An Operational Framework for Evaluating the Performance of Learning Record Stores

Chahrazed Labba, Azim Roussanaly, Anne Boyer

An operational Framework for Evaluating the Performance of Learning Record Stores

Chahrazed Labba[0000−0002−7921−273X], Azim Roussanaly[0000−0002−3311−3613], and Anne Boyer

Lorraine University, Loria, KIWI Team
{chahrazed.labba,azim.roussanaly,anne.boyer}@loria.fr

Abstract. Nowadays, Learning Record Stores (LRS) are increasingly used within digital learning systems to store learning experiences. Multiple LRS software have made their appearance in the market. These systems provide the same basic functional features including receiving, storing and retrieving learning records. Further, some of them may offer varying features like visualization functions and interfacing with various external systems. However, the non-functional requirements such as scalability, response time and throughput may differ from one LRS to another. Thus, for a specific organization, choosing the appropriate LRS is of high importance, since adopting a non-optimized one in terms of non-functional requirements may lead to a loss of money, time and effort. In this paper, we focus on the performance aspect and we introduce an operational framework for analyzing the performance behaviour of LRS under a set of test scenarios. Moreover, the use of our framework provides the user with the possibility to choose the suitable strategy for sending storing requests to optimize their processing while taking into account the underlying infrastructure. A set of metrics are used to provide performance measurements at the end of each test. To validate our framework, we studied and analyzed the performances of two open source LRS including Learning Locker and Trax.

Keywords: Test scenarios · non-functional requirements · Learning Record Store · xAPI specifications.

1 Introduction

Nowadays, the learning process is digitized and can happen through different activities such as learning management systems, online training, simulations and games. Given the heterogeneity of the learning activity keeping track of the learning experience of the learner is becoming challenging. The specification xAPI or Tin Can [1] came to solve the above challenge. According to the Advanced Distributed Learning (ADL) [2], xAPI is a technical specification that aims to facilitate the documentation and communication of learning experiences. This

[1] https://adlnet.gov/projects/xapi/
[2] https://www.adlnet.gov

specification defines the way to describe learning experiences by using a structure in the form of "Actor Verb Object". The above structure is known as a learning record or a statement. Further, the xAPI specifies how these statements can be exchanged electronically by transmission over HTTP or HTTPS to an LRS, which is defined by the ADL [3] as follows: "A server [4] that is responsible for receiving, storing, and providing access to Learning Records."

Multiple LRS products have made their appearance in the market such as Learning Locker [5], Watershed LRS [6] and Trax [7]. The competition, in terms of being the market leader, is high. All the LRS systems provide the same basic features including recording and retrieving learning records. Further, some of them offer additional varying functionalities such as visualization functions and interfacing with various external systems. ADL has setup a rational process [2] for choosing an LRS. The process gives more importance to the selection based on the functional features such as analytics, reporting and external integrations rather than the non-functional requirements such as the response time, scalability and throughput. However, these non-functional requirements may differ from one LRS to another. Thus, for a specific organization, choosing the appropriate LRS is of high importance, since adopting a non-optimized one may lead to both monetary and data losses. There exist many tools [1][4][5] that can be used to run performance tests in order to select the appropriate LRS. However, the test settings need to be constantly edited to run different scenarios. Furthermore, adjusting the performance tests is a time consuming task and error prone.

The aim of this work is to provide those involved in the process of selecting an LRS with automatic test plans requiring minimum settings effort. To do so, an operational framework for studying and analyzing the performance behaviour of LRS is proposed. Our framework, called LOLA[8]-meter, provides the user with two test plans: 1) The performance plan consists of two test types including the load and stress tests. The aim is to study the LRS behaviour under expected and not expected load conditions. The load can be expressed in terms of the number of simultaneous requests and/or the number of statements sent within a request[9]; and 2) The strategy selection plan consists of two types of tests including the Post_Chunk_Time and the Post_Chunk_Statement tests. The aim is to determine the suitable strategy to adopt for sending learning experiences to be stored in the LRS while taking into consideration the infrastructure and the size of the generated data. For both types of test plans a set of performance indicators are defined to analyze the output results of the test scenarios e.g response time and throughput. To validate our test scenarios, we studied and analyzed the performances of two LRSs including Learning Locker (LL) [5] and

[3] https://adlnet.gov/news/2017/04/06/xapi-learning-record-store-test-suite-and-adopter/
[4] system capable of receiving and processing web requests
[5] https://www.ht2labs.com/learning-locker-community/overview/
[6] https://www.watershedlrs.com
[7] http://traxproject.fr/traxlrs.php
[8] Laboratoire Ouvert en Learning Analytics
[9] the size of a post request

Trax [7]. Both of the LRS were selected because they are open source and conform to the specification requirements [10] defined by ADL.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 introduces the proposed test scenarios. Section 4 presents the experiments and the validation using two open source LRS including LL and Trax. Section 5 enumerates the threats to validity. Section 6 presents the conclusions and the future work.

2 Related Work

There are few works on how to select an LRS. Some vendors, for example, LL[5], watershed LRS[6] and waxLRS[11] provide on their websites case studies and demos concerning the adoption of their systems. Further, some vendors provide white papers that discuss how to choose the LRS partner and what questions need be asked to evaluate the LRS products. For example Yet Analytics[12] provides a set of questions [8] that are organized into three categories "analytics and reports", "customer support" and "database security, stability and scalability". In each category, a set of questions are defined and need to be asked to evaluate the LRS partner. While, H2Labs defines 28 questions[13] to conduct an LRS needs analysis. The questions cover the types of deployment (on-site or SaaS), the data sending, storing and retrieving. In [2], the author introduces a rational process to select an LRS. The process gives more importance to the selection based on the functional features rather than the non-functional requirements. Further, the paper does not provide in anyway a comparative rating or evaluation of existing LRS software. In [7], the author highlights the special features and issues to consider when selecting an LRS such as conformance requirement, cost, hosting options and data analytics requirements. In [9], the authors present a set of decisive factors to consider, when searching for an LRS, such as analytics and reporting, security, integration and scalability. In [3], the authors proposed a web-based learning environment dedicated for training how to command and control unmanned autonomous vehicles. One of the main issues revealed in the work is the scalability and performance requirements of the integrated LRS for storing stream data. The authors found that the existing LRS may not perform well under certain circumstances. So, they proposed a storage system based on the use of an adhoc server over SQLLite3. Even though the proposed solution presents an efficient storage system, however it leaves out many facilities provided by the LRS. To summarize, both research works and case studies deal with the selection of LRS from a pure functional point of view. However, the evaluation of the LRS based on the non-functional requirements is not well taken into consideration. In this work, we focus on providing an operational framework to evaluate and rate the existing LRS software. LOLA-meter provides the users

[10] https://lrstest.adlnet.gov
[11] https://www.elucidat.com/blog/using-wax-lrs/
[12] LRS provider https://www.yetanalytics.com
[13] https://www.ht2labs.com/conducting-lrs-needs-analysis/

with a set of automatic test plans requiring minimum settings effort to study to which extent an LRS fulfills the non-functional requirements. There exist many tools that can be used to run performance tests in order to select the appropriate LRS. However, the test settings in these tools need to be constantly edited to run different scenarios. Furthermore, adjusting the performance tests is a time consuming task and error prone. In the next section we introduce an updated version of the selection process of an LRS and we present the test plans implemented within LOLA-meter.

3 How To Select an LRS?

ADL has setup a rational process [2] for choosing the appropriate LRS. It emphasizes the LRS selection based on the functional features rather than the non-functional requirements. In this section, we introduce an updated version of the selection process. An additional step that considers the analysis of the non-functional requirements is added. Section 3.1, introduces the modified selection process and Section 3.2 presents LOLA-meter for evaluating LRS software using the non-functional requirements.

3.1 Updated Process for choosing an LRS

The recommended process for choosing the LRS is composed originally from four big steps including 1) **Project scope**, 2) **Develop an LRS requirement matrix**, 3) **Develop a feature rating matrix** and 4) **Decision making**. The process focuses more on the LRS features and what the system can do to fulfill the functional requirements and less on the performance aspect. So we extend the current process version and we include an additional step "**Develop a non-functional requirement matrix**". In the following, we explain the different steps of the updated selection process introduced in Fig.1.

Project Scope: This phase is primordial for any organization that wishes to adopt the use of LRS. Meetings with all the involved stakeholders need to be organized to discuss the following steps: i) Study and analyze the feasibility of acquiring an LRS system; ii) Determine the critical and high-level requirements for the LRS. The definition of such requirement types will allow the organization exclude many unsuitable LRS candidates; iii) Fix the budget knowing that there are different pricing models and iv) Select the required LRS category.

Develop an LRS Requirement Matrix: This phase consists of the following steps: i) identify the LRS products that better match the LRS category identified in the previous step; and ii) Develop and populate a functional requirement matrix. The matrix allows assessing the identified LRS systems against the high-level requirements defined in the project scope phase.

Develop an LRS Feature Matrix: During this phase, the following steps need to be achieved: i) Filter the list of the potential candidates by eliminating those who do not fulfill the minimum of the functional requirements or are over the fixed budget; ii) Develop a detailed list of the features of the remaining LRS;

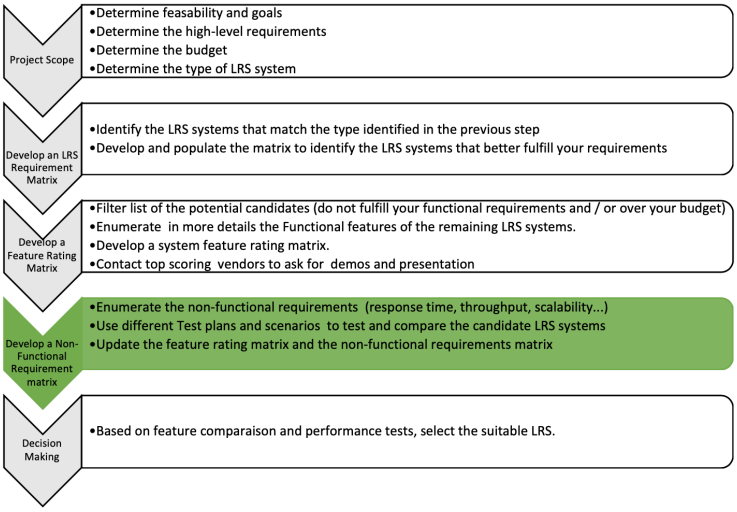

Fig. 1: Updated Process for choosing an LRS

iii) Use the system featuring rating matrix proposed in [2]. It consists in using numerical ratings of the LRS features for example "0" if the LRS does not offer the feature and "10" if the LRS presents a strong implementation of the feature. The rating process can be achieved through the use of available documentation, consultation of the feedback of some LRS users or by the contact of LRS representatives; and iv) Contact the top scoring vendors to ask for presentations and demos.

Develop a non-functional Requirement Matrix: The main contribution of the current work is elaborated in this phase. Indeed, considering the non-functional requirements is of high importance to select the right LRS. A suitable LRS for one organization does not mean the same for others, since each has its own requirements. In this phase, we introduce the non-functional requirements matrix, presented in Table 1, which presents a set of metrics measurements. Those measurements will allow the user to determine the suitable LRS that fulfills his needs. The matrix is populated after running each scenario with a given test plan. Indeed, we propose two different test types including the performance test plans and the strategy test plans (More detailed in Section 3.2). The matrix can be used as follows: i) Replace the "X" with the name of the executed test scenario; ii) Replace the top row (LRS1, LRS2, ..) with the name of the LRS that have been identified in the previous steps of the selection process; iii) Replace the row names (metric1, metric2, ..) with the names of the considered metrics for example response time and throughput; iv) Run the test scenario for each of the LRS software and fill the matrix with the average measurements. To summarize, this phase consists of the following steps: i) Define the non-functional requirements; ii) Use our automated test plans for all the LRS candidates and iii) Update the non-functional matrix.

LRS NFR rating matrix for Scenario X				
Metric Name	LRS 1	LRS 2	LRS 3	LRS 4
Metric 1				
Metric 2				
Metric 3				
Metric 4				
Metric 5				

Table 1: Non-functional requirements matrix for Scenario X

Decision Making: The final phase consists of selecting one of the LRS products. Based on the feature and non-functional requirements comparisons as well as the discussions with the LRS vendors, the organization can make its decision about which LRS to adopt for their use.

3.2 Design of Test Plans

In this section we describe the proposed test plans that need to be used during the fourth step in the selection process presented in Figure 1. Indeed, we distinguish two different plans including the performance and the strategy selection. Each plan encompasses a set of test types. For both types of test plans a set of performance indicators are defined to analyze the output results of the test scenarios e.g response time and throughput. The test plans are implemented and made available through an operational framework [14] that provides 24 different test scenarios. Our framework is based on the use of the Apache Jmeter API [15]. The selection of Jmeter was not arbitrary. Indeed, according to many evaluations [1] [4] [5] [6], Jmeter was proven to be one of the best open source testing tools. Compared to the Jmeter application, our framework exempts the user from the burden of preparing manually the configurations files like the json and the csv files. Everything will be generated automatically through graphical interfaces or a simple configuration file (for a non-gui version of the framework).

Performance Test Plan: This scenario encompasses two types of tests including the load and stress tests. Each test requires a set of inputs and provides a set of outputs.

– Load test: is designated to study the LRS performance behaviour under real-life and expected load conditions. In our case, the load can be either expressed in terms of simultaneous requests [16] or the number of statements that are sent within requests [17]. If the LRS is used by one organization, so the number of connected users is equal to 1. In this case the load is expressed in terms of the number of statements that are sent within a post

[14] Open source: https://github.com/Chahrazed-l/Operational_Framework
[15] https://jmeter.apache.org
[16] denotes also the number of the concurrent users connected to the LRS
[17] specific to the requests of type post

request. While if the LRS is dedicated for multi-organizational use, the load is expressed in terms of both number of concurrent users as well as the number of statements sent within requests. The test is provided with real-life statistics of statements generation[18]. The user can use those statistics to run the load test, else he can enter and use his own information about the number of statements that can be generated by learners.
- Stress test: This test is designated to study the robustness of the LRS beyond the limits of normal load conditions. The load can be expressed in terms of the simultaneous requests or the number of statements sent within the requests. This test consists of generating sudden heavy loads either by increasing the number of simultaneous requests (the number of concurrent users) and/or the number of statements (for post request).

For both test scenarios , a set of inputs needs to be provided to ensure a smooth execution. The inputs include: 1) the request type (get, post or both), 2) Number of test runs: the tests need to be repeated so many times in order to ensure the stability and the accuracy of the results; 3) Time interval between runs: depicts the time period that separates the end of a test run and the start of another; 4) number of requests; 5) time interval between requests; 6) number of concurrent users [19];7) statements number (for post request); 8) LRS information: required to connect to the deployed LRS.

Strategy Selection Test Plan: This plan allows the selection of the suitable strategy to adopt to send post requests. It encompasses two types of tests including the Post_Chunk_Time test and the Post_Chunk_Statement tests. The aim is to discover which of the strategies allows reducing the LRS load and optimizing the statements storage by taking into consideration the underlying computational infrastructure. Each test requires a set of inputs and provides a set of outputs. For both tests, the load is expressed in terms of the number of statements sent within a post request.

- Post_Chunk_Time test: is designated to study the LRS performance under dynamic loads sent periodically (second, minute, hour, day). Indeed, we fix the time interval that separates the end of sending a request and the start of sending a new one. The number of the statements keeps changing for each new request. Different methods are used to generate the statements number including random, Poisson and Gaussian methods. For this test, the user selects the corresponding time interval to consider as well as the method for generating the statements.
- Post_Chunk_Statement test: is designated to study the LRS performance under static loads sent aperiodically. This test consists of: 1) fixing the number of statements sent within each request and 2) Using dynamic time intervals to send the requests. Different methods are used to generate the time intervals including random, Poisson and Gaussian methods. For this test, the

[18] extracted from a Moodle dataset, more information are provided in the section 4: Experiments and validation

[19] One LRS can be used to store the statements coming from different sources

user selects the corresponding statements number to be sent per request as well as the method for generating the time intervals.

Both the performance and strategy selection test plans provide as output a set of metrics measurements such as the response time, response time distribution, throughput, latency, the error rate and connect time (to name a few). We provide an exhaustive set of metrics to visualize the performance behaviour of the LRS under the different test scenarios. At the end of each scenario, a report with all the measurements is generated. The report contains dashboards and charts.

4 Experiments and validation

Two open source LRS have been used to validate the test plans, including Trax[7] and LL[5]. Both of the LRS have been deployed on machines having the same characteristics in terms of computing power and memory. The evaluation was carried out using a real-life Moodle dataset [20]. The dataset contains the data logged during 769 days (more than 2 years). It contains 2169 users , almost 2 millions events and we count 57 different actions such as viewed, updated and submitted. Moodle is an important source of learning data. Nowadays many plugins [21,22] have been developed to ensure the generation of xAPI statements from moodle contents. In the moodle traces, one can easily detect the format of an xAPI statement. For the rest of this section, we admit that an event in moodle is equivalent to an xAPI statement. We investigated in more details the behavior of the students in terms of event generation. We considered the 10 students who interacted the most with the Moodle content. For each of these students, we investigated in more details the maximum number of events that have been generated on different time intervals including second, minute, hour and day. Table 2 summarizes the extracted information. For example, the maximum events numbers that have been generated by Student 1 on a second, a minute , an hour and a day are respectively 11, 28, 385 and 1439. For each student, the observed values on the different time intervals are not necessarily perceived on the same day. In section 4.1, we used the information of Student 1 as a reference to run the different test scenarios including load and stress tests. The rest of the information are provided through LOLA-meter to run the same scenarios with different inputs in terms of statement generation. Due to space limitations, only some of the test scenarios are presented.

4.1 Performance Test Plan

For this plan, two tests have been performed: the load test and the stress test.

The load test is used to analyze the behaviour of the LRS under real-life expected loads. The test is performed using different configurations, shown in

[20] https://research.moodle.org/158/
[21] https://moodle.org/plugins/mod_tincanlaunch
[22] https://moodle.org/plugins/logstore_xapi

Username	Student 1	Student 2	Student 3	Student 4	Student 5	Student 6	Student 7	Student 8	Student 9	Student 10
Second	11	14	17	23	33	64	86	224	227	459
Minute	28	42	60	62	75	198	230	3634	5196	6220
Hour	385	604	608	718	737	893	1113	9938	10959	45959
Day	1439	1457	1861	2079	2952	3884	6139	11119	11799	53506

Table 2: Statistics for event generation on different time intervals

Users Number	Students Number	Statements per Student	Total statement/req	Iteration Duration (S)	Iterations number	Break Time (S)	Config. Label
1	200	11	2200	900	5	120	Config.1
	400		4400				Config.2
	600		6600				Config.3
	800		8800				Config.4
	1000		11000				Config.5
	1100		12100				Config.6
	1200		13200				Config.7

Table 3: Configurations for the Load Test

Table 3. Each configuration is characterized by the number of users, the number of students per user, the number of statements generated per student, the test duration, the iterations number and the break time between two successive iterations. Each of the LRS is deployed as it is furnished by the providers without modifying any used technologies including the database and the application server. LL and Trax use respectively nginx/mongo and apache2/mysql. In Table 4, we present the results provided at the end of each test scenario. Both of the LRS have been evaluated in the same manner and using the same configurations. In the evaluation we can take into consideration, the number of requests that has been processed during the test duration, the error rate which indicates the rate of requests that the LRS failed to store due to a given issue, the response time and the throughput. Based on the results of the performed tests, one can notice that LL has, sometimes, the best min and max values for response time. However, Trax outperforms LL in all the tests scenarios. The performance difference is noticeable in the number of processed requests, the error rate, the response time (average, 90%, 95% and 99%) and the throughput. For example, as shown in Table 4, for both configurations (Config.6 and Config.7) LL failed to store all the received requests. For Config.6 and Config.7 LL failed to process respectively 5.58 % and 37.78 % of the requests. The LL application server throughout an internal error to the server (code 500). Whereas, for these same configurations, Trax succeeded in storing all the received requests. To investigate in more details the results presented in Table 4, additional tests have to be performed. Indeed, we need to use the same technologies including the application server and the database for both of the LRS.

Config. Label	LRS	Executions			Response Time (ms)						debit
		#req	KO	%Error	Avg	Min	Max	90%	95%	99%	
Config.1	LL	294	0	0.0%	15431.88	5227	42734	24426.00	33678.25	42213.35	0.06
	Trax	421	0	0.00%	10751.18	9429	23198	12355.80	14309.50	19049.06	0.08
Config.2	LL	152	0	0.0%	30187.72	14113	51952	36565.80	40472.40	48886.48	0.03
	Trax	234	0	0.0%	19469.05	18616	23681	20047.00	20534.00	22352.90	0.05
Config.3	LL	98	0	0.0%	47315.87	20227	81839	71867.50	74172.85	81839.00	0.02
	Trax	150	0	0.0%	30782.15	28790	76040	32640.60	36278.65	62869.25	0.03
Config.4	LL	98	0	0.0%	47432.11	14217	88432	60185.60	62722.10	88432.00	0.02
	Trax	103	0	0.0%	44925.97	39406	109670	54009.00	59329.20	107892.12	0.02
Config.5	LL	59	0	0.0%	80935.97	14778	171988	100402.00	101631.00	171988.00	0.01
	Trax	91	0	0.0%	53152.45	48587	297079	52377.40	54405.80	297079.00	0.02
Config.6	LL	51	3	5.58%	93050.45	15914	128741	118633.80	123194.60	128741.00	0.01
	Trax	83	0	0.0%	56662.07	53186	128302	57786.80	61309.20	128302.00	0.02
Config.7	LL	45	17	37.78%	107262.82	2091	149505	142720.80	145747.90	149505.00	0.01
	Trax	75	0	0.0%	61524.87	57843	144986	62219.80	69537.00	144986.00	0.01

Table 4: Execution Results for both Trax and LL using different configurations

The stress test is used to study the performance of the LRS under unexpected load conditions. The test is performed using the configuration shown in Table 5. The configuration is characterized by the number of concurrent users (3 users), the number of requests per iteration (20 requests), the break time between two successive requests sent by each user, the number of statements sent within each request, the number of iterations (5 iterations) and the time separating two successive iterations (2 minutes). Each request contains a different number of statements. As shown in Table 5, the first request contains 551 statements, while the last one contains 804. The statements are written in separated Json files that will be sent within the requests bodies. Among the 20 requests, one user sends from 6 to 7 requests. The users send their requests concurrently to the LRS. For example user 1, user 2 and user 3 send respectively 551, 595 and 613 statements within their first requests and so on for the rest. The results of the stress test are shown in Table 6. For both LRS, we provide the total execution time where we subtracted the break time between the iterations, the number of statements successfully stored, the error rate, the average response time as well as the Min and Max response times. Even though Trax has a greater error rate (17 %), the number of the final statements successfully stored is superior to the one stored within LL. Further, the overall execution time with Trax is smaller 28 minutes compared to 68 minutes for LL. This is due to the response time, which is better for Trax. One can notice that from the average and maximum response time values in Table 6.

4.2 Strategy Selection Test Plan

The strategy test allows to select the way of sending the xAPI data to the LRS. As we explained in Section 3.2, we differentiate two types of this test including the Post_Chunk_Statement test and the Post_Chunk_Time test.

Number of users	Number of requests/ iteration	Break Time request (Second)	#statement for all requests	#iterations	break time iteration (Second)
3	20	10	551,595,613,700,11281,11967,804 12944,1588,1602,13537,1881,14149 613,2088,13537,1881,14149,11967,804	5	120

Table 5: Configuration for the stress test

LRS	total exec Time (Minute)	#statements	Error rate	Avg response Time (ms)	Min (ms)	Max (ms)
LL	68	542 924	15%	104077.99	1524	267548
Trax	28	552 129	17%	41348.32	2312	105941

Table 6: Performance results for a Stress Test

The Post_Chunk_Statement test is performed using the configurations presented in Table 7. Each configuration is characterized by the size of the statement chunk, the time intervals, the iteration duration and the iteration number. The main idea is to analyze the LRS performance when sending fixed statements number within the request body while using different time intervals. The time separating the sent of one request from another can be generated using three different methods including Poisson, Gaussian and Random. For each method, the user can fix the range for generating the values. As shown in Table 7, we used two different chunks (500 and 1000). The time interval for 500 is generated using a random generation method. While the time interval for the 1000 is deducted from the first one[23]. The aim is to show the impact of the chunk size on the LRS performance while using the same data generation scenario.

The Table 8 presents the response time of both LRS using the statement chunk sizes 500 and 1000. One can notice that LL shows better results compared to Trax in terms of response time when sending requests with small number of statements separated by a considerable time interval (one minute minimum).

Further, one can notice that by using a chunk of 1000, both LRS (LL and Trax) show better performance in terms of response time. As shown in Table 8, for LL and Trax the response times are respectively less then 2500 ms and 5200 ms. However, this values have been exceeded more than once for both LRS while using the chunk of 500 statements. We can justify this results by the fact that in the first scenario the number of the sent requests is twice the number of the ones sent within the second scenario. Further, the time interval separating the sending of two successive requests with 500 statements each, is smaller compared to the one used during the second test.

[23]The first time interval to wait to send a request with 1000 statements corresponds to the sum of the two first ones to wait to send requests with 500 statements each.

Statement Chunk_Size	Time Intervals (Second)	Iteration Duration (Second)	Iterations Number
500	59.94, 180.16, 59.79, 180.13, 59.67, 179.71, 60.3, 180.15, 60.04	900	5
1000	240.103, 239.922, 239.388, 240.448		

Table 7: Chunk Statement Strategy configuration

Chunk statement size	LRS	Response Time (ms)					
		Average	Min	Max	90%	95%	99%
500	LL	1184.67	878	2864	1651.50	2419.00	2864.00
	Trax	2896.84	2220	10234	4152.80	5519.80	10234.00
1000	LL	1943.90	1722	2435	2124.00	2420.00	2435.00
	Trax	4505.60	4235	5127	4811.40	5111.30	5127.00

Table 8: Results of the chunk statement test

The Post_Chunk_Time test is performed using the configurations presented in Table 9. Each configuration is characterized by the time chunk, the number of statements to be sent in each request, the iteration duration and the iteration number. The main idea is to analyze the LRS performance when sending variable statements number within each request body periodically. The number of statements in one request can be generated using three different methods including Poisson, Gaussian and Random. For each method, the user can fix the range for generating the values. As shown in Table 9, we used two different time chunks (1 and 5 minute). The number of statements for 1 minute is generated using a random generation method. While the number of statements for the 5 minutes is deducted from the first one. The aim is to show the influence of the selection of the time chunk size for the same generated amount of data. The Table 10 presents the results of running both scenarios with LL and Trax in terms of response time. We can notice that LL is more dedicated for batch strategy where the size of the body request is not too much significant and the time separating two successive requests is considerable. However, this was not the case during the load and stress tests, where the request size is important in terms of statement number and the break time between two successive requests was less than 10 seconds.

For both tests including the Post_Chunk_Time and the Post_Chunk_statement, the final number of statements to be stored in the LRS is the same. However, we can notice that the use of the Post_Chunk_statement strategy is more appropriate for sending data to the LRS. In overall, the response times recorded for both statement chunks (500 and 1000) are better than those recorded using fixed time chunks. Indeed, we think that the use of fixed time chunks may not be appropriate. During peak use, there may be generation of a significant number of statements that will be encapsulated and sent within a request, which the LRS will be unable to process in a reasonable time. If the chunk time is small,

Time Chunk_Size (Minute)	Statement Number	Iteration Duration (Second)	Iterations Number
1	120, 500,50, 100, 300, 50, 10, 700, 1000, 900, 30 100, 100, 100, 0	900	5
5	1070, 2660, 330		

Table 9: Chunk Statement Strategy configuration

Chunk Time	LRS	%Error	Response Time					
			Avg	Min	Max	90%	95%	99%
1	LL	0.00%	1179.04	404	5784	2225.20	2452.20	5784.00
	Trax	0.00%	1551.92	76	6276	4412.60	4728.20	6276.00
5	LL	0.00%	3501.80	1176	8663	6925.40	8663.00	8663.00
	Trax	0.00%	8196.00	1681	19530	16224.00	19530.00	19530.00

Table 10: Execution Results for chunk Time scenario

the LRS will be submerged by requests with huge number of statements. This situation may be even worse in case the LRS is used by more than one organization. The use of fixed statement chunks may be more appropriate to keep a stable performance especially in terms of response time provided that we select the suitable chunk size.

5 Threats to validity

The current work presents some limitations that we tried to mitigate when possible:(i) We used one single machine to deploy the LRS that we used to perform our tests. We intend to run more tests while using load-balancing and distributed deployment. Cloud computing may be an appropriate environment to do so. (ii) Our developed test tool can be used on a single machine, which presents a limitation when it comes to the use of many concurrent users with heavy loads. We plan to extend the current version to a distributed one where many machines can be used to run large-scale scenarios. (iii) The test tool has been validated using only two open source LRS. We plan to contact LRS vendors to carry out a large performance evaluation to publish benchmarking studies.

6 Conclusion

Multiple LRS have made their appearance in the market.The ADL provided a rational process for selecting the appropriate LRS. However, this process emphasizes the selection based on the functional features rather the non-functional requirements. Thus, in this paper we proposed an updated version of the LRS selection process by adding another step called "Develop a non-functional matrix". This step is enriched by the development and implementation of a set

of test plans to evaluate the performance of the LRS as well as determine the suitable strategy to adopt for sending learning data. A set of metrics have been used to provide the performance measurements at the end of each test. Our automated test plans have been validated using two open source LRS including LL and Trax. Evaluating the performance of LRS depends on the organization requirements as well as the context of use. An appropriate LRS for one organization does not mean the same for another. In the current work, the performed evaluation of LL and Trax does not provide in any way a recommendation of one LRS over the other. Indeed, we provide just a snapshot in the time of the results that a user may have by using LOLA-meter.

As a future work, we intend to enhance LOLA-meter to support distributed testing in order to perform large-scale tests. Moreover, we plan to use the cloud environment to perform additional performance evaluation by including multiple LRS products with different deployment settings.

Acknowledgement

This work has been done in the framework of the LOLA[8] project, with the support of the French Ministry of Higher Education, Research and Innovation.

References

1. Abbas, R., Sultan, Z., Bhatti, S.N.: Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege. In: 2017 International Conference on Communication Technologies (ComTech). pp. 39–44 (2017)
2. Berking, P.: Technical report: Choosing a learning record store. Tech. Rep. version1.13, Advanced Distributed Learning (ADL) Initiative (December 2016)
3. Dodero, J.M., González-Conejero, E.J., Gutiérrez-Herrera, G., Peinado, S., Tocino, J.T., Ruiz-Rube, I.: Trade-off between interoperability and data collection performance when designing an architecture for learning analytics. Future Generation Computer Systems **68**, 31 – 37 (2017). https://doi.org/https://doi.org/10.1016/j.future.2016.06.040, http://www.sciencedirect.com/science/article/pii/S0167739X16302813
4. Khan, R.B.: Comparative Study of Performance Testing Tools: Apache JMeter and HP LoadRunner. Master's thesis, , Department of Software Engineering (2016)
5. Maila-Maila, F., Intriago-Pazmiño, M., Ibarra-Fiallo, J.: Evaluation of open source software for testing performance of web applications. In: Rocha, Á., Adeli, H., Reis, L.P., Costanzo, S. (eds.) New Knowledge in Information Systems and Technologies. pp. 75–82. Springer International Publishing, Cham (2019)
6. Paz, S., Bernardino, J.: Comparative analysis of web platform assessment tools. In: WEBIST. pp. 116–125 (2017)
7. Presnall, B.: Choosing a learning record store (the 2019 edition). DEVLEARN (October 2019)
8. Tscheulin, A.: How to Choose Your LRS Partner. Yet Analytics Blog (2017)
9. Vermeulen, M., Wolfe, W.: In search of a learning record store. DEVLEARN (October 2019)