EasyChair Preprint

№ 2743

Lodestone: A Streaming Approach to Behavior Modeling and Load-Testing

Chester Parrott and Doris Carver

February 21, 2020

Lodestone: A Streaming Approach to Behavior Modeling and Load-Testing

Chester Parrott, Doris Carver
Division of Electrical Engineering and Computer Science
Louisiana State University
Baton Rouge, Louisiana
Email: cparro5@lsu.edu, dcarver@lsu.edu

***Abstract*—It is evident that our technologically-dependent society rightly expects systems engineers to produce systems having increasing levels of performance, efficiency, and reliability. As such, we must convolve academic and industrial approaches to provide theory, systems, and working technologies which can catalyze and propel engineers, developers, and technical professionals of various disciplines toward the ultimate goal of consistent delivery of quality systems. There exist tools and processes for improving the quality-oriented posture of the systems engineering industry; in practice, the most perpetual form of software testing continues to be rote repetition of test cases through either manual testing or scripted automation of those same manual tests.**

We describe Lodestone: a real-time approach for generating workload in software systems. This real-time approach to LT uses streaming log data to generate and dynamically update user behavior models, cluster them into similar behavior profiles, and instantiate distributed workload of software systems. We show that Lodestone outperforms Markov4JMeter through a qualitative comparison of key feature parameters as well through experimentation based on shared data and models.

***Index Terms*—Software quality, Software performance, System performance, Performance analysis, Software testing, System testing, Automatic testing, Automatic test pattern generation**

I. Introduction

Our society has become wholly reliant upon the continuous creation, operation, maintenance, and improvement of human-machine systems. The metrics and tools for verifying and validating system quality can range from the opaquely theoretical to the overly simplistic - having few solutions of practical use between the two extremes. The number of approaches to automatically evaluate the quality of a software system are continuously increasing; however, such approaches do not readily incorporate real-time understanding of and feedback from a system under test (SUT). Systems which are continuously monitored (or systems under observation (SUO)) allow for logs and metrics to be captured. To preempt and reduce the monetary cost of system maintenance, quality assurance professionals (QA) use manual processes and automated tools to validate an acceptable level of quality within a SUT. One set of tools and processes, load-testing (LT), is a sub-discipline of automated testing which focuses upon the non-functional qualities of a SUT (such as security, stability, and scalability) through exposing the SUT to various simulations of expected workload. Widely available tools (similar to JMeter [1]) are well-established as an effective means of LT in both the industrial and research communities. Tools in the same vein as JMeter primarily rely upon recorded or configured scenarios to submit a stream of statically defined stimuli to an SUT. Static scenarios can be a powerful tool for discovering flaws in an SUT by building up a service-side state/cache of information necessary to repetitively test a specific condition or circumstance; they are typically based on behavioral traces, system logs, or hand-crafted scripts. As artifacts, static scenarios must be regularly pruned, updated, and maintained by QA in order to combat the risk of diminished expected value. The bulky nature of testing artifacts can emit costs such as: storage, transmission, security, execution, and maintenance. Model-based LT approaches draw from various bodies of research (data-mining, statistics, process mining, and machine learning) in order to reduce the operational and computational cost associated with static scenarios. However, model-based LT approaches can also suffer from the same costs as static scenarios: execution, maintenance, and modification. A Markov-chain-based model of behavioral interaction was proposed by Menasce et al. in [2], [3] and extended by Markov4JMeter [4] (a free, open-source extension to JMeter) by van Hoorn et al. [5]. Subsequent approaches such as WESSBAS by Vögele et al. [6] have been shown to be an effective approach to LT. However, Qusef et al. point out that JMeter was principally


```
{
  "event_chain": [
    "login",
    "role_view",
    "logout"
  ],
  "login": 1,
  "logout": 1,
  "role_view": 1,
  "session": "9474dec4-f3f8",
  "session_end": 1500000137,
  "session_length": 50,
  "session_start": 1500000087,
  "uid": "bbicke"
}
```


Fig. 1. Recorded Session Data from SUO

designed for LT of traditional web systems [7] - a potentially notable challenge when dealing with the prolific evolution of modern system infrastructures and engineering patterns. Indeed, industry has found that JMeter has performance limitations when emulating a large amount of concurrent requests [8].

The rapid propagation of cloud technology has sparked the widespread adoption of architectural patterns such as stateless microservices based on REST (representational state transfer) and similar protocols. Heinrich et al. state in [9] that LT of modern microservice architectures provides additional challenges over classical architectures, to wit: the need for LT to align with agile development practices such as dynamic deployment and automation, the importance of understanding the infrastructure in the test cases, and the need to continuously update testing artifacts (such as models and scenarios). The capabilities of cloud-native test systems partially address the concerns of Heinrich et al. [9] and [7] through ready availability, swift scale-out/scale-in, distributed computation spanning geographic and infrastructure boundaries (similar to users of cloud-based systems), and component-level monitoring.

To the best of our knowledge, there does not exist a streaming cloud-based approach to LT which uses a Markov approach (as in WESSBAS and the Markov4JMeter extension to JMeter) and is designed to meet the accelerating modeling and responsiveness requirements of the agile development practices mentioned above. We proffer Lodestone: a learning, online, distributed engine for simulation and testing based on the operational norms of entities within a system. Our work with Lodestone represents a novel, cloud-based approach to ingesting system logs, modeling and simulating human-machine behaviors, and executing realistic LT upon a human-machine system. The following research questions inform the direction and intent of our investigation regarding Lodestone:

RQ_1: Can Lodestone extend the features provided by an open-source tool for Markov-based LT?

RQ_2: Can Lodestone provide measurable performance benefits in certain scenarios, when compared to an open-source tool for Markov-based LT?

The rest of this paper is structured as follows. The Background (II) contains information related to previous efforts. We then discuss User Behavior Modeling for LT (III), in order to provide additional context and terminology. We describe the overall approach of the system along with the architecture of Lodestone (IV). The Evaluation (V) describes the quantitative and qualitative guidelines used for our study and a discussion of our results. We conclude with a Summary (VIII) of the work and potential future directions.

II. Background and Related Work

There are many approaches to LT of scale-conscious software systems; for recent literature reviews of academic work, we recommend the reader examine the work of Jiang [10] as well as the collaboration between Jiang and Hassan [11]. In addition, we suggest a review of Leitner and Bezemer's work [12] for a recent overview of industrial tooling. Chen and

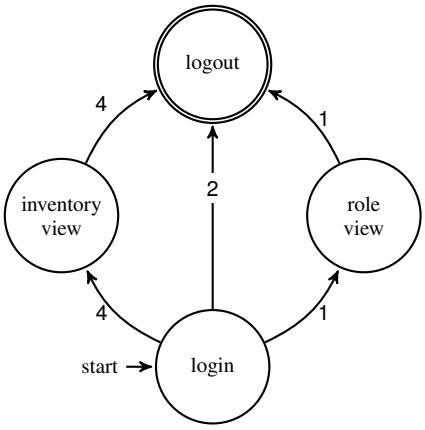

Fig. 2. Aggregated User Behavior Count Graph

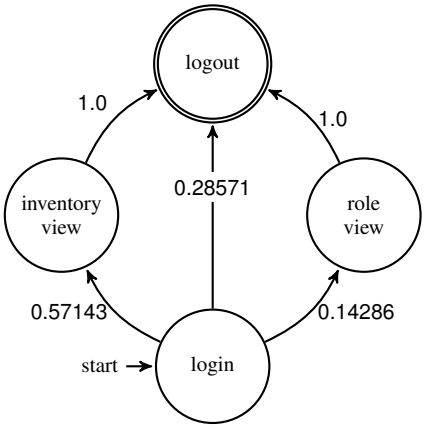

Fig. 3. Markov Chain Directed Graph for User Behavior

Shang [13] perform a longitudinal study of several releases of the same open source products to analyze regression in performance quality, asserting that most performance regressions come from bug fixes and are not noticed until after they are deployed to production - a key reason for QA efforts such as LT before software is deployed to production. Chen et al. [14] discuss many open problems related to performance and regression testing partially informing the direction of our work. Performance is also a security concern in the form of Distributed Denial of Service attacks, as addressed by Jiang et al. in [15].

A. Adaptive LT

Realistic scenarios and rapidly closing the feedback response loop are two of the most important qualities for legitimate system testing; in addition, natural feedback is why human testing will continue to be critical for performance testing of production-critical systems. The ability of a testing system to adapt to incoming data is paramount toward bridging the gap between human efforts and rote machine script execution. Lenz et al. [16] describe a machine-learning approach to clustering of various load and performance testing metrics toward improving results of the quality assurance lifecycle;

however, their approach is offline and does not mention dynamic generation of tests or test data from the learned information. In cloud-based systems, Shariffdeen et al. [17] describe adaptive auto-scaling strategies to meet performance goals; similarly, Iqbal et al. [18] describe such adaptive scaling approaches in web applications. The tendency of modeling typically relies upon a manual process consisting of a period of observation, followed by building or rebuilding models, and then executing the performance testing itself. The process of LT must remain adaptive and reactive to current data (instead of requiring manual intervention from QA).

B. Model-based LT

Modeling system or user behavior is a powerful means of using data from an SUO which can impact approaches toward testing the SUT. Gao and Jiang describe an ensemble-model based approach to performance testing and show how it can outperform baseline models under environmental changes in the SUT [19]. Apte et al. describe *AutoPerf* [20] for modeling performance metrics of a system under test while simultaneously driving the performance testing process. Ramakrishnan et al. [21] discuss metrics for tracking user interaction times. Vögele et al. as well as van Hoorn et al. describe several means of modeling of users and workload in session-based systems [5], [6], [22]–[25]. Trubiani et al. in [26] discuss using operational profiles in LT in order to detect and correct performance-based anti-patterns. Wienke et al. describe a domain specific language approach in [27] for modeling performance testing in robotics. In all, the modeling of user, machine, operational, and performance characteristics of a system is a powerful means of improving that system's measurable quality. For better results in a prospective automated performance testing approach, adaptive methods should be included for improved speed in testing feedback. We proceed to discuss the central approach to catalyzing model-based LT through Markov modeling of user behavior from system logs.

III. User Behavior Modeling for Load-Testing

We describe microservices, system logs, and behavior modeling as related to LT. A data-driven model-based approach to LT allows for a certain degree of human facility to be imposed on the process.

A. Cleaning and Filtering of System Logs

Logs are a semi-structured representation of data points recorded throughout the process of operating and maintaining the SUO. When extracted from a SUO, such logs can be a treasure trove of valuable information for understanding the health of the SUO; moreover, if certain metrics and semantic data (such as HTTP request headers, query parameters, unique identifiers, and access tokens) are available within the SUO's logs, data aggregation can facilitate statistical models of expected usage behavior for the SUO. We expect a log to consist of a sextuple (σ_s and $\sigma_t \in S, \tau, u, e, s$), where σ_s and σ_t are source states and target states (respectively) within a set of possible states S in the system, τ is a time-stamp of when the state transition occurred, u is a unique identifier for a user or session, e represents whether the state transition resulted in an error, and s is any attached satellite information. (For simplicity, we may also refer to such sextuples as logs.) The process of discovering and modeling through log structures is described by Menasce et al. [2] as Customer Behavior Model Graphs (to be extended by Menasce in [3]); further, a streaming log processing approach is discussed by Du and Li [28]. A Markov approach to behavior modeling is based on the observed relative activation frequency of each transition between states (see the `event_chain` attribute as shown in Figure 1). Karlin and Taylor define a Markov Process as:

> "a process with the property that, given the value of X_t, the values of $X_s, s > t$, do not depend on the values of $X_u, u < t$; that is, the probability of any particular future behavior of the process, when its present state is known exactly, is not altered by


```
{
  "uid":"bbicke",
  "average_session_length": 28,
  "transition_counts": {
    "(login,logout)": 2,
    "(login,inventory_view)": 4,
    "(login,role_view)": 1,
    "(inventory_view,logout)": 4,
    "(role_view,logout)": 1
  },
  "number_of_sessions": 7,
  "session_lengths": [
    33,33,33,33,7,7,50
  ],
  "user_behavior_profile": {
    "(login,logout)": 0.28571,
    "(login,inventory_view)": 0.57143,
    "(login,role_view)": 0.14286,
    "(inventory_view,logout)": 1,
    "(role_view,logout)": 1
  }
}
```


Fig. 4. Aggregated User Behavior Model


```
{
  "average_session_length": 30.35,
  "number_of_users": 2,
  "user_ids": ["bbicke", "thuels"],
  "clustered_behavior_profile": {
    "(login,logout)": 0.34286,
    "(login,inventory_view)": 0.43571,
    "(login,role_view)": 0.22143,
    "(inventory_view,logout)": 1,
    "(role_view,logout)": 1
  },
  "probability": 0.0253164557,
  "profile_id": 5
}
```


Fig. 5. Clustered Profile Behavior Model

	login	$	inventory_add	inventory_modify	inventory_view	role_add	role_modify	role_view
login*	0.0	0.34286	0.0	0.0	0.43571	0.0	0.0	0.22143
inventory_add	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
inventory_modify	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
inventory_view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
role_add	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
role_modify	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
role_view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Fig. 6. Sample Learned Markov4JMeter Model

additional knowledge concerning its past behavior." [29]

Karlin and Taylor describe a Markov Process as being a Markov Chain (MC) if it is composed of a distinct set of states which are countable, and finite [29]. We can visualize the MC as a directed graph, in which nodes of the graph are states of the chain, and edges represent the transition between states (see Figure 2). A value or weight associated with an edge between two nodes is representative of the probability of transitioning between the two states. More formally, given two states in an MC (σ_a, and σ_b), let edge $< \sigma_a, \sigma_b >$ have a weight of p. It stands that $Pr(\sigma_b|\sigma_a) = p$, or, given that the currently observed state is σ_a, the probability that σ_b is the next observed state is equivalent to p. The aggregation of user sessions can be illustrated as the directed graph in Figure 2 and condensed to the MC shown in Figure 3. An example of how these data can be stored as shown in Figure 4. If the size of S is large (as is typical in modern web-based software systems), it is more computationally and spatially efficient to represent the counts and frequencies of extracted behavior models as sparse vectors (defined by Tinney et al. [30]). Sparse vectors can be structured from session data as represented by the `transition_counts`, `user_behavior`, and `clustered_behavior_profile` attributes in Figure 4 and Figure 5. By assuming that the set of states S is fully known, sparse vectors can be reconstructed into a non-sparse matrix having the same dimensionality as $S \cdot S$. The Markov4JMeter software depends on a non-sparse matrix, such as shown by Figure 6. Note that the `$` symbol in the matrix represents the final accepting state of the MC (such as logout) and the $\star$ symbol represents the initial state of the MC. By measuring the session length and average transition time (or think time, as termed by Menasce [2]), we capture a metric for the estimated proficiency of each observed user within the SUO as well as the relative complexity involved in the tasks being performed by the user and the SUO.

B. Modeling Behavior Profiles

Generating behavior profiles from users relies on machine learning algorithms such as clustering [6]. Lodestone executes the DBSCAN [31] clustering algorithm on aggregated user profiles (illustrated by `user_behavior_profile` attribute in Figure 4). The members of this cluster do not have identical behavior patterns, but there exists enough similarity that there does not need to exist a replication of their individual behavior patterns in study or in simulation. We calculate a centroid matrix which may be used in place of the individual elements of the cluster for storage, simulation, or additional analysis. An example profile behavior model extracted from the data within our SUO is shown in Figure 5. The number of users represented by each profile model (as divided by the total number of observed users in the test population) is the frequency associated with that profile's behavior mix (as described in [6]). The clustered user profiles and behavior mix frequency rate are the final parameters required to setup the Markov4JMeter tool and Lodestone. We refer the reader to the documentation for Markov4JMeter[4] for additional details required to configure its usage. It is key to note that both Markov4JMeter and Lodestone rely on these data pre-processing steps, data structures, and resultant models.

IV. LODESTONE

In Figure 7, we show the architecture and dataflow of our implementation of Lodestone in Amazon Web Services. Our implementation consists of (1) an API Gateway serving as the External SUO (closely mirroring (8) the External SUT). The SUO and SUT are built as a microservice consisting of various resources (such as `login`, `inventory`, and `role`) and actions (`add`, `modify`, `view`). As the API is used, the event data are collected into (2) a triggered Data Processor Lambda which cleans, processes, and writes to (3) DynamoDB (Knowledge Store) - capturing live and completed session information (represented as in Figure 1). (This step serves to replace the more traditional offline method of collecting behavioral data for analysis and storage such as WESSBAS.) As the session information is written, another Lambda (4) is triggered - the User Behavior Model Builder. The User Behavior Model Builder analyzes the session data, updates the model of the user associated with the session and stores the user model (represented as in Figure 4) within the Behavioral Model Cache (implemented also in DynamoDB). As these user models are updated, the model builder runs another processing step (5), the Profile Behavior Model Builder, to cluster the user models into profile models (as represented in Figure 5). Since clustering happens as the data are processed, no additional processing needs to be performed by QA. The LT Manager (6) is another Lambda which can run in a scheduled manner using CloudWatch Event scheduling (through UNIX-style CRON triggers) or ad-hoc as part of a continuous integration or testing process. The Behavioral Model Executor Lambda (7), when triggered, will scale out the requisite number of Lambda operations, representing the users to simulate (based on the number of user models represented by a profile model). As Lodestone is cloud-based, it is scalable to within the limits of

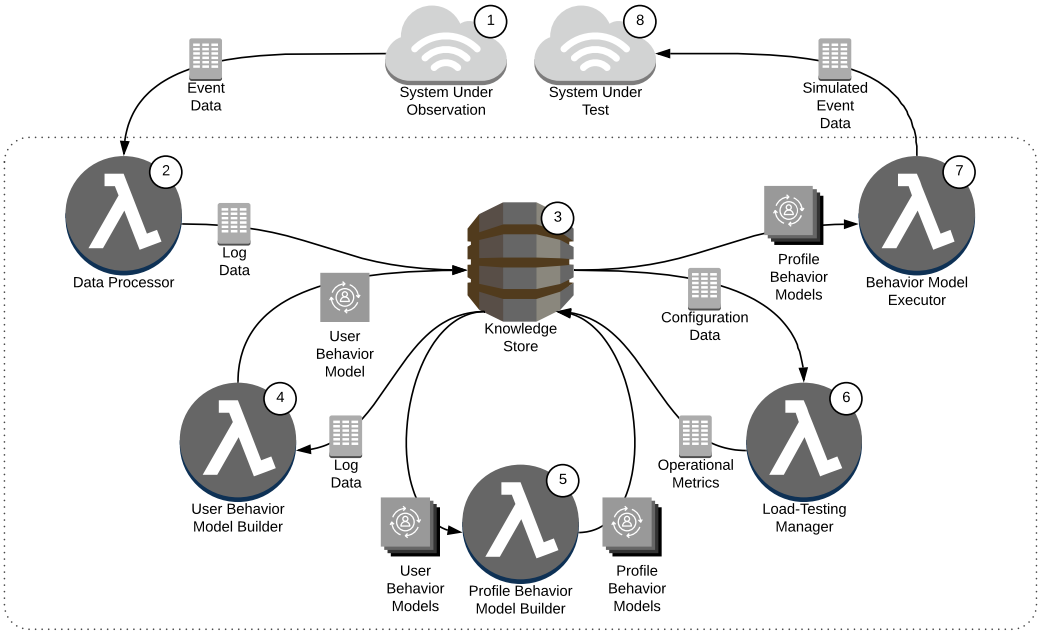

Fig. 7. Lodestone Physical Architecture and Data Flow

the cloud ecosystem being used. Through API Gateway and CloudWatch, we can determine the number of errors in the SUT (8) in real-time, in order to evaluate the load-bearing capability of the SUT.

V. Evaluation

We describe a methodology used for evaluating LT systems through their measurable run-time performance (RQ_2) and selected desirable key features (RQ_1). We show the results on Lodestone and Markov4JMeter as well as provide discussion within the boundaries of our methodology.

A. Qualitative Methodology

We define qualitative parameters informed by the literature, express why having these parameters affects the desirability of an LT system, and describe what an LT system must exhibit in order to achieve these parameters.

Behavioral: The LT system must replicate interactions between users and the SUO. An LT having this quality is desirable as it will be data-driven, rather than strictly configured or programmed - reducing the cost to maintain and use the LT.

Modeled: The LT system must use models representing behaviors of users of the SUO. An LT having this quality is desirable as models are easier to store, maintain, and update than massive testing artifacts - in addition, models can morph or completely remove sensitive and private information, catalyzing LT capabilities within restricted environments.

Distributed: The LT system must use distinct operating instances to represent the distribution of users of the SUO. An LT having this quality is desirable as distributed instances may expose infrastructure flaws (such as hardware failures or network bottlenecks) which might be otherwise invisible to small sets of dedicated testing machines.

Compressed: The LT system must use components which are efficiently stored and executed. An LT having this quality is desirable due to the performance cost of transmitting and running inefficient artifacts compounding the necessary additional fiscal cost to store and compute workload.

Streaming: The LT system must dynamically adapt to observable changes within the SUO. An LT having this quality is desirable due to the the associated cost of maintaining rapidly-changing modern software systems.

Scalable: The LT system must be capable of immediately and massively scaling. An LT having this quality is desirable due to the need to verify the expected demands on scalability in modern software systems.

Cloud-based: The LT system must be built on cloud-based technology stacks. An LT having this quality is desirable due to the migration of software systems to the cloud, the data storage and transfer synergy available on cloud-native stacks, and the distributed networking capabilities available for validating non-functional infrastructure requirements.

Think Time: The LT system must be capable of representing the time it takes for an instance to transition between states in the SUT at the profile level. An LT having this quality is desirable due to the need to accurately replicate expected workload on the SUT.

B. Quantitative Methodology:

We describe the approach for evaluating the measurable capability of the LT systems under analysis. For the purposes of our experiments, we were solely concerned with the capability

of the LT system to produce sustained testing volume. To measure and control for such, we introduce two quantitative parameters to extend the desired qualities above, namely: user volume and sustained requests per minute.

User Volume (UV): represents the number of users being simulated - the primary contributing factor determining the required amount of computational and storage resources required to operate the LT system under analysis.

Sustained Requests-Per-Minute (SRPM): represents the workload that the LT system is able to produce while maintaining expected operational norms. We gather data points to measure the SRPM by providing sufficient time for the LT system to warmup, operate, and cooldown.

We conducted four experiments, each having UV as the primary variation in input. The first two experiments consisted of running each LT system with UV=100 for at least five minutes (plus sufficient warmup and cooldown time). The second two experiments consisted of running the LT systems with UV=1000 for at least five minutes (plus sufficient warmup and cooldown time). As part of our steps to ensure consistency, we used the same models to execute both LT systems; these are the same behavioral models learned from the streaming-behavioral training of the SUO. We trained the models based on 4470 live requests from 79 users against the SUO API and noted an average latency of 485ms per request while training, with no observable non-functional errors (e.g. API responses of 400, 404, or 500). For executing the Markov4JMeter load tests, we used a 2018 MacBook Pro with 2.2GHz 6-Core Intel Core i7 with 16GB 2400 MHz DDR4 RAM. For the Lodestone testing, our instances operated on Lambdas configured at 3008MB of RAM with a 15m timeout period. As Markov4JMeter does not allow for profile-level think time between steps, we used an average of the profile think-time averages learned when configuring Markov4JMeter (30s per step). In order to ensure the SUT properly entered a dormant state with no cached information, we cleared all caches and cookies before executing each experiment.

C. Results and Discussion

We compare the features between Markov4JMeter and Lodestone, as shown in Figure 8. JMeter [1] is a well-studied open-source tool that has classically been used for LT of systems in research as well as industry. While it can be deployed in the cloud in virtual servers, it is not a cloud-native technology. We show JMeter as the base case for comparison, it does not support model-based LT by default, it is client-based instead of being distributed; however, it can be based on user behavior if explicitly configured from static artifacts. Markov4JMeter extends JMeter with the capability to configure user-behavior Markov models within the LT system [4]. However, Markov4JMeter does not continuously capture or model user behavior; thus, it not built for online distributed systems or cloud-native operations. WESSBAS, an extension to Markov4JMeter [6], relies on batch-based learning instead of online learning. Lodestone provides a cloud-native means to learn, store, and execute user behavior models from streaming data in the form of Markov models. As Lodestone is based on the serverless compute model, our approach is both distributed and scalable. In addition, Lodestone provides analysis of think-time per user as well as per profile, where Markov4JMeter's documentation [4] shows a standard Gaussian think-time per behavior test iteration.

	JMeter[1]	Markov4JMeter[4], [6]	Lodestone
Behavioral	✓	✓	✓
Modeled	✗	✓	✓
Distributed	✗	✗	✓
Compressed	✗	✗	✓
Streaming	✗	✗	✓
Scalable	✗	✗	✓
Cloud-based	✗	✗	✓
Time-variant	✗	✗	✓

Fig. 8. Feature Matrix of Select Load-testing Tools

For the first set of tests, we used UV=100 users on both LT engines. In Figure 9, we observed the SUT warming up to around 200 sustained concurrent requests per minute while running Markov4JMeter with 100 users; by comparison, in Figure 9, we observed the SUT warming up to 1000 sustained concurrent requests per minute while running Lodestone with 100 simulated users. For the second set of tests, we used UV=1000 users on both LT engines observing 2000 sustained concurrent requests per minute in Figure 10 for Markov4JMeter; for Lodestone, we observed around 9500 sustained concurrent requests per minute (see Figure 10). With our second test, we were able to hit the throttling limit of the SUT, an operational upper bound in the SUT). The SRPM of Lodestone was an order of magnitude higher than Markov4JMeter when generating workload on the same models for UV=100 and UV=1000. In addition, our results show that Lodestone was able to elicit an operational limitation that Markov4JMeter was not capable of detecting with the hardware and configurations available to our machine. Any additionally perceived slowdown in Markov4JMeter is a well known limitation of the product (see [8] for more details) due to the underlying product (JMeter) being primarily single-client based. We control for this factor through the use of a lower order of magnitude UV in addition to the higher UV to account for low throughput scenarios. Other factors for us to consider are the variations in run-time and unknown environmental factors between our experiments; however, we controlled for these factors by running multiple iterations of our experiments and waiting a sufficient time between iterations to allow for such variations to clear. Throughout the multiple iterations of the experiment detailed above, the SRPM of Lodestone remained an order of magnitude higher than the SRPM of Markov4JMeter. Our results show that Lodestone extends features provided by classic MC-based approach to LT (RQ_1) while providing performance benefits (RQ_2).

VI. Summary

We have described the processes and architecture used to implement Lodestone - a real-time approach to ingest event

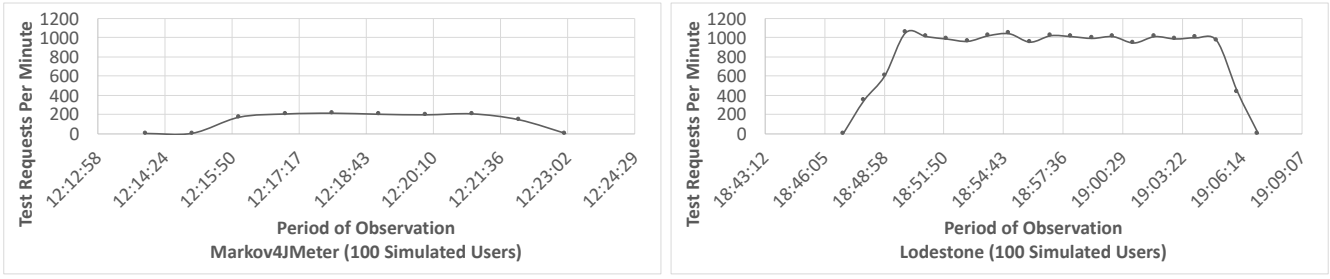

Fig. 9. Test Per Minute Volume (UV=100)

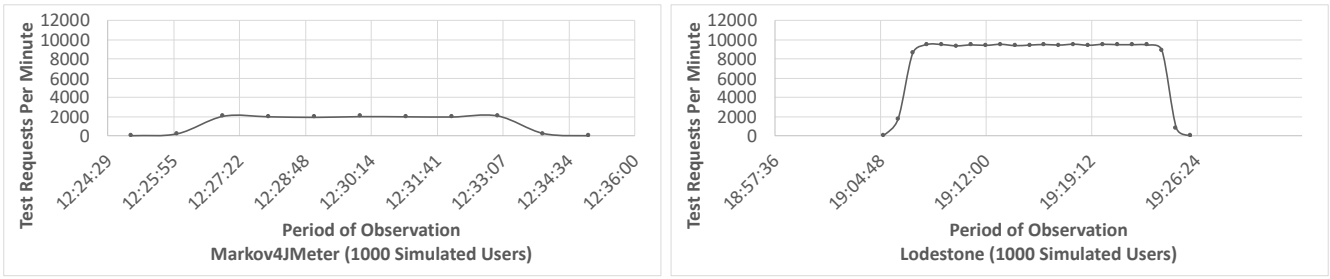

Fig. 10. Test Per Minute Volume (UV=1000)

logs, model user behavior, and simulate scalable workload on software systems. Lodestone is capable of real-time behavior modeling in systems where richly-populated event logs are readily available from an SUO and a representative SUT is available for evaluating software before it is released to users. We compared the features and operational measurements of Lodestone with an extension to a well-researched open-source product JMeter - Markov4JMeter. Based on the boundaries established in our evaluation, we have shown Lodestone to perform favorably when using the same learned models as Markov4JMeter; moreover, Lodestone extends the features provided by Markov4JMeter to scale to cloud-scale workload requirements. Our approach also has potential for future extension toward dynamic execution of massive simulations, agile requirements mining (through examination of the behavior models), automated regression testing, and adaptive security testing. Additional work can be done to extend our results by varying the configuration parameters of our experimentation and providing additional metrics for comparing the two LT systems. This real-time approach to LT uses streaming log data to generate and dynamically update user behavior models, cluster them into similar behavior profiles, and instantiate distributed workload of software systems.

REFERENCES

[1] The Apache Software Foundation, "Apache JMeter." https://jmeter.apache.org, 2019. [Online; accessed: 2019-05-19].

[2] D. A. Menascé, V. A. Almeida, R. Fonseca, and M. A. Mendes, "A methodology for workload characterization of e-commerce sites," in *Proceedings of the 1st ACM conference on Electronic commerce*, pp. 119–128, 1999.

[3] D. A. Menasce, "Load testing of web sites," *IEEE Internet Computing*, vol. 6, pp. 70–74, Jul 2002.

[4] A. van Hoorn, "Markov4JMeter - Probabilistic and Intensity-Varying Workload Generation for Session-Based Software Systems." https://www.se.informatik.uni-kiel.de/en/research/projects/markov4jmeter, 2008. [Online; accessed: 2019-05-19].

[5] A. Van Hoorn, M. Rohr, and W. Hasselbring, "Generating probabilistic and intensity-varying workload for web-based software systems," in *SPEC International Performance Evaluation Workshop*, pp. 124–143, Springer, 2008.

[6] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. Krcmar, "Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems," *Software & Systems Modeling*, vol. 17, no. 2, pp. 443–477, 2018.

[7] A. Qusef, L. Issa, E. Ayoubi, and S. Murad, "Challenges and opportunities in cloud testing," in *Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems*, p. 15, ACM, 2019.