2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Scalable Memcached design for InfiniBand Clusters using Hybrid Transports*

Jithin Jose[1], Hari Subramoni[1], Krishna Kandalla[1], Md. Wasi-ur-Rahman[1], Hao Wang[1], Sundeep Narravula[2], and Dhabaleswar K. Panda[1]

[1] *Department of Computer Science and Engineering, The Ohio State University*
{jose, subramon, kandalla, rahmanmd, wangh, panda}
@cse.ohio-state.edu

[2] *Yahoo! Inc. Sunnyvale, California*
{sundeepn}
@yahoo-inc.com

***Abstract*—Memcached is a general-purpose key-value based distributed memory object caching system. It is widely used in data-center domain for caching results of database calls, API calls or page rendering. An efficient Memcached design is critical to achieve high transaction throughput and scalability. Previous research [1] in the field has shown that the use of high performance interconnects like InfiniBand can dramatically improve the performance of Memcached. The Reliable Connection (RC) is the most commonly used transport model for InfiniBand implementations. However, it has been shown that RC transport imposes scalability issues due to high memory consumption per connection. Such a characteristic is not favorable for middlewares like Memcached, where the server is required to serve thousands of clients. The Unreliable Datagram (UD) transport offers higher scalability, but has several other limitations, which need to be efficiently handled.**

In this context, we introduce a hybrid transport model which takes advantage of the best features of RC and UD to deliver scalability and performance higher than that of a single-transport. To the best of our knowledge, this is the first effort aimed at studying the impact of using a hybrid of multiple transport protocols on Memcached performance. We present comprehensive performance analysis using micro-benchmarks, application benchmarks and realistic industry workloads. Our performance evaluations reveal that our Hybrid transport delivers performance comparable to that of RC, while maintaining a steady memory footprint. Memcached Get latency for 4 byte data size, is 4.28μs and 4.86μs for RC and hybrid transports, respectively. This represents a factor of twelve improvement over the performance of SDP. In evaluations using Apache Olio benchmark with 1,024 clients, Memcached execution time using RC, UD and hybrid transports are 1.61, 1.96 and 1.70 seconds, respectively. Further, our scalability analysis with 4,096 client connections reveal that our proposed hybrid transport achieves good memory scalability.

***Keywords*-Data-centers, Memcached, InfiniBand, Cloud Computing and Transaction Processing.**

I. Introduction

Over the past several years, dynamic web sites and applications have increased tremendously. Web applications, especially in the field of entertainment, social networking, media, etc. trend clearly toward dynamic application content. These increased dynamic data accesses put significant amount of load on origin site databases. Memcached [2] was proposed to alleviate this problem by introducing a "caching layer." Memcached is an open source, high-performance, distributed memory object caching system. It helps reduce the data access times and improves the ability for the underlying database to accommodate consistently higher loads. Memcached has been termed as the 'pillar' of Web 2.0 architecture. It provides an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering. Some of the popular Memcached users are Facebook, LiveJournal, Wikipedia, Flickr, Twitter and Youtube.

Performance and scalability are the key characteristics of an efficient Memcached deployment. The existing open-source Memcached implementation is designed to use the traditional BSD Sockets interface. Previous research in the field [1] has shown that the use of high performance interconnects like InfiniBand [3] (IB) and modern technologies such as RDMA can significantly improve the performance of Memcached. In [1], Memcached was layered on top of the Unified Communication Runtime (UCR) [4], a light-weight high performance runtime based on InfiniBand's connection-oriented Reliable Connection (RC) transport. However, exclusive use of RC transport hinders scalability due to high memory consumption. This may not be favorable for middlewares like Memcached, where the server may communicate with several thousands of clients, concurrently. InfiniBand's Unreliable Datagram (UD) transport addresses the scalability issue, but at the cost of reliability and large message performance. In this paper, we introduce a hybrid transport model which leverages the best features of RC and UD to deliver *both* high performance and scalability for Memcached. To the best of our knowledge, this is the first effort aimed at studying the impact of using a hybrid of multiple transport protocols on Memcached performance. We address the following challenges in this paper:

1) Can we leverage the best features of RC and UD to deliver *both* high performance and scalability to Memcached?
2) Can we design UCR with hybrid transports in a transparent manner so that Memcached can directly leverage our work?
3) How do we efficiently handle the reliability, flow control and large message communication latency problems associated with InfiniBand's UD transport?

Our performance evaluations reveal that hybrid transport delivers performance comparable to that of RC, while main-

This research is supported in part by U.S. Department of Energy grant #DE-FC02-06ER25755; National Science Foundation grants #CCF-0916302, #CCF-0937842 and #OCI-0926691; and grant from Wright Center for Innovation #WCI04-010-OSU-0.

978-0-7695-4691-9/12 $26.00 © 2012 IEEE
DOI 10.1109/CCGrid.2012.141

taining a steady memory footprint. Memcached Get latency for 4 byte data size, is 4.28μs and 4.86μs for RC and UD transports, respectively. This represents a factor of twelve improvement over the performance of SDP. In evaluations using Apache Olio benchmark with 1,024 clients, Memcached execution time using RC, UD and hybrid transports are 1.61, 1.96 and 1.70 seconds, respectively. Further, our scalability analysis with 4,096 client connections reveal that the hybrid transport achieves good memory scalability.

