Mahesh Sunil Patil, 2026, 14:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science,
Engineering and Technology
An Open Access Journal

Impact of Caching on Spring Boot Application Performance

Mahesh Sunil Patil[1] Dr. Deepak Singh[2]
Department Of Computer Science,
Suryadatta College Of Science And Commerce, Pune, Maharashtra, India Research Guide:

ABSTRACT- Web applications often use caching to minimize repeated accesses to the database. This paper reports on a preliminary Evaluation of application-level caching of a Spring Boot REST application backed by MySQL and Redis as the application level cache store. cache layer. The manual mode was used to record response times using Postman with the following reported conditions: caching disabled,The following conditions were reported: caching disabled, caching enabled (with a cold cache), and enabled (with a warm cache). Five observations were provided for each of the following :condition. The 5 observations that were not cached were 4.10 seconds, 11 ms, 20 ms, 42ms and 9 ms. The cold-cache observations were 10.9 ms, 6.4 ms, 10 ms, 11 ms and 10 ms; the warm-cache observations were 16 ms, 18 ms, 13 ms, 153ms and 9 ms. The first observation is marked as a 'caching-disabled' observation, which includes a substantial outlier and has a strong influence on the arithmetic mean. The Due to the high value of 153ms in the warm-cache observations, the medians are also reported. but means and results should be considered as tentative. All the measurements available do not confirm a This is a consistent improvement in response-time due to caching. Count of database queries, throughput with concurrent load, CPU No measurements of and claims for memory utilization are made. The study is done by small scale measurement. understands and recognizes the extra level of control needed for a more dependable assessment.

INDEX TERMS Spring Boot, Redis, MySQL, application-level caching, response time, Postman, performance evaluation.

I. INTRODUCTION

Performance of an application has an impact on the responsiveness and usability of a web-based system. Interactive services that respond Slow ones are not seen as being as fluid or natural to users as those that are slower [1] are perceived within approximately 100 milliseconds. A typical Spring When Boot REST request is called, it can run business logic and get data from the relational database and return the data. The Internet-scale distributed hypermedia systems were designed by REST [2] that is an architectural style; response. If the same

If information is repeatedly requested, then the application could be running similar database operations many times. Caching stores The outcome of one operation, to allow subsequent operations to use it again. Once the caching logic is added into the application's If it is going to store processed content in memory, then it is application level caching and the developers have to make a choice. How and what to cache [3]. Redis is an in-memory key-value data store, which can be used as a cache backing service; Such key-value stores are typically implemented using hash indexes stored in RAM, like Memcached or Redis.

[4]. A demand-filled look-aside cache, also called a cache-aside pattern [5] is one in which the application tests the cache first.The application can fetch the value from the and return it in the case of a hit, or it can go back to the server to get the value in the case of a miss.

Saves it in the database and caches it. On a write, the stale cache entry is invalidated [5]. Caching is not guaranteed to... improve every request. The effect observed is dependent on the cost of the database query, cache hit rate, network latency, Serialization,

© 2026 A Mahesh Sunil PatilThis is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

cache configuration and test environment. The hit rate will be based on the workload: request:

It has been reported that popularity on the Web follows Zipf-like distributions with different exponents for different traces. In the more recent proxy-based studies [7], popularity of web pages was discovered to be much more skewed than in [6]. Application data stored in caches can also cause consistency issues with respect to the underlying database [8]. Thus it is desirable to evaluate the effect by using measured observations and well described conditions.

1. Research Problem And Questions

Avoidable database activity may occur when using the same records multiple times in a Spring Boot application.

- Although Redis caching can help minimize repeated database reads, the impact on end-to-end response time will vary depending on the application. and workload. The study considers the following questions:
- RQ1: What are the difference in response times between the two configurations (no application level caching and Redis caching enabled)?
- RQ2: What is the difference in response time for a repeated call for a cold-cache versus warm-cache response?
- RQ3: What are the limitations and considerations with regards to the results that are derived from the measurement? The

The initial response-time observations in this report are preliminary and should not be used to draw final conclusions. Does not give enough evidence to answer

Database-query questions, throughput questions, or resource-utilization questions.

2. Research Objectives

- To examine response times of a selected Spring Boot REST API under caching-disabled and caching-enabled conditions.
- To record cold-cache and warm-cache observations when Redis caching is enabled.
- To summarize the collected response times using arithmetic mean and median.
- To identify unusual observations and limitations that may affect interpretation.
- To outline further measurements needed for a reproducible performance comparison.

