UDC 004.732.056 **DOI: 10.20998/2411-0558.2026.01.03**

S. LEONOV, Dr. Tech. Sci., Prof, National technical University "Kharkiv polytechnic institute",
D. TYRTYSHNYI, PhD student, National technical University "Kharkiv polytechnic institute"

PERFORMANCE OPTIMISATION OF REAL-TIME DATA INGESTION: ENHANCING THE INFLUXDB BACKEND LISTENER FOR JMETER

The object of the study is the process of information processing and real-time data ingestion during performance testing in load testing tools like JMeter. **The subject of the study** is the methods of optimizing software tools implementation for data forwarding to time-series databases, specifically for the InfluxDB Backend Listener Plugin used in JMeter. The purpose of the paper is to resolve inefficiencies in the plugin's performance and significantly enhance throughput and resource utilization of computer systems during high-load testing scenarios. **The results obtained.** An optimized implementation of the InfluxDB Backend Listener Plugin was developed, addressing bottlenecks in the software code that led to underutilization of system resources during tests. The enhanced version incorporates techniques such as asynchronous data write operations and batching, enabling efficient utilization of computing resources. Performance benchmarking demonstrated a remarkable 14.5x improvement in throughput, increasing from 480 requests/sec with spiky pattern to stable 7000 requests/sec. Advanced capacity testing further validated its ability to handle up to 20000 requests/sec under optimal conditions. **Conclusions.** Experiments confirmed the efficiency of the proposed solution. Code-level optimizations successfully resolved the inefficiencies in real-time metric forwarding and allowed the plugin to operate at maximum capacity. These improvements enable large-scale performance validation by generating sufficient load for backend systems while reliably forwarding real-time metrics for analysis in CI/CD workflows. The new implementation establishes the InfluxDB Backend Listener as a scalable, efficient, and reliable component of modern distributed systems and performance testing frameworks. Fig.: 12. Tabl.: 1. Refs.: 13 items.

Keywords: performance testing; InfluxDB; real-time data ingestion; throughput optimization; time-series database; JMeter; asynchronous write operations; batching; software tools; computer systems; distributed systems; information processing.

Introduction. Performance analysis plays a critical role in ensuring the reliability, stability, and scalability of modern software applications. With the growing complexity of software systems and the increasing demand for real-time monitoring, there is a heightened need for efficient ways to collect, process, and visualize performance metrics during tests. Tools like JMeter, widely adopted for load and performance testing, are crucial in simulating high system loads to uncover bottlenecks and measure key performance indicators (KPIs). To provide actionable insights in real-time, JMeter often integrates with time-series databases such as InfluxDB for storing and processing test metrics [1, 2]. However, inefficiencies in how these tools interact can severely impact their ability to scale and deliver high-throughput data under heavy testing scenarios.

Inefficient real-time metric ingestion during performance testing is one of the main challenges in achieving maximum system analysis capabilities [3]. Specifically, JMeter's InfluxDB Backend Listener Plugin, designed to forward test metrics to InfluxDB in real-time, can encounter significant limitations when subjected to high throughput testing scenarios. These issues stem primarily from suboptimal software implementation, which results in underutilization of system resources, delayed data transmission, and reduced throughput. Such inefficiencies hinder the ability to process and handle large-scale performance test results in real-time, limiting the effectiveness of load testing in continuous integration/continuous delivery (CI/CD) workflows [4].

Several factors contribute to the limitations of current real-time data ingestion processes between JMeter and InfluxDB:

– Underutilization of system resources: The original implementation of the Backend Listener Plugin fails to efficiently leverage available CPU and memory. Even under heavy load, system resources remain idle due to synchronous execution patterns and lack of batching, leading to bottlenecks in data throughput.

– Scalability challenges: The absence of proper optimization techniques, such as asynchronous writes and connection pooling, restricts the plugin's ability to handle the volume of data typically generated during high-load performance tests.

– Latency in metric transmission: Without batching and optimized network utilization, frequent individual data writes introduce significant delays, making it difficult to process metrics in real time.

– Impact on CI/CD workflows: Integrating load testing into modern CI/CD pipelines requires fast and efficient real-time analysis of performance metrics [5]. Existing inefficiencies in the metric ingestion process can delay feedback loops, reducing development agility and limiting system scalability.

Addressing these challenges requires substantial improvements to the existing implementation of the InfluxDB Backend Listener Plugin. Previous solutions in this domain have largely focused on server-side optimizations for time-series databases, such as indexing improvements or caching mechanisms, while leaving client-side plugins underexplored. Consequently, the need for optimizing the software design of the JMeter plugin to unlock its full potential for high-throughput performance testing and resource utilization remains critical.

Related work and Literature overview.

Numerous tools and frameworks have been developed to facilitate load testing, with JMeter emerging as one of the most versatile and widely adopted solutions due to its ability to simulate significant volumes of user traffic and generate performance metrics. While JMeter offers many features for performance engineers, its InfluxDB Backend Listener Plugin – frequently used for forwarding performance metrics to databases – has faced challenges in handling high-throughput scenarios. Integrating JMeter with time-series databases like InfluxDB is essential for real-time test analysis and visualization, especially in continuous integration / continuous delivery (CI/CD) workflows. However, inefficiencies in the existing plugin often result in delayed transmission of test data and underutilization of system resources.

InfluxDB, a time-series database optimized for real-time metrics, is commonly integrated into performance engineering pipelines. It is used for storing, querying, and visualizing real-time data. Several studies have explored its use in performance testing due to its scalable nature and built-in support for analyzing trends and anomalies [6], but few have focused on optimizing the data ingestion process between client tools like JMeter and the database. The issue is less about database limitations and more about the inefficiencies in the software implementation of such plugins. This reveals a gap in research on improving client-side plugins, which serve as intermediaries in performance monitoring pipelines.

Although significant advancements have been made in real-time performance testing tools and workflows, several limitations persist:

– Studies of JMeter have primarily addressed server-side optimizations like caching and indexing in InfluxDB [6, 13], while client-side optimization - such as improving the data forwarding process - remains underexplored. Most plugins, including the InfluxDB Backend Listener, do not efficiently leverage system resources during high-load tests. This limits the scalability and throughput of performance monitoring workflows [7].

– Existing time-series database ingestion systems tend to operate synchronously, preventing optimal resource utilization during real-time ingestion. Solutions using batching and asynchronous data forwarding, widely studied in fields like distributed data systems, have not been adapted well for load testing scenarios.

– Few studies have focused on improving integration efficiency in CI/CD workflows [4]. The ability to process real-time metrics efficiently is critical for providing feedback in automated pipelines and supporting rapid testing cycles. However, the absence of scalable plugin designs for high-load scenarios limits their applicability to large-scale systems.

While InfluxDB's use in real-time monitoring applications (such as IoT and edge computing scenarios [8]) has been explored, there is little work addressing its optimal integration with tools like JMeter.

Optimizing client-side plugins for metric forwarding is essential for achieving scalable performance monitoring in CI/CD workflows. Studies on batching, asynchronous operations, and connection pooling have proven effective in distributed systems to minimize write latency and streamline data ingestion [11, 12]. However, applying such techniques to plugin-level software remains underexplored.

In this context, our study focuses on addressing the unique bottlenecks of the InfluxDB Backend Listener Plugin in JMeter, with the goal of improving throughput [9] and resource utilization in real-time data ingestion workflows. By resolving inefficiencies in the plugin's software implementation, we aim to fill the identified gaps in research and enable seamless integration of performance testing workflows into scalable CI/CD pipelines.

