - –Valid up to 8 Racks
- Many multi-racks sites

Multiple Rack Case – Up to 8 Racks



- Distribute 8 links from every “leaf” switch to every “spine” switch
  - 2 rack case – 4 links from every leaf to spine switch
  - 3 rack case - 3,3,2 links from each leaf to the 3 spine switches
  - 4 rack case – 2 links from each leaf to spine switch
  - 5,6,7,8 rack case – 1 or 2 links from each leaf to spine (8 links from each leaf)
- Greater than 8 rack requires larger external switches

Current Operational Model

- RDS kernel mode driver
  - used for RAC messages
  - Exadata I/0
- Socket interface with extensions for async I/O
  - Atomics
  - RDMA read / write
  - FMRs - we register / deregister every I/O to fence memory.
  - 1.7 million 8k IOPS over 4 HCAs..

Current Status cont…

- Certified RDS / TCP for RAC
- ROCEE being validated for RAC
- Ongoing work in RDS driver..
  - NUMA performance
  - Interrupt balancing
  - Ordering hinting.. enables multiple paths..
  - SRQ..
  - Resource re-use / preallocation.. For deterministic behavior under load..

NFS - RDMA

- User mode NFS client - based on TCP
  - many connections to filer.
  - Implemented NFS / RDMA from user mode via user mode verbs.
  - Only uses rdma for data xfers.. Still uses TCP for control plane.. Meta data ops..
  - We extended verbs to support FMRs = relaxed = lazy invalidate.
  - New ibv_sync_mr().. To force invalidation of unregistered fmrs…

New Oracle IPC

- RDS is great for Exadata…
  - relatively long operation times (I/Os).. 100 usecs.. From flash..
  - Event based model.. Client yields while waiting..
- New IPC.. (75% reduction) With udp, rds, rc, xrc transports.. 50% shorter code paths..
- Much simpler interface (dropped the MIT effect).
- rc + xrc are based on libibverbs.

RAC requires much lower latencies

- Moving to MPI like interfaces for some operations.
  - User mode busy wait for completion
  - Stalls hardware thread execution… and polls for CQ completions..
- Introducing Remote Memory Access Model
  - Clients operate on a declared data structure.. As if structure is always local..
  - Transparent where location of structure is
  - Uses rdma read, write, + atomics..

Oracle RMA model

```
struct my_obj { int var1;       char data[256]; };

peer1() {
        struct my_obj obj;      struct *rma_obj;

        init_rma_obj(&my_obj, rma_obj);

        while (1) {
                rma_update_barrier_begin(rma_obj);

                my_obj.var1++;
                my_obj.data = "this is a peer1";

                rma_update_barrier_end(rma_obj);
        }
}
```

Oracle RMA model

- Barrier / data sync operations
  - Dirty read, Dirty write
  - Consistent Read - remote host can be updating local memory while remote reader is reading it..
  - Consistent Update – local host can be reading data while remote host is updating it..
  - Serialized Read / Write – lock, read, write, unlock
- Atomics
  - Fetch add, Compare swap, variable sized data..
  - Transactional.. If updater dies in middle of update.. Update is rolled back..

Oracle RMA model

- Dirty Read – 2 usecs.. 256 bytes..
- Consistent Read – 3 usecs for 256 bytes..
  - Uses CRC for small data structures.
  - Uses sequence of 3 RDMA fenced reads for larger structs..
- PGAS model for “fixed” RMA object space.
  - All nodes contribute chunk of shmem address space.
  - Same size on all nodes.
  - All nodes can access shmem on all other nodes
  - All objects are at same offset in PGAS.. For each node

Oracle MSGQ

- Based on RDMA writes to remote rings
- Allocate ring in private or shared memory
- Client has access to remote ring and inserts variable sized msgs with rdma writes..
- 4 usecs update latency for 1kb msgs..
- If q is empty reader writes that it is waiting to writer via rdma write same if q is full..
- Wake involves sending small msg.. To get remote side scheduled again..

Future work..

- RMA helps with some problems..
  - Where simple structure manipulation meets run time requirements and operation time permits busy wait.
- However, if complex processing is required..
  - Lock and search hash list..if entry not found then insert new entry.. Etc..
  - requires multiple RMA operations.. Resulting in client yielding.. Can not busywait() for 20 usecs..

Offload Procedures

- Procedure structure
  1. Lock local host memory struct..
  2. Read in chunk of local host memory to HCA.
  3. Process chunk in local HCA scratch pad memory.
  4. Write chunk back to local host memory.
  5. Unlock local host memory struct.
  6. Return procedure result to initiator.
- Procedures complete in < 5 usecs.. Allowing client to busy wait..

Offload Procedures

- Embedded processors with O/S and support library..
- Host side daemons for debug / tracing / mgmt tasks etc..
- Procedure compiler + Load procedures into HCA..
- New QP type.. “offload” – supports all IB operations.. Procedures().. With IB ordering semantics.
- Procedures look like data packet to IB.. Except that they are picked off by remote HCA and handed to local handler running in HCA..
- Handler sends response which looks like incoming msg to verbs app..
- Error conditions can cause procedure to be forwarded to verbs app at target..