Configuration and Deployment Guide For Memcached on Intel® Architecture

About this Guide

This *Configuration and Deployment Guide* explores designing and building a Memcached infrastructure that is scalable, reliable, manageable and secure. The guide uses experience with real-world deployments as well as data from benchmark tests. Configuration guidelines on clusters of Intel® Xeon®- and Atom™-based servers take into account differing business scenarios and inform the various tradeoffs to accommodate different Service Level Agreement (SLA) requirements and Total Cost of Ownership (TCO) objectives.

Table of Contents

1.0 Introduction

With the growth of online commerce and social media, more Web pages have become dynamic—their content varies based on parameters provided by users or by software. Dynamic Web pages store content in databases rather than HTML pages to support uses like online credit card transactions and ad serving.

The surging use of dynamic Web content can overwhelm traditional data centers as they strive to comply with Service Level Agreements (SLAs). Specifically, the memory capacity of any given server in the data center may be insufficient to support the in-memory operations required for timely execution and a good user experience.

Originally developed by Brad Fitzpatrick in 2003 to address dynamic Web performance and scalability for his *LiveJournal* Website, Memcached is an open-source, multi-threaded, distributed, Key-Value caching solution commonly used for delivering software-as-a-service, reducing service latency and traffic to database and computational servers. It implements a coherent in-memory RAM cache that scales across a cluster of servers. Memcached exploits the fact that the typical access time for DRAM is orders of magnitude faster than disk access times, thus enabling considerably lower latency and much greater throughput. So a caching tier is employed that can cache "hot" data and reduce costly trips back to the database to read data from disk.

Memcached delivers a performance boost for applications because it caches data and objects in memory to reduce the number of times an external data source must be read from disk. By making frequently accessed information immediately available in memory, Memcached provides an important speed advantage when application performance is critical and resource constraints make accessing the original data from disk too time consuming and expensive.

In dynamic Web applications, where a large percentage of the information served to users results from back-end processing, the ability to cache the values in RAM has proved to be advantageous, especially in response to fast-trending social media peak usage patterns and so-called "hot-key" requests (e.g., a surge of requests for a "hot" video clip, picture or other piece of popular content).

During the past 10 years, Memcached has been implemented by major cloud and Web service delivery companies such as Facebook, Twitter, Reddit, YouTube, Flickr, Craigslist to reduce latency for serving Web data to consumers and to ease the demand on back-end database and application servers. In addition, many enterprise data centers also implement Memcached to improve the response-times for a range of database-oriented applications.

2.0 Driving Forces and Strategic Considerations

The extreme rate of data growth associated with Cloud usage models, and the cost advantage of spinning media over alternatives for long-term retention of that data put a crunch on data centers for both public cloud service providers and for internal enterprise operations. Front-end service demands already go well beyond the capabilities of conventional disk-based storage and retrieval technologies.

Just a few examples:

- 15 out of 17 U.S. business sectors have more data stored per company than the U.S. Library of Congress (McKinsey Global Institute, 2011)
- 62 percent of organizations use between 100 terabytes and 9 petabytes of data (ODCA, 2012)
- Volume of data is doubling every 18 months (IDC, 2011)
- Twitter receives over 200 million tweets per day (ODCA, 2012)
- Facebook collects an average of 15 terabytes per day (ODCA, 2012)

Even relatively static content cannot be fetched fast enough from disk to keep up with escalating demands, especially when hot-key trends occur with millions of requests for the same information.

To make matters even more challenging, a growing percentage of dynamic Web requests now require a process or logic to run to create the requested item. The prospect of recalculating and serving the same result potentially millions of times to update Web content presents an intolerable bottleneck.

Users don't care about all of these difficulties going on behind the scenes to scale performance to meet their aggregate demands. If end-users perceive unacceptable delays, then from the users' perspective, "the system is broken", and they give up or move on to another site. Implementation of an efficient and scalable caching architecture between the Web tier and the application and database tiers has emerged as a viable and affordable solution.

Memcached has become the dominant open-source architecture for enabling Web tier services to scale to tens of thousands of systems without either 1) overwhelming the database, or 2) accruing geometric queuing delays that cause unacceptable end-user experiences.

Well-designed Memcached clusters can consistently support a latency SLA in the 100 µsec range while scaling to support very large Web server farms. In addition to caching database queries and other Web service information, enterprise data centers also use Memcached to minimize database load for a variety of client/server applications (e.g., order processing). Memcached also offers the ability to add/remove incremental data caching capacity in response to quickly changing requirements.

3.0 Overview of the Memcached Architecture

3.1 Primary Objective: Reduce Transaction Latency

Memcached intercepts data requests and satisfies a high percentage of them out of its logical cache (i.e. system memory), thereby avoiding trips to read data from back-end disk storage. Memcached also

helps reduce compute-intensive tasks by retrieving pre-computed values from the logical cache, thereby avoiding the need for repetitive computations of the same value. In both cases, Memcached can reduce the overall time for responding to requests with cached data. That improves the sustained level of transaction latency.

3.2 Theory of Operation

Memcached is a high-performance, distributed memory object caching system, originally intended for use in speeding-up dynamic Web applications.