Methodology.

2.1. Backend Performance Analysis Framework.

to conduct performance testing of backend systems and monitor detailed metrics in real time, I developed a Backend Performance Testing Framework, as illustrated in Fig. 1. This framework incorporates several components working together to generate high throughput, collect performance data, and visualize test results.

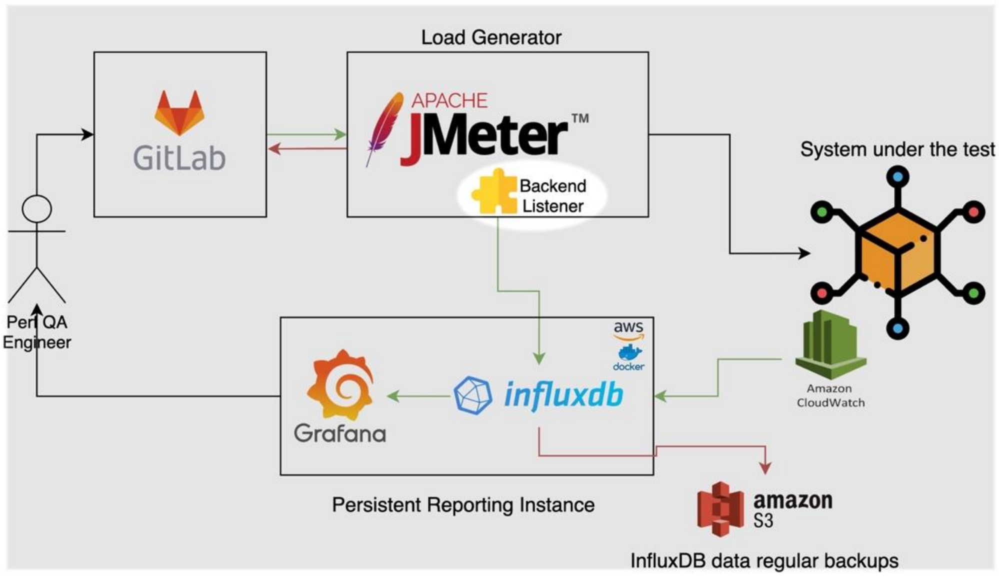


Fig. 1. Architectural Diagram of Backend Performance Testing Framework

Key Components:

– GitLab CI/CD Pipeline: The framework is triggered by automation pipelines within GitLab. Performance QA engineers initiate performance testing as part of the CI/CD workflows. Test scripts are stored there.

– Load Generator (Apache JMeter): JMeter acts as the load generator, simulating concurrent virtual users that interact with the System Under Test (SUT) to execute predefined test scenarios.

– Backend Listener Plugin: The InfluxDB Backend Listener plugin, the subject of this study, is integrated within JMeter to forward raw performance metrics during test execution for real-time monitoring.

– InfluxDB: Metrics forwarded by the Backend Listener plugin are stored in an InfluxDB - time-series database for central long-term metrics storage.

– Grafana: Visual dashboards are integrated with InfluxDB to display test results in real time for Performance QA engineers, main tool for results analysis [10].

– Amazon S3: Regular backups of test results are performed to ensure data integrity and long-term storage.

– Amazon CloudWatch: Monitors the health and infrastructure metrics of the Persistent Reporting Instance to ensure smooth operation during testing.

– System Under Test (SUT): The tested backend system includes distributed components running on cloud infrastructure. The framework is used to apply sustained load on the SUT and analyze its performance, stability and reliability under varying traffic conditions.

2.2. Role of the Backend Listener Plugin.

The key role of the Backend Listener plugin is to forward raw test metrics, such as response times, request throughput, and error rates, from JMeter to InfluxDB in real time. This enables:

– Accurate Performance Analysis: Collected metrics are stored in their raw form, enabling fine-grained analysis without losing critical outlier data.

– Real-Time Monitoring: Metrics are visualized live in Grafana, allowing engineers to identify bottlenecks, functional errors, reproducible only under high load, or resource constraints during test execution.

However, two existing approaches to forward JMeter metrics to the InfluxDB – the standard JMeter InfluxDB Backend Listener and my previous Custom Listener (Version 1) – presented significant performance challenges.

2.3. Standard JMeter InfluxDB Backend Listener.

The standard listener, while convenient for lightweight performance monitoring, exhibited major limitations under high-load scenarios:

– Synchronous Writes: The listener processes and sends metrics via synchronous HTTP requests (influxDB.write), causing test execution threads to block until writes complete.

– Limited Scalability: During tests generating around 400 requests/sec load, the listener consistently failed, throwing a TimeoutException:

```
ERROR o.a.j.v.b.i.HttpMetricsSender: failed to send data to influxDB server.
java.util.concurrent.TimeoutException: Connection lease request time out
```

- Functional Limitations:

Aggregates metrics before transmission, sacrificing accuracy and granularity.

Lacks support for custom tags, impeding detailed filtering and grouping of metrics for deeper analysis.

2.4. Custom Listener (Version 1). To address the functional limitations of the standard listener, Version 1 of a custom listener was previously implemented. While Version 1 supported raw metric ingestion and custom tags, it suffered from architectural inefficiencies that made it unsuitable for high-load scenarios:

– Blocking Behavior: Metrics were forwarded synchronously (synchronized writes), blocking the execution of test threads. At loads exceeding 500 requests/sec, the listener struggled to keep pace, leading to frequent throughput dips and an inability to generate sustained loads on the System Under Test (SUT). The throughput graph in Fig. 2 illustrates this behavior, where dips occurred due to blocking behavior at the listener level. The throughput varied from 670 requests/sec down to 16 requests/sec (see green line):

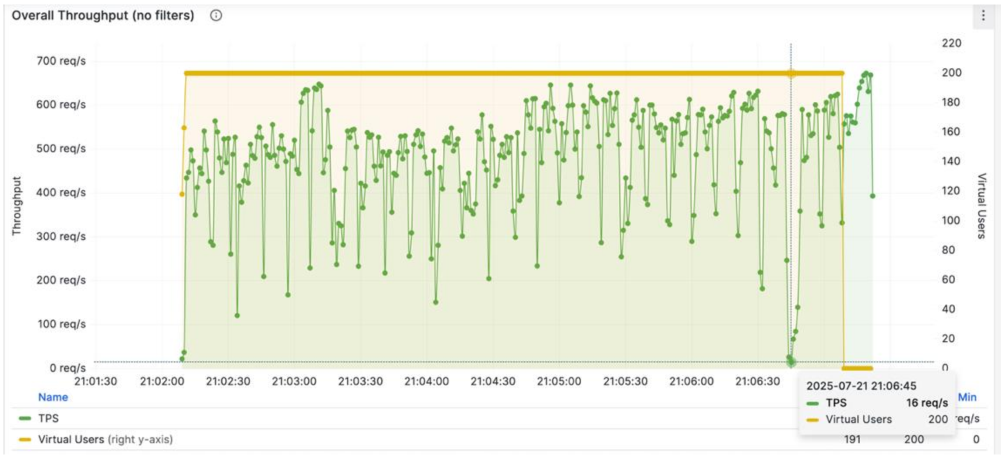


Fig. 2. Throughput graph with spikes - listener v.1

– Underutilization of Resources: CPU utilization on the system under the test remained as low as 10%, meaning that the system was not tested properly. With the v.1 implementation of the listener, the proper test execution was not possible, as test execution threads were delayed, waiting for metric writes.