II. Motivation

In [1], we observed that Memcached's performance could be significantly improved through our Unified Communication Runtime (UCR) layer, based on InfiniBand's RC transport. However, a typical RC connection with standard configuration parameters requires around 64 KB of memory. Since a Memcached server may need to serve thousands of clients simultaneously, we need to explore design alternatives to improve the scalability.

InfiniBand provides a connectionless transport service, called Unreliable Datagram (UD). As a connectionless transport, the maximum memory used for connections remains static even as the number of connections increases. However, the UD transport layer brings several challenges, including providing reliability and flow-control. Additionally, the UD transport limits the maximum message size to the Message Transfer Unit (MTU) size and lacks support for Remote Direct Memory Access (RDMA). The MTU size on current Mellanox [5] hardware is limited to 2 KB.

In Figure 1, we compare the performance and scalability of Memcached, across three dimensions, *latency, throughput and scalability*, with three different InfiniBand transports *RC, UD and Hybrid*. Memcached based purely on RC may deliver good latency and throughput, but may not have the best scalability. Pure UD based Memcached designs may deliver good scalability, but may not deliver the same performance as Memcached with RC. Memcached based on a hybrid transport could potentially deliver the same performance as Memcached with RC and similar scalability as Memcached with UD.

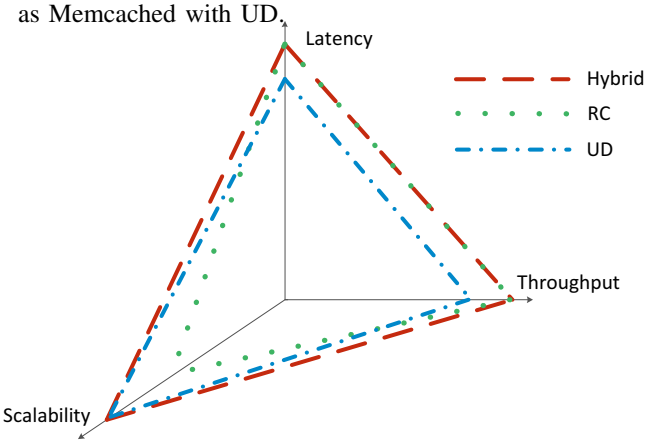


Figure 1. Memcached Performance with different InfiniBand Transports

III. Background

In this section we give the necessary background information for our work.

A. Memcached Overview

Memcached was proposed by Fitzpatrick [6] to cache database request results. Memcached was primarily designed to improve performance of the web site LiveJournal. Due to its generic nature and open-source distribution [2], it was quickly adopted in a wide variety of environments. Using Memcached, spare memory in data-center servers can be aggregated to speed up lookups of frequently accessed information, like database queries, results of API calls or web-page rendering elements.

Memcached clients can store and retrieve items from servers using "keys". Identification of destination server is done at the client side using a hash function on the key. Therefore, the architecture is inherently scalable as there is no 'central server' to consult while trying to locate values from keys. Even though the underlying architecture of Memcached is scalable, the implementation needs careful attention in order to scale to thousands of clients accessing data simultaneously.

B. InfiniBand Overview

InfiniBand [7] is a very popular switched interconnect standard being used by almost 41% of the Top500 Supercomputing systems [8]. InfiniBand offers various software layers through which it can be accessed. They are described below:

1) InfiniBand Verbs Layer: The *verbs* layer is the lowest access layer to InfiniBand. The verbs interface follows a Queue Pair (QP) and Completion Queue (CQ) model. Upper-level software layers can place work requests on a QP, which are processed by the Host Channel Adapter (HCA). Completions can be detected by either polling the CQ or by asynchronous events (interrupts).

2) IP-over-IB (IPoIB): InfiniBand also provides a driver for implementing the IP layer. This exposes the InfiniBand device as just another network interface, with an IP address. However, this layer does not provide OS-bypass.

3) InfiniBand Sockets Direct Protocol: Sockets Direct Protocol (SDP) is a byte-stream transport protocol that closely mimics TCP sockets stream semantics and is layered on top of IB message-oriented transfer model. SDP may offer complete OS-bypass, but it does not match the performance of verbs.

C. InfiniBand Transport Services

InfiniBand offers four transport modes - Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC) and Unreliable Datagram (UD). RC is a connection-oriented service and it requires a distinct QP per communicating peer. It provides RDMA capability, atomic operations and reliable service. UD is a connectionless and unreliable

transport. A single UD QP can communicate with any number of other UD QPs. However, UD does not offer RDMA, reliability and message ordering. Moreover, messages larger than an MTU size (2 KB in current Mellanox hardware) have to be segmented and sent in MTU size chunks.

D. 10 Gigabit Ethernet and iWARP Overview

10 Gigabit Ethernet was standardized in an effort to improve bandwidth in data-center environments, because traditional sockets interface may not offer high communication rates [9]. Towards that effort, iWARP standard was introduced for performing RDMA over TCP/IP [10]. Hardware accelerated versions of TCP/IP, TCP Offload Engines (TOE) can offer full socket streaming semantics efficiently in hardware. We used 10 Gigabit Ethernet adapters from Chelsio Communications in this study.