At its heart, Memcached is a simple but very powerful key/value store. Its uncomplicated design promotes ease of deployment while alleviating performance problems facing large data caches.

A typical Memcached implementation consists of the following basic elements:

- Client software, which is given a list of available Memcached servers
- A client-based hashing algorithm, which chooses a server based on the "key" input
- Server software, which stores the values with their keys into an internal hash table
- Server algorithms that determine when to discard old data to free up or reuse memory

A Memcached deployment is implemented partially in a client, and partially in a server. Clients understand how to send items to particular servers, what to do when a server cannot be contacted, and how to fetch keys from the servers. The servers understand how to receive items, how to service requests and how to expire items when more memory is needed.

The Memcached interface provides the basic primitives that hash tables provide–insertion, deletion, and retrieval–as well as more complex operations built atop them. Data are stored as individual items, each including a key, a value, and metadata. Item size can vary from a few bytes up to 1MB, although most Memcached implementations are heavily skewed toward smaller items.

Memcached servers generally are not aware of each other. There is no crosstalk, no synchronization, no broadcasting or other need for inter-server communications. The lack of interconnections between Memcached servers means that adding more servers will add more capacity, offering excellent scalability.

The rest of this Guide focuses on the issues involved with configuring and tuning Memcached clusters to deliver optimal performance, security and scalability on Intel® Architecture-based servers.

4.0 Configuration and Deployment

As data size increases in the application, Memcached scales smoothly to provide consistent performance, even with very large databases and low-latency application demands. Two primary scaling methods are to add more RAM to a server and/or to add more servers to the network.

4.1 Memcached Cluster Topology

In a typical Web-serving datacenter deployment, Memcached servers operate in a cluster located between the front-end Web tier and the back-end database storage tiers (See Figure 1). The size of the cluster (e.g. number of servers and RAM per server) depends upon application requirements and performance criteria as described in subsequent sections. See Section 4.3 on Performance Considerations for more detail regarding the metrics to consider when scaling out the number of servers in a Memcached cluster.

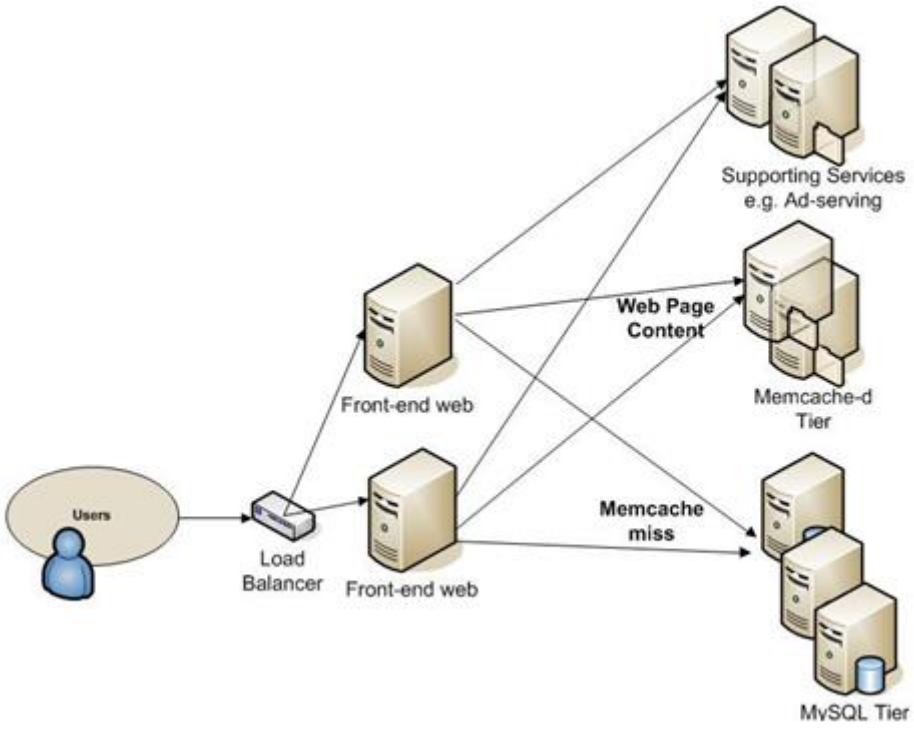


Figure 1 Memcached Cluster Architecture

The Memcached cluster reduces latency by intercepting requests that would otherwise be handled by application servers fetching the data from database storage every time an item is requested.

Figures 2 and 3 illustrate the differences between a non-cached Web infrastructure versus a Memcached Web infrastructure. Figure 2 shows a typical client server topology without a caching tier. Each request must pass through an application server and requires one or more queries to the database. In Figure 3, the same system is fitted with a Memcached cluster sitting between the client(s) and the server(s). It matches-up requests with the memory cache before returning results to the client. In the rare case of a cache-miss, the request goes to the application server, as before.