– Network Overhead: Frequent, small HTTP transactions consumed substantial bandwidth and increased latency, further accelerating throughput issues.

2.5. Optimized Architecture: Custom Listener (Ver. 2) To overcome these limitations, Version 2 of the custom InfluxDB Backend Listener Plugin was developed with a focus on asynchronous architecture and high-throughput scalability. The key architectural changes in Version 2 include:

a) Asynchronous Write Architecture.

Concept: Decouple metric forwarding from test execution to ensure that test throughput is unaffected by the listener's data transmission processes.

Implementation: metrics are temporarily buffered in an in-memory queue (pointsBatch). Writes to InfluxDB are managed by the InfluxDB WriteApi, configured with asynchronous WriteOptions, which utilize background threads to handle data transmission independently of the test execution threads.

Key WriteOptions configurations:

Batch Size (500 points): Ensures efficient transmission by bundling multiple metrics into a single write operation.

Flush Interval (100ms): Automatically flushes buffered metrics to ensure timely updates.

Buffer Limit (10000 points): Provides fault tolerance by allowing high-load scenarios without data loss.

Advantages:

Non-blocking behavior: Test threads proceed independently of metric forwarding, ensuring sustained load generation.

High throughput: Scalable up to 7000 requests/sec, maintaining stability even under extreme load conditions.

b) Batching Mechanism.

Concept: Reduce the communication overhead between JMeter and InfluxDB by consolidating multiple metrics into batches.

Implementation: Metrics in the pointsBatch buffer are sent to InfluxDB when: The batch size exceeds the configured threshold (500 metrics); or the flush interval (100ms) expires, whichever occurs first. Batching reduces the frequency of HTTP write operations, improving the efficiency of network usage.

Advantages:

Reduces the number of HTTP requests, minimizing latency and bandwidth utilization.

Leverages InfluxDB's optimized batch processing capabilities for greater write efficiency.

c) Connection Pooling.

Concept: Optimize the reuse of network connections to minimize the latency caused by establishing and tearing down new connections for each write.

Implementation: A pool of reusable HTTP connections is maintained for metric forwarding. Connections are allocated dynamically, with idle connections timing out after a configurable period.

Advantages:

Reduces connection setup overhead, improving response times for HTTP operations.

Provides stability during long-running tests with consistent connection reuse.

Key Comparison: Standard vs. Custom Listeners.

The following table compares the architectural features and performance of the Standard JMeter Listener, Version 1, and Version 2:

Table 1

Backend listeners side-by-side comparison

Feature	**Standard Listener**	**Custom Listener (Version 1)**	**Optimized Listener (Version 2)**
Write Mechanism	Via HTTP	Synchronous writes	Asynchronous batch writes
Connection Management	Limited connection reuse	No pooling	Connection pooling
Throughput	400 requests/sec (fails)	480 requests/sec (spikes)	7000 requests/sec (stable)
Blocking the test	Yes	Yes	No
Custom Tagging	No	Yes	Yes
Granularity of Metrics	Aggregated/Raw - new version	Raw Metrics	Raw Metrics

Experiment. The experiments aimed to evaluate the performance improvements introduced in the optimized InfluxDB Backend Listener Plugin (Version 2), compared with the legacy implementation (Version 1) and the standard JMeter Backend Listener. The testing was conducted using the Backend Performance Testing Framework described in the Methodology section.

3.1. Experimental setup.

a) Load Generator:

– Tool: Apache JMeter v5.6.3 with custom-developed Backend Listener Plugin (both Version 1 and Version 2).

– Test Plan: Predefined test scenario simulating high-throughput API traffic on the System Under Test (SUT).

– Load Simulation: Fixed-load, ranging from 500 to 7000 requests per second was sustained for 5 minutes, using up to 500 virtual users.

b) Persistent Reporting Instance:

- Database: InfluxDB v.2.7 – time-series database used for capturing and storing metrics forwarded by the listener.

– Visualization: Grafana v10.1.1 – configured to display real-time metrics from InfluxDB during tests.

c) Infrastructure:

– System Under Test (SUT): Amazon EC2 instance with 4 vCPUs, 32 GB RAM, 1 Gbps Internet speed, fronted by Load Balancer.

– Load Generator: Apple M1 Pro, 32 GB RAM, 100Mbps Internet speed.

3.2. Test Scenarios. Testing consisted of four phases to evaluate plugin performance and scalability:

a. Baseline Test:

Testing without the Backend Listener to assess the maximum throughput achievable without reporting overhead.

b. Version 1 vs Version 2:

Comparative testing to evaluate improvements in throughput and resource utilization with the optimized plugin.

c. Standard Listener Comparison:

Validation of the optimized listener’s scalability and stability against the standard JMeter Backend Listener.

d. Capacity and Stress Testing:
Focused on identifying the maximum throughput capacity of Version 2 using mocked sample generators.

3.3. Measured Metrics. The following metrics were collected during the experiment phase:
a) Throughput: Number of requests processed by the System Under the test and forwarded to InfluxDB per second. Used to determine maximum sustainable throughput without failures.
b) Resource Utilization: CPU and memory usage of the listener plugin. Additionally, the CPU usage of the System Under the Test, to verify that Load Generator could satisfy load testing requirements.
c) Failure Rates: Number of requests dropped or failed during transmission. Used to assess reliability of the developed plugins under high-load conditions.

3.4. Progress, Results and Analysis.
Old backend listener (version 1) results. The practical part of the experiment started with the test with the old custom backend listener (version 1) to figure out current issues and limitation. Results of one of the iterations of the initial testing could be seen in the Fig.2 at the “Methodology” section. The throughput graph displayed lots of spikes, meaning that the flow was unstable. First idea was to verify that the System Under the test could keep up with the load. For that, the resources monitoring, using Telegraf, on the server has been set up. Results have shown that the load on the SUT is barely visible: just 11% CPU utilization, and 2Mb/min network throughput (see Figure 3).
Next, I examined the resource utilization of the Load Generator, considering the possibility that resource constraints could be contributing to the issue, given that load generation also demands system resources. However, the resource usage was found to be within normal levels and well below peak capacity.

Initially, it was assumed that the issue was limited to reporting, as the load generation process appeared to remain stable despite problems with storing metrics in InfluxDB. However, a detailed analysis of the aforementioned information, along with logs from the System Under Test (SUT), confirmed that the SUT receives messages at a throttled rate, consistent with the throughput patterns observed in the Grafana graph.

