Open Source In-Memory Data Grid Systems: Benchmarking Hazelcast and Infinispan

Conference Paper · April 2017

DOI: 10.1145/3030207.3053671

4 authors, including:

Haytham Salhi
Birzeit University
3 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Adel Taweel
Birzeit University
126 PUBLICATIONS **1,117** CITATIONS
SEE PROFILE

Open Source In-Memory Data Grid Systems: Benchmarking Hazelcast and Infinispan

Haytham Salhi
hsalhi89@gmail.com
Faculty of Engineering and Technology
Birzeit University, Palestine

Feras Odeh
ferasodh@gmail.com
Faculty of Engineering and Technology
Birzeit University, Palestine

Rabee Nasser
rabinasser@gmail.com
Faculty of Engineering and Technology
Birzeit University, Palestine

Adel Taweel
ataweel@birzeit.edu
Department of Computer Science, Faculty of Engineering and Technology
Birzeit University, Palestine

ABSTRACT

In this paper, we studied the performance of two popular open source distributed cache systems (Hazelcast and Infinispan) indifferently. The conducted performance analysis shows that Infinispan outperforms Hazelcast in the simple data retrieval scenarios as well as most of SQL-like queries scenarios, whereas Hazelcast outperforms Infinispan in SQL-like queries for small data sizes.

Keywords

Benchmarking, Hazelcast, Infinispan

1. INTRODUCTION

In computing, a cache is a software component that stores portions of datasets which would otherwise either take a long time to calculate, process, or originate from an underlying backend system [8]. A cache system could be used either to decrease application latencies or gain additional performance [8].

In this paper, the performance of Hazelcast (version 3.6.1) and Infinispan (version 8.1.2.Final) was studied, with specific focus on two key factors: *number of concurrent clients* and *size of processed data*. The study focused on the data retrieval aspects, that they are the most common operations in the use cases of distributed caches. Yardstick was used as a primary benchmarking framework [5], however, an additional mechanism was also developed, to benchmark distributed caches, to enable capturing the varying number of clients and data sizes to ensure proper synchronization of run-times.

Other benchmarking tools have been considered, which some have been specially extended to produce metrics for database operations within transactions to detect anomalies from workload processing, e.g. "Yahoo! Cloud Serving Benchmark" (YSCB) [6, 4] extended to YSCB+T [7]. However, Yardstick [5] has been chosen because it provides a more extensible feature-rich open source framework to develop and customize. Also Yardstick provides benchmarks that are relatively easier and faster to develop than other frameworks, e.g. Java Microbenchamrking Harness (JMH) [2], Radar Gun [3], and YSCB [6, 4]. The results show that there is a clear relationship between the performance of data retrieval operations and the number of concurrent clients and size of data.

ICPE'17 April 22-26, 2017, L'Aquila, Italy

ACM ISBN 978-1-4503-4404-3/17/04.

DOI: http://dx.doi.org/10.1145/3030207.3053671

2. FACTORS OF INTEREST

The performance of a distributed cache system depends on several different factors. Since the dependent variable of interest is *the performance of data retrieval operations*, in this experiment, we are particularly concerned with the effect of two key independent variables as follows: *(1) Number of concurrent clients:* ***1, 2, 4, 8, 16, 32, 64, and 128****. (2) Data sizes:* **100, 1000, 10000, 100,000, and 1,000,000**. The developed benchmarks and shell scripts, used in this experiment, can be found on our public Github[1] repository.

3. BENCHMARKING RESULTS

Figures 1 and 2 draw obtained results, each representing the behaviour of performance for each of the two studied systems. The Y-axis represents the throughput (ops/sec), while the X-axis shows the number of concurrent clients.

For *get* query, as shown in Figure 1, the throughput of Infinispan is generally better than Hazelcast. However, for *SQL-like* queries, which are more complex than the primitive *get* query, the throughput of both systems is significantly smaller compared to the case for *get* query. There is a significant drop in throughput for both systems for changes in data size from 100 to 100000. The results also show that the effect of the number of concurrent clients becomes less significant for larger data sizes.

[1]https://github.com/ferasodh/Distributed-Caches-Benchmarking-Experiment

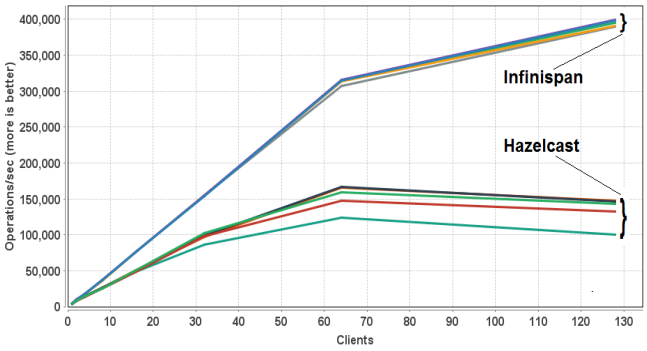

Figure 1: Behaviour of *get* operation performance in term of throughput (ops/sec) as a function of number of clients (each colour represents a data size; as shown the larger the data size the smaller the average throughput).

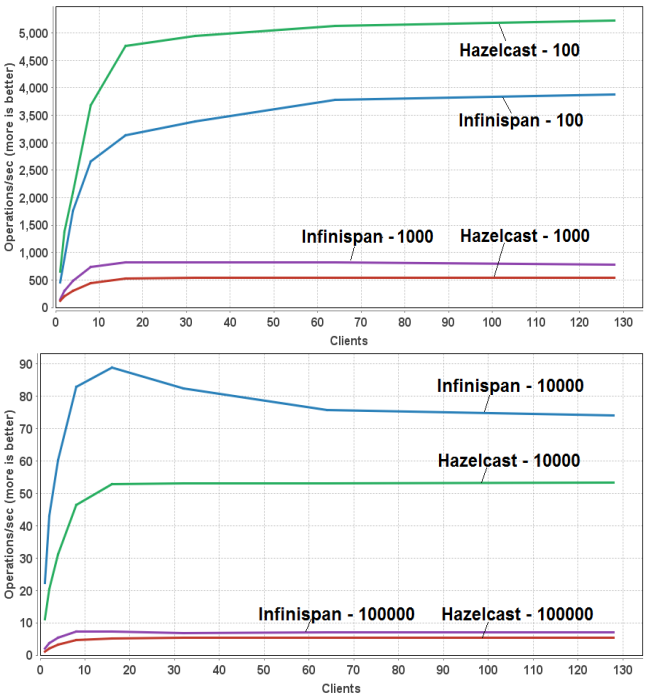

Figure 2: General behaviour of *SQL-like* performance in term of throughput (ops/sec) as a function of number of clients for all data sizes.

4. DISCUSSION

There are several factors that affect the performance of Infinispan and Hazelcast that could be reasons for performance bottlenecks. Some of these factors are discussed below:

- **Data serialization:** For each request a cache system processes, nearly 20% of the processing time is spent in serialization and deserialization in most configurations [1], which can be computationally expensive.
- **In-memory objects format:** In Hazelcast, the default format is the binary format. However, this format is not efficient if the application is processing large number of SQL-like queries, where then serialization and deserialization happen on the server side [9].
- **Indexing** is one of the most significant factors in query performance. Although single-attribute indexes were added to both systems, the engine for Infinispan, which is based on hibernate search and Apache Lucene, is more optimized than Hazelcast default indexing mechanism, thus may have resulted in improved performance.

5. CONCLUSION AND FUTURE WORK

The above results show that studying performance analysis of cache systems with dynamically varying number of concurrent clients and data sizes is critical in determining a more accurate performance readings. Measuring performance with static independent variable or factors may provide misleading results, particularly in systems where cache is a critical part of a system function or design. These require building benchmarking tools that consider such dynamically changing variables to reflect and replicate real-life usage of systems.

In addition, the results show that Infinispan (version 8.1.2.Final) outperforms Hazelcast (version 3.6.1) in all the tested cases except in SQL-like queries with small data sizes. They show also that the concurrent clients, where each client opens its own connection, has a considerable impact on the performance of *get* and *SQL-like* queries. The data size, on the other hand, has very small impact on the performance of *get* query but large impact on the performance of *SQL-like* queries.

Further, based on the mechanism followed in this experiment, a more integrated benchmarking framework, as proposed above, needs to be developed which takes into account the varying number of concurrent clients and data sizes for distributed caches. To test more accurately, future work may include developing new techniques that improves the performance with respect to data representation and communication protocols.

6. REFERENCES

[1] Infinispan. http://www.aosabook.org/en/posa/infinispan.html#fn10. Accessed on: 25/06/2016.

[2] Jmh - java microbenchmark harness. http://tutorials.jenkov.com/java-performance/jmh.html. Accessed on: 28/05/2016.

[3] Radargun documentation. https://github.com/radargun/radargun/wiki. Accessed on: 28/05/2016.

[4] Yahoo! cloud serving benchmark. https://github.com/brianfrankcooper/YCSB/wiki. Accessed on: 28/05/2016.

[5] Yardstick - benchmarking framework. https://github.com/yardstick-benchmarks/yardstick. Accessed on: 28/05/2016.

[6] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[7] Dey, A., Fekete, A., Nambiar, R., and Röhm, U. Ycsb+t: Benchmarking web-scale transactional databases. In *Proceedings of Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on* (2014), IEEE, pp. 223–230.

[8] Engelbert, C. White paper: Caching strategies. Tech. rep., Hazelcast Company.

[9] Evans, B. White paper: An architect's view of hazelcast. Tech. rep., Hazelcast Company.