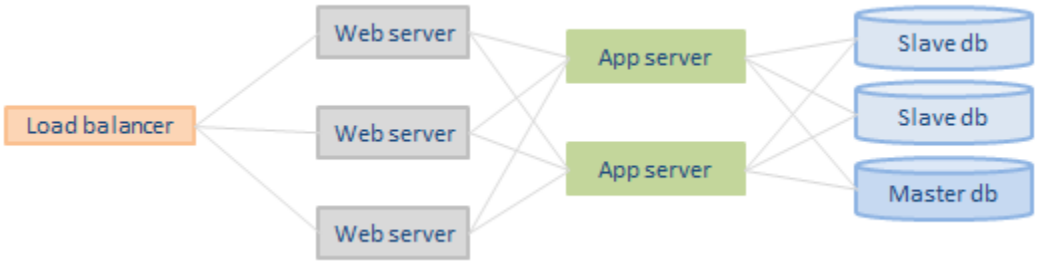


Figure 2 - Typical Client/Server Web Topology

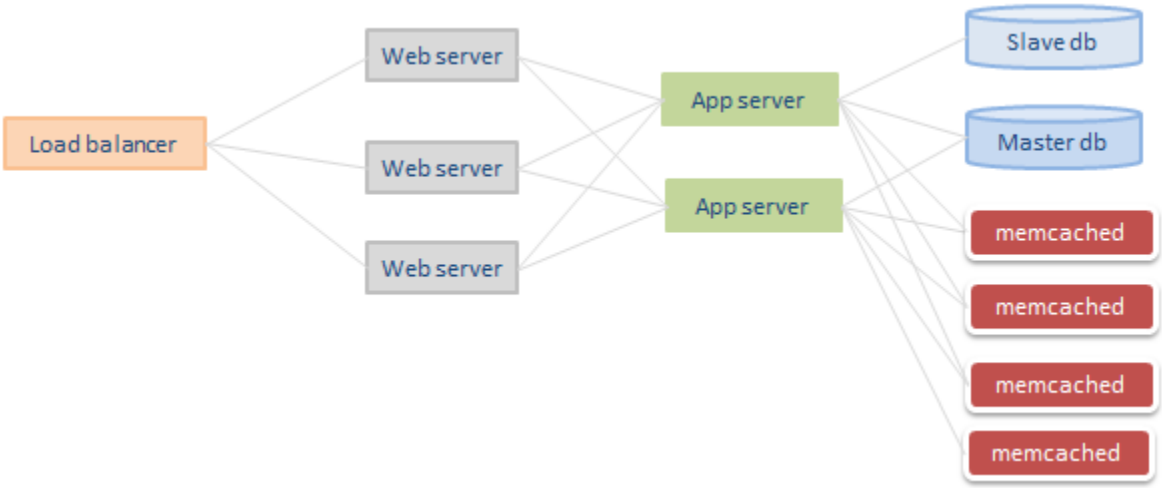


Figure 3 - Memcached-Enabled Web Server Topology

4.2 Software Architecture

Memcached uses a client/server software model in which all servers maintain a key-value associative array that clients populate and query. Keys are up to 250 bytes long and values can be up to 1MB in size, however typical deployments average less than 1KB value sizes.

Memcached clients make use of consistent hashing methodology to select a unique server per *key*, requiring only the knowledge of the total number of servers and their IP addresses. This technique presents the entire aggregate data in the servers as a unified distributed hash table, thereby keeping all servers completely independent and facilitating scalability as the overall data size grows.

Each client knows all of the servers. The servers do not communicate with each other. If a client wants to set or read the value corresponding to a certain key, the client's library computes a hash of the key to determine the server that will be used. Then the client contacts that server, which computes a second hash of the key to determine where to store or read the corresponding value.

In a Memcached cluster, all client libraries use the same hashing algorithm to determine servers, which enables any client to read other clients' cached data. That offers a significant advantage for "peanut buttering" hot-trending values within the common cache array and having them immediately available for serving to any client upon request.

The servers keep all current values in RAM for fast access. If a server runs out of RAM, it discards the oldest values. That means that clients cannot treat any of the cache locations as persistent memory, but it also means that the Memcached can dynamically respond to changing trends in end user request patterns; thereby providing faster responsiveness to the most recent, repetitive and hot-trending information requests.

4.3 Performance Considerations

As one of the top contributers to upstream open source projects, Intel provided numerous optimizations to improve Memcached performance on Intel Architecture-based servers. Through those efforts, we learned that three main considerations impact performance when deploying, tuning and scaling a Memcached server cluster. These are: hit-rate; cache request throughput; and latency.

4.3.1 Hit-Rate

Memcached should serve most data requests generated by the Web tier. To accomplish that, the aggregate size of the key/value store must suffice to cache the majority of requested data objects in order to achieve an acceptable hit-rate.

Because the Web tier workload varies by user and is dynamically changing over time, large transient peaks of usage can occur, especially in fast-trending social media environments. General caching theory still applies: the hit-rate through the tier will improve roughly by half with the quadrupling of the overall cache capacity; however the acceptable miss-rate must be defined by the specific application requirements.

A fixed ratio of Memcached servers to Web servers can achieve a target traffic rate through the cache. The exact ratio varies depending on specific workload objectives, but a useful center-point is one Memcached server to every four Web servers, based on testing and real-world experience with Intel® Xeon® E5 family dual-socket servers.