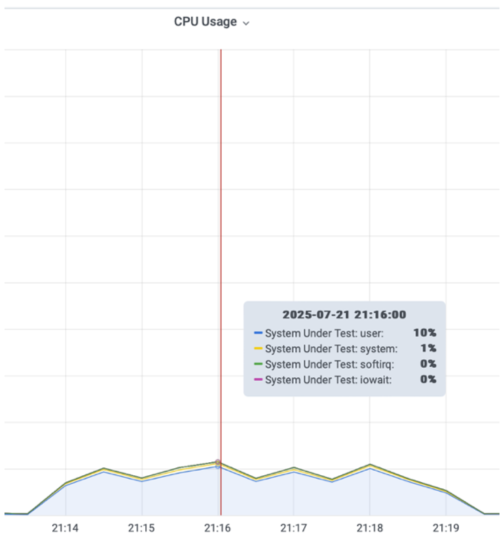
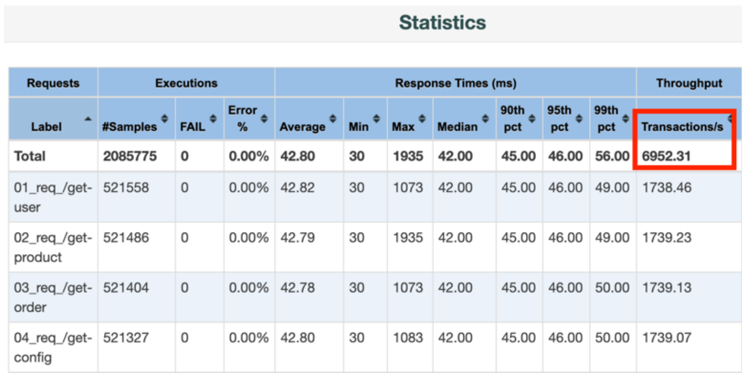


Fig. 3. CPU usage of the SUT during test with the Listener v.1

Baseline Test. All these facts indicate that the issue is somewhere in the software. The best approach here is to exclude different pieces one-by-one. Thus, I moved to the next part of the experiment – test execution without the Backend Listener at all. This approach allows me to store results in the standard JMeter report only. During the initial test run without the Backend Listener, it became evident that the insufficient load generation was caused by the suboptimal implementation of the Influx Backend Listener plugin for JMeter. The test run without Backend Listener has shown outstanding results – 6950 requests per second throughput, which is 14x more than the initial testing with the old implementation of the Backend Listener. The results generated from the standard JMeter report are presented in Figure 4 below.



Statistics

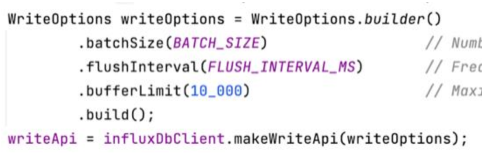
Requests	Executions			Response Times (ms)							Throughput
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s
Total	2085775	0	0.00%	42.80	30	1935	42.00	45.00	46.00	56.00	6952.31
01_req_/get-user	521558	0	0.00%	42.82	30	1073	42.00	45.00	46.00	49.00	1738.46
02_req_/get-product	521486	0	0.00%	42.79	30	1935	42.00	45.00	46.00	49.00	1739.23
03_req_/get-order	521404	0	0.00%	42.78	30	1073	42.00	45.00	46.00	50.00	1739.13
04_req_/get-config	521327	0	0.00%	42.80	30	1083	42.00	45.00	46.00	50.00	1739.07

Fig. 4. Report for the test without the listener.

This test established the performance benchmark that I aimed to achieve with the newly optimized Backend Listener implementation (version 2).

Implementation details. Before proceeding with the optimization, the shortcomings of the existing Backend Listener (version 1) were thoroughly analyzed. The identified gaps included:

a) Synchronous blocking metric writes.



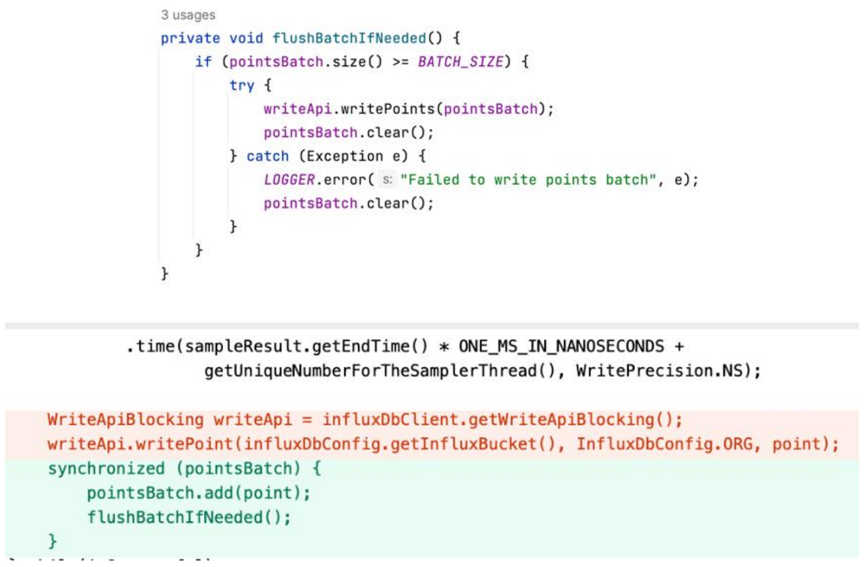
```
WriteOptions writeOptions = WriteOptions.builder()
        .batchSize(BATCH_SIZE)                  // Numt
        .flushInterval(FLUSH_INTERVAL_MS)       // Frec
        .bufferLimit(10_000)                    // Maxi
        .build();
writeApi = influxDbClient.makeWriteApi(writeOptions);
```

b) Sending InfluxDB records (points) separately rather than in batches.

c) Lack of a buffer mechanism for metrics.

After outlining all the gaps in the previous implementation, all of them were resolved by leveraging an updated version of the InfluxDB Java Library, which includes support of asynchronous batch writes. In the Figure 5 below you can observe that the WriteApiBlocking class and writeApi.writePoint method call were replaced by pointsBatch.add(point) and flushBatchIfNeeded methods. This allows to avoid saving each metric individually to InfluxDB but rather collect them in batch and transfer as a group. Additionally, the writeApi object was configured with specific WriteOptions consists of optimal values for my use case.

Optimized Listener testing. After the changes have been applied, an updated plugin was packaged into a .jar file and placed in the Jmeter plugins folder. After several iterations of testing and tuning the batch size, flush interval and the buffer limit – the goal was achieved.



```
3 usages
private void flushBatchIfNeeded() {
    if (pointsBatch.size() >= BATCH_SIZE) {
        try {
            writeApi.writePoints(pointsBatch);
            pointsBatch.clear();
        } catch (Exception e) {
            LOGGER.error( s: "Failed to write points batch", e);
            pointsBatch.clear();
        }
    }
}
```

```
        .time(sampleResult.getEndTime() * ONE_MS_IN_NANOSECONDS +
                getUniqueNumberForTheSamplerThread(), WritePrecision.NS);

    WriteApiBlocking writeApi = influxDbClient.getWriteApiBlocking();
    writeApi.writePoint(influxDbConfig.getInfluxBucket(), InfluxDbConfig.ORG, point);
    synchronized (pointsBatch) {
        pointsBatch.add(point);
        flushBatchIfNeeded();
    }
```

Fig. 5. Main code changes in the optimized Listener

I was able to generate a load of ~7000 requests/sec and hold it pretty stable with the optimized InfluxDB Backend Listener enabled. As demonstrated in Figure 6, just for 5 minutes there were sent 2.1 million requests to the SUT with 0 errors and no blocking throughput dips. Side-by-side comparison of optimized listener (after) and old listener (before) emphasizes the results that were achieved in scope of this work:

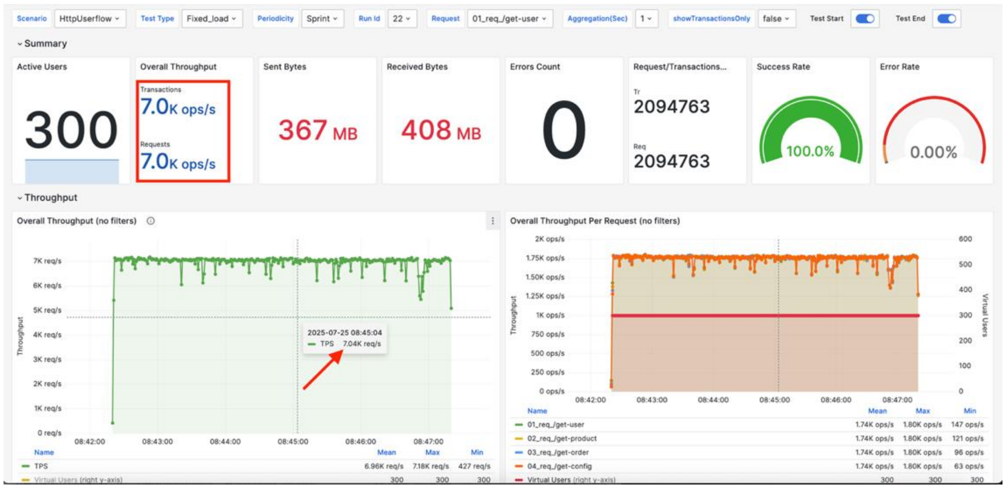
– 2.1 million vs 150k requests sent for 5 minutes

– Stable behavior, without significant dips

– ~7000 vs 480 requests/min average throughput.

– SUT CPU utilization 99% vs 11%, meaning that the generated load was more than enough to achieve goals of testing.

Fig. 7 and Fig. 8 provide a visual comparison of the performance metrics before and after optimization.


Scenario: HttpUserflow · Test Type: Fixed_load · Periodicity: Sprint · Run Id: 22 · Request: 01_req_/get-user · Aggregation(Sec): 1 · showTransactionsOnly: false · Test Start · Test End

Summary

Active Users	Overall Throughput	Sent Bytes	Received Bytes	Errors Count	Request/Transactions...	Success Rate	Error Rate
300	Transactions 7.0K ops/s; Requests 7.0K ops/s	367 MB	408 MB	0	Tr 2094763; Req 2094763	100.0%	0.00%

Throughput

Overall Throughput (no filters)

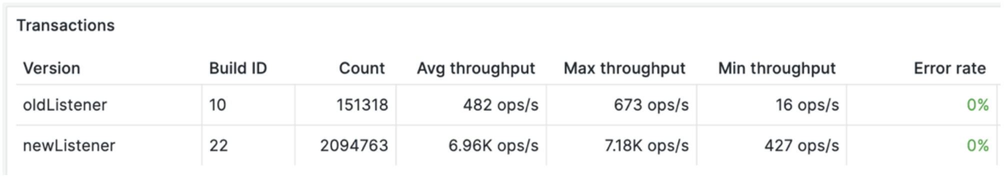
2025-07-25 08:45:04 TPS 7.04K req/s

Name	Mean	Max	Min
TPS	6.96K req/s	7.18K req/s	427 req/s
Virtual Users (right y-axis)	300	300	300

Overall Throughput Per Request (no filters)

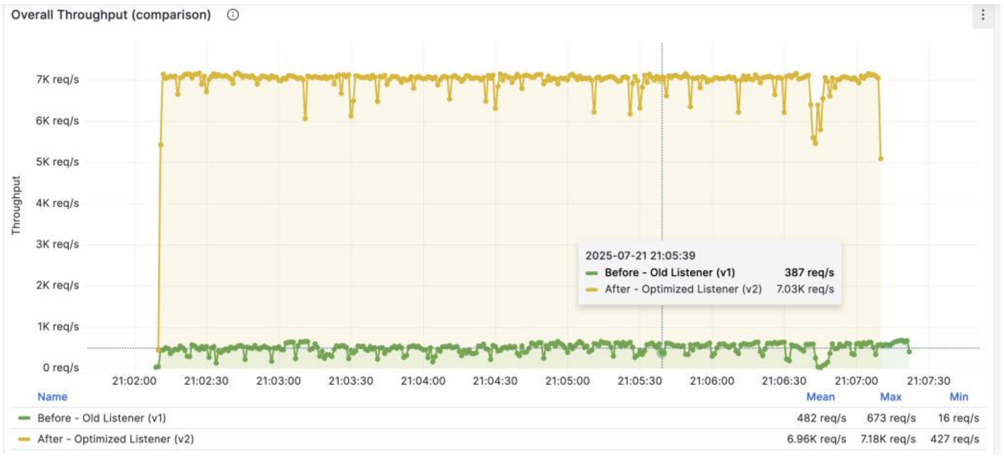
Name	Mean	Max	Min
01_req_/get-user	1.74K ops/s	1.80K ops/s	147 ops/s
02_req_/get-product	1.74K ops/s	1.80K ops/s	121 ops/s
03_req_/get-order	1.74K ops/s	1.80K ops/s	96 ops/s
04_req_/get-config	1.74K ops/s	1.80K ops/s	63 ops/s
Virtual Users (right y-axis)	300	300	300

Fig. 6. Optimized backend listener throughput and other summary metrics for the Fixed-load test


Transactions

Version	Build ID	Count	Avg throughput	Max throughput	Min throughput	Error rate
oldListener	10	151318	482 ops/s	673 ops/s	16 ops/s	0%
newListener	22	2094763	6.96K ops/s	7.18K ops/s	427 ops/s	0%

Fig. 7. Side-by-Side comparison of the old vs new listener


Overall Throughput (comparison)

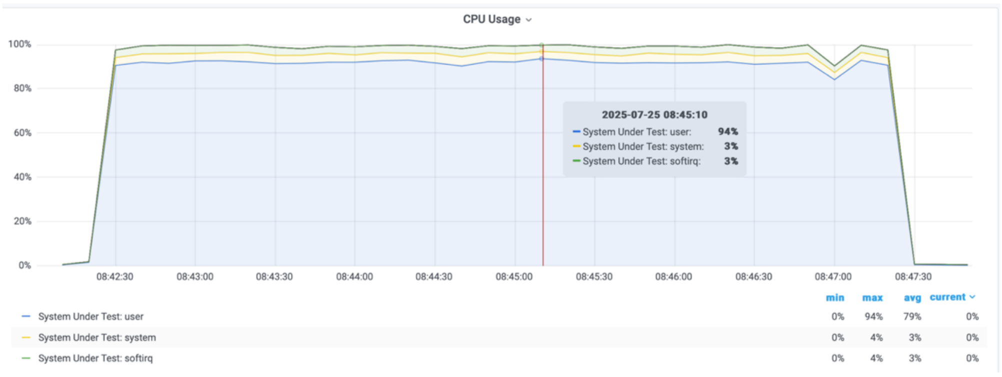
2025-07-21 21:05:39
Before - Old Listener (v1) 387 req/s
After - Optimized Listener (v2) 7.03K req/s

Name	Mean	Max	Min
Before - Old Listener (v1)	482 req/s	673 req/s	16 req/s
After - Optimized Listener (v2)	6.96K req/s	7.18K req/s	427 req/s

Fig. 8. Side-by-Side throughput graph comparison of the old vs new listener

Fig. 8 highlights the significant performance improvement achieved with the new optimized Backend Listener. Yellow line represents throughput of the

Listener v.2, when the green line corresponds to the Listener v.1. Although the dips in the green graph (Listener v.1) might appear less drastic at this scale, the optimized listener (v.2) is demonstrably more stable. Its throughput dips are typically within 10%, whereas the dips associated with the original listener often ranged between 75% and 90%. Additionally, the spikes visible in the performance of the optimized listener are not attributable to the listener or the load generation process itself. Instead, they are linked to the performance and capacity of the System Under Test (SUT). In contrast, the original listener failed to meet the requirements for application load testing, as it utilized only 10% of the SUT's resources. The optimized listener, however, enables the testing of the SUT at its maximum capacity while maintaining stability. Notably, there is evidence to assume that neither the optimized listener nor the load generator has yet reached its full potential. Fig. 9 illustrates the CPU utilization of the SUT when subjected to a sustained load of 7000 requests per second.



