

Supporting Strong Coherency for Active Caches in Multi Tier Centers over Infiniband

SAN-03 Workshop (in conjunction with HPCA), Feb 2004

Cache Consistency and Cache Coherence

- Consistency
 - Non decreasing views of system state
 - Updates seen by all or none
- Coherency
 - Average staleness of data
 - Strong Coherence
 - Staleness is zero
 - Online banking, Airline reservation, Time-sensitive data
 - Weak Coherence
 - Bounded Staleness
 - Score Reporting Services , Non time-sensitive data

Approaches to ensure cache coherency

- No-Cache Approach
 - Each request processed at the home node
 - Propagation of request is expensive
 - Computation and communication overhead
- Client Polling
 - On each request, local cache is checked for availability
 - If available, coherence status (version) of object checked
 - If object is outdated, updated version is fetched

Objective: Design an architecture that very efficiently supports strong cache coherency on Infiniband

- External modules are used
 - Proxy Module
 - Validates cached objects associated with the request
 - Application Module
 - Arbitrates access in a multiple-reader single writer access pattern
- On Cache Hit
 - Back-end version check
 - If version current, use cached object
 - If version outdated, invalidate cached object
- On Cache Miss
 - Get data to cache
 - Initialize local versions

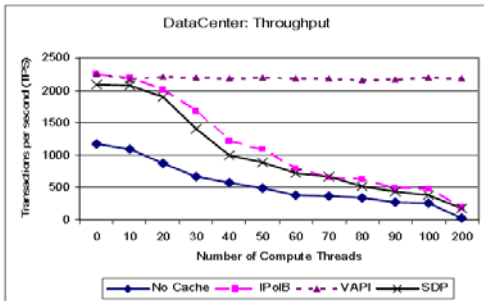
Implementation

- Every server has an associated module that uses either IPoIB, VAPI or SDP to communicate
- VAPI
 - VAPI module is contacted when request arrives
 - Latest version from back-end read using RDMA-Read operation
 - Latest version is cached and earlier one is invalidated

Experimental Test-bed

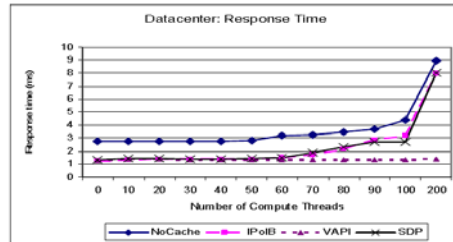
- Eight Dual 2.4GHz Xeon processor nodes
- 64-bit 133MHz PCI-X interfaces
- 512KB L2-Cache and 400MHz Front Side Bus
- Mellanox InfiniHost MT23108 Dual Port 4x HCAs
- MT43132 eight 4x port Switch
- SDK version 0.2.0
- Firmware version 1.17

Data-Center: Performance



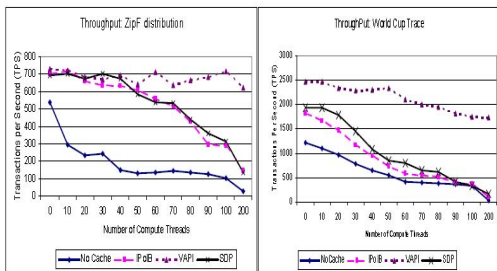
- The VAPI module can sustain performance even with heavy load on the back-end servers

Data-Center: Response Time



- The VAPI module responds faster even with heavy load on the back-end servers

Data Center : Application Throughput



Conclusions

- Architecture for supporting strong coherence
 - Compatibility with existing applications
 - Choice of transport
- Sockets API limitation
 - Two sided communication
- RDMA based design
 - Largest benefit due to one sided nature of RDMA calls

On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over Infiniband

IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05)

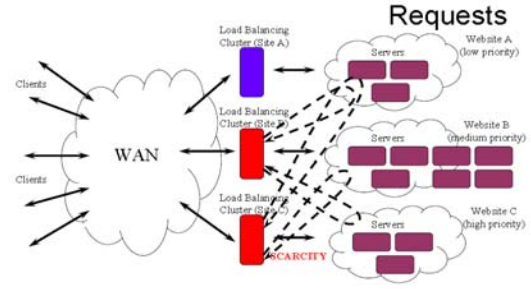
Why Dynamic Reconfigurability ?

