INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2024, VOL. 70, NO. 1, PP. 161-167
Manuscript received August 1, 2023; revised March, 2024. doi: 10.24425/ijet.2024.149526

Comparison of openEHR open-source servers

Jacek Kryszyn, Waldemar T. Smolik, Damian Wanta, Przemysław Wróblewski, and Mateusz Midura

***Abstract*—Medical information systems could benefit from electronic health records management using openEHR. On the other hand, such a standard adds an additional software layer to the system, which might impact performance. In this article, we present an in-depth comparison of open-source openEHR servers and propose tools for testing them. Load tests for selected open-source servers were prepared using Apache JMeter. Statistics of elapsed time of requests and throughput of each solution were calculated. Results show that open-source openEHR servers significantly differ in performance and stability and prove that load testing should be a crucial part of a development process.**

***Keywords*—medical information systems; electronic health record; openEHR**

I. Introduction

THE openEHR [1] consists of specifications, clinical models, and software, which together create a healthcare technology allowing the management of electronic health records (EHRs). Using multi-level modeling, data representation is separated from domain content, significantly improving the ability to design and process patient-related data. The basic unit of data describing medical terms is known as an archetype, and archetypes can be combined into so-called templates. This approach allows to model virtually any medical-related event or examination, which can be stored securely in a repository of EHRs. openEHR guarantees interoperability. The standard has already been applied to many existing medical information systems [2]-[6], as well as been studied in many research works which show that openEHR is suitable for various applications in medicine [7]-[17].

In our recent work, we presented a performance comparison of custom and openEHR-based solutions for medical information systems [18]. As the name suggests, openEHR is an open solution, but unfortunately, there are currently only two supported open-source openEHR servers that allow incorporating this standard into medical information systems: EHRServer [19] and EHRbase [20]. An organization responsible for maintaining the standard also indicates Ethercis [21], but this solution seems unsupported for several years. In the previous work, we selected EHRServer as an EHR repository. We also provided a simple comparison between this software and EHRbase in which we showed the superiority of EHRServer in terms of performance. Those tests simulated the usage of both servers by one user. After a more thorough analysis, we came to the conclusion that this approach was flawed. Because of that, we have developed more comprehensive load tests of both servers to perform an objective and complex comparison presented in this paper. Apache JMeter load tests were prepared for both solutions in order to test the most basic operations (creating and fetching EHRs, creating and fetching compositions). Moreover, we made the test framework created in this work available open-source to allow others to perform their own studies, which might be crucial in developing new medical information systems based on openEHR.

II. Methods

EHRServer and EHRbase are both open-source web applications that allow the storage of openEHR data. Both provide REST APIs that enable communication with them using HTTP protocol and JSON/XML data. EHRbase is only a repository with no graphical user interface (GUI), whereas EHRServer provides GUI for browsing and managing data, so it might be used as a standalone solution. EHRbase is pure Java, whereas EHRServer is developed using Grails 3.3.10. Described systems differ in a database system used for data storage. EHRbase uses PostgreSQL, whereas EHRServer uses MySQL. In this study, we used EHRbase version 0.24.0 (commit c1a6db20, 22 Feb 2023) running on Java 17.0.6 with PostgreSQL 13.10 and EHRServer version 2.3 (commit 1215f58, 7 Dec 2022) running on Java 1.8.0_351 with MySQL 5.7.40.

Both systems can be run in Docker, which simplifies configuration and deployment. Both are configured so that the web application and the database run in separate containers that communicate through a network. In this study, we decided to test each server in different configurations (Figure 1) to test if the selected architecture might affect the servers' performance.

In addition to running both applications in containers, we decided to run each app locally in two scenarios: the web application and the database running on the same host machine (Figure 1 b) and the web application and the database on separate machines (Figure 1 c). EHRServer can be run in different environments: development, test, and production. The containerized version, by default, runs in production. Because of that the local version was always run using grails prod run-app command which ensures the production environment.

This work was financed thanks to an internal grant for employees of the Warsaw University of Technology supporting scientific activity in the discipline of Biomedical Engineering [grant number 504/04763/1034/43.052202].

All Authors are with Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Radioelectronics and Multimedia Technology (e-mail: damian.wanta@pw.edu.pl).

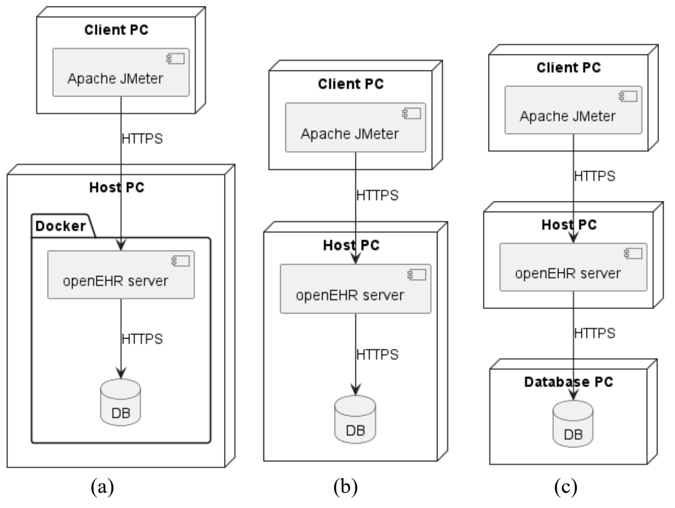

Fig. 1. Different architectures tested (a) an openEHR server and a database running in a Docker container on a host machine (b) an openEHR server and a database running locally on a host machine (c) an openEHR server and a database running locally, each on separate machine.