E. Olio Application and Faban Benchmark Framework

Olio [11] is a Web 2.0 toolkit developed at Sun Microsystems for evaluating the performance of web technologies. It defines a Social Event Calendar application and provides implementations on PHP, Java (EE) and RubyOnRails (ROR). It can be used to evaluate web technologies such as AJAX, Memcached, MobileFS, etc. It simulates a high read-to-write ratio for database accesses, which is common to social web sites. Faban [12] is a benchmark harness and driver development framework. It can generate workloads for benchmarks like Olio. Faban driver framework defines operations, transactions and associated statistics collection and reporting. It has several configuration options for selecting different web technology services, number of concurrent users, database load, etc.

IV. Unified Communication Runtime (UCR)

The Unified Communication Runtime (UCR) [4] is a light-weight, high performance communication runtime, designed and developed at The Ohio State University. It aims to unify the communication runtime requirements of scientific parallel programming models, such as MPI and Partitioned Global Address Space (PGAS) along with those of data-center middle-ware, such as Memcached [6], HBase [13], Map Reduce [14], etc.

The requirements imposed by these disparate models are quite challenging. For example, in the parallel computing domain, there is a notion of a parallel "job." Processes belonging to a job may communicate with others using a rank or a thread id (in case of PGAS). This is quite different from the Memcached model, where clients maintain a list, or a pool of available Memcached servers. Based on the data they want to access, keys are assigned to the data and a server is chosen using a hash function. Another important distinction is fault tolerance. In MPI or PGAS, when a process belonging to a job unexpectedly fails, the entire job fails. However, in the data-center domain, failure of one Memcached server or client must be tolerated.

UCR is designed as a native library to extract high performance from advanced network technologies. The UCR project draws from the design of MVAPICH and MVAPICH2 software. MVAPICH and MVAPICH2 [15] are very popular implementations of the MPI-1 and MPI-2 specifications. They are being used by more than 1,850 organizations in 66 countries and also distributed by popular InfiniBand software stacks, such as OpenFabrics Enterprise Distribution (OFED) and Linux distributions like RedHat.

V. Design

As discussed in Section II, Memcached based on UCR with UD transport could potentially achieve better scalability, while delivering similar performance when compared to Memcached based on UCR with RC transport. However, the UD transport has several limitations (Section III-C). We propose the following solutions to address these challenges and improve the scalability of Memcached.

Reliability and Flow Control: With UD transport, it is critical to efficiently handle reliability and flow control. Our UCR-UD design is based on the traditional sliding window protocol. The sender issues packets depending on the space available in the window, which represents the maximum number of unacknowledged message segments. Send operations that may occur when the window is already full are queued until outstanding operations have been acknowledged. We assign a time-stamp to each segment that is sent and if an ACK has not been received within a given timeout, the segment is re-transmitted.

Large Message Transfers: RC transport can employ Remote Direct Memory Access (RDMA) efficiently for large message transfers. With UD, messages larger than the MTU size have to be segmented and sent as separate packets and they have to be reassembled at receiver, which involves two memory copies. We employ the Zero Copy Transfer protocol [16] to address these limitations.

Hybrid Connection Management: We also consider hybrid connection models [17]. Using this approach, we try to leverage the advantages of both RC and UD. We can limit the maximum number of RC connections to a specified threshold. Further connections can be made over UD transport. This ensures that the memory requirement does not increase above a particular limit. Another possibility is to dynamically change the transport mode from UD-to-RC (upgrade) or RC-to-UD (downgrade), based on the communication frequency between the server and a specific client, and we can also impose a limit on the maximum number of RC connections. These hybrid modes can be selected depending on platform characteristics.

Communication Performance: Along with the transport level enhancements, we have also introduced a 'Registration Cache' in UCR. For InfiniBand, the communication buffers have to be registered with the adapter. However, the memory registration and de-registration operations are expensive and

if they are done for every communication operation, it significantly affects performance. Registration cache allows us to re-use registered memory pages and these are deregistered only when they are evicted from the cache.

Memcached based on UCR-UD: We have designed the UCR-UD interface to allow Memcached to transparently use the same set of API's. The connection transport type is identified by the `transport_type` argument, during the creation of the communication end-point. Our design guarantees that Memcached can conveniently leverage our work without any changes. Hybrid configuration parameters can be set using run-time environment variables. UCR transparently switches between transport protocols in hybrid mode for best performance. We have also made several enhancements over the Memcached design presented in [1]. These include support for multiple Memcached servers and handling of Memcached operations such as 'delete' and 'flush'. We use the multi-server configuration in one of the application benchmarks. Detailed description and results are presented in Section VI-D.

