Towards Bringing Vitruvius into the Cloud II: Against Attacks from the Internet

Martin Armbruster
martin.armbruster@kit.edu
Karlsruhe Institute of Technology, Karlsruhe, Germany

Fatma Chebbi
fatma.chebbi@kit.edu
Karlsruhe Institute of Technology, Karlsruhe, Germany

Thomas Weber
thomas.weber@kit.edu
Karlsruhe Institute of Technology, Karlsruhe, Germany

Anne Koziolek
koziolek@kit.edu
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

Consistency management for models is a field of active research. While performance is usually of limited interest, it is a necessity for the real-world applicability of such tools. Last year, we introduced a client-server architecture for our consistency management tool Vitruvius, focusing on transferring the concepts from monolithic to client-server. The contribution of this paper is the extension of this client-server architecture with security mechanisms, concretely, authentication, authorization, and encryption, and a first look on the influence on the performance of the tool. We extended the existing implementation and took performance measurements of the monolithic, the client-server and the secured client-server architectures.

1 Introduction

Tools for model management in general and for consistency between these models are an area of active research, such as the Vitruvius tool [5]. We presented an extension of the Vitruvius tool, migrating from a monolithic architecture to a server-client architecture, enabling a more diverse set of use cases [8]. However, this extension was limited by not considering security, e.g., the lack of user authentication and authorization. As part of the *CONVIDE* research project [7], we plan to use Vitruvius as a research platform, which motivates our need for secured access and usage control to Vitruvius instances when used with industrial or industry-inspired data. Similarly to the application of Vitruvius in practice for the development of cyber-physical systems [7], planned for the future of Vitruvius, this necessitates security considerations. To mitigate this issue, we extended the Vitruvius server-client architecture by integrating transport security with TLS, different HTTP protocols, OAuth, and OpenID Connect using the Eclipse Jetty server and client implementations[1]. While the integration of these mechanisms is necessary for the application of Vitruvius, they might have an impact on the performance, which we want to investigate in this paper. Therefore, in order to assess the performance of these additions, we carried out several performance measurements, comparing the monolithic architecture, the client-server architecture, and the secured client-server architecture.

2 Background and Related Work

Vitruvius is an approach for the view-based development of consistent systems [5]. It focuses on two main aspects: the management of consistency between overlapping parts of different models, and the definition and use of views, which enable developers to interact with custom views on the underlying models, tailored for specific use cases. The information reduction employed through defining and using views helps developers cope with the complexity of the models. Views are modified by developers interacting with Vitruvius, and the resulting changes are propagated back into Vitruvius, which connects the different models and keeps them consistent. Therefore, we exposed the use of views (checkout, changing, committing changes) over a REST interface in Vitruvius Remote [8].

Related Work Related approaches for consistency preserving system development, e.g., DesignSpace [3] or Comprehensive Systems [6] focus on the theoretical foundations and less on the practical application, so they are missing security considerations.

3 Vitruvius Security Server

To secure the Vitruvius Remote architecture, we employ three primary mechanisms on top of it. First, the communication is secured through HTTPS, leveraging TLS for encryption, integrity checks, and certificates

[1] `https://jetty.org/`, accessed 26.08.2025

Configuration Name	Number of Families	Number of Members per Family	Repetitions
Small	10	5	1000
Medium	100	10	500
Large	300	20	100

Table 1: Characterization of the evaluation configurations [8]. The different number of repetitions is due to the increasing runtime.

to maintain confidentiality, integrity, and authenticity. Second, identity and access management is realized with the OpenID Connect[2] protocol, enabling secure authentication and token-based access control. Third, server logging is introduced to support non-repudiation and auditing.

We realized these mechanisms in a Vitruvius Security Server based on Eclipse Jetty, which acts as an API gateway to authenticate clients, enforce authorization, and handle requests. It supports all HTTP versions (HTTP/1.1, HTTP/2, and experimentally HTTP/3). The actual Vitruvius requests can be directly handled by the Vitruvius Security Server (*direct operation mode*) or be forwarded to an internal, unsecured Vitruvius Server (*proxy operation mode*).

4 Performance Evaluation

The evaluation's goal is to investigate whether the Vitruvius Security Server adds overhead compared to the monolithic Vitruvius and Vitruvius Remote.

4.1 Setup

To ensure comparability of our results with those from last year, we planned a setup similar to [8]. Thus, we reuse the same V-SUM consisting of two metamodels: family and persons. For the family model, we generate a model and record these changes, which are propagated to the persons model. In the evaluation, we measure the end-to-end execution time of the change propagation. Furthermore, we consider three model sizes for our measurements, depicted in Table 1.