To achieve price/performance objectives, Memcached servers typically utilize mainstream DRAM devices in standard DIMM form-factors (RDIMM for mainstream servers and UDIMM where supported).

4.3.2 Cache Request Throughout

The Web tier generates a steady but non-uniform rate of requests to the caching tier. A good way to think of the workload requirement is:

- The Web tier contains "X" servers to support a given capacity target
- Each active Web server supports a target of "N" page requests per second
- Each page request generates "M" key/value look-ups (gets) to build content
- Thus the average load on the Memcached tier is X * M * N gets per second

For example, a Web tier of 10,000 server instances, each targeted to 64 requests per second, with a Web workload that generates an average of 100 key/value gets per second, would generate a cumulative load of 64M gets/second. Assuming the 1:4 ratio, and a peanut-buttering of the load across all cache servers, the Memcached tier would need 2,500 servers, each of which must handle 25,000 gets/second.

The relative popularity of objects (i.e., the number of requests per second) in the key/value store varies a great deal, with "hot" and "cold" keys in the array. Therefore two distinct considerations regarding throughput must be taken into account when designing your Memcached tier: (1) the average expected utilization; and (2) the peak throughput on servers for hot-key demand. In the above example, the average utilization is 25,000 gets/second, however the peak hot-key throughput must be defined by the specific application requirements and the targeted user experience (i.e., response-time). For example, in 2011, Facebook's stated target for peak throughput for their Memcached tier was 1,000,000 gets/second.

The inherent variability of Web services and the non-uniform nature of the load make it inevitable that much of the Memcached tier will be underutilized much of the time. So the efficiency challenge in designing the Memcached tier is to meet the target hot-key peak throughput rate while maintaining the lowest capital expenditures and operating costs.

4.3.3 Latency

The third performance metric is the latency to respond to each request, as measured from the client end.

Your business requirements should determine your latency policy, but a useful rule-of-thumb is that users may tolerate a range of 100 microseconds to 1 millisecond of response-time. Latency that exceeds that range may create visible delays for end-users which impinge on their satisfaction and reflect badly

on your business. Again, the specific amount of tolerable latency varies with the business model, so your latency polices may differ. In subsequent sections, we will explore ways to change your latency SLA, as needed.

While occasional excursions in latency can be tolerated, if a high fraction of the requests take too long, the service has failed to meet the SLA. Therefore, datacenter administration typically monitors the frequency of SLA violation events and triggers alerts at a specified level.

The latency of a key/value look-up breaks-down into the following elements:

- **Client**: code path to hash the key and identify the target cache server
- **Client**: network stack to transmit the cache look-up request
- **Network**: multi-hop switching delay between client and server nodes
- **Server**: network stack to receive the cache look-up request
- **Server**: code path to look up the value and determine outcome (hit/miss)
- **Server**: network stack to transmit the cache look-up response
- **Network**: multi-hop switching delay between server and client nodes
- **Client**: network stack to receive the cache look-up response

Note that the network stack must be called twice on each end, and the physical network switching path is traversed twice for each look-up request. Therefore, the two-way communications between client and server nodes makes up a significant amount of the overall latency. Even if the client and server nodes are co-resident under the same top-of-rack switch, the network latency could approach 20 µsec for just the hops back and forth between servers.

Also note that there is no significant advantage in beating the SLA requirement because end-user experience is either satisfactory or it is not. For example, if the SLA calls for latency of no more than 100 µsec with 3-sigma at 1 msec, then a Memcached cluster achieving 50 µsec with 3-sigma at 500 µsec actually wastes money; the datacenter could improve TCO by cutting the provisioning to run closer to the requirement.

With all of these performance considerations in mind, the following sections provide additional detail regarding the configuration of two different kinds of Memcached clusters, one using Intel Atom™-based microservers and one using Xeon®-based dual-socket servers.

4.4 Server Hardware Configurations

Scenario 1: High SLA requirements/low tolerance for latency

This approach uses Intel Xeon® dual-socket E5-based servers with 10Gbps network interfaces, adequate RAM for the size of the data store in question, and a full complement of security and manageability solutions.

Scenario 2: Relatively high tolerance for latency and non-mission-critical peak throughput

This approach uses Intel Atom™-based microservers with 1Gbps network interfaces, and adequate RAM. With Thermal Design Points (TDPs) as low as 6 Watts, Intel Atom Systems on a Chip (SoCs) can deliver

substantial electrical power savings. That is especially important for cloud service provides who typically find as much as 40 percent of their operating costs come from the electricity to power and cool their data center infrastructures.

4.4.1 Memory Sizing

Memcached uses each server's memory for its operations. All keys and values are stored in the servers' DRAM. Therefore, choosing how much memory per server in the Memcached cluster depends on the size of the data store planned by the system architect. For example, to support a distributed cache of 2GB, a cluster of four Memcached servers requires at least 500MB additional RAM per server, above and beyond what's required to run the Memcached servers, themselves.

4.4.2 Networking

As previously described, one of the most critical factors in Memcached performance is the network latency, which makes it imperative to balance network throughput and server capacity. Because the Memcached server constantly communicates with the client and the backend, adequate networking infrastructure among all those components is crucial.