VI. Performance Evaluation

We describe the experimental setup and the results of our performance analysis of Memcached in this section. We present micro-benchmark and application-benchmark level analysis as well as results based on real application workloads. The micro-benchmarks used in experiments are based on the popular `memslap` benchmark distributed along with Memcached client library. The application benchmark used is Apache Olio [11] benchmark. We compare the performance of Memcached design over UCR (UD and RC) with that of the socket variants (SDP, IPoIB, 1GigE and 10GigE). We used Memcached server [2] version 1.4.9 and client (libmemcached) [18] version 0.52. For socket versions, the client behavior was set using `MEMCACHED_BEHAVIOR_TCP_NODELAY`. This resulted in better and more predictable latency performance. The clusters which we used for our experiments have 8 CPU cores per node. Memcached Server was configured with 8 'worker threads' for better utilization of CPU cores.

A. Experimental Setup

We used two generations of multi-core architectures from Intel distributed across two different clusters for our evaluation.

Intel Westmere Cluster (Cluster A): This cluster consists of 144 compute nodes with Intel Xeon Dual quad-core processor nodes, operating at 2.67 GHz. Each node has 12 GB of memory and is equipped with MT26428 QDR ConnectX HCAs (32 Gbps data rate) with PCI-Ex Gen2 interfaces. The nodes are interconnected using 171-port Mellanox QDR switch. The operating system used is Red Hat Enterprise Linux Server release 5.4 (Tikanga), with kernel version 2.6.18-164.el5 and OpenFabrics version 1.5.1. This cluster represents the leading edge of the technologies used in commodity clusters. It is termed as "Cluster A" in experiment results.

Intel Clovertown Cluster (Cluster B): This cluster consists of 64 compute nodes with Intel Xeon Dual quad-core processor nodes operating at 2.33 GHz with 6 GB RAM, a PCIe 1.1 interface. The nodes are equipped with a ConnectX DDR IB HCA (16 Gbps data rate) as well as a Chelsio T320 10GbE Dual Port Adapter with TCP Offload capabilities. The operating system used is Red Hat Enterprise Linux Server release 5.5 (Tikanga), with kernel version 2.6.30.10 and OpenFabrics version 1.5.1. The IB cards on the nodes are interconnected using a 144 port Silverstorm IB DDR switch, while the 10 GigE cards are connected using a Fulcrum Focalpoint 10 GigE switch. This cluster represents the trailing edge of the technologies used in commodity clusters. This cluster is termed as "ClusterB" in experiment results.

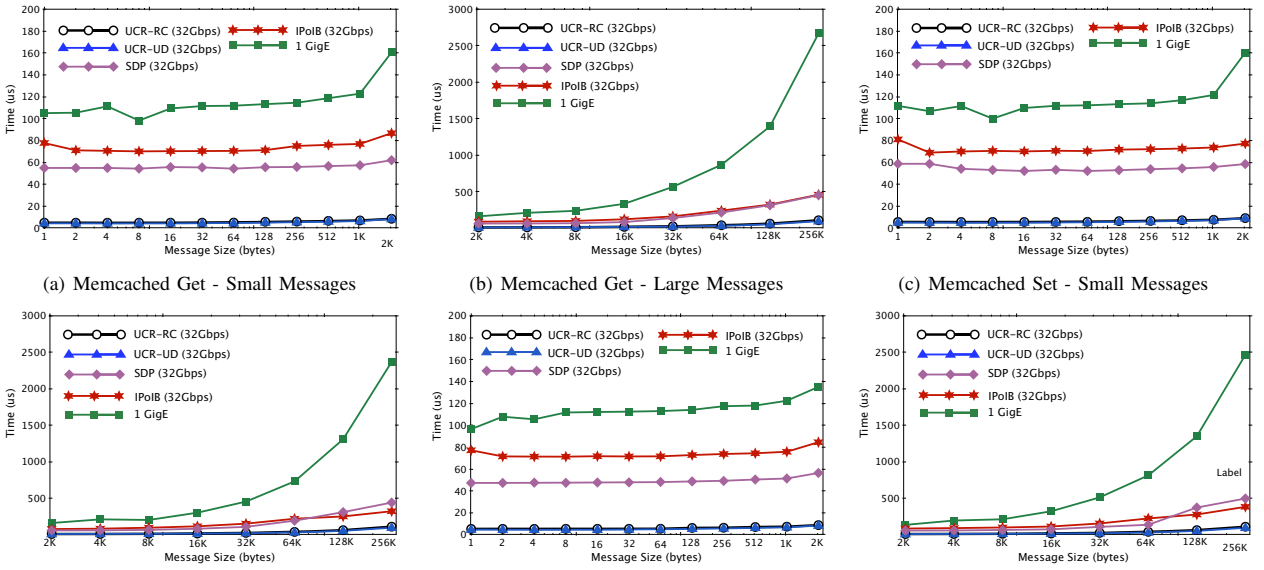
B. Microbenchmark Analysis

We present the performance results of Memcached Set/Get operations, transaction throughput evaluation and scalability analysis in this section.

1) Performance of Memcached Set/Get operations: This set of experiments measure the performance of the key Memcached operations, *Set* and *Get*. In the experiment, Memcached client repeatedly issues *Set* (or *Get*) operations for a fixed number of iterations. We repeated this for different message sizes. The average operation latency is reported in the results. We did this experiment for different *Set*/*Get* mixtures: 100% *Set*, 100% *Get* and 50% *Set* - 50% *Get*. We performed the evaluation on clusters A and B and results are shown in Figure 2 and Figure 3, respectively.