We conducted two experiments. Experiment 1 runs the monolithic Vitruvius, Vitruvius Remote, and the Vitruvius Security Server on one machine. Experiment 2 runs Vitruvius Remote and the Vitruvius Security Server, where the server part is one machine and the client part on another one. As the factors *HTTP version* and *operation mode* of the Vitruvius Security Server can be varied, we selected a subset of the possible settings that does not surpass the scope of this paper. Accordingly, we consider the Vitruvius Security Server in the proxy mode with HTTP/1.1 and HTTP/2 (only for experiment 2).

For the actual measurements, we used two machines S_1 and S_2. S_1 is a Dell Precision 5820 Tower with an Intel Xeon W-2245 CPU (8 cores, 2 threads per core), 125 GB RAM, and Arch Linux. S_2 is a tower PC with an Intel Core i9-14900KF CPU (24 cores, 2 threads per core), 62 GB RAM, and Arch Linux. On both machines, we ran the Vitruvius server and client programs within Docker containers and limited the CPU core count to 4 and RAM to 16 GiB to avoid inferences with other applications. Both machines were located in the same building, but different rooms, connected in the same virtual LAN over Ethernet.

[2]`https://openid.net/specs/openid-connect-core-1_0.html`, accessed 26.08.2025

4.2 Analysis Process

To statistically analyze the measurements, we calculate their average execution time and corresponding standard deviation. To check if differences in averages $\hat{\theta}$ are statistically significant, we perform pairwise significance tests within each configuration and experiment. Here, the null hypothesis is no difference in the averages ($\theta = 0$). As we cannot assume normality, equal variances, or equal shapes for the measurements, we apply the bootstrap-based approach proposed by Johnston and Faulkner [4]. According to this process, for two measurement series m_1 and m_2, N random resamples with replacement ($m^*_{1,i}$ and $m^*_{2,i}$, $1 \leq i \leq N$) are taken for each measurement series. Then, the resamples are turned into a null distribution $\hat{\delta}_i = \left| mean(m^*_{1,i}) - mean(m^*_{2,i}) - \hat{\theta} \right|$. Finally, the $\hat{p}$ value, which expresses the probability to observe a $\hat{\delta}_i$ at least as extreme as $\hat{\theta}$, is estimated by

$$\hat{p} = \frac{\sum_{n=1}^{N} I(\hat{\delta}_i \geq \left|\hat{\theta}\right|) + 1}{N + 1},$$

where $I(\cdot)$ is the indicator function. For $\hat{p}$, a 95 % confidence interval (CI) can be calculated, too.

We set $N = 10,000$. In addition, different from the proposed approach, we check if a measurement series is independent and identically distributed (iid) before we resample it, since iid measurements are required for the resampling [4]. Thus, we check if the estimated autocorrelation coefficients fall within the 95 % CI of $\pm 2 * \sqrt{n}$, n being the number of measurements [2]. If this is not the case, we assume a time dependency in the measurements and employ the stationary bootstrap for resampling [1]. The stationary bootstrap resamples blocks of measurements to retain the dependency structure. At last, we choose $\alpha^* = 0.01$ as significance level and applied a Bonferroni correction to $\alpha = 0.00037037$ based on 27 comparisons.

4.3 Results

Table 2 and Table 3 show the average execution times in experiment 1 and 2, respectively. In Table 4, the calculated $\hat{p}$ values for the significance tests are depicted. There are only three cases, in which the values differ from all other values. Only Vitruvius Remote in the large configuration in experiment 1 can be assumed as iid.

Config	Monolithic VITRUVIUS	VITRUVIUS Remote	VITRUVIUS Security Server (HTTP/1.1)
Small	11.8 ms (sd = 2.0 ms)	80.7 ms (sd = 8.9 ms)	39.4 (sd = 10.3 ms)
Medium	849.7 ms (sd = 38.1 ms)	1,054.6 ms (sd = 38.4 ms)	1,002.8 ms (sd = 38.3 ms)
Large	20.3 s (sd = 685.5 ms)	21.5 s (sd = 113.3 ms)	21.4 s (sd = 150.1 ms)

Table 2: Results for experiment 1 on S_1: average execution times and standard deviation.