The optimization of networking technology involves not only the bandwidth of the network interface (e.g., 10Gbps vs 1Gbps) but also the queue affinity between the processor and the network interface. Queue affinity can best be achieved by using script-level control of multi-queue handling, spreading requests between cores, controlling interrupts, thread migration control, etc. (See Section 4.5.2 Memcached Performance Tuning for more detail).

Bandwidth efficiency can generally be addressed by making network interface choices that properly match to the Memcached server platform (see the following two sections on Intel Xeon- and Intel Atom-based servers).

4.4.3 Intel Xeon-based Platforms

Per Scenario 1, to achieve best performance, a dual-socket Intel Xeon E5-based platform typically requires a 10Gbps network interface. That ensures the server's aggregate processing performance is matched by the ability to handle and dispatch high numbers of requests. In addition, selecting a network interface capable of multi-queue handling will ensure that Memcached requests are simultaneously distributed across multiple network interface queues. Even though the use of a 1Gbps network interface with an Intel Xeon platform is feasible, the performance observed is less than ideal because the constrained bandwidth creates a latency bottleneck.

4.4.4 Intel Atom-based Platforms

Per Scenario 2, when using Atom-based microservers, the use of a 1Gbps network interface typically achieves optimum performance. Our testing and real-world deployments show that using a faster network interface does not significantly contribute to better performance for Memcached on Atom-based platforms. Keeping in mind the need to match costs to SLA requirements, using a 1Gbps network interface with Atom-based platforms is a cost-effective choice. However, even with that bandwidth, we recommend using a network interface capable of multi-queue handling.

4.5 Memcached Tuning and Optimization

4.5.1 Hardware and Software Optimization

For Intel Xeon platforms, Hyperthreading and Turbo Boost features should be enabled. Depending on the type of Intel Atom platform used, similar features, when available, also should be enabled. This helps the Memcached server use CPU core affinity to limit thread migration, a detriment to performance. Similarly, BIOS Speed Step states, C States and EIST should be enabled in those systems that support them. While Hyperthreading and Turbo Boost help with query performance, the implementation of Speed Step and C States, along with EIST, helps optimize power efficiency and improves overall TCO.

4.5.2 Memcached Performance Tuning

As previously described, Memcached clusters often exhibit a high load profile on a minority of CPUs, while other CPUs sit idle. Therefore, the network interface needs to distribute requests across all available CPUs running Memcached. Setting Interrupt Request (IRQ) affinity on the network interface for each Memcached server ensures that all CPUs are set to handle the requests.

Along with IRQ affinity, network interface interrupt coalescing enables best performance on both processor classes. By observing soft IRQ handling, you can determine if the system spends more time servicing interrupts than processing Memcached requests. One way to balance that is to throttle the network interface. By tuning its "rx-usecs" parameter, you can increase Memcached server performance.

For Intel Xeon platforms, you can start with the number of microseconds per interrupt (usually around 350), and then vary it to observe changes in performance. The Linux command to use is: ethtool <nic_id> -C -rx-usecs 350.

In addition to network interface IRQ affinity and interrupt coalescing, thread affinity for the Memcached server needs to be enabled. Thread affinity ensures that each Memcached thread is pinned to a given CPU, and only to that CPU.

For certain versions of Memcached, such as 1.6BagLRU and 1.4.15-ThreadAffinity, you can set the CPU affinity from the Memcached settings file. This typically consists of an incremental value set to an integer, allowing the server to pin its threads to given CPUs according to the specified value. You can use the "taskset" shell command in Linux to set CPU affinity for Memcached, as well. Note that the value in question depends on the CPU layout of the platform, and it can vary from system to system, even within the same processor class.

5.0 Performance Measurements and Platform Comparisons

The performance information in this section compares the results of extensive Memcached testing, which was conducted across a range of Intel architectures, including the latest Avoton release of the Atom-based microserver family as well as current Xeon E3 (Haswell) and Xeon E5 (Ivybridge) processors.

The testing process encompassed both raw performance data and power-related (performance/watt) calculations. With power and cooling costs now representing the largest expenditures for operating most data centers, data center managers must carefully balance the power-performance tradeoffs when choosing the appropriate architecture.

In addition, to optimize Memcached for some high-performance, high-availability application environments, system designers often need to manage the "blast radius" by using more nodes to minimize the amount of data that is at risk on each node in the cluster. This provides higher availability rate of Hot Keys to support requirements for fast trending and high peak demand scenarios. For these situations, increasing the number of nodes while still managing optimal performance, power and cost is an important factor for overall success.