II. LITERATURE REVIEW

Web caching. Web caching has received much research. Wang [9] reviewed the existing web caching schemes and presented the elements of a caching system and its desirable properties were discussed, while Podlipnig and Böszörmenyi [10] classified the many cache configurations. replacement strategies that have been proposed. Breslau et al. [6] showed that page requests follow a Zipf-like rather than In fact, it was observed that the exponent of the Zipf distribution is not constant, but depends on the traces. Padmanabhan and Qiu [7] have investigated the server side ofa busy news site, found that file popularity was Zipf-like, but with a higher value of the Zipf parameter than in previous proxy-based studies. that some set of files were modified over and over, which is important when dealing with freshness of cached copies.

1. Application-Level

caching. In this context, approaches to designing, implementing and managing application level caching were surveyed by Mertz and Nunes [3]. and mentioned that developers have to deal with the cache on their own, using possibly existing libraries or creating their own, frameworks. Application-level caches like memcached place the responsibility for the cache on the application as observed by Ports et al. [8]

For caching and keeping data in cache up-to-date; and a transactional cache, which he proposed to provide higher throughput.

In their experiments, the a web application was improved up to 5.2 times. Nishtala et al. [5] explained how Facebook uses memcached as a A cache that can be written to and invalidated on writes, in front of MySQL and filled on demand. Bronson et al. [11] reported That Facebook's web servers were originally the ones that had direct access to MySQL, using memcache as a look-aside cache, and that they've since switched to Memcached exclusively as their cache.That

Facebook's web servers used to be direct to MySQL, but now use Memcached as their cache.

Introduced TAO - a graph-aware cache that prioritises availability and efficiency over strong consistency. In-memory stores and workloads. Zhang et al. [4] overviewed in-memory data management and processing systems, such as key-value stores. such as Memcached and Redis. Atikoglu et al. [12] studied workloads of the Facebook's Memcached deployment.

noted that a key-value store will be as efficient as other caching heuristics, depending on its workload. Latency and measurement. Dean and Barroso [1] pointed out that short episodes of high latency, unimportant in moderate-size As systems grow, can come to dominate overall service performance, so look at unusually slow systems. and not at average rates. From the methodological perspective, Cooper et al. [13] presented a framework of standards. important enough to warrant platform-specific optimizations is typically not worth the effort.Georges et al. [14] found that data analysis that is not important enough to optimize for platform-specific performance is usually not worth the effort. Measures like a single average or best of several runs, which are statistically rigorous, may be misleading, and may be pushed. confidence intervals. It was shown by Mytkowicz et al. [15] that seemingly innocuous details of an experimental setup can introduce some kind of measurement error resulting in incorrect inferences, as was noted by Kalibera and Jones [16] that many evaluations. "Don't despair about how much repetition or enough statistics and report effect sizes with confidence intervals." Taken Overall, the literature justifies the assessment of caching in the context of a specific workload, repeated many times and with sufficient duration, They documented it as a setup, not assuming that in every application the same performance impact will be observed.

2. Research Gap And Proposed Contribution

Published studies show that caching approaches are effective in their particular environments [5, 8, 11, 12] but the results cannot be directly extrapolated to other environments. Be directly mapped to this particular Spring Boot application. This study offers a small preliminary comparative study. Using the same general API workflow responsetime observations. It is a small number that limits its contribution Lack of observability and independent measurements of database queries and system resources. The reported values are therefore made available for the pilot as measurements, and not as proof of a caching benefit in general.

III. METHODOLOGY

This was done through an experimental approach to see the API response time in three conditions.

- **Caching disabled:** Application level caching was turned off for the service operation tested. An API call was made The response time was recorded using Postman.
- **Caching enabled** — cold cache: Redis caching was enabled with the application's cache configuration, @Cacheable annotation. A cold cache request is one which is made for a cache entry that has not yet been filled; and Thus a request may need to retrieve a database and may be used as such.
- **Caching enabled** — warm cache: When the cache entry was populated, it was repeated although requests were made for the same data. These observations have been considered as warm-cache observations. The observations reported by the user are divided into five

The values of response-time for each condition. The same general application/API workflow was used, but a complete laboratory

Event logs with information on hardware, exact software versions, size of datasets, request endpoint, cache TTL, Redis deployment mode, This manuscript did not contain and request timing protocol. The details are not assumed or invented, therefore. The interval of time that Postman shows for a single request is known as Response time. It symbolizes what clients see from end-to-end.It is a representation of what the client sees end-to-end. may contain application processing time, database access time or cache time, network time etc. The observations The arithmetic mean and median were used to summarize Because extreme values can have a significant influence on a small sample, value, both are reported, consistent with the general recommendation

that performance summary reflect value. run-to-run variation [14], [16].