We compare the performance of Memcached design using UCR (both RC and UD transports) with that of pure Memcached implementation on various socket protocols. Cluster A is configured with the following socket protocols - SDP, IPoIB, 1GigE, and 10GigE. Figure 2(a) and Figure 2(b) depict the performance of Memcached *Get* operations on Cluster A. We have split the results into two graphs for fine grained analysis based on message size. As it can be observed from the figure, Memcached over UCR achieves lowest latency. For 4 byte message size, the *Get* operation latency is just about 4.28 μs and 4.86 μs for Memcached design over UCR-RC and UCR-UD, respectively. The operation latency of the best performing socket version, SDP is 54.32 μs. This is a factor of 12 X higher than the latency of Memcached over UCR. The performance gain exists for larger message sizes also. The operation latencies of *Set* operation and mixed *Get*/*Set* operations are presented in Figures 2(c) - 2(f). The performance patterns of these experiments are similar - Memcached over UCR achieves the lowest latency.

Figure 3 depicts the results in Cluster B. Since this cluster contains 10 GigE cards, we present Memcached performance results over 10 GigE also. For 4 byte message



(a) Memcached Get - Small Messages

(b) Memcached Get - Large Messages

(c) Memcached Set - Small Messages

(d) Memcached Set - Large Messages

(e) Memcached Mixed Operations (50% Get, 50% Set) - Small Messages

(f) Memcached Mixed Operations (50% Get, 50% Set) - Large Messages

Figure 2. Latency of Memcached Operations - Cluster A

size, the *Get* operation latency is just about 6.82 μs and 7.73 μs for Memcached design over UCR-RC and UCR-UD, respectively. The operation latency of the best performing socket version, 10 GigE is 48.86 μs. This is a factor of 7 X higher than the latency of UCR versions. The latency results of UCR-RC and UCR-UD are a little higher than those observed in Cluster A. This is because Cluster B contains a slower InfiniBand adapter (16 Gbps) as compared to Cluster A (32 Gbps).

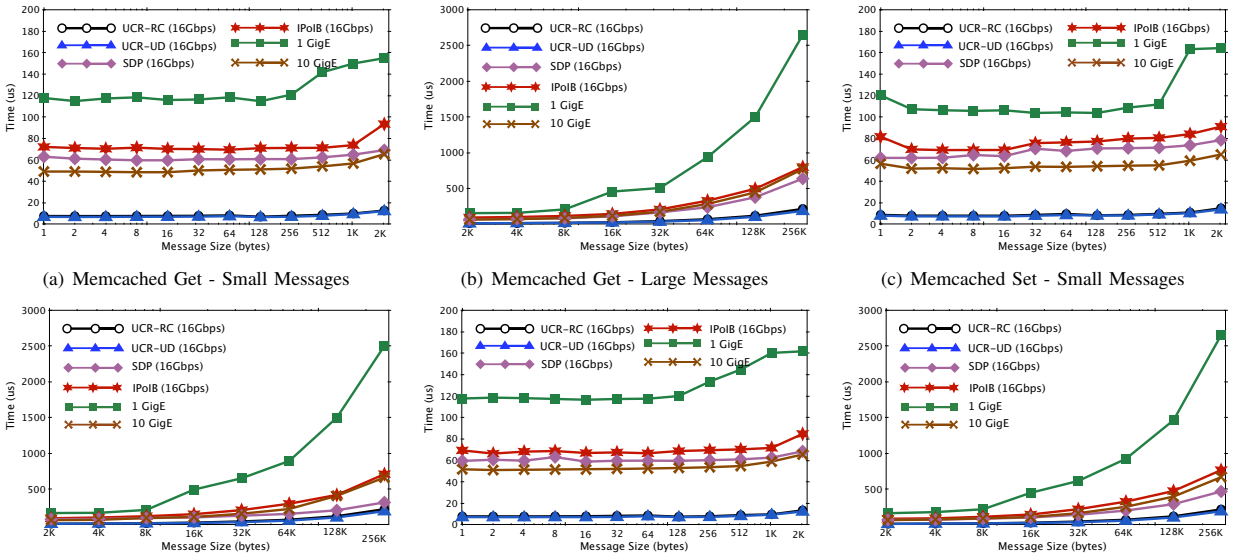
It is to be noted that the performance of UCR-UD is a little lower as compared to UCR-RC in both the clusters. RC transport offers built-in reliability, which is implemented in hardware layer. But, UD transport does not guarantee reliability and flow control, and these mechanisms shall be implemented in the software layer. Because of these software overheads, UD performance is a little lower than RC. Moreover, messages larger than MTU size (2 KB) have to be segmented and shall be sent in chunks. On the other hand, RC transport offers high performance features such as RDMA for large message transfers.

2) Transaction Throughput: Memcached transaction throughput indicates the maximum number of *Get/Set* operations that the server can handle per second. In the experiment, we used one Memcached server and varied the number of clients from 1 to 1,024. We used *Get* of 4 bytes as the operation. We ran this experiment on Cluster A using Memcached over UCR (RC and UD) and Memcached over socket interfaces (SDP, IPoIB and 1 GigE). Results are shown in Figure 4(a). As it can be observed from the figure, Memcached over UCR delivers higher throughput. Memcached over UCR-RC and UCR-UD provide peak throughput of 1.4 M operations per second and 1.3 M operations per second, respectively. This is 3 X times higher than that of Memcached over SDP.

