Towards performance prediction in massive scale datastores

Francisco Cruz[1], Fábio Coelho[1] and Rui Oliveira[1]
[1]*INESC TEC & Universidade do Minho, Braga, Portugal*
fmcruz@di.uminho.pt, fabio.a.coelho@inesctec.pt, rco@di.uminho.pt

Keywords: Performance, Cloud Computing, NoSQL databases

Abstract: Buffer caching mechanisms are paramount to improve the performance of today's massive scale NoSQL databases. In this work, we show that in fact there is a direct and univocal relationship between the resource usage and the cache hit ratio in NoSQL databases. In addition, this relationship can be leveraged to build a mechanism that is able to estimate resource usage of the nodes composing the NoSQL cluster.

1 INTRODUCTION

Massive scale distributed key-value datastores (popularly referred to as NoSQL databases) are becoming pivotal systems in nowadays infrastructures. They support some of the most popular Internet services and have to cope with huge amounts of data while offering stellar performance. In fact, their highly desirable performance, scalability and availability properties cannot be achieved without carefully choosing its underlying infrastructure and without adequate data allocation. All requiring real scenario testing and performance prediction.

Similar to traditional relational databases, NoSQL databases make heavy use of buffer caching mechanisms, in order to improve the performance of read requests. Moreover, the effectiveness of such mechanisms is directly related to the performance and, as a consequence, to the resource utilization of the database. This effectiveness can be measured in terms of the hit ratio that the caching mechanism exhibits. The higher the cache hit ratio the more effective the cache mechanism is, and thus more performant is the database.

In this position paper, we set up some experiments to demonstrate that there is a direct relationship and univocal relationship between the cache hit ratio and the resource utilization. Stemming from this relationship, we envision that it is possible to take advantage of this relationship in order to estimate the resource usage of NoSQL databases simply by knowing its cache hit ratio, as it is a reflection of the data size and the request distribution, and the incoming throughput.

2 BACKGROUND

Caching mechanisms are crucial to improve the performance of computing systems. In particular, databases make use of buffer caching to improve their read performance. When using caching one of the main goals is to try to keep the cache hit rate as high as possible. The cache hit rate measures the percentage of requests that result in a cache hit. A high hit cache rate means that a good number of requests are being served exclusively by the cache, and thus optimizing resource consumption. In other words, this means using less CPU and less I/O operations. As a result, the cache hit ratio is directly related to resource consumption. When the data size exceeds the cache size, eventually, some data in the cache needs to be removed to give room to more frequently accessed data. One of the most widely used cache replacement algorithms (Puzak, 1985). is the Least Recently Used (LRU) algorithm (Sleator and Tarjan, 1985), and it is used by NoSQL databases in their implemented buffer caches (George, 2011) (L. and M., 2009).

NoSQL databases focus on high throughput, high scalability and high availability, in addtiion they run in a distributed setting with tenths to hundreds of nodes, usually composed of commodity hardware. The application data is partitioned and these partitions are assigned to the available nodes according to a data placement strategy. In HBase, nodes are called *RegionSevers* and data partitions are referred as *regions*. Contrasting with relational database management systems (RDBMS), these databases only provide a simple key-value interface to manipulate data by means of put, get, delete, and scan operations and they do not offer strong consistency criteria. Based on

Bigtable (Chang et al., 2006), HBase's data model implements a variant of the entity-attribute-value (EAV) model and can be thought of as a multi-dimensional sorted map. This map is called *HTable* and is indexed by the row key, the column name and a timestamp. HBase has a *block cache* implementing the LRU replacement algorithm. Several key-values are grouped into block of configurable size and these blocks are the ones used in the cache mechanism. The block size within the *block cache* is a parameter but defaults to 64KB.

3 INTERDEPENDENCE OF RESOURCE USAGE AND CACHE HIT RATIO

The cache hit ratio has a great impact on how a system performs and is thus directly related to its resource consumption. By resource consumption we mean the amount of main memory used, the number of I/O operations to distinct storage mediums and the amount of memory/disk swapping needed. The server usage encompasses the CPU time waiting for I/O operations to complete (I/O wait), the time spent on user space (CPU_{user}) and the time spent on kernel space (CPU_{system}).

$$Server_{usage} = I/Owait + CPU_{user} + CPU_{system}$$

With this measure it is possible to have an accurate picture of how the machine is using its resources. Although the I/O wait corresponds to a period when the CPU is free for other computational tasks, we are addressing a specific scenario that focus on a NoSQL database where we cannot achieve a perfect parallelism between I/O wait and CPU usage. In fact, as most operations require network and/or disk resources we must consider I/O wait to accurately represent the cost of such operations in the metric. Thus, if the combined I/O wait and CPU usage reaches 100%, the throughput does not increase by adding more clients.

In order to demonstrate that effectively the cache hit ratio is related to server usage, we set up two different experiments using a HBase deployment and YCSB (Cooper et al., 2010) as the workload generator. These experiments, while not necessarily representative of real-world workloads, cover a wide spectrum of possible behaviors. With these we are able to show a clear and direct relationship between the cache hit ratio and server usage in NoSQL databases.