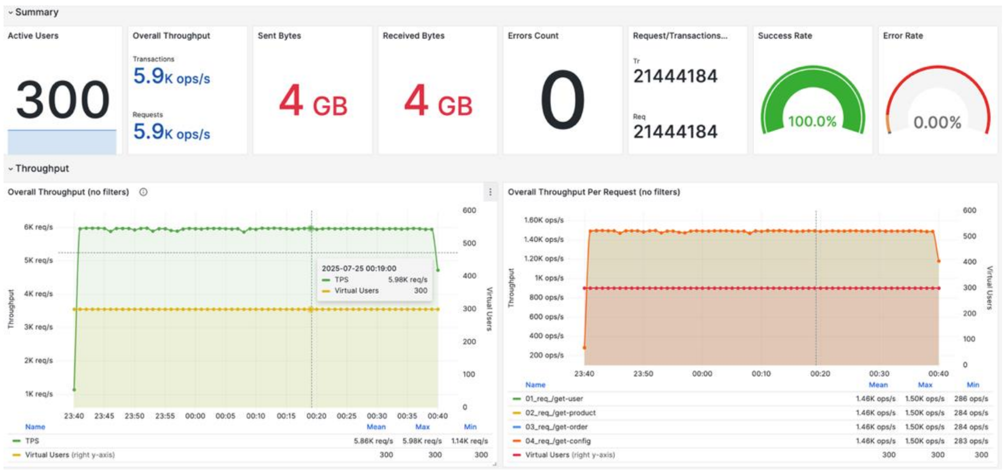
2025-07-25 08:45:10
System Under Test: user: 94%
System Under Test: system: 3%
System Under Test: softirq: 3%

	min	max	avg	current
System Under Test: user	0%	94%	79%	0%
System Under Test: system	0%	4%	3%	0%
System Under Test: softirq	0%	4%	3%	0%

Fig. 9. CPU utilization of the SUT during 7000 rps test

As shown in Figure 9, the SUT’s CPU utilization was constantly at level of 98-100%, indicating that the primary factor limiting throughput in this experiment was no longer the InfluxDB Backend Listener but rather the processing capacity of the SUT itself.

To further validate the stability of the optimized solution, I conducted an extended 1-hour test at 85% of the maximum load (~6000 requests per second). The results of this test, presented in Figure 10, demonstrate that throughput remained exceptionally stable throughout the entire duration of the experiment. A total of 21.4 million requests were successfully sent to and processed by the SUT, confirming that neither large-scale request volumes nor prolonged test durations could exhaust the capabilities of the new optimized listener. Additionally, the Figure 11 shows that this load is optimal for High Load test, since the CPU usage of the System under the test during the test execution was at ~85%. the load generator's CPU usage was consistent with earlier observations from testing without the listener, remaining around 40%. This suggests that the new optimized listener still has untapped potential for handling even higher workloads under appropriate conditions.

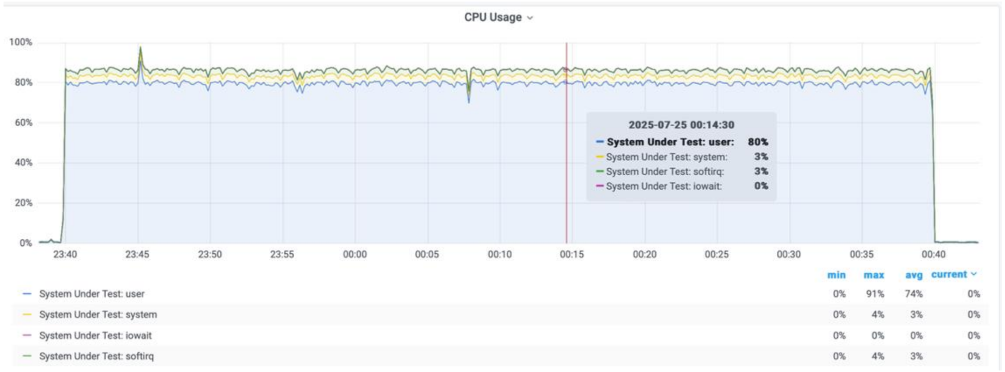


Active Users	Overall Throughput	Sent Bytes	Received Bytes	Errors Count	Request/Transactions...	Success Rate	Error Rate
300	Transactions 5.9K ops/s; Requests 5.9K ops/s	4 GB	4 GB	0	Tr 21444184; Req 21444184	100.0%	0.00%

Name	Mean	Max	Min
TPS	5.86K req/s	5.98K req/s	1.14K req/s
Virtual Users (right y-axis)	300	300	300

Name	Mean	Max	Min
01_req_/get-user	1.46K ops/s	1.50K ops/s	286 ops/s
02_req_/get-product	1.46K ops/s	1.50K ops/s	284 ops/s
03_req_/get-order	1.46K ops/s	1.50K ops/s	284 ops/s
04_req_/get-config	1.46K ops/s	1.50K ops/s	283 ops/s
Virtual Users (right y-axis)	300	300	300

Fig.10. Optimized backend listener throughput and other summary metrics for the 1 hour long Fixed-load test



	min	max	avg	current
System Under Test: user	0%	91%	74%	0%
System Under Test: system	0%	4%	3%	0%
System Under Test: iowait	0%	0%	0%	0%
System Under Test: softirq	0%	4%	3%	0%

Fig. 11. SUT CPU utilization during 1-hour long test

Capacity test. Finally, with an assumption that the Backend Listener is not yet at its maximum capacity, I prepared different test scenario, where the SUT was excluded from the workflow. Rather than sending requests to a real application, mock requests were generated using the JSR223 sampler in Jmeter. This approach enabled the generation of samples purely for the Backend Listener to process and send to InfluxDB, eliminating the SUT as a potential bottleneck and allowing for an isolated evaluation of the listener's absolute capacity. The best suited test type for these needs is the Capacity test. This is a type of performance testing that focuses on determining the maximum load a system can handle before performance degrades or fails. The capacity test load model was the following:

- Start with 1 request/sec(rps) and increase to 15000 rps in 30 seconds
- Hold 15000 rps for 1 minute
- Increase from 15000 rps to 20000 rps in 30 sec
- Hold 20000 rps for a minute
- And so on... until the saturation and instability points or zones are found.

As shown in Fig. 12, the saturation zone was identified in the range of 20000 to 25000 rps. In this zone, marked in orange, noticeable spikes in throughput were observed, but no errors were encountered, indicating that the Backend Listener still maintained functionality despite nearing its limits. However, during the next rate increment (from 25000 to 30000 rps), the system entered the instability zone. At this stage, the frequency of throughput spikes increased, and eventually, at approximately 29000 rps, throughput dropped drastically to around 3000 rps, accompanied by error messages such as: *"com.influxdb.exceptions.InfluxException: Could not emit buffer due to lack of requests*", which indicates a backpressure issue within the client-side library, specifically when using a reactive or asynchronous client. Specifically, it suggests that the InfluxDB client was attempting to flush a buffer of points but was unable to do so because the underlying mechanism for handling requests could not accept additional data, likely due to resource limits within the client or network.

