IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

Performance Analysis in Memory Distribution Key Value Store and Design of Storage Driver for SBK Framework

Prof. Satheesh Chandra Reddy S[1], Navyashree L[2], Neharika L[3], Shubha Gurudev[4]
Professor and HOD, Department of Information Science and Engineering[1]
Students, Department of Information Science Engineering[2,3,4]
S. J. C. Institute of Technology, Chikballapur, Karnataka, India

Abstract: *This paper aims to write the Memcached storage driver and performance analysing the Memcached. Memcached is a free and open source software, high performance, distributed memory object caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times, an external data source must be read. The SBK itself is a high-performance benchmark tool/framework. It massively writes the data to the storage system and reads the data from the storage system. The SBK supports multi writers and readers and also the End to End latency benchmarking. The main objective is to performance benchmark the Memcached storage driver using Storage Benchmark Kit(SBK), to speed up the performance of websites and web applications using SBK software, to extract the results of the operations which were performed on the data. The performance benchmarking is done by adding SBK driver which specifies the input/output operations of storage systems. SBK delivers the throughput and latency values for a every specific interval for live performance analytics. Memcached is a free and open-source software, high performance, distributed memory object caching system. The SBK itself is a high-performance benchmark tool/framework. The SBK supports multi writers and readers and also the End-to-End latency benchmarking. Memcached is a key-value store that keeps data in the DRAM i.e., data is not persistent. Clients communicate with Memcached through a simple set of APIs: Set, Add, replace are used to store data, Get or Remove are used to retrieve or remove data.*

Keywords: Storage Area Network, Performance Analysis, Storage Benchmarking Kit

I. INTRODUCTION

The interface to Memcached supports the following methods for storing and retrieving information in the cache, and these are consistent across all the different APIs, although the language specific mechanics might be different. When practical, use Memcached to store full items, rather than caching a single column value from the database. For example, when displaying a record about an object (invoice, user history, or blog post), load all the data for the associated entry from the database, and compile it into the internal structure that would normally be required by the application. Save the complete object in the cache. To share data stored in Memcached instances between different language interfaces, consider using a common serialization solution such as JSON (JavaScript Object Notation). Memcached allows you to take memory from parts of your system where you have more than you need and make it accessible to areas where you have less than you need. Memcached also allows you to make better use of your memory. Memcached is a popular component of modern Web architectures, which allows fast response times for serving popular objects. In this work, we study how memory partitioning in Memcached works and how it affects system performance in terms of hit rate. Memcached divides the memory into different classes proportionally to the percentage of requests for objects of different sizes. Once all the available memory has been allocated, reallocation is not possible or limited, a problem called “calcification”. Modern Web architectures are designed to provide low latency response times to thousands of requests per second originated by a large number of clients trying to access. Memcached is a popular and well-integrated component of such Web architectures: it is a key-value store that exposes a simple API to store and serve data residing in the DRAM of a machine. Due to its simplicity and efficiency, Memcached has been adopted by many companies, such as Wikipedia, Flickr, Digg, WordPress.com, Craigslist, and, with additional customizations,

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

Facebook and Twitter. The basic unit of memory is called a slab and has fixed size, set by default to 1 MB. A slab is logically sliced into chunks that contain data items.

By design, Memcached partitions the memory dedicated to store data objects into different classes; each class is dedicated to objects with a progressively increasing size, which takes into account the typical size distribution of web data objects. When a new object that is, an object that was never requested before – has to be stored, Memcached checks if there is available space in the appropriate class, and stores it. If there is no space, Memcached evicts a previously stored object in favour of the new one.

Clearly, if the statistical properties of the objects (i.e, the distribution of the object sizes) that are stored in Memcached does not change over time, then calcification does not affect performance, because the baseline eviction policy currently implemented in Memcached works well for storing

"hot" objects. However, if the statistical properties of object sizes change over time, calcification has a direct impact on the hit rate achieved by Memcached – this problem has been reported by Twitter and Facebook.

For increasing scalability Memcached is used by many websites with heavy traffic like Facebook, YouTube, twitter, raddit etc. In next section Memcached architecture is described with its basic usage. The data structures used in Memcached are described in detail. How these data structures are proper for Memcached is also described. In Basic Operations section get, set and delete are described and some other operations are also listed in the same. In the next section process flow of Memcached is explained with example. While Memcached already offers excellent performance, Memcached architecture can be studied on some of its limitations and can give better results than of now.

The Linux kernel has some interesting system calls called splice and vmsplice which implement zero- copy. We modify Memcached to use these system calls and document the effect of this modification on performance.

Multiple clients accessing the same objects must have the same list of servers because the location of an object is determined by hashing the list of servers together with the object's key.