Config	VITRUVIUS Remote	VITRUVIUS Security Server (HTTP/1.1)	VITRUVIUS Security Server (HTTP/2)
Small-S_1	82.5 ms (sd = 6.2 ms)	45.2 ms (sd = 11.1 ms)	43.6 ms (sd = 4.9 ms)
Medium-S_1	918.2 ms (sd = 27.1 ms)	900.5 ms (sd = 40.0 ms)	969.3 ms (sd = 36.9 ms)
Large-S_1	19.8 s (sd = 660.4 ms)	19.5 s (sd = 613.1 ms)	20.7 s (sd = 253.5 ms)
Small-S_2	65.9 ms (sd = 2.9 ms)	30.4 ms (sd = 7.9 ms)	32.6 ms (sd = 5.1 ms)
Medium-S_2	526.9 ms (sd = 8.5 ms)	467.0 ms (sd = 9.4 ms)	487.8 ms (sd = 10.0 ms)
Large-S_2	9.3 s (sd = 51.6 ms)	9.0 s (sd = 50.6 ms)	9.2 s (sd = 90.4 ms)

Table 3: Results for experiment 2: average execution times and standard deviation. The machine after the configuration name operated as server, while the other machine acted as client.

4.4 Discussion

Considering the $\hat{p}$ values in Table 4, we can reject the null hypothesis with $\alpha = 0.00037037$ in almost all cases and can assume a statistically significant difference in the averages, except for the three cases with a higher $\hat{p}$ value. However, it should be noted that the upper limit of the 95 % CI (0.000566) is greater than α.

In the results of experiment 1, we can observe that the client / server architectures increase the average execution times, in particular relatively for the small model size. In experiments 1 and 2, the average execution time of the VITRUVIUS Security Server with HTTP/1.1 is (except for two cases) statistically significantly lower than the average execution time of VITRUVIUS Remote, suggesting a slight improvement in performance. A potential factor can be the employed client / server implementations. While the VITRUVIUS Security Server uses Eclipse Jetty, VITRUVIUS Remote builds upon the Java-internal HTTP Server [8]. Nevertheless, the insignificant differences occurred in the large configuration, which opens the question if larger models can outweigh the overhead of the client / server communication. For HTTP/2 in experiment 2, the average execution time is in five of six cases above the one of HTTP/1.1 and in two of six cases above the one of VITRUVIUS Remote (one of the four remaining cases does not differ statistically significantly). This implies that HTTP/2 was slower than HTTP/1.1. There is no clear indication how HTTP/2 compares to VITRUVIUS Remote. For overall more insights, further measurements are necessary in future work.

Comparison	Config	$\hat{p}$	95 % CI
Nearly all	All	0.0001	[0.0000051, 0.000566]
VITRUVIUS Remote - Security Server (HTTP/1.1) (Exp. 1)	Large	0.0019998	[0.001295, 0.003087]
VITRUVIUS Remote - Security Server (HTTP/1.1) (Exp. 2)	Large-S_1	0.28167	[0.27294, 0.29057]
VITRUVIUS Security Server HTTP/1.1 - HTTP/2 (Exp. 2)	Small-S_1	0.05379	[0.04954, 0.05839]

Table 4: Results for the calculated $\hat{p}$ values for the significance tests.

5 Conclusion

In this paper, we presented an extension of VITRUVIUS Remote [8], which includes the integration of transport security, authentication, and authorization. For the future of the VITRUVIUS server-client architecture, we plan to improve its general usability by conducting field tests with domain experts, as well as further performance analysis and security improvements.

All data in this paper are available online.[3]

Acknowledgements

This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF), supported by KASTEL Security Research Labs, Karlsruhe, and supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263.

References

[1] D. N. Politis and J. P. Romano. “The Stationary Bootstrap”. In: *Journal of the American Statistical Association* 89.428 (1994), pp. 1303–1313.

[2] G. Kirchgässner, J. Wolters, and U. Hassler. “Introduction and Basics”. In: *Introduction to Modern Time Series Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–25.

[3] A. Demuth et al. “Designspace: an infrastructure for multi-user/multi-tool engineering”. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 1486–1491.

[4] M. G. Johnston and C. Faulkner. “A bootstrap approach is a superior statistical method for the comparison of non-normal data with differing variances”. In: *The New Phytologist* 230.1 (2021), pp. 23–26.

[5] H. Klare et al. “Enabling consistency in view-based system development—the vitruvius approach”. In: *Journal of Systems and Software* 171 (2021), p. 110815.

[6] P. Stünkel et al. “Comprehensive Systems: A formal foundation for Multi-Model Consistency Management”. In: *Formal Aspects of Computing* 33.5 (2021), pp. 1067–1114.

[7] R. Reussner et al. “Consistency in the view-based development of cyber-physical systems (convide)”. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2023, pp. 83–84.

[8] M. Armbruster, T. Weber, and L. König. “Towards Bringing Vitruvius into the Cloud”. In: *Softwaretechnik-Trends Band 44, Heft 4*. Gesellschaft für Informatik e.V., 2024.

[3] `https://doi.org/10.5281/zenodo.17311676`