The final test confirmed that the Backend Listener is capable of handling ~20000 rps load per load generator. This level of performance is exceptional and typically exceeds the requirements of most performance testing scenarios, as very few Systems Under Test (SUTs) can sustain such load levels. The Backend Listener's capacity, therefore, is more than sufficient for most high-load testing use cases.

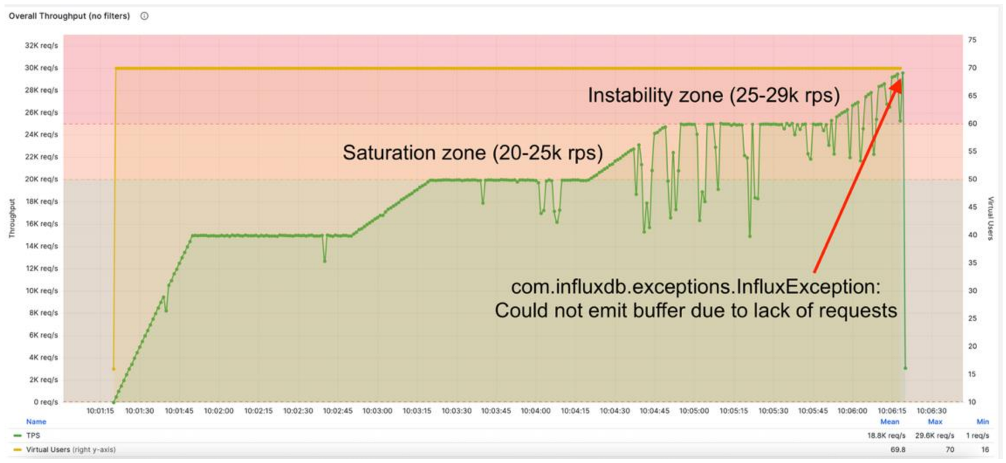


Fig.12. Capacity test with the Mock application to find the maximum rates the Backend listener could operate at

4. Results analysis and discussion. 4.1. Comparative Evaluation. The experiment results clearly validate the performance improvements in the optimized InfluxDB Backend Listener Plugin (Version 2) compared to Version 1 and the standard listener. The following comparisons serve to demonstrate:

a) Throughput:

– The optimized plugin increased throughput significantly, achieving ~7000 requests/sec over sustained periods during fixed-load tests compared to 480 requests/sec in Version 1. (Figure 6, 7, and 8).

– The standard Raw JMeter listener failed at 400 requests/sec, confirming its inability to scale for high-throughput scenarios

This **14.5x improvement** directly translates to the ability to conduct large-scale performance tests and generate realistic system pressure – an essential requirement for modern backend benchmarking.

b) Stability and Reliability:

– V.2 consistently maintained stable throughput under stress and capacity testing, processing up to 21.4 million metrics in 1 hour in fixed-load tests (Figure 10), and sustaining 20k requests/sec without errors under ideal conditions during capacity tests (Figure 12).

– This stability eliminates throughput dips observed in Version 1 (Figure 2), allowing steady generation of load and reducing performance bottlenecks in CI/CD workflows reliant on real-time metrics forwarding.

c) Resource Utilization:

– CPU utilization increased significantly: from 11% with Version 1 to 99% with Version 2 on the System Under Test (Figure 9). This indicates that the new listener removed previous limitations, allowing the load generator to fully exercise SUT capacity.

4.2. Implications for Practical Applications. The optimized Backend Listener Plugin (Version 2) offers significant advantages for performance engineering, particularly within modern CI/CD pipelines, large-scale systems, and real-time monitoring workflows:

a) Scalability and Load Generation:

– With stable throughput of 7000 requests/sec for real-applications tests and up to 20k requests/sec under ideal conditions, the plugin ensures realistic load simulations for demanding software architectures like microservices or distributed cloud systems.

– This improvement supports performance testing of systems designed to operate under high user concurrency and throughput scenarios.

b) Real-Time Monitoring:

– By processing and forwarding raw metrics to InfluxDB efficiently, the plugin enables real-time performance dashboards via Grafana, providing actionable insights during live testing.

– Reliable integration ensures seamless feedback loops in continuous testing workflows.

c) Performance Bottlenecks Addressed:

– The elimination of blocking behavior in the listener reduces the risk of underutilized resources or delayed tests, ensuring consistent system pressure during performance benchmarking.

4.3. Limitations of the Current Implementation. Although the optimized plugin performed well, certain limitations remain:

a) Client-Side Saturation Zone:

– During capacity testing, the plugin began to exhibit saturation beyond 20–25k requests/sec, with performance outage observed near 29k requests/sec due to client-side backpressure in the InfluxDB library. This exceeded typical real-world requirements but suggests further optimization potential.

b) Single Machine Testing:

– Testing was limited to single load generator instances. While results demonstrate sufficient capacity per instance, further scalability across distributed load generators could be validated to support ultra-large systems.

Future enhancements could focus on additional optimization strategies for handling extreme-scale workloads and minimizing client-side saturation issues.

4.4. Contribution to Performance Engineering

The improvements introduced in Version 2 of the Backend Listener reflect meaningful contributions to performance testing frameworks:

a) Efficiency in Metric Forwarding:

– The asynchronous batch-processing design optimizes network utilization, saving resources while scaling throughput.

b) Framework Stability:

– Extended-duration tests confirmed that the listener performs reliably under long-term load generation, addressing pain points often experienced in real-time reporting systems integrated into JMeter workflows.

c) Capacity Validation:

– The discovery of saturation zones (20k–25k rps) further emphasizes the practical reliability of the optimized plugin in supporting large-scale testing without exceeding typical technical limitations.

References:

1. *Leonov, S. and Tyrtyshnyi, D.* (2024), "Research of server side testing and framework development", *Control, Navigation and Communication Systems*, Vol. 1, No. 75, pp. 122–126. doi: https://doi.org/10.26906/SUNZ.2024.1.122

2. *Leonov, S. and Tyrtyshnyi, D.* (2025), "Development of a software platform for testing the performance of the client part of a web application", *Control, Navigation and Communication Systems*, Vol. 1, No. 79, pp. 111–115. doi: https://doi.org/10.26906/SUNZ.2025.1.111-115

3. *Murarka, S., Jain, A. and Singh, L.* (2024), "Advanced Techniques in Data Ingestion and Pipelining for Scalable Big Data Platforms: A Comprehensive Review", *2024 IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG)*, Indore, India, pp. 1–6. doi: https://doi.org/10.1109/ICTBIG64922.2024.10911053

4. *Pratama, M.R. and Kusumo, D.S.* (2021), "Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing", *2021 9th International Conference on Information and Communication Technology (ICoICT)*, Yogyakarta, Indonesia, pp. 230–235. doi: https://doi.org/10.1109/ICoICT52021.2021.9527496

5. *Yang, S.* (2025), "The Impact of Continuous Integration and Continuous Delivery on Software Development Efficiency", *Journal of Computer, Signal, and System Research*, Vol. 2, No. 3, pp. 59–68. doi: https://doi.org/10.71222/pzvfqm21