The key features of Memcached are as follows:

- It is open source.
- Memcached server is a big hash table.
- It significantly reduces the database load
- It is perfectly efficient for websites with high database load.
- It is distributed under Berkeley Software Distribution (BSD) license.
- It is a client-server application over TCP or UDP.

Unlike databases that store data on disk or SSDs, Memcached keeps its data in memory. By eliminating the need to access disks, in-memory key-value stores such as Memcached avoid seek time delays and can access data in microseconds. Memcached is also distributed, meaning that it is easy to scale out by adding new nodes. And since Memcached is multithreaded, you can easily scale up compute capacity. As a result of its speed and scalability as well as its simple design, efficient memory management, and API support for most popular languages Memcached is a popular choice for high-performance, large-scale caching use cases. Amazon offers a fully managed Memcached service, Amazon Elastic Cache for Memcached:

- Easily set up, operate, and scale Memcached deployments in the cloud. Failed nodes are automatically detected and replaced.
- Take advantage of cost-efficient and resizable hardware capacity.
- Use an Elastic Cache Memcached Cluster Client with Auto Discovery to speed up application development by simplifying node management.

By eliminating the need to access disks, in- memory key-value stores such as Memcached avoid seek time delays and can access data in microseconds. Memcached is also distributed, meaning that it is easy to scale out by adding new nodes. And since Memcached is multithreaded, you can easily scale up compute capacity. As a result of its speed and scalability as well as its simple design, efficient memory management, and API support for most popular languages Memcached is a popular choice for high- performance, large-scale caching use cases.

Even within a trusted organization, the flat trust model of Memcached may have security implications. For efficient simplicity, all Memcached operations are treated equally. Clients with a valid need for access to low-security entries

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

within the cache gain access to all entries within the cache, even when these are higher-security and that client has no justifiable need for them. If the cache key can be either predicted, its cache entry may be retrieved.

II. RELATED WORK

Before building our application, the following system is taken into consideration:

Title: Design and Implementation of Storage Benchmark Kit

Abstract: This design solves the synchronization issues between multiple writers, readers/callback (push) readers. It describes the best suited data structures such as multiple concurrent queues to measure the maximum throughput and low latency for any storage device/client. The SBK benchmark parses and processes the application/user supplied or command line arguments, configures the multiple writers, readers, and the component "SBK performance processor."

Title: Storage performance - metrics and benchmarks

Abstract: Describes a new approach to I/O performance evaluation-the Self- scaling Benchmark with Predicted Performance. The Self-Scaling Benchmark scales automatically to current and future systems by scaling the workload to the system under test

Title: Memory partitioning in Memcached: An Experimental Performance Analysis.

Abstract: Memcached is a popular and well- integrated component of such Web. It is a key-value store that exposes a simple API to store and serve data residing in the DRAM of a machine. It is known for its simplicity and efficiency. Memcached has been adopted by many companies, such as Wikipedia, Flickr, WordPress and, with additional customizations, Facebook and Twitter

Title: A Memory Efficient and High- Performance Key-Value Store

Abstract: This paper presents the design and implementation of data retrieval of a system aiming at reducing the memory overhead and index building performance of Memcached. We present a new high- performance key-value store retrieval system called Memcached. Designing a multi-level system, including a read- write optimized hash store algorithm, in order to optimize the space cost.

Title: Characterizing Facebook's Memcached Workload

Abstract: This paper analyses the Memcached workload at Facebook, looking at server-side performance, request composition, caching efficiency, and key locality. Caching as a general research topic has been extensively studied. The observations presented here lead to several design insights and new research directions for key- value caches, such as the relative inadequacy of the least recently used (LRU) replacement policy.

III. METHODOLOGY USED

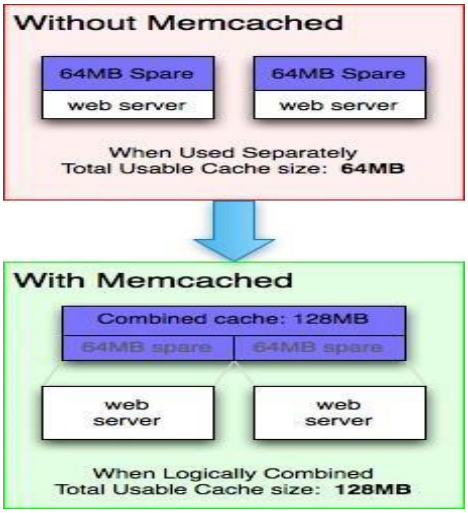

Figure 1: MEMCACHED

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

Memcached allows you to take memory from parts of your system where you have more than you need and make it accessible to areas where you have less than you need. Memcached also allows you to make better use of your memory. If you consider the diagram to the right, you can see two deployment scenarios:

1. Each node is completely independent(top).
2. Each node can make use of memory from other nodes(bottom).

The first scenario illustrates the classic deployment strategy, however you will find that it's both wasteful in the sense that the total cache size is a fraction of the actual capacity of your web farm, but also in the amount of effort required to keep the cache consistent across all of those nodes With Memcached, you can see that all of the servers are looking into virtual pool of memory. This means that a given item is always stored and always retrieved from the same location in your entire web cluster. Also, as the demand for your application grows to the point where you need to have more servers, it generally also grows in terms of the data that must be regularly accessed.