In both experiments, one node acts as master for both HBase, and it also holds a Zookeeper (Hunt et al., 2010) instance running in standalone mode, which is required by HBase. Our HBase cluster was composed of 1 *RegionServer*, configured with a heap of 4 GB, and 1 *DataNode*. HBase's LRU *block cache* was configured to use 55% of the heap size, which HBase translates into roughly 2.15 GB. We used one other node to run the YCSB workload generator. The YCSB client was configured with a *readProportion* of 100% i.e. only issue *get* operations, and with a fixed throughput of 2000 operations per second with 75 client threads so we solely analyze the impact of cache hit ratio in server usage. All experiments were set to run for 30 minutes with 150 seconds of ramp up time and the results are the computed average of 5 individual runs. The server usage was logged every second in the *RegionServer* node using the UNIX *top* command. All nodes used for these experiments have an Intel i3 CPU at 3.1GHz, with 8GB of main memory and a local 7200 RPM SATA disk, and are interconnected by a switched Gigabit local area network.

In the first experiment, a single *region* was populated using the YCSB generator with 4,000,000 records (4.3 GB). This means that the *region* cannot be fitted entirely into the *block cache*: about 1.1 millions records (1.21 GB) remain on secondary memory and must be brought into main memory when requested. There were two different scenarios each with a different configured request popularity:

1. A *uniform* popularity distribution, that is all records have equal probability of being requested (the is the case where the cache hit ratio is minimum (Sleator and Tarjan, 1985));
2. A *zipfian* popularity distribution, highly skewed, and clustered, meaning that the most popular keys are contiguous, which makes them fall in the same cache block.

The results for this experiment are depicted in Table 1. As expected, the *uniform* request popularity is the one that achieves the lower cache hit ratio (49%) and thus consumes more server resources (58.35%). On the contrary, the *zipfian* request popularity attains the higher cache hit ratio (93%), because the most popular records are clustered and as a result fall in the same block served by HBase's block cache. And because of that it consumes the least Server resources (only 19.28 %) for the same 2000 ops/s.

Distribution	p_{hit}	Average $Server_{usage}$	#Records
Uniform	49%	58.35%	4,000,000
Zipfian	93%	19.28%	4,000,000

Table 1: Average $Server_{usage}$ and cache hit ratio results under 2 different distributions, for a *region* not fitting in *block cache*.

In the following experiment we show that two dif-

ferent distributions, with different data sizes but with the same cache hit ratio, will have the same server resources consumption if subject to the same fixed throughput. Consequently, in this experiment we used the same setting as the first experiment for the *zipfian*, again populated with 4,000,000 records (4.3GB), but we changed the number of records of the *uniform* distribution to 2,141,881(2.3 GB) so its cache hit ratio could also be 93%. The throughput is again fixed at 2000 operations per second. From Table 2 it is possible to see that the amount of resources used by both distinct distributions is identical. Despite the fact that they have been populated with different data sizes. Therefore, for some throughput all it takes to have an identical server usage is an identical cache hit ratio, regardless of the data size and the distribution.

Distribution	p_{hit}	Average $Server_{usage}$	#Records
Uniform	93%	19.76%	2,141,881
Zipfian	93%	19.28%	4,000,000

Table 2: Average $Server_{usage}$ and cache hit ratio results for 2 distributions with different sizes, but with same cache hit ratio.

From both experiments we can infer that the cache hit ratio is related to database resource usage: for a given throughput, the higher the cache hit ratio, the lower the server usage. In addition, the cache hit rate reflects not only the data size, but also the underlying distribution of requests which, in combination with an incoming throughput, corresponds to a given server usage.

As a result, we envision that the server usage of any workload can be estimated simply by knowing its cache hit ratio and incoming throughput, regardless of the distribution of requests and data size. In that regard, it should be possible to to build a tridimensional model, that models the server usage for a NoSQL database, when the cache hit ratio and the throughput vary. It is worth noting that by collapsing the data size and the request distribution into a single metric, the cache hit ratio, the construction of such model should be simplified. Nonetheless, that model will be hardware dependent and has to be rebuilt when the hardware changes or when there are changes in core configuration parameters of the database.

4 CONCLUSION

In this paper, we demonstrated that there is a direct and univocal relationship between the server usage and the cache hit ratio of NoSQL databases. Furthermore. we propose that instead of a specific workload is characterized by the three common parameters, namely: i) data size; ii) distribution of requests and iii) incoming throughput; a workload can be characterized by the incoming throughput and by its cache hit ratio, as the latter is a reflection of the i) data size and of the ii) distribution of requests. This simplification can simplify the construction of a resource usage server model that could then be used to estimate the server usage for any combination of cache hit ratio and incoming throughput.

Acknowledgements

This work was part-funded by project CoherentPaaS: A Coherent and Rich PaaS with a Common Programming Model (FP7-611068).

REFERENCES

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2006). Bigtable: a distributed storage system for structured data. In *OSDI*.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *SoCC*.

George, L. (2011). *HBase: The Definitive Guide*. O'Reilly.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11.

L., A. and M., P. (2009). Cassandra - a decentralized structured storage system. In *LADIS*.

Puzak, T. R. (1985). *Analysis of Cache Replacement-algorithms*. PhD thesis. AAI8509594.

Sleator, D. D. and Tarjan, R. E. (1985). Amortized efficiency of list update and paging rules. *Commun. ACM*, pages 202–208.