6. *Zhu, X., Nie, X. and Liu, J.* (2023), "Time Series Database Optimization Based on InfluxDB", *2023 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEC),* Athens, Greece, pp. 879–885. doi: https://doi.org/10.1109/PEEEC60561.2023.00172

7. *Ramdass, K., Mulka, A., Agarwal, N., Avvaru, S., Gupta, G.K. and Chauhan, S.S.* (2025), "Scalable Performance Testing for Distributed Big Data Frameworks", *2025 First International Conference on Advances in Computer Science, Electrical, Electronics, and Communication Technologies (CE2CT),* Bhimtal, Nainital, India, pp. 1360–1364. doi: https://doi.org/10.1109/CE2CT64011.2025.10939442

8. *Kuchuk, H. and Malokhvii, E.* (2024), "Integration of IoT with Cloud, Fog, and Edge Computing: A Review", *Advanced Information Systems*, Vol. 8, No. 2, pp. 65–78. doi: https://doi.org/10.20998/2522-9052.2024.2.08

9. *Alnuhait, H., Alzyadat, W., Althunibat, A., Kahtan, H., Zaqaibeh, B. and Al-Khawaja, H.A.* (2024), "Web application performance assessment: A study of responsiveness, throughput, and scalability", *International Journal of Advanced and Applied Sciences*, Vol. 11, No. 9, pp. 214 226. doi: https://doi.org/10.21833/ijaas.2024.09.023

10. *Kirešová, S., Guzan, M., Fecko, B., Somka, O., Rusyn, V. and Yatsiuk, R.* (2023), "Grafana as a Visualization Tool for Measurements", *2023 IEEE 5th International Conference on Modern Electrical and Energy System (MEES),* Kremenchuk, Ukraine, pp. 1–5. doi: https://doi.org/10.1109/MEES61502.2023.10402486

11. *Gupta, K. and Mathuria, M.* (2017), "Improving performance of web application approaches using connection pooling", *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA),* Coimbatore, India, pp. 355–358. doi: https://doi.org/10.1109/ICECA.2017.8212833

12. *Barbosa, K., Ferreira, R., Pinto, G., d'Amorim, M. and Miranda, B.* (2023), "Test Flakiness Across Programming Languages", *IEEE Transactions on Software Engineering*, Vol. 49, No. 4, pp. 2039–2052. doi: https://doi.org/10.1109/TSE.2022.3208864

13. *Tong, H., Jeon, S. and Nah, Y.* (2024), "Performance Analysis of Time Series Databases: A Comparison in Cloud and Physical Environments", *2024 International Conference on AI x Data and Knowledge Engineering (AIxDKE),* Tokyo, Japan, pp. 108–111. doi: https://doi.org/10.1109/AIxDKE63520.2024.00027

Статтю представив д-р техн. наук, проф. НТУ "ХПІ" Заковоротний О.Ю.

Received 19.12.2025
Стаття прийнята до друку 21.01.2026
Опублікована 27.02.2026

Sergey Leonov, Doctor of Technical Sciences, Professor.
National Technical University “Kharkiv Polytechnic Institute”
2, Kyrpychova str., Kharkiv, Ukraine, 61002
Tel.: +38 (099) 911-91-13, e-mail: serleomail@gmail.com
ORCID ID: 0000-0001-8139-0458

Dmytro Tyrtyshnyi, PhD student.
National Technical University “Kharkiv Polytechnic Institute”
2, Kyrpychova str., Kharkiv, Ukraine, 61002
Tel.: +38 (095) 459-33-95, e-mail: dmytro.tyrtyshnyi@gmail.com
ORCID ID: 0009-0000-4935-7156

УДК 004.732.056

Оптимізація продуктивності отримання даних у реальному часі: покращення InfluxDb backend listener для Jmeter / Тиртишний Д.А, Леонов С.Ю. // Вісник НТУ "ХПІ". Серія: Інформатика та моделювання. – Харків: НТУ "ХПІ". – 2026. – № 1 (15). – С. 43 – 67.

У роботі розглядається проблема неефективності обробки інформації та отримання даних у режимі реального часу під час тестування продуктивності з високим навантаженням за допомогою JMeter та InfluxDB. Об'єктом дослідження є процес пересилання даних до баз даних часових рядів. Предметом дослідження є методи оптимізації програмних засобів та реалізації плагіна InfluxDB Backend Listener. Метою роботи є усунення недоліків у продуктивності плагіна та значне підвищення пропускної здатності та використання ресурсів комп'ютерних систем. Розроблено оптимізовану реалізацію, яка включає такі методи, як асинхронні операції запису та пакетна обробка даних, що дозволило уникнути блокування потоків виконання тестів. Експериментальні дослідження показали значне покращення пропускної здатності у 14,5 разів – з 480 запитів/с (з піковими падіннями) до стабільних 7000 запитів/с. Тестування ємності підтвердило здатність обробляти до 20 000 запитів/с за оптимальних умов. Результати підтверджують ефективність запропонованого рішення для використання у масштабованих розподілених системах та CI/CD процесах. Іл.: 12. Табл.: 1. Бібліогр.: 13 назв.

Ключові слова: тестування продуктивності; InfluxDB; отримання даних у реальному часі; оптимізація пропускної здатності; бази даних часових рядів; JMeter; асинхронний запис; програмні засоби; комп'ютерні системи; розподілені системи; обробка інформації.

UDC 004.732.056

Performance Optimisation of real-time data ingestion: Enhancing the InfluxDB backend listener for Jmeter / Tyrtyshnyi D., Leonov S. // Herald of the National Technical University "KhPI". Series of "Informatics and Modeling". – Kharkiv: NTU "KhPI". – 2026. – № 1 (15). – P. 43 – 67.

The object of the study is the process of information processing and real-time data ingestion during performance testing in load testing tools like JMeter. **The subject of the study** is the methods of optimizing software tools implementation for data forwarding to time-series databases, specifically for the InfluxDB Backend Listener Plugin used in JMeter. The purpose of the paper is to resolve inefficiencies in the plugin's performance and significantly enhance throughput and resource utilization of computer systems during high-load testing scenarios. **The results obtained.** An optimized implementation of the InfluxDB Backend Listener Plugin was developed, addressing bottlenecks in the software code that led to underutilization of system resources during tests. The enhanced version incorporates techniques such as asynchronous data write operations and batching, enabling efficient utilization of computing resources. Performance benchmarking demonstrated a remarkable 14.5x improvement in throughput, increasing from 480 requests/sec with spiky pattern to stable 7000 requests/sec. Advanced capacity testing further validated its ability to handle up to 20000 requests/sec under optimal conditions. **Conclusions.** Experiments confirmed the efficiency of the proposed solution. Code-level optimizations successfully resolved the inefficiencies in real-time metric forwarding and allowed the plugin to operate at maximum capacity. These improvements enable large-scale performance validation by generating sufficient load for backend systems while reliably forwarding real-time metrics for analysis in CI/CD workflows. The new implementation establishes the InfluxDB Backend Listener as a scalable, efficient, and reliable component of modern distributed systems and performance testing frameworks. Fig.: 12. Tabl.: 1. Refs.: 13 items.

Keywords: performance testing; InfluxDB; real-time data ingestion; throughput optimization; time-series database; JMeter; asynchronous write operations; batching; software tools; computer systems; distributed systems; information processing.