It is interesting to note that the transaction throughput of Memcached over UCR increases and reaches a peak at 8 clients and then stabilizes to around 750 K operations per second. Cluster A has compute nodes with 8 CPU cores. We configured Memcached server (both UCR and socket versions) with 8 worker threads, so that all the cores are utilized efficiently. This is the reason we obtain peak throughput at 8 clients. When the number of clients is increased further, each worker thread serves more than one client and CPU performance becomes the bottleneck. Even though network offers more bandwidth, CPU performance bottlenecks the throughput. This trend is not visible in socket versions because network bandwidth is the bottleneck for these. Similar characteristics were observed in previous Memcached performance analysis [19].

3) Scalability Analysis: In this section we analyze the memory requirement of Memcached server, when scaled up. We measured the memory footprint for varying number of clients connections - 1 to 4 K. The results are presented in Figure 4(b). As it can be observed from the figure, the socket based versions have higher memory footprint, around 630 MB. Socket based design keeps per-thread shared connection buffer pool for TCP and UDP sockets. Each buffer pool is about 64 MB, and the experiments were run using 8 worker thread configuration. It is to be noted that keeping per thread buffer pool might not be efficient especially when multi/many core clusters are evolving.

InfiniBand RC transport needs to create QP for each client connection (Section III-C). Size of each QP depends on the queue size. We used a typical configuration of 64 'send work queue' elements and 128 'receive work queue' elements. As



(a) Memcached Get - Small Messages

(b) Memcached Get - Large Messages

(c) Memcached Set - Small Messages

(d) Memcached Set - Large Messages

(e) Memcached Mixed Operations (50% Get, 50% Set) - Small Messages

(f) Memcached Mixed Operations (50% Get, 50% Set) - Large Messages

Figure 3. Latency of Memcached Operations - Cluster B

the number of client connections increase, memory footprint of Memcached server with UCR-RC increases. For 4 K client connections, it almost doubles the initial memory footprint. For UD transport, there is no need to create new QP for each connection. As a result, it keeps a steady memory footprint size. This behavior is favorable for applications like Memcached. If there is an increase in memory requirement as the connections increase, it can adversely affect the memtable size and degrade the performance when scaled up. The initial memory footprint (when number of client connections is zero) of UCR-UD is a little higher as compared to UCR-RC. This is because more receive buffers are posted in UCR-UD for handling connections at scale.

We ran the experiment for hybrid connection model as well. In this model, we limit the maximum number of RC connections to 512. It can be observed that the memory footprint of UCR-Hybrid remains steady after 512 client connections. Socket based versions keep a steady memory footprint, but the performance is not at all comparable to UCR based variants.

C. Apache Olio Benchmark Results

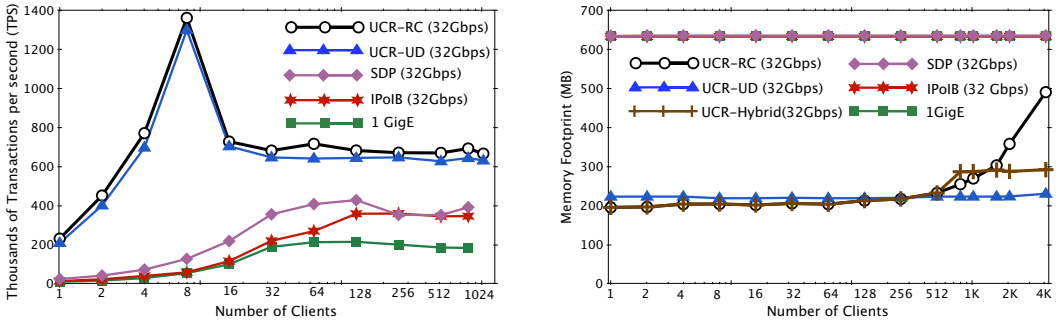
We evaluated our design using application benchmarks as well. We used Apache Olio benchmark [11] and simulated the workload using Faban [12]. Olio is a social web calendar application. A detailed description of Olio Benchmark and Faban Workload Generator is given in Section III-E.

In our experiment, we used the workload for 25,000 application users per Memcached client. We varied the number of Memcached clients from 1 to 1,024 and the average time for handling workloads per user is reported. The workload contains a mixture of Memcached *Set* and *Get* operations (90% *Get*s and 10% *Set*s). The data size of these operations vary in the range of 4 bytes to 32 Kbytes.

Results of this experiment are shown in Figure 5. Figure 5(a) compares the performance of Memcached over UCR with that of the socket variants. As it can be observed from the figure, Memcached-UCR over RC and UD performs significantly better than the socket versions. For 8 client (25,000 x 8 users) benchmark, UCR-RC and UCR-UD takes around 12.95 ms and 17.52 ms for complete. This is a factor of four better than the IPoIB results. It is to be noted that even though SDP performed better than IPoIB in micro-benchmarks, it does not hold true in case of application results. For 8 client benchmark, Memcached over SDP took 99.94 ms whereas IPoIB took 48.13 ms. This holds true for other application benchmarks also, as discussed in Section VI-D.