To test performance, we prepared Apache JMeter load tests of four operations:

1. creating EHRs
   - PUT /ehr/${ehr_uuid} in EHRbase,
   - POST /ehrs in EHRServer,
2. fetching EHRs
   - GET /ehr/${ehr_uuid} in EHRbase;
   - GET /ehrs/${ehr_uuid} in EHRServer,
3. creating compositions
   - POST /ehr/${ehr_uuid}/composition in EHRbase,
   - POST /ehrs/${ehr_uuid}/compositions in EHRServer,
4. fetching compositions
   - GET /ehr/${ehr_uuid}/composition/${composition_uuid} in EHRbase,
   - GET /ehrs/${{her_uuid}/${composition_uuid} in EHRServer.

In EHRbase tests, random UUIDs were generated by Apache JMeter, whereas in EHRServer tests, UUIDs were generated randomly by the server. Basic auth was used in EHRbase, whereas token authorization was used in EHRServer. This means a token valid for 24 hours had to be issued before tests. This token was later added to the Authorization header in the HTTP request. Apache JMeter 5.5 in CLI (NON-GUI) mode was used for load tests. Batch scripts were prepared to automate tests as much as possible. Each test lasted 3 minutes, but only the last 60 seconds of the tests were analyzed. This was done because we wanted to obtain as stable conditions as possible and omit the phase in which Apache JMeter starts consecutive threads that send requests. Each test was repeated for the different number of concurrent users: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 25, 30, 35, 40, 45, 50. The maximum number of users was selected according to the settings of servers which by default allow 50 connections.

The following scenario was executed: first, N concurrent users create EHRs for 3 minutes. Identifiers of these EHRs are saved in a CSV file later used in the test, which fetches EHRs for 3 minutes. Next, ten compositions for each EHR are created in the third test, their identifiers are saved in a CSV file, and those compositions are fetched in the last test.

Compositions were created using an ePrescription (FHIR) template downloaded from Clinical Knowledge Manager, an openEHR repository of archetypes and templates [22]. This template is an openEHR Medication order. An instance (composition) of this template was generated using openEHR-OPT tool [23], which was later randomized in each request sent by Apache JMeter.

The host and database machines had the same parameters: Intel Core i7-11700, 128 GB RAM, SSD NVMe SK Hynix 512 GB, Windows 11 Pro 22H2. The client machine used in the experiments had the following specifications: Intel Core i7-9700K, 32 GB RAM, SSD ADATA SX8200PNP, Windows 10 Pro 22H2. We also modified Java parameters available to JVM regarding RAM to ensure as many resources as possible: -XX:InitialRAMPercentage=50.0 (% of available initial RAM), -XX:MinRAMPercentage=50.0 (% of minimum available RAM), -XX:MaxRAMPercentage=80.0 (% of maximum available RAM). All machines were in the same local area network with 1 Gpbs speed and ping below 1 ms.

Data processing was done in MATLAB and was very similar to our previous work [18]. Elapsed times of each request (total time of the HTTP request and HTTP response) measured by Apache JMeter were used to calculate: minimum elapsed time (calculated using min function), maximum elapsed time (calculated using max function), mean elapsed time (calculated using mean function), standard deviation of the elapsed time (calculated using std function), 90-th (P90), 95-th (P95), 99-th (P99), 99.99-th (P99.99) percentiles (maximum time in which given percentage of requests was handled calculated using prctile function) and throughput (number of handled requests in one second).

The test framework used in this study and MATLAB scripts used to process the collected data are available publicly at [24].

III. Results

Table I and Table II show metrics of all tests made for EHRbase and EHRServer, respectively. For clarity, we decided to show data for 1, 2, 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 concurrent users in the main part of the article limited to mean elapsed time of requests, standard deviation of elapsed time of requests and throughput. All tests were performed many times in case of any temporary problems that could occur. Presented results are representative in this sense that all other instances of certain test gave similar results.

A. EHRbase – PUT /ehr endpoint

PUT /ehr is an endpoint used for creating EHRs in EHRbase. The throughput of each architecture stabilizes and doesn't increase for more than 20 users. This means that every new user results in a slower request processing time for all other users. The difference between each architecture is clearly visible – Docker being the slowest, whereas EHRbase running locally on one machine along the database being the fastest.

Assuming that the maximum throughput of EHRbase running in Docker is equal to about 157 requests per second and the maximum throughput of the server and database running locally on one machine is about 1280 requests per second, we can see that the difference in performance is equal to about eight times. The performance of a server running locally on one machine and a database running locally on another machine is somewhere in the middle – faster than Docker but visibly slower than both components running on the same machine.