1. Experimental Environment

The project is implemented using technologies such as Java, Spring Boot, MySQL, Redis, a REST API, and Maven. Postman, and Windows. Certain versions and hardware requirements are not provided and not specifically mentioned. The chosen API operation fetches data from MySQL and accesses Redis caching with the help of spring's @Cacheable annotation.

When caching is on, this will be the case. Without these details there is less ability to be reproducible and the potential sources of these are unknown.

There is no assurance of measurement bias [15]. The three conditions are different as follows: (1) application-level caching

The three different states are: (1) disabled, (2) caching enabled with a cold cache, and (3) caching enabled with a warm cache. The response-time samples

The 5 requests made for each condition are recorded in these . This is not intended to be used as a sequential Postman measurement. This was interpreted as a concurrent-load/throughput benchmark.

2. Results

Table I presents the five observations recorded under each condition. The value 4.10 s was converted to 4,100 ms for comparison with the millisecond observations. This conversion follows the unit written in the supplied result. The unusually high value is retained rather than silently removed.

TABLE I. Response-time observations

Test	Caching disabled	Cold cache	Warm cache
1	4.10 s	10.9 ms	16 ms
2	11 ms	6.4 ms	18 ms
3	20 ms	10 ms	13 ms
4	42 ms	11 ms	153 ms
5	9 ms	10 ms	9 ms
Mean	836.4 ms	9.66 ms	41.8 ms
Median	20 ms	10 ms	16 ms

Note: The caching-disabled first observation was supplied as 4.10 s and converted to 4,100 ms. All other observations were supplied in milliseconds.

IV. RESULTS AND DISCUSSION

The disabled group (caching = disabled) had an arithmetic mean of 836.4ms and a median of 20ms. The large difference between these The first observation (4,100ms) is the major contributor to values. The other four observations were in the range of 9 msec to 42 ms. The first "high value" cannot be explained from the available records, including: The initial application or database setup is a hypothesis only and not verified. Run-to-run non-determinism in Java applications, for instance from just-in-time compilation and garbage collection [14], is another general possibility.

This was not explored here. The mean was 9.66ms, and the median was 10ms for the case when caching is enabled with a cold cache. For With a warm cache, the mean was 41.8 ms and the median was 16 ms when the caching was enabled. The warm-cache set has one observation of 153 ms will increase its mean; isolated slow requests like this are exactly what latency-focused applications are looking for. Studies caution that summary figures can be skewed by the use of studies [1].

In the 5 paired observations from last time that you provided, 4 warm-cache values were provided. One of them was lower than its cold-cache value, while the others were higher. These are the small and varying samples that do not Show a steady reduction of response time when using warm cache. The average of the set that is not being cached should not equal the average of the set that is being cached. be compared as if it were a stable baseline, since it is dominated by observation of 4,100ms. Likewise, the warm-cache mean is affected by the 153 ms observation.

The medians give a more comprehensive summary for these small samples, but five Compared to a successful development claim, the number of observations per condition is not yet adequate [14], [16]. It will also log the timings of the Postman. Not provide the number of queries to MySQL, Redis hit ratio, throughput, CPU or memory usage.

1. Limitations

There are some limitations of the study. First, there are only 5 response-time (RT) observations per condition, which is too small. In the benchmarking literature [14] and [16] it is recommended to use comparisons based on confidence intervals, which we have few of. Second, the caching-disabled observations have a very high first value and warm-cache observations have a high value of 1. ms. Third, exact hardware, versions of software, size of a dataset, endpoint information, cache TTL, and network conditions. Measurement bias cannot be ruled out since they were not noted in the provided notes [15]. Fourth, the measurements were

The sequential Postman requests instead of a controlled concurrent-load test. These are the only ones that cannot be cached by Redis.The following ones cannot be cached by Redis:No counters, throughput, CPU or memory measurements were provided, as cache benefit is a function of hit rate and workload [6], [12] the study does not make any claim as to how the tested request pattern relates to a realistic workload. These limitations Do not allow a final opinion regarding the impact about all performance of Redis caching.