The illustration to the right only shows two web servers for simplicity, but the property remains the same as the number increases. If you had fifty web servers, you'd still have a usable cache size of 64MB.

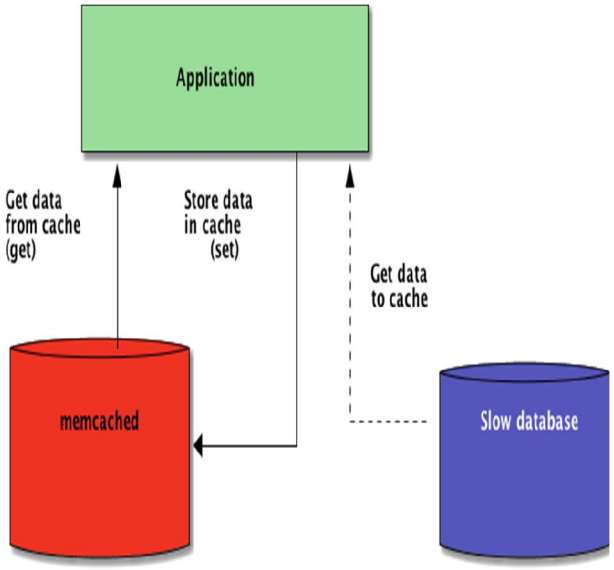

Figure 2: INTERACTION

Figure 2 represents memcached architecture Memcached is designed to scale from a single server to many servers using horizontal scaling approach also known as scale. It uses 'JAVA' programming language. It has mainly two components. Memcached client and Memcached server.

Memcached Server Instance:

In distributed caching, Memcached server is a standalone process that basically handles three tasks.

It manages memory allocation and restoration. It keeps track of objects stored in memory.

It serves client requests regarding object retrieval and storage.

Memcached Client Instance

Memcached client are intended for providing a common way of accessing the Memcached server. Their main objective is to serialize objects, which will be stored in the cache. As the caching system is meant to be distributed, a client can connect to multiple servers by specifying their IP addresses and ports. Multiple clients accessing the same objects must have the same list of servers because the location of an object is determined by hashing the list of servers together with the object's key.

Simple and lightweight client libraries have been implemented in different programming languages like C, Java, Python, Perl etc.

There are number of client and server instances are possible at one time for large data set.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

IV. IMPLEMENTATION

4.1 Basic Operations

There many operations in Memcached to deal with data. Memcached provides a simple set of operations (set get, and delete) that makes it attractive as an elemental component in a large-scale distributed system.

- **Set:** Value which is to be stored on cache is passed with a key with help of this function. Memcached client node wants to set the key "test key" with the value "test value". Memcached client node takes the list of available Memcached server nodes and performs hash operation on them in order to find the Memcached server node to store particular value for particular data. Proper Memcached server node is selected. Memcached client node directly connects to Memcached server node and sets key "test key" with the value "test value".
- **Get:** Memcached client node wants to get key "test key". Memcached client node is able to use the same hashing process to determine that key "test key" is on which server. Memcached client node directly requests key "test key" from Memcached server node and gets back "test value".
- **Delete:** Memcached client node wants to delete key "test key". Memcached client node is able to use the same hashing process to determine that key "test key" is on which server. Memcached client node directly requests key "test key" from Memcached server node and deletes value "test value". Replace, increment, decrement are some other important operations of Memcached used for different purpose.