The Memcached test methodology is summarized below:

- The test environment consisted of Memcached Version 1.4.15 with optimization for thread affinity added to the open source code and IRQ affinity enabled for the network interface. For Xeon E5 data, the 1.6BagLRU optimization was also enabled.
- Two workload elements were measured
    - Random Key GET transactions
    - Hot Key GET transactions
- Performance metric = Max GET transaction Requests Per Sec (kRPS) under SLA
- SLA requirement = 99.5% of transactions must have average latency of $\leq$ 1 ms
- Overall performance is based on the following key measurements:
    - Max Random Key GET transactions per second under SLA (RandomKeyGetRpsPerf)
    - Max Hot Key GET transactions per second under SLA (HotKeyGetRpsPerf)
- Power is measured at the wall for each of the two key performance measurements
- Power/watt (PPW) is expressed as:
    - Max Random Key GET transactions per second per watt ((RandomKeyGetRpsPPW)
    - Max Hot Key GET transactions per second per watt (HotKeyGetRpsPPW)
- Overall performance and performance/watt metrics are:
    - Overall Perf = (0.8 * RandomKeyGetRpsPerf + 0.2 * HotKeyGetRpsPerf)
    - Overall PPW = (0.8 * RandomKeyGetRpsPPW + 0.2 * HotKeyGetRpsPPW)

Three versions of Memcached are relevant to the performance testing.

- Memcached 1.4.15 is the official Memcached version.

- Memcached 1.4.15 Thread affinity, used in the testing is an Intel optimized version of Memcached 1.4.15. The optimizations added enable Memcached 1.4.15 to make better use of core efficiency by making sure that Memcached thread are pinned each to a single core. Not enabling threads to move around cores, allows for better performance, as it limits cache to cache transfers.

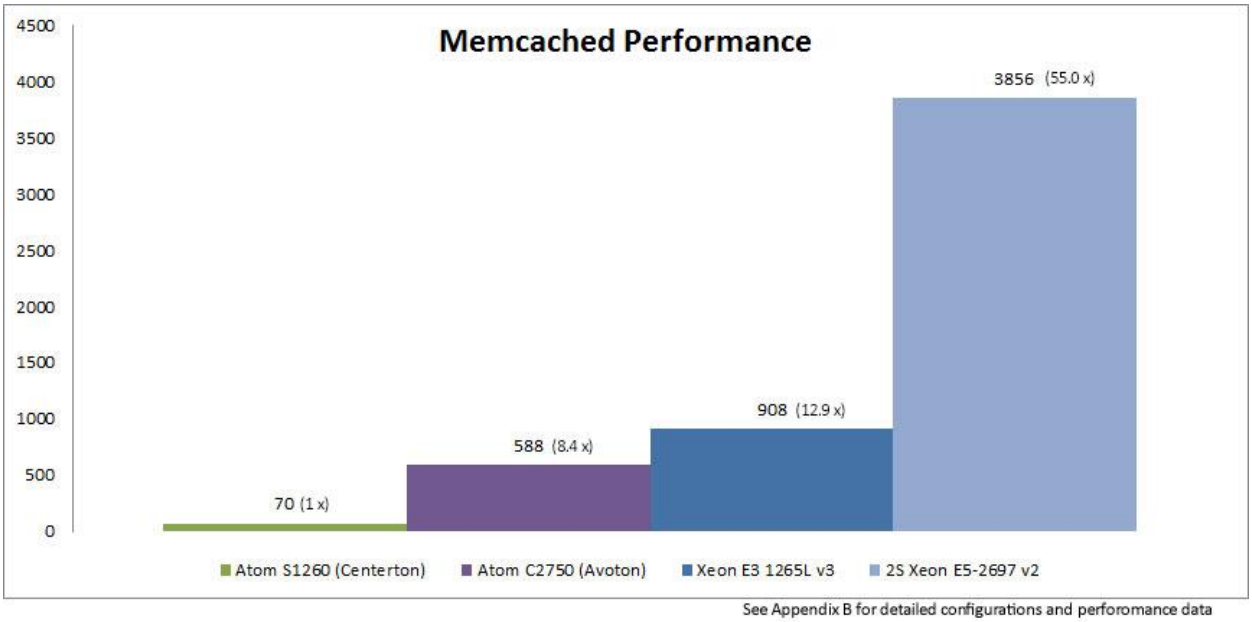
- Memcached 1.6BagLRU version is different from Memcached 1.4.15, as it not only features thread affinity much like Memcached 1.4.15 Thread Affinity, but the 1.6BagLRU version also allows for a finer grained lock mechanism on the Memcached Hash Table Array. This makes for less lock contention, and promotes not only performance, but also thread scalability.

Intel recommends Memcached 1.4.15 with Thread Affinity as this version provides the optimal combination of performance and compatibility.

Memcached 1.4.15 Thread Affinity can be downloaded from: https://github.com/zfadika/memcached