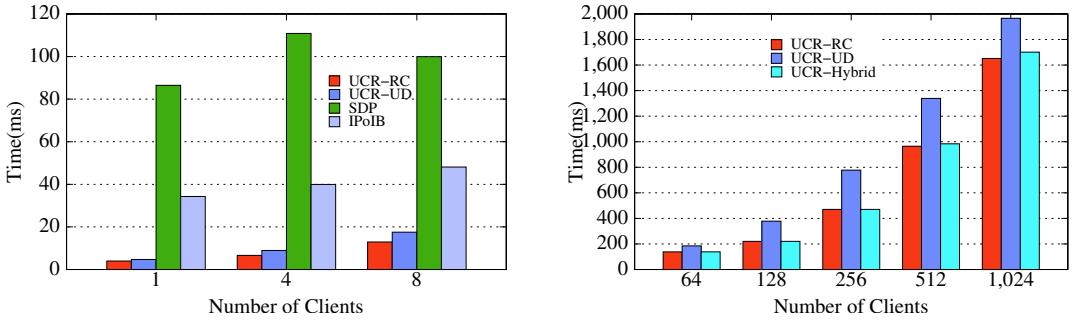
We performed the experiment for higher scale also. These results are depicted in Figure 5(b). In order to do a fine grain analysis of UCR-RC and UCR-UD, we did not include the results of socket based variants. Since the workload contains large data sizes, there is a visible performance difference between UCR-UD and UCR-RC results. This is because UCR-RC employs RDMA for large message transfer. Since RDMA support is not there in UD transport, messages are segmented and sent in 2 KB packets.

We used UCR in a hybrid connection mode as well, in which some connections use RC transport and some other use UD. We limited the maximum number of RC connections to 512. This can be set as a runtime parameter and it can be tuned as per system/application characteristics. The hybrid mode results are indicated as 'UCR-Hybrid' in the results. It can be noted that UCR-RC and UCR-Hybrid results are almost same for less than 512 client connections.



(a) Transaction Throughput (b) Memory Footprint

Figure 4. Scaling Memcached - Transaction Throughput and Memory Footprint



(a) Comparison with socket-based designs (b) Comparison with RC Transport

Figure 5. Apache Olio Benchmark Results

For 1,204 clients, UCR-RC took 1651.56 ms, whereas for hybrid mode took 1701.35 ms. With hybrid mode we achieve comparable performance to that of RC, while keeping a limit on memory footprint.

D. Results based on Real Application Workloads

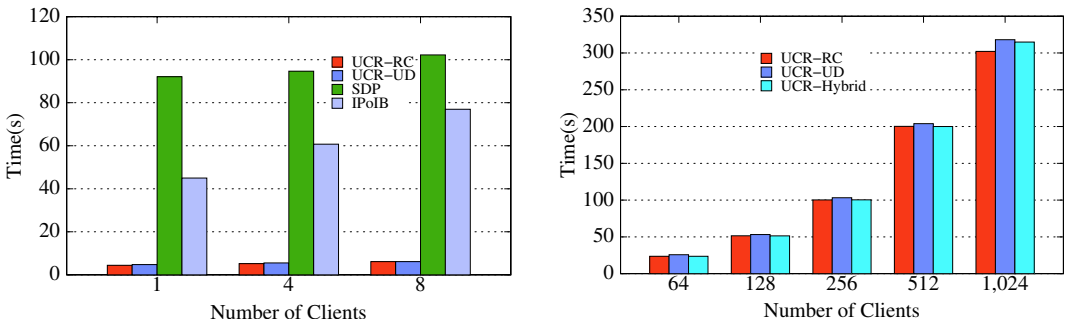
In order to further evaluate our design, we ran experiments based on real application workloads. Our benchmarks are based on Internet workloads handling user actions. User actions on the Internet are typically kept lightweight, in order to avoid slow browsing experience. These generate condensed logs on the server side with information such as browser cookies, session cookies and other such concise identifiers. Processing of these user action logs involves attribution/lookups with the actual complete information needed for further action. Such functions are often batched and processed in small chunks. With current Internet scales, such lookup data can be very large and performance requirements are quite high.

In our experiment, we used the following configuration. Each Memcached client issues 0.5 million *Get* operations followed by *Set* operations which update 20% of data set. The value length varies from 1 KB to 4 KB and key length varies from 3 to 20 bytes. These benchmarks exhibit 'weak scaling' characteristics. We increased the total data size, with increase in number of clients. Each client adds 2,000 key-value pairs. The entire data set was cached using three Memcached servers. Read and update operations were repeated over several iterations and the average time is reported.