V. RESULTS

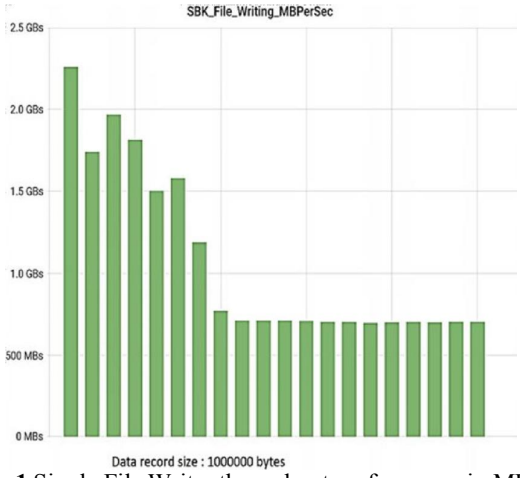

Fig. 1 Single File Writer throughput performance in MB/s

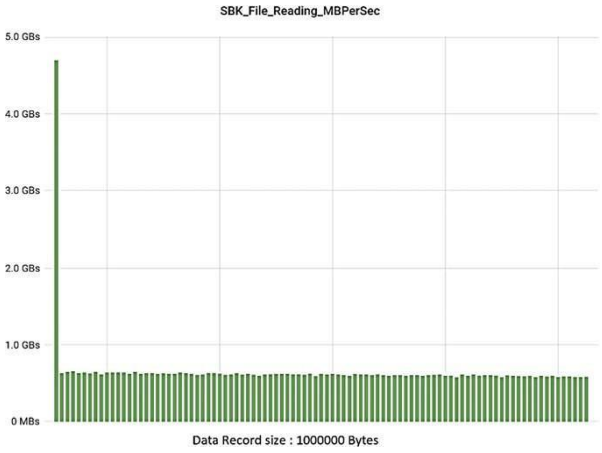

Fig. 3 Single File Reader throughput performance in MB/s

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

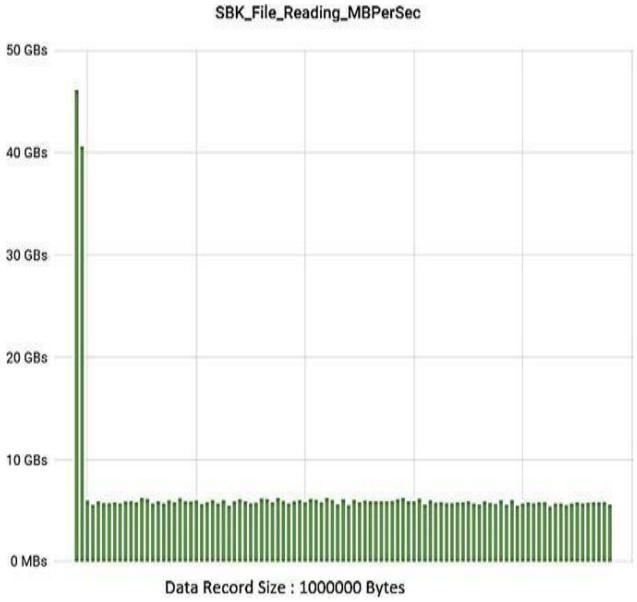

Fig. 4 10 File Readers throughput performance in MB/s

VI. CONCLUSION

The design and implementation of the sbk framework, presented in this paper, scales high with multiple readers and writers. This design solves the synchronization issues between multiple writers, readers/callback (push) readers, and it also describes the best suited data structures such as multiple concurrent queues to measure the maximum throughput and low latency for any storage device/client/cluster. The design of SBK. Exports the standard storage interface apis which can be extended to include a storage driver to conduct the performance benchmarking of any custom storage device/client. The sbk currently supports benchmarking of a wide category of storage systems such as local mounted file systems, distributed file systems, distributed messaging, streaming storage platforms, key-value storage systems, database systems, and object storage systems. Sbk can be used as a common framework to conduct performance benchmarking among similar category storage system. The main agenda is to add a driver to the successfully built sbk storage. To build sbk we have to follow few steps that is, to build storage benchmarking kit (sbk) we use the ubuntu operating system. We are supposed to install the jdk 17 package that there many operations in Memcached to deal with data. Memcached provides a simple set of operations (set get, and delete) that makes it attractive as an elemental component in a large-scale distributed system.

Allows us to write the framework in java. Gradle is an automation tool that is used for multi language software development. Sbk is an open source software framework for the performance benchmarking of any storage system. In the end we have to add a driver memcached to sbk framework , and analyse the performance of driver.

ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions of Professor and HOD, Prof. Satheesh Chandra Reddy S and authors who made their contributions to the earlier part of the work which had been previously published.

REFERENCES

[1]. Design and Implementation of Storage Benchmark Kit: K. Munegowda and N. V. Sanjay Kumar https://github.com/kmgowda/SBK/blob/master/docs/sb k.pdf

[2]. Storage Performance-metrics and Benchmarks: Peter M. Chen And David A. Pamrson, FELLOW, IEEE Proceedings of the IEEE, Vol. 81. No. 8, August 1993, https://Ieeexplore.Ieee.Org/Document/236192

[3]. Memcached: Memcached CAS Operations. https://cloud.google.com/appengine/docs/python/memcache/#Python_Using_compare_and_set_in_Pyth on (2014)

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, June 2022
Impact Factor: 6.252

[4]. Memcached: Memcached Web Page. http://memcached.org

[5]. Storage Benchmark Kit (SBK). https://github.com/kmgowda/SBK(2020)

[6]. A. Wiggins and J. Langston, "Enhancing the Scalability of Memcached," in Intel document, unpublished, http://software.intel.com/en- us/articles/enhancing-the-scalability-of-memcached, 2012.