- Evolution of shared multi-tier data centers
 - Data centers provide multiple independent services
- Satisfy dynamically varying resource Requirements of a website
- Service differentiation required
- QoS guarantees
 - Hard QoS guarantees
 - Soft QoS guarantees
- Static assignment of nodes
 - Under utilization of resources

Design (Cont)

- Support for Existing Applications
 - External Helper Modules (external programs running on each node) take care of load monitoring, reconfiguration
- Remote Memory Operation Based Design
 - Load balancer reconfigures with no interrupt to server
 - Highly resilient to server load

Issues with Reconf on High Priority Requests



High Priority website may get lesser number of servers compared to medium/low priority websites since Reconf does not have any idea about Prioritization between websites

Dynamic Reconfigurability with Prioritization

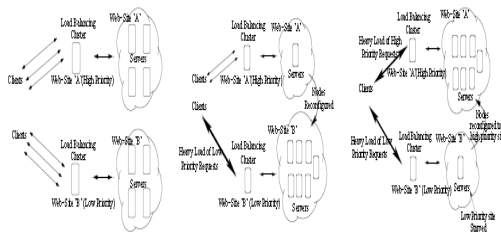
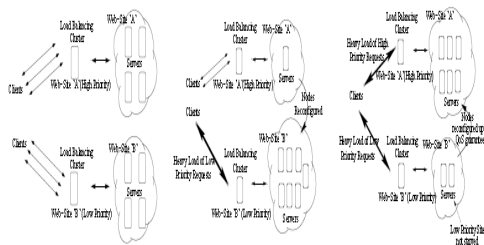


Figure 3. Dynamic Reconfigurability with Prioritization: (a) Step 1, (b) Step 2 and (c) Step 3

Dynamic Reconfigurability with Prioritization and Soft QoS guarantees

- Issues with Priority-Based
 - Starvation of lower priority websites
- Reconf-PQ (Priority and QoS) design used
 - Hard QoS guarantee
 - Minimum number of resources allotted to the website at any point of time
 - Soft QoS guarantees
 - Maximum resources allotted, if not already allotted to a higher priority website

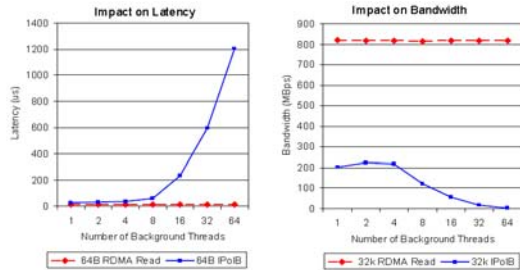
Reconf-PQ



Experimental Testbed

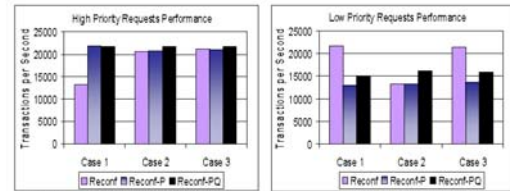
- Cluster 1 with:
 - 8 SuperMicro SUPER X5DL8-GG nodes; Dual Intel Xeon 3.0 GHz processors
 - 512 KB L2 cache, 2 GB memory; PCI-X 64-bit 133 MHz
- Cluster 2 with:
 - 8 SuperMicro SUPER P4DL6 nodes; Dual Intel Xeon 2.4 GHz processors
 - 512 KB L2 cache, 512 MB memory; PCI-X 64-bit 133 MHz
- InfiniBand Interconnect with:
 - Mellanox MT23108 Dual Port 4x HCAs; MT43132 24-port switch
- Apache 2.0.50 Web and PHP 4.3.7 servers; MySQL 4.0.12 Database

Testing Load Resilience



• Remote memory operations are not affected AT ALL with remote server load

Reconf-PQ Performance



> **Case 1:** A load of high priority requests arrives when a load of low priority requests already exists

> **Case 2:** A load of low priority requests arrives when a load of high priority requests already exists

> **Case 3:** Both high and low priority requests arrive simultaneously

• Reconf does not perform any additional reconfiguration

• Reconf and Reconf-P allocate maximum number of nodes to the low-priority website whereas Reconf-PQ allocates nodes to the QoS guaranteed to that website.

