Architecture for Caching Responses with Multiple Dynamic Dependencies in Multi-Tier Data-Centers over InfiniBand


The Ohio State University
Presentation Outline

- Introduction/Motivation
- Design and Implementation
- Experimental Results
- Conclusions
Introduction

- Fast Internet Growth
  - Number of Users
  - Amount of data
  - Types of services

- Several uses
  - E-Commerce, Online Banking, Online Auctions, etc

- Web Server Scalability
  - Multi-Tier Data-Centers
  - Caching – An Important Technique
Presentation Outline

- Introduction/Motivation
  - Multi-Tier Data-Centers
  - Active Caches
- Design and Implementation
- Experimental Results
- Conclusions
A Typical Multi-Tier Data-Center

- Tier 0: Proxy Nodes
- Tier 1: Application Servers
- Tier 2: Database Servers

Clients → WAN → Tier 0 → Tier 1 → Tier 2 → WAN

Apache

SAN
InfiniBand

- High Performance
  - Low latency
  - High Bandwidth
- Open Industry Standard
- Provides rich features
  - RDMA, Remote Atomic operations, etc
- Targeted for Data-Centers
- Transport Layers
  - VAPI
  - IPoIB
Presentation Outline

- Introduction/Motivation
  - Multi-Tier Data-Centers
  - Active Caches
- Design and Implementation
- Experimental Results
- Conclusions
Caching

- Can avoid re-fetching of content
- Beneficial if requests repeat
- Important for scalability
- Static content caching
  - Well studied in the past
  - Widely used
Active Caching

- **Dynamic Data**
  - Stock Quotes, Scores, Personalized Content, etc
  - Complexity of content

- **Simple caching methods not suited**

- **Issues**
  - Consistency
  - Coherency

![Diagram]

- User Request
- Proxy Node Cache
- Back-End Data
- Update
Cache Coherency

- Refers to the average staleness of the document served from cache
- Strong or immediate (Strong Coherency)
  - Required for certain kinds of data
  - Cache Disabling
  - Client Polling
Basic Client Polling

* SAN04: Supporting Strong Cache Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. Narravula, et. Al.
Multiple Object Dependencies

- Cache documents contain multiple objects
- A Many-to-Many mapping
  - Single Cache document can contain Multiple Objects
  - Single Object can be a part of multiple Documents
- Complexity!!
Client Polling

Request

Cache Hit

Version Read

Response

Cache Miss

Front-End

Back-End

Single Check Possible

Single Lookup counter essential for correct and efficient design
Objective

To design an architecture that very efficiently supports strong cache coherency with multiple dynamic dependencies on InfiniBand
Presentation Outline

- Introduction/Motivation
- Design and Implementation
- Experimental Results
- Conclusions
Basic System Architecture

Cache Lookup Counter maintained on the Application Servers
Basic Design

- **Home Node based Client Polling**
  - Cache Documents assigned home nodes

- **Proxy Server Modules**
  - *Client polling* functionality

- **Application Server Modules**
  - Support “Version Reads” for *client polling*
  - Handle updates
Many-to-Many Mappings

- Mapping of updates to dynamic objects
- Mapping of dynamic objects with Lookup counters
- Efficiency
  - Factor of dependency

Lookup counters  ────> Objects  ────> Updates
Mapping of updates

- Non-Trivial solution
- Three possibilities
  - Database schema, constraints and dependencies are known
  - Per query dependencies are known
  - No dependency information known
Mapping Schemes

- Dependency Lists
  - Home node based
  - Complete dependency lists

- Invalidate All
  - Single Lookup Counter for a given class of queries
  - Low application server overheads
Handling Updates

Application Server

Local Search and Coherent Invalidate

Update VAPI Send

Notification

 Ack (Atomic)

HTTP Request

HTTP Response

Application Server

Application Server

Application Server

Database Server

DB Query (TCP)

DB Response
Presentation Outline

- Introduction/Motivation
- Design and Implementation
- Experimental Results
- Conclusions
Experimental Test-bed

- **Cluster 1**: Eight Dual 3.0 GHz Xeon processor nodes with 64-bit 133MHz PCI-X interface, 512KB L2-Cache and 533 MHz Front Side Bus
- **Cluster 2**: Eight Dual 2.4 GHz Xeon processor nodes with 64-bit 133MHz PCI-X interface, 512KB L2-Cache and 400 MHz Front Side Bus
- Mellanox InfiniHost MT23108 Dual Port 4x HCAs
- MT43132 eight 4x port Switch
- Mellanox Golden CD 0.5.0
Experimental Outline

- Basic Data-Center Performance
- Cache Misses in Active Caching
- Impact of Cache Size
- Impact of Varying Dependencies
- Impact of Load in Backend Servers
- Traces Used
  - Traces 1-5 with increasing update rate
  - Trace 6: Zipf like trace
Basic Data-Center Performance

- Maintaining Dependency Lists perform significantly well for all traces
Cache Misses in Active Caching

- Cache misses for **Invalidate All** increases drastically with increasing update rates
• Maintaining Dependency Lists perform significantly well for all traces
• Possible to cache a select few and still extract performance
Impact of Varying Dependencies

- Throughput drops significantly with increase in the average number of dependencies per cache file
Impact of Load in Backend Servers

- Our design can sustain high performance even under high loaded conditions with a factor of improvement close to 22
Conclusions

- An architecture for supporting Strong Cache Coherence with multiple dynamic dependencies
- Efficiently handle multiple dynamic dependencies
  - Supporting RDMA-based Client polling
- Resilient to load on back-end servers
Web Pointers

http://nowlab.cis.ohio-state.edu/

E-mail: {narravul, balaji, vaidyana, jinhy, panda}@cse.ohio-state.edu
Back-up Slides
Cache Consistency

- Non-decreasing views of system state
- Updates seen by all or none

Diagram:
- Proxy Nodes
- Back-End Nodes
- User Requests
- Update
Performance

- Receiver side CPU utilization is very low
- Leveraging the benefits of One sided communication
The VAPI module can sustain performance even with heavy load on the back-end servers

* SAN04: Supporting Strong Cache Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. Narravula, et. Al.
### Mechanism

- **Cache Hit:**
  - Back-end Version Check
  - If version current, use cache
  - Invalidate data for failed version check
  - Use of RDMA-Read

- **Cache Miss**
  - Get data to cache
  - Initialize local versions
Other Implementation Details

- Requests to read and update are mutually excluded at the back-end module to avoid simultaneous readers and writers accessing the same data.
- Minimal changes to existing application software