Figure 6(a) compares the performance of Memcached over UCR with that of socket based variants. The performance characteristics are similar to that of Olio Benchmark results. Memcached over UCR performs significantly better than other socket based variants. For 8 clients, UCR-RC and UCR-UD take 6.1 ms and 6.2 ms, respectively. This is about 12 X times better than IPoIB performance (72.6 ms). As observed in Olio results, SDP does not perform as good as IPoIB. Figure 6(b) depicts the performance of Memcached over UCR at scale. Since the data size is comparatively smaller to that of Olio, the performance of UCR-RC and UCR-UD are almost similar. UCR-Hybrid (number of RC connections capped to 512) results are also presented. For 1024 clients, the execution times using UCR-RC, UCR-UD and UCR-Hybrid were 302.18, 318.04 and 314.93 ms, respectively.

VII. Related Work

Memcached has been actively researched and widely used in recent years. In [20], Paul Saab et al. from Facebook enhanced Memcached scalability and throughput through using UDP to reduce network traffic and global lock contention. In [19], Shanti et al, enhanced Memcached performance on Sun's Nehalem system through using per-thread stats to remove the stats lock. The authors in [21] and [22] introduced methods to improve Memcached scalability through improved server deployment strategies. These ideas were based on traditional Ethernet networks. In [1], we proposed a high performance design for Memcached over InfiniBand networks and we observed significant improvements in terms



(a) Comparison with socket-based designs

(b) Comparison with RC Transport

Figure 6. Real Application Workload Results

of operation latency and throughput. In this paper, we explore the challenges of leveraging the benefits of a hybrid RC/UD transport to deliver *both* performance and scalability to Memcached.

VIII. Conclusion

In this work, we developed a scalable and high performance Memcached design using hybrid InfiniBand transports. We implemented the hybrid RC/UD transport in the Unified Communication Runtime (UCR) and layered Memcached on top of UCR. We addressed several issues with the UD transport including reliability, flow control and communication performance. We also proposed schemes that allow connections to transparently switch from UD and RC to offer a hybrid transport to the Memcached stack. Through our work, we could achieve a good balance of performance and scalability by using efficient hybrid schemes. Our experimental evaluations revealed that our design is highly scalable, with comparable performance as that of RC.

Memcached Get latency for 4 byte data size is 4.28μs and 4.86μs for RC and UD transports, respectively. Latency results for 4 KB data size is about 9μs and 10μs for RC and UD transports, respectively. This is a factor of 6 X to 12 X better than SDP performance. In evaluations using Apache Olio benchmark with 1,024 clients, Memcached execution time using RC, UD and hybrid transports are 1.61, 1.96 and 1.70 seconds, respectively.

IX. Acknowledgment

The authors would like to thank Dr. Sayantan Sur of Intel Corporation, Dr. Chet Murthy of IBM Research and Devendar Bureddy of The Ohio State University for many useful discussions on this topic and constant encouragement.

References

[1] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. W. Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda, "Memcached Design on High Performance RDMA Capable Interconnects," in *International Conference on Parallel Processing (ICPP)*, Sept 2011.

[2] "Memcached: High-Performance, Distributed Memory Object Caching System," http://memcached.org/.

[3] "InfiniBand Trade Association," http://www.infinibandta.com.

[4] J. Jose, M. Luo, S. Sur, and D. K. Panda, "Unifying UPC and MPI Runtimes: Experience with MVAPICH," in *Fourth Conference on Partitioned Global Address Space Programming Model (PGAS)*, Oct 2010.

[5] Mellanox Technologies, "ConnectX Host Channel Adapters," http://www.mellanox.com/pdf/products/hca/ConnectX_IB_Card.pdf.

[6] B. Fitzpatrick, "Distributed caching with memcached," *Linux Journal*, vol. 2004, pp. 5–, August 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1012889.1012894

[7] Infiniband Trade Association, http://www.infinibandta.org.

[8] Top500 Supercomputing System, http://www.top500.org.

[9] P. Balaji, H. V. Shah, and D. K. Panda, "Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck," in *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT), in conjunction with IEEE Cluster*, 2004.

[10] RDMA Consortium, "Architectural Specifications for RDMA over TCP/IP," http://www.rdmaconsortium.org/.

[11] Olio, http://incubator.apache.org/olio.

[12] Faban Harness and Benchmark Framework, http://java.net/projects/faban.

[13] Apache HBase, "The Apache Hadoop Project," `http://hbase.apache.org//`.

[14] Hadoop Map Reduce, "The Apache Hadoop Project," `http://hadoop.apache.org/mapreduce///`.

[15] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, http://mvapich.cse.ohio-state.edu/.

[16] M. Koop, S. Sur and D. K. Panda, "Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram," in *Cluster'07*, 2007.

[17] M. Koop and T. Jones and D. K. Panda, "MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand," in *IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS 2008)*, April 2008.

[18] "libmemcached: Open Source C/C++ Client Library and Tools for Memcached," http://libmemcached.org/.

[19] Memcached Performance on Sun's Nehalem System, http://137.254.16.27/shanti/entry/memcached_on_nehalem1.

[20] Paul Saab, "Facebook Engineering Notes: Scaling Memcached at Facebook," https://www.facebook.com/note.php?note_id=39391378919.

[21] Designing and Implementing Scalable Applications with Memcached and MySQL, http://www.mysql.com/why-mysql/white-papers/mysql_wp_memcached.php.

[22] Scalling, and Deploying Memcached with Libmemcached, http://opensourcebridge.org/proposals/701.