Conclusion

Shared Data-Centers are commonly used by several ISPs

- Resource Fragmentation
- Prioritization for high paying websites
- QoS guarantees for all websites

Extended our previous Dynamic Reconfigurability scheme

- Prioritization improves the performance of high priority websites
- QoS guarantees protect the low priority websites from scarcity of resources

High Performance Distributed Lock Management Services using Network-based Remote Atomic Operations

Int'l Symposium on Cluster Computing and the Grid, May 2007

Motivation

- Effective and efficient cooperation among processes across nodes is required
- Existing Approaches
 - Distribute per-lock overload (one server manages all operations for a pre defined set of locks)
 - Distributing each individual lock's overload (distributing the queue associated with the lock)

DLM (Distributed Lock Management) Approaches

- Send/Receive –based Server
 - Two sided send-recv primitives
 - Local node sends requests to remote node responsible for the key
 - Inherent ordering of messages
 - Free of deadlock and starvation
- Distributed Queue-Based Locking:
 - RDMA CS is used to create a distributed queue
 - 64-bit value used to represent the tail of queue
 - When a process wants to acquire a lock, it does a CS on this 64-bit value; If successful, the lock is granted. If not, RDMA CS is repeated to set the requesting process' rank – which is the tail of the queue.

Design of DLM (Distributed Lock Manager)

- Client-server model
- DLM server daemons on every node
- Applications make lock/unlock requests to their DLM daemons using IPC
- DLM maintains information of every lock associated with a key
- The keys (and hence the lock info) are partitioned among nodes

NCoSED – Exclusive Locking Protocol

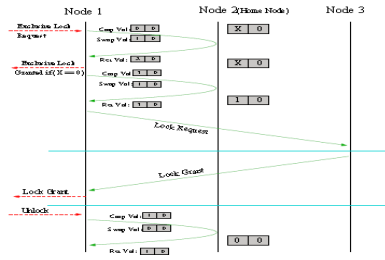


Figure 3. Locking protocols: (a)

Shared Locking Protocol

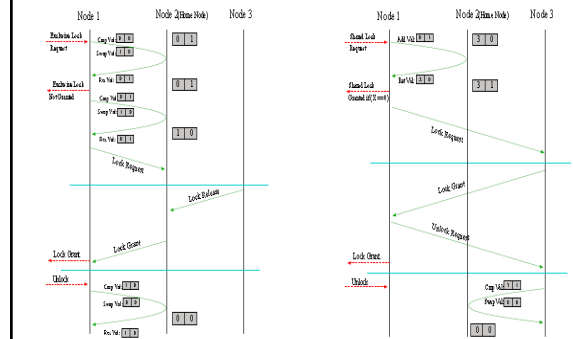
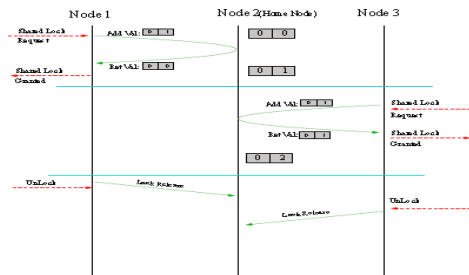


Figure 4. Locking protocols: (a) Shared followed by Exclusive (b) Exclusive followed by Shared

Experimental Test Bed

- 32-node Intel Xeon cluster
- Each node – 3.6 GHz Intel processor and 2GM main memory
- 64-bit CPUs
- MT25208 HCAs with PCI Express interfaces
- OFED 1.1.1 software distribution is used

Locking Operation Latency

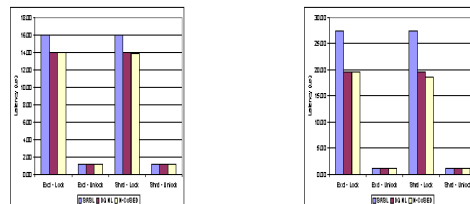


Figure 5. Basic Locking Operations' Latency: (a) Polling (b) Notification

Lock Cascading Effect

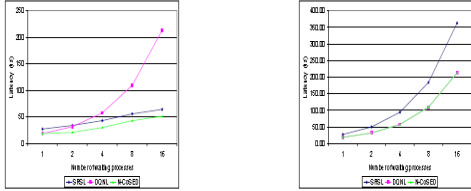


Figure 7. Lock Cascading Effect: (a) Shared Lock Cascade (b) Exclusive Lock Cascade

Conclusion

- Novel DLM protocol
- Eliminates two-sided communication protocols in the critical path
- Achieves significantly higher performance

Exploiting RDMA operations for Providing Efficient Fine Grained Resource Monitoring

Workshop on RDMA – Applications, Implementations and Technologies, Sept 2006

Approaches

- Back end Based Monitoring
 - Back end informs the front-end on detecting a high load
 - Dependent on system scheduling resolution
- Front end Based Monitoring
 - Front-end sends a request to retrieve load information
 - Asynchronous
 - Load calculating phase and load requesting phase are independent
 - Synchronous
 - Back-end calculates load for every request received from the front-end

Sockets Based Resource Monitoring

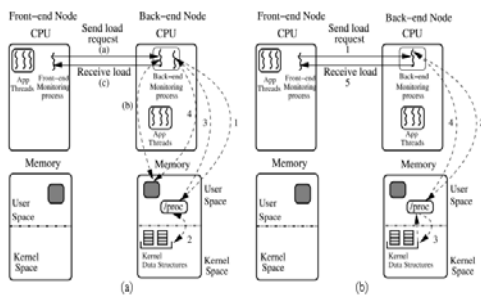


Figure 1. Sockets-based Resource Monitoring Mechanism (a) Asynchronous (b) Synchronous

RDMA Based Resource Monitoring

- RDMA Async
 - Back-end creates registered memory regions
 - Front-end monitoring process does RDMA read operations to retrieve load information
 - Similar to socket-async but one-sided
- RDMA Sync
 - RDMA – one sided, theoretically not sync
 - Accuracy of sync achieved by registering kernel data structure holding load info
 - Retriever can do a RDMA read directly on /proc
 - Removes the need of an extra load calculating thread

Performance- Latency

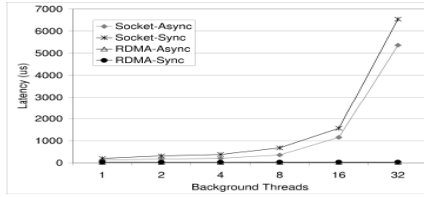


Figure 3. Latency of Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync schemes with increasing background threads

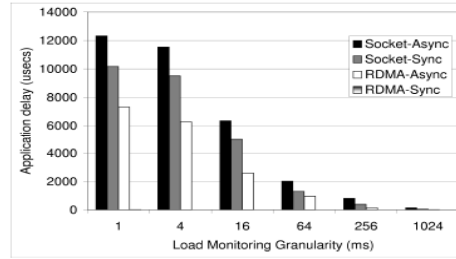
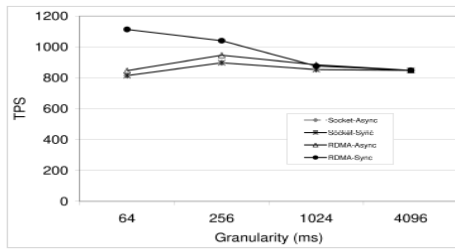


Figure 4. Impact on application performance with Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync schemes

Fine Grained vs Coarse Grained Processing



Thank You