B. EHRbase – GET /ehr

GET /ehr allows to fetch data of a certain EHR. Similar relationships between the throughput of each architecture, as in PUT /ehr tests, are visible. Local architecture with the server and the database running on one machine overperforms the other two architectures.

C. EHRbase – POST /composition

POST /composition is used in EHRbase to add a composition to a specific EHR. These tests result in a slightly different outcome than the previous. Still, a host as a server and a database running locally on one machine is faster, but this time EHRbase running in a container tuned out to be able to host more concurrent users than the third architecture. Actually, the third's throughput was higher for fewer users but then saturated

TABLE I
LOAD TESTS METRICS. EHRBASE INSTANCES.

	Docker			Server and DB locally on one machine			Server and DB locally on separate machines		
Users	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]
PUT /her									
1	62.05	1.59	15.60	5.25	1.12	170.60	25.83	2.08	38.02
2	62.26	1.32	31.63	4.97	1.24	357.25	25.53	3.26	77.23
5	69.98	2.24	69.17	6.66	1.84	575.03	16.83	1.40	290.25
10	88.35	4.44	112.18	8.06	3.31	1037.28	14.76	1.83	655.25
15	102.15	6.30	145.67	10.83	5.27	1287.03	20.87	5.43	701.83
20	127.75	22.51	155.20	14.74	8.85	1246.08	27.58	10.74	711.60
25	158.58	40.43	156.68	17.47	8.58	1337.98	34.55	14.73	712.50
30	192.90	57.64	154.93	21.59	11.00	1317.70	41.55	25.21	713.60
35	221.48	69.85	157.40	25.52	13.84	1300.50	48.72	24.69	711.12
40	252.60	81.79	157.75	30.30	18.92	1264.53	55.77	30.75	710.72
45	284.82	99.72	157.35	33.85	19.16	1285.20	62.71	34.97	712.65
50	316.75	112.04	157.68	37.88	20.72	1280.97	70.42	45.65	705.00
GET /ehr									
1	68.24	5.96	14.62	5.36	0.87	184.70	67.48	3.82	14.80
2	65.40	1.47	30.53	5.53	0.71	355.32	56.96	5.10	35.08
5	76.36	5.15	65.33	7.65	0.65	639.20	44.25	2.53	112.93
10	91.22	4.49	109.48	8.71	0.67	1138.08	41.40	2.15	241.43
15	102.92	7.22	145.55	11.29	1.87	1322.83	61.74	17.34	242.95
20	128.05	25.51	155.98	14.63	4.56	1357.30	81.02	35.61	246.73
25	157.78	42.27	158.48	18.17	7.43	1367.73	101.12	60.23	247.32
30	188.03	57.28	159.42	21.88	11.35	1364.88	121.58	67.28	246.65
35	219.52	85.00	159.32	26.69	15.30	1308.28	139.15	79.76	251.32
40	250.45	100.43	159.35	30.02	17.76	1329.93	159.14	148.80	251.23
45	281.78	117.42	159.55	33.75	21.14	1331.78	184.40	114.74	244.23
50	312.42	101.71	159.85	36.60	22.71	1363.03	208.16	134.92	240.08
POST /composition									
1	73.49	1.84	13.37	19.28	0.90	50.37	49.88	2.77	19.85
2	75.93	10.61	26.02	20.00	1.13	97.57	53.93	5.23	36.80
5	89.82	12.38	54.80	26.09	2.12	185.40	62.01	3.17	80.10
10	115.20	21.35	86.07	34.69	4.01	281.27	96.79	16.73	102.42
15	139.78	36.97	106.57	46.66	5.42	316.90	144.36	40.87	103.35
20	172.78	40.38	115.05	61.85	10.45	320.27	192.98	58.78	103.22
25	211.89	55.96	117.38	77.15	12.19	321.30	241.50	107.15	103.12
30	253.53	74.89	117.90	92.07	14.18	322.77	290.82	111.86	102.87
35	298.68	98.20	116.85	109.16	40.28	317.97	335.62	177.94	104.28
40	336.14	124.15	118.55	124.08	41.02	320.97	382.45	178.30	104.98
45	373.58	142.51	120.00	140.58	58.54	318.17	432.64	248.09	103.83
50	415.62	176.69	120.02	154.32	31.66	322.60	481.12	236.88	104.65
GET /composition									
1	69.00	1.25	14.47	10.35	0.99	96.03	63.20	3.36	15.80
2	72.45	5.65	27.57	10.43	0.94	190.48	52.19	3.05	38.32
5	79.79	3.04	62.57	15.44	1.21	322.92	55.34	2.49	90.27
10	99.18	4.57	100.67	30.90	4.17	323.12	69.00	2.86	144.88
15	111.79	6.50	134.02	46.37	44.06	323.38	101.31	21.46	148.00
20	137.86	22.20	144.98	61.82	85.48	323.22	135.19	40.22	147.93
25	170.45	41.08	146.60	77.17	124.90	324.68	168.68	58.56	148.28
30	203.93	60.53	146.98	92.74	152.18	329.27	200.65	75.27	149.45
35	237.03	75.31	148.32	107.64	183.29	324.13	234.19	94.51	149.47
40	272.73	91.72	147.13	123.02	203.94	326.08	268.04	117.30	149.23
45	308.96	116.42	145.37	138.92	241.98	323.40	304.53	136.76	148.05
50	352.90	133.12	142.97	154.36	258.69	326.45	338.49	151.66	148.25

TABLE II
LOAD TESTS METRICS. EHRSERVER INSTANCES.

	Docker			Server and DB locally on one machine			Server and DB locally on separate machines		
Users	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]	Mean elapsed time [ms]	Std of elapsed time [ms]	Throughput [requests/s]
POST /ehrs									
1	26.29	2.08	35.92	11.56	1.15	84.00	18.51	3.70	53.03
GET /ehrs									
1	19.90	1.36	49.93	12.32	1.12	80.90	15.37	2.56	64.97
2	20.13	2.26	98.70	13.23	0.93	150.77	17.67	2.53	113.05
5	23.40	2.90	212.77	18.28	2.25	273.25	18.85	3.04	264.93
10	28.91	4.01	344.67	26.30	3.52	379.73	21.18	3.92	471.57
15	33.65	5.24	443.62	32.44	3.48	461.93	23.57	5.24	635.67
20	41.73	6.87	477.47	40.29	11.01	495.95	26.58	6.15	751.67
25	50.64	8.96	492.27	49.31	20.27	506.68	30.56	5.57	817.50
30	59.59	11.16	502.13	58.84	26.84	509.50	34.94	9.72	858.03
35	68.11	12.96	512.90	68.20	33.06	513.00	39.58	10.91	883.85
40	77.42	16.14	515.77	78.61	44.89	508.67	61.98	57.22	645.07
45	86.45	18.55	520.38	101.68	94.63	446.72	49.39	15.09	910.68
50	95.40	21.77	523.57	281620.86	83983.98	0.23	55.79	71.29	895.83
POST /compositions									
1	11562.17	210.16	0.10	277.73	15.14	3.60	241.01	6.56	4.15
GET /compositions									
1	19.22	1.72	51.42	13.54	2.47	73.77	28.61	2.82	34.90
2	24.98	2.83	79.52	14.31	2.06	139.55	22.60	2.37	88.42
5	40.31	3.56	123.52	14.75	1.75	338.17	21.88	2.40	228.25
10	60.55	9.47	164.62	19.74	2.76	505.87	25.34	2.87	394.30
15	64.97	29.14	229.77	24.36	3.71	614.55	28.52	3.42	525.45
20	79.00	38.07	252.72	29.03	5.06	688.18	31.18	2.90	640.90
25	93.33	53.96	267.67	33.71	11.07	740.90	33.56	4.71	744.28
30	100.27	55.03	299.05	40.44	11.54	741.33	36.29	5.91	826.00
35	110.39	72.95	317.63	57.21	72.59	615.82	40.12	7.88	872.02
40	123.82	88.83	323.55	252759.35	92701.07	0.28	44.78	10.83	892.92
45	137.41	100.29	326.92	31550.14	732.90	0.23	575.29	11060.03	367.62
50	151.85	118.57	329.47	1086.70	21146.56	414.40	86.93	76.68	234.10

and achieved a lower value overall. The ratio of a Docker architecture's throughput and a system running on two machines locally for 50 concurrent users equals 1.15. The difference is not big but noticeable.

D. *EHRbase – GET /composition*

GET /composition is the last EHRbase endpoint tested. It allows to fetch a specific composition of a selected EHR. Results are similar to POST /composition except that this time the Docker version of the system and the version running locally on two separate machines achieved almost identical throughput, although for lower number of users EHRbase running in a container was able to process lower number of requests.

E. *EHRServer – POST /ehrs*

POST /ehrs is EHRServer's version of PUT /ehr in EHRbase. The same problem occurred as in our previous work [18] – no matter what kind of architecture was used, EHRServer was unable to process more than one concurrent user while creating resources – both EHRs and compositions. Trying to send requests by more than one user almost always resulted in errors with the following description: "Batch update returned unexpected row count from update [0]; actual row count: 0; expected: 1" which suggests that the system might have some problems with transactions and concurrent inserts to the database. Because of that we decided to modify the scenario. Only one thread was used to create EHRs and compositions whereas multiple threads were used for GET operations. Results are similar to EHRbase PUT /ehr endpoint – EHRServer running in a container is slower than EHRServer running locally. The fastest version is running both the server and the database on the same host machine. Discrepancies between architectures are smaller than in case of EHRbase – Docker version is 2.3 times slower than EHRServer running locally on the same machine as the database.

F. *EHRServer – GET /ehrs*

In the case of GET /ehrs there were no such errors as in the case of POST /ehrs, but other issues manifested. In case of 50 concurrent users sending requests to EHRServer running locally on the same host machine as the database, mean time achieved values bigger of the order of magnitude than in other cases. Figure 2 shows elapsed time of all requests sent in this test. The server virtually stopped responding after about 100 seconds of the test. This is why there are almost no requests in the last 60 seconds of the test, and their elapsed time is enormous. We repeated this test many times, and similar results were obtained, sometimes for fewer users. It is worth noting that the server running locally on a separate machine than the database had issues with dealing with 40 concurrent users. Figure 3 shows the problem. This time requests didn't stop to be processed, but their elapsed time was periodically much higher, especially in the last 60 seconds. This resulted in much lower throughput of the server. Tests were repeated many times in order to exclude some temporary issues with the network/host machine/any other reason. Such slowdowns for many users occurred every time. This time, the server running locally on a separate machine than the database was the fastest. This might be due to resources needed by both applications running on the same machine.

G. *EHRServer – POST /compositions*

The POST /compositions endpoint of EHRServer had the same issue as POST /ehrs – obtaining a test with no errors regarding many concurrent inserts was impossible. Because of that, only one user was used to test each architecture. Once again, the Docker version was much slower than EHRServer running locally. Interestingly, this time the version running separately from the database was a little bit faster, but overall all instances were not able to process more than a few requests per second.

H. *EHRServer – GET /compositions*

This endpoint allows to fetch a given composition of a certain EHR. Yet again, the same problems as in GET /ehrs tests occurred. For some tests, the server simply stopped responding in an acceptable amount of time (45 concurrent users, EHRServer running locally on the same machine as the database), whereas in some tests, there were large slowdowns in response time. Interestingly, EHRServer running locally on a different machine than the database had better throughput for a smaller number of users. In the end, both locally running deployments had big issues with handling requests for more than 35 users.

IV. Discussion

Results obtained for EHRbase running in a container for one user are consistent with what we presented in [18]. It is also true for EHRServer except POST /compositions endpoint, which was much slower this time. It seems that the performance of this endpoint depends on uploaded data. The template selected in this study (ePrescription (FHIR)) is much more sophisticated than the one used in the previous study. This suggests that EHRServer should be tested with many different templates because results obtained for one template might not be comparable to other templates. EHRServer was much more problematic overall. It seems that load testing was missing during the development of this software. It is possible that during this process, it was tested manually or by unit testing but with no load from more than one user. This means there is a need for testing presented in this paper and tools which we publish along this paper. They allow to profile the performance of a system and find errors that are impossible to observe on a test deployment by a single user.

Interestingly, the best stability was achieved using Docker and running the system as containers. This suggests that those containers have some specific configuration that should be done when running the system locally. Moreover, local deployment using separate machines performed better than the local deployment of the server and the database on one machine except POST /ehrs. This means that other operations have to be so consuming that the resources of the host machine (processor/RAM) are insufficient for both components working on one machine.

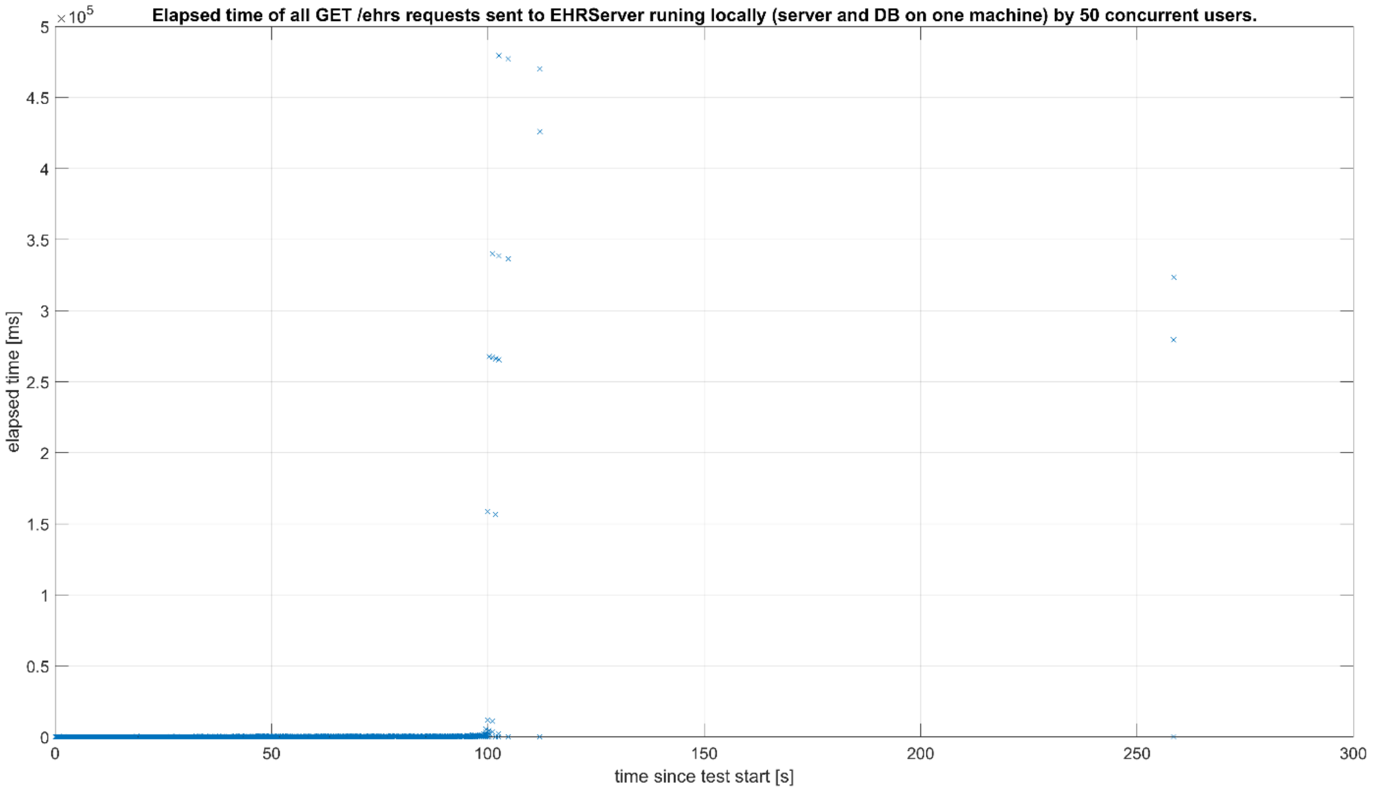

Fig. 2. Elapsed time of all requests sent by 50 users to GET /ehrs – EHRServer running locally on the same machine as the database.

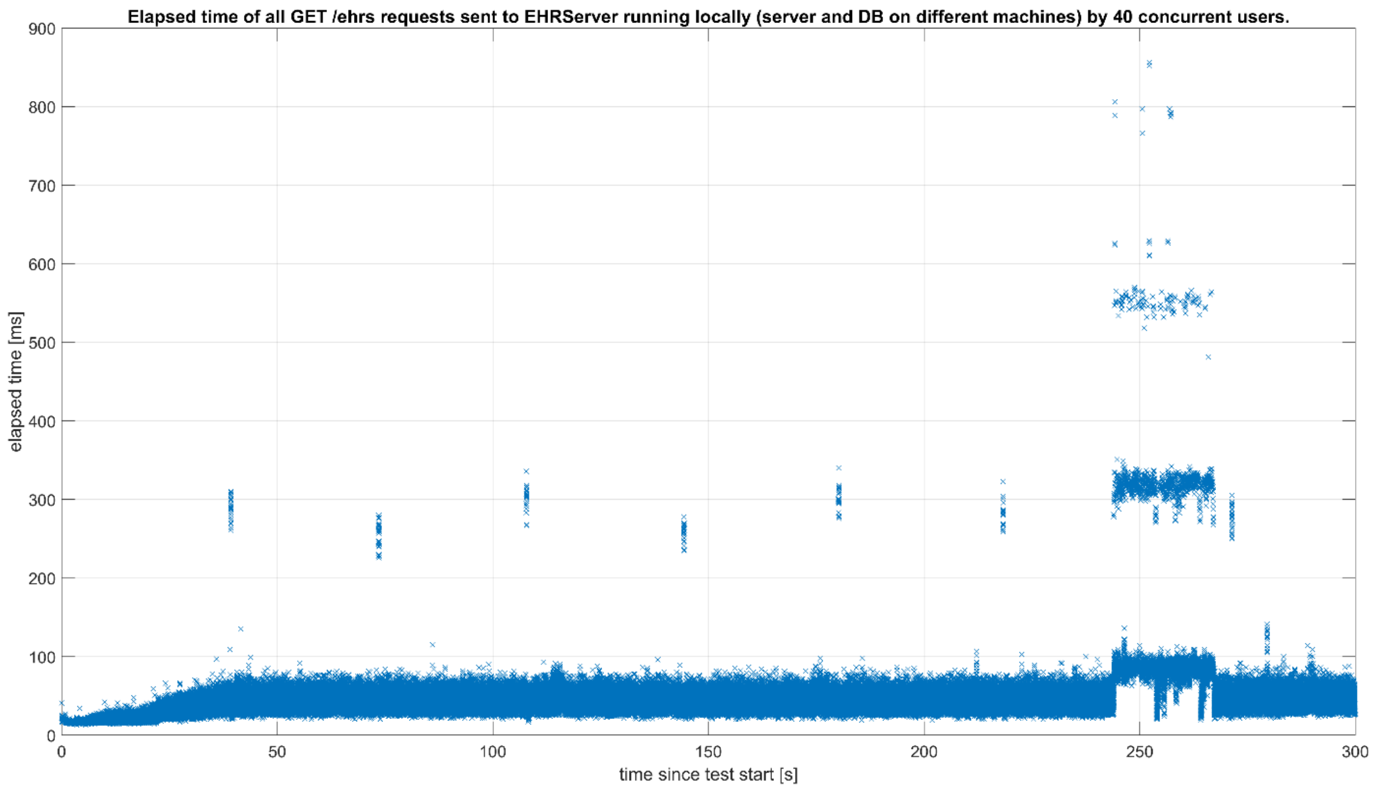

Fig. 3. Elapsed time of all requests sent by 40 users to GET /ehrs – EHRServer running locally on different machine than the database.

In the case of EHRbase, we can see that the best performance is achieved when the server and database run locally on the same machine. This means that the need to communicate between two machines over the network adds latency and increases the processing time of requests. In the case of the other two architectures, there is a major difference in the processing time of endpoints related to EHR (in favor of locally running server and database on two separate machines), but in the case of endpoints related to compositions, this difference is small. In the end, the throughput of GET /composition is identical, whereas the throughput of POST /composition is even higher for the Docker version of the system. This suggests that there has to be a common bottleneck for both architectures – probably the speed of connection between machines hosting the server and the database.

Table III shows the throughput achieved for 50 concurrent users (or less in certain cases of EHRServer) for each endpoint of EHRbase and EHRServer, respectively. The ratio of throughput of each instance is also calculated. This allows to notice some interesting relations. There is a big difference in the performance of each instance of EHRbase regarding endpoints related to EHRs, but the difference is smaller in the case of composition endpoints. This shows that the processing of compositions must be much more complicated, and the advantage of a localhost connection between the database and the server is less important. In the case of EHRServer, we selected the best throughput before errors started to occur in each test. Of course, this means throughput for only one user in the case of POST endpoints. Because of this, EHRbase seems to be faster overall for POST operations since it can serve more than one user, and throughput usually saturates after 15 concurrent users. Due to problems with stability, it is hard to make a comparison between both servers. When EHRServer works, it seems to work faster. Nonetheless, speed is overshadowed by problems with stability.

V. Conclusion

In this work, we presented a detailed comparison of the performance of two open-source openEHR servers: EHRbase and EHRServer. These are two constantly maintained solutions referenced by openEHR Foundation. Load tests (sending requests by many concurrent users for a certain amount of time) showed that EHRServer has problems with stability: it is not able to correctly process POST requests from many concurrent users at all and has a problem with handling larger traffic for GET requests. EHRbase is clearly more stable and didn't result in any errors in all performed tests. It is hard to compare the performance of both systems due to errors during EHRServer. EHRbase processed requests from one user slower than EHRServer, but due to ability to serve more users without errors its throughput of POST operations was usually higher. On the other hand, EHRServer showed better performance of fetching compositions.

We tested servers deployed in different ways: using Docker, locally along the database, and locally on a separate host machine than the database machine. It is not surprising, but it turned out that each architecture is characterized by different performance. Putting a system in a container degrades its performance, so some tweaking of a container parameters should also be done. In the case of EHRbase, the best performance was achievable with the server and the database running locally on the same machine, which mitigated the latency of a network connection between software. EHRServer was rather faster when deployed on a separate machine than the

TABLE III
THROUGHPUT AND RATIO OF THROUGHPUTS FOR 50 CONCURRENT USERS (OR LESS IF GIVEN IN BRACKETS) OF TESTED EHRBASE AND EHRSERVER INSTANCES.

	Docker (A)	Server and DB on one machine locally (B)	Server and DB on separate machines locally (C)	B/A	B/C	C/A
			EHRbase			
PUT /ehr	157.68	1280.97	705.00	8.12	1.82	4.47
GET /ehr	159.85	1363.03	240.08	8.53	5.68	1.50
POST /composition	120.02	322.60	104.65	2.69	3.0827	0.87
GET /composition	142.97	258.69	148.25	1.81	1.75	1.0369
			EHRServer			
POST /ehrs	35.92 (1 user)	84.00 (user)	53.03 (1 user)	2.34	1.58	1.4763
GET /ehrs	523.53	513.00 (35 users)	910.68 (45 users)	0.98	0.56	1.74
POST /compositions	0.10 (1 user)	3.60 (1 user)	4.15 (1 user)	36.00	0.8675	41.50
GET /compositions	329.47	741.33 (30 users)	892.92 (40 users)	2.25	0.8302	2.71

database. These results are important because they show how specific conditions can dramatically affect the results or even the correct functioning of the system. This should be taken into account when developing software. Load testing should be the standard when validating web applications because it exposes problems that the developer is unable to notice when working locally with a small set of data.

During this work we developed tools for load testing of openEHR servers. They are published online as open-source and available for everyone. We hope that these tools will contribute to the development of openEHR servers and medical information systems.

REFERENCES

[1] "openEHR Home." https://www.openehr.org/ (accessed Dec. 07, 2022).

[2] I. D. Păun *et al.*, "Local EHR management based on openEHR and EN13606," J Med Syst, vol. 35, no. 4, pp. 585–590, Aug. 2011, doi:10.1007/s10916-009-9395-1

[3] B. Christensen and G. Ellingsen, "Evaluating Model-Driven Development for large-scale EHRs through the openEHR approach," *Int J Med Inform*, vol. 89, pp. 43–54, May 2016, doi:10.1016/j.ijmedinf.2016.02.004

[4] L. Min, Q. Tian, X. Lu, and H. Duan, "Modeling EHR with the openEHR approach: An exploratory study in China Philip Payne," BMC Med Inform Decis Mak, vol. 18, no. 1, pp. 1–15, Aug. 2018, doi:10.1186/s12911-018-0650-6

[5] F. Hak, D. Oliveira, N. Abreu, P. Leuschner, A. Abelha, and M. Santos, "An OpenEHR Adoption in a Portuguese Healthcare Facility," in Procedia Computer Science, Elsevier B.V., Jan. 2020, pp. 1047–1052. doi:10.1016/j.procs.2020.03.075

[6] G. M. Bacelar-Silva, H. César, P. Braga, and R. Guimarães, "OpenEHR-based pervasive health information system for primary care: First Brazilian experience for public care," Proceedings of CBMS 2013 - 26th IEEE International Symposium on Computer-Based Medical Systems, pp. 572–573, 2013, doi:10.1109/CBMS.2013.6627881

[7] J. Buck, S. Garde, C. D. Kohl, and P. Knaup-Gregori, "Towards a comprehensive electronic patient record to support an innovative individual care concept for premature infants using the openEHR approach," Int J Med Inform, vol. 78, no. 8, pp. 521–531, Aug. 2009, doi:10.1016/j.ijmedinf.2009.03.001

[8] C. Pahl et al., "Role of OpenEHR as an open source solution for the regional modelling of patient data in obstetrics," J Biomed Inform, vol. 55, pp. 174–187, Jun. 2015, doi: https://doi.org/10.1016/j.jbi.2015.04.004

[9] R. Chen, P. Georg-Hemming, and H. Åhlfeldt, "Representing a chemotherapy guideline using openEHR and rules," in Studies in Health Technology and Informatics, IOS Press, 2009, pp. 653–657. doi: https://doi.org/10.3233/978-1-60750-044-5-653

[10] M. Li et al., "Development of an openEHR Template for COVID-19 Based on Clinical Guidelines," J Med Internet Res, vol. 22, no. 6, p. e20239, Jun. 2020, doi: https://doi.org/10.2196/20239

[11] L. Min, Q. Tian, X. Lu, J. An, and H. Duan, "An openEHR based approach to improve the semantic interoperability of clinical data registry," BMC Med Inform Decis Mak, vol. 18, no. 1, p. 15, Mar. 2018, doi: 10.1186/s12911-018-0596-8

[12] S. Garde, E. Hovenga, J. Buck, and P. Knaup, "Expressing clinical data sets with openEHR archetypes: A solid basis for ubiquitous computing," Int J Med Inform, vol. 76, no. SUPPL. 3, pp. S334–S341, Dec. 2007, doi:10.1016/j.ijmedinf.2007.02.004

[13] C. D. Kohl, S. Garde, and P. Knaup, "Facilitating secondary use of medical data by using openEHR archetypes," in Studies in Health Technology and Informatics, IOS Press, 2010, pp. 1117–1121. doi:10.3233/978-1-60750-588-4-1117

[14] A. Wulff, B. Haarbrandt, E. Tute, M. Marschollek, P. Beerbaum, and T. Jack, "An interoperable clinical decision-support system for early detection of SIRS in pediatric intensive care using openEHR," Artif Intell Med, vol. 89, pp. 10–23, Jul. 2018, doi:10.1016/j.artmed.2018.04.012

[15] F. Khennou, Y. I. Khamlichi, and N. E. H. Chaoui, "Improving the use of big data analytics within electronic health records: A case study based OpenEHR," in Procedia Computer Science, Elsevier B.V., Jan. 2018, pp. 60–68. doi:10.1016/j.procs.2018.01.098

[16] J. N. S. Rubí and P. R. L. Gondim, "IoMT platform for pervasive healthcare data aggregation, processing, and sharing based on oneM2M and openEHR," Sensors (Switzerland), vol. 19, no. 19, p. 4283, Oct. 2019, doi:10.3390/s19194283

[17] Y. Yang, H. Xu, B. Qi, X. Niu, M. Li, and D. Zhao, "Stroke screening data modeling based on openEHR and NINDS Stroke CDE," Proceedings - 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020, pp. 2147–2152, Dec. 2020, doi:10.1109/BIBM49941.2020.9313127

[18] J. Kryszyn, K. Cywoniuk, W. T. Smolik, D. Wanta, P. Wróblewski, and M. Midura, "Performance of an openEHR based hospital information system," Int J Med Inform, vol. 162, p. 104757, Jun. 2022, doi:10.1016/J.IJMEDINF.2022.104757

[19] "GitHub - ppazos/cabolabs-ehrserver: Open platform to manage and share standardized clinical data, designed by @ppazos at CaboLabs Health Informatics." https://github.com/ppazos/cabolabs-ehrserver (accessed May 13, 2023).

[20] "ehrbase/ehrbase: An open source openEHR server." https://github.com/ehrbase/ehrbase (accessed May 13, 2023).

[21] "GitHub - ethercis/ethercis." https://github.com/ethercis/ethercis (accessed May 13, 2023).

[22] "Clinical Knowledge Manager." https://ckm.openehr.org/ckm/templates/1013.26.80 (accessed May 26, 2023).

[23] "ppazos/openEHR-OPT: Java/Groovy Support of openEHR Operational Templates, Reference Model, Data Generators and other tools for www.CaboLabs.com projects." https://github.com/ppazos/openEHR-OPT (accessed May 26, 2023).

[24] "jkryszyn/openehr-test-suite: JMeter test suite for openEHR servers: EHRServer and EHRbase." https://github.com/jkryszyn/openehr-test-suite (accessed May 26, 2023).