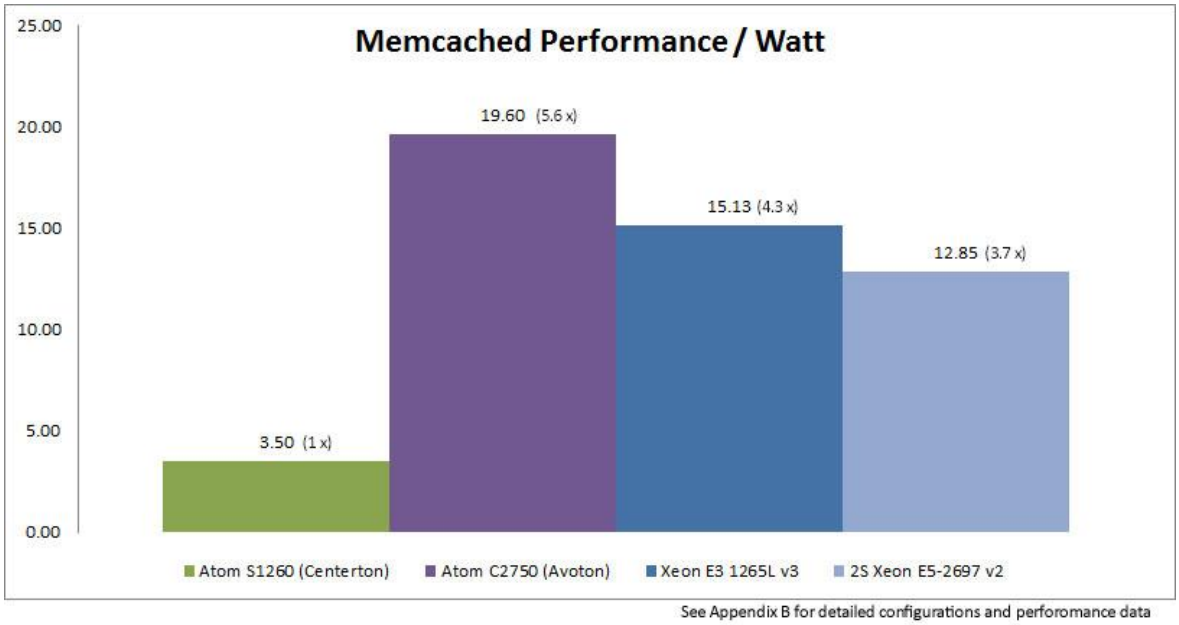
The following graphics summarize the results of these specific tests and provide comparisons between the different processor architectures. (See Appendix B for configuration and performance details)

Overall raw performance measurements show the 2-socket Xeon E5 processor-based platform can provide approximately 55x the kRps rate of the baseline Atom S1260 (Centerton) based system. The Xeon E3 processor-based platform is 12.9x higher and the two-node Atom C2750 (Avoton) delivers 8.4x higher performance than the Atom S1260 (Centerton) based microserver.



See Appendix B for detailed configurations and perforomance data

The performance/watt measurements show a tighter grouping with the two-node Atom C2750 (Avoton) delivering 5.6x higher perf/watt than the Atom S1260 (Centerton) followed by the Xeon E3 providing 4.3x and the 2-socket Xeon E5 providing 3.7x perf/watt over the Atom S1260 baseline.



See Appendix B for detailed configurations and perforomance data

Depending on the specific Memcached application, workload demands, SLA requirements, data center configurations, blast-radius issues and growth projections, system designers can either "dial-up" or "dial-down" their processor choices to achieve optimal performance, power and cost objectives.

6.0 Summary

Memcached provides a performance boost for Web services because it caches data and objects in memory to reduce the number of times that an external data source must be read. The Memcached tier can cache "hot" data and reduce costly trips back to the database to read data from disk, thereby providing a speed advantage and enabling Web services to respond quickly to fast trending demands.

Well-designed Memcached clusters can consistently support a transaction latency SLA in the 100 µsec range while scaling to support very larger Web server farms. As data size increases, Memcached scales smoothly by adding more RAM to a server and/or adding more servers to the network. Because all client libraries use the same hashing algorithm to determine servers, any client can read other clients' cached data, which offers the advantage of peanut-buttering hot-trending values within the common cache array and making them immediately available to any client upon request.

Intel has been a leading contributor to upstream open source projects and industry efforts to advance Memcached performance, conducting extensive tests and proposing numerous optimizations for Memcached on Intel Architecture. Through these efforts, we have identified three main considerations impacting Memcached performance: 1) Hit-Rate, 2) cache request throughput, and 3) latency.

Three primary deployment scenarios require specific hardware approaches:

Scenario 1: High SLA requirements and low tolerance for latency
Using Intel Xeon® dual-socket E5-based servers with 10Gbps network interfaces, adequate RAM for the size of the data store in question, and a full complement of security and manageability solutions.

Scenario 2: Relatively high tolerance for latency and non-mission-critical peak throughput
Using Intel Atom™-based microservers with 1Gbps network interfaces, and adequate RAM. Intel Atom Systems on a Chip (SoCs) can deliver substantial electrical power and cost savings.

Scenario 3: Application requirements to minimize "blast radius" failure risks
Using Intel Atom™-based microservers, such as Avoton, to increase number of nodes and reduce blast radius, while optimizing performance/cost/power tradeoffs.

In all of these approaches, proper memory sizing is important because each server's memory is used for the key/value caching. The amount of memory per server depends on the total Memcached memory designed by the system architect, divided by the number of servers plus the amount of memory needed to run each server.

Networking is another key performance factor in both scenarios due to the critical impact of network throughput on latency. Properly matching the network interface to the server is the first issue, with a 1Gbps interface generally adequate for the Intel Atom-based systems in Scenario 2 and a 10Gbps interface better for the Intel Xeon-based systems in Scenario 1. Hyperthreading, Turbo Boost and distributing requests through IRQ affinity tuning also are helpful to optimize performance.

Using Intel Architecture across the Web tier, cache tier and back-end tier reduces overall complexity and improves TCO by providing a common hardware and software architecture that can be optimized for performance, cost and maintainability for all functional levels throughout the data center.

7.0 Appendix A – Additional Resources

"Ways to Speed-Up Your Cloud Environment" blog post

"Enhancing the Scalability of Memcached" technical paper

P. Saab, "Scaling Memcached at Facebook," 12 December 2008. [Online]. Available: https://www.facebook.com/note.php?note_id=39391378919&ref=mf

Twitter Engineering, "Memcached SPOF Mystery," Twitter Engineering, 20 April 2010. [Online]. Available: http://engineering.twitter.com/2010/04/Memcached-spof-mystery.html

Reddit Admins, "reddit's May 2010 "State of the Servers" report," Reddit.com, 11 May 2011. [Online]. Available: http://blog.reddit.com/2010/05/reddits-may-2010-state-of-servers.html

C. Do, "YouTube Scalability," Youtube / Google, Inc., 23 June 27. [Online]. Available: http://www.youtube.com/watch?v=ZW5_eEKEC28

Memcached.org, "Memcached- a distributed memory object caching system," Memcached.org, 2009. [Online]. Available: http://Memcached.org/about

8.0 Appendix B: Performance Test Configuration Details

Testing was conducted using the following Intel-based configurations which yielded the performance results as indicated:

Atom S1260: Intel® Bordenville Server platform codenamed Double Cove Fab3 with one Intel® Centerton B0 Stepping (2-Core, 2.0 GHz), Hyper-Threading Enabled, 8GB memory 8GB DDR3 1333 UDIMM ECC), 2xHDD, 1x1GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys. Random Key Gets: 77kRPS. Hot Key Gets: 44kRPS. Performance = 80% Random + 20% Hot = 70k RPS. Estimated node system power: 20W

Atom C2750 : Intel® Edisonville Server platform codenamed Mohon Peak with one Intel® Avoton B0 Stepping (8-Core/8-Thread, 2.40 GHz), 8GB memory DDR3-1600 UDIMM ECC), Turbo Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 620kRPS. Hot Key Gets: 460kRPS. Performance = 80% Random + 20% Hot = 588k RPS. Estimated node system power: 30W

Xeon E3-1265L v3: Intel® Xeon E3-1265L v3(4C, 2.50 GHz), 8GB memory DDR3-1600 UDIMM ECC, Turbo Disabled, Hyper-Threading Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 892kRPS. Hot Key Gets: 976kRPS. Performance = 80% Random + 20% Hot = 908k RPS. Estimated node system power: 60W

2S Xeon E5-2697 v2: Intel® Xeon E5-2697 v2(12C, 2.7 GHz), 32GB memory DDR3-1600, Turbo Enabled, Hyper-Threading Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 4160kRPS. Hot Key Gets: 2640kRPS. Performance = 80% Random + 20% Hot = 3856k RPS. Estimated node system power: 300W

9.0 Notices:

Intel, the Intel logo, Intel Atom, Intel Core, and Intel Xeon are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance. Results have been estimated based on internal Intel analysis and are provided for informational purposes only.

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase

Intel Hyperthreading Technology available on select Intel® Core™ processors. Requires an Intel® HT Technology-enabled system. Consult your PC manufacturer. Performance will vary depending on the specific hardware and software used. For more information including details on which processors support HT Technology, visit http://www.intel.com/info/hyperthreading.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other

optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations:

Atom S1260: Intel® Bordenville Server platform codenamed Double Cove Fab3 with one Intel® Centerton B0 Stepping (2-Core, 2.0 GHz), Hyper-Threading Enabled, 8GB memory 8GB DDR3 1333 UDIMM ECC), 2xHDD, 1x1GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys. Random Key Gets: 77kRPS. Hot Key Gets: 44kRPS. Performance = 80% Random + 20% Hot = 70k RPS. Estimated node system power: 20W

Atom C2750 : Intel® Edisonville Server platform codenamed Mohon Peak with one Intel® Avoton B0 Stepping (8-Core/8-Thread, 2.40 GHz), 8GB memory DDR3-1600 UDIMM ECC), Turbo Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 620kRPS. Hot Key Gets: 460kRPS. Performance = 80% Random + 20% Hot = 588k RPS. Estimated node system power: 30W

Xeon E3-1265L v3: Intel® Xeon E3-1265L v3(4C, 2.50 GHz), 8GB memory DDR3-1600 UDIMM ECC, Turbo Disabled, Hyper-Threading Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 892kRPS. Hot Key Gets: 976kRPS. Performance = 80% Random + 20% Hot = 908k RPS. Estimated node system power: 60W

2S Xeon E5-2697 v2: Intel® Xeon E5-2697 v2(12C, 2.7 GHz), 32GB memory DDR3-1600, Turbo Enabled, Hyper-Threading Enabled, 1xHDD, 1x10GbE, Mcblaster - From gitHUB modified to distribute data & transactions over multiple memcached servers & support hot keys in addition to random keys, Random Key Gets: 4160kRPS. Hot Key Gets: 2640kRPS. Performance = 80% Random + 20% Hot = 3856k RPS. Estimated node system power: 300W

For more information go to http://www.intel.com/performance