V. CONCLUSION

This paper presented a preliminary response-time evaluation of a Spring Boot application using MySQL and Redis. Five observations were reported for each of three conditions: caching disabled, caching enabled with a cold cache, and caching enabled with a warm cache. The median response times were 20 ms, 10 ms, and 16 ms respectively. However, the caching-disabled mean was strongly affected by a 4,100 ms observation, and the warm-cache mean was affected by a 153 ms observation. The small sample and variability mean that the available results do not establish a consistent performance improvement from caching. The study therefore reports the observed values without claiming that Redis reduced response time or database workload. A larger controlled experiment with documented environment details and database-side measurements is required for a conclusive evaluation.

1. Future Scope

A larger number of repeated observations under controlled conditions should be collected for future work, and confidence should be reported.

The following is a list of the parameters: interval for the difference between conditions, [14], [16]; document hardware, software versions, dataset size, endpoint, Measure MySQL query counts and Redis hit/miss behaviour; use a dedicated loadtesting tool for; measure response time for MySQL queries; cache key and TTL; measure Redis cache key size; First, the number of concurrent users, and second, the workloads, which are defined as explicitly as in the standard benchmarking frameworks [13]. Further work can Discuss cache invalidation and consistency following database updates, cache expiration and eviction policies [10]. [5], [8], [11] and assess performance on larger data sets and various request patterns – skewed popularity distributions [6], [7].

REFERENCES

1. J. Dean and L. A. Barroso, "The tail at scale," Commun. ACM, vol. 56, no. 2, pp. 74–80, Feb. 2013.
2. R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," ACM Trans. Internet Technol., vol. 2, no. 2, pp. 115–150, May 2002, doi: 10.1145/514183.514185.
3. J. Mertz and I. Nunes, "Understanding application-level caching in web applications: A comprehensive introduction and survey of state-of-the-art approaches," ACM Comput. Surv., vol. 50, no. 6, Art. no. 98, pp. 1–34, 2018, doi: 10.1145/3145813.
4. H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: A survey," IEEE Trans. Knowl. Data Eng., vol. 27, no. 7, pp. 1920–1948, Jul. 2015, doi: 10.1109/TKDE.2015.2427795.
5. R. Nishtala et al., "Scaling Memcache at Facebook," in Proc. 10th USENIX Symp. Networked Syst. Design Implement. (NSDI 13), Lombard, IL, USA, Apr. 2013, pp. 385–398.
6. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in Proc. IEEE INFOCOM, vol. 1,

1999, pp. 126–134, doi: 10.1109/INFCOM.1999.749260.
7. V. N. Padmanabhan and L. Qiu, "The content and access dynamics of a busy web site: Findings and implications," in Proc. ACM SIGCOMM, Stockholm, Sweden, 2000.
8. D. R. K. Ports, A. T. Clements, I. Zhang, S. Madden, and B. Liskov, "Transactional consistency and automatic management in an application data cache," in Proc. 9th USENIX Symp. Operating Syst. Design Implement. (OSDI 10), Vancouver, BC, Canada, Oct. 2010.
9. J. Wang, "A survey of web caching schemes for the Internet," ACM SIGCOMM Comput. Commun. Rev., vol. 29, no. 5, pp. 36–46, Oct. 1999, doi: 10.1145/505696.505701.
10. S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," ACM Comput. Surv., vol. 35, no. 4, pp. 374–398, Dec. 2003, doi: 10.1145/954339.954341.
11. N. Bronson et al., "TAO: Facebook's distributed data store for the social graph," in Proc. USENIX Annu. Tech. Conf. (USENIX ATC 13), 2013, pp. 49–60.
12. B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in Proc. ACM SIGMETRICS, London, U.K., Jun. 2012, pp. 53–64, doi: 10.1145/2254756.2254766.
13. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in Proc. 1st ACM Symp. Cloud Comput. (SoCC), Indianapolis, IN, USA, Jun. 2010, pp. 143–154, doi: 10.1145/1807128.1807152.
14. A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in Proc. OOPSLA, Montréal, QC, Canada, Oct. 2007, pp. 57–76, doi: 10.1145/1297027.1297033.
15. T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!," in Proc. ASPLOS, Washington, DC, USA, Mar. 2009, pp. 265–276, doi: 10.1145/1508244.1508275.
16. T. Kalibera and R. Jones, "Rigorous benchmarking in reasonable time," in Proc. ACM SIGPLAN Int. Symp. Memory Manage. (ISMM), Seattle, WA, USA, Jun. 2013, pp. 63–74, doi: 10.1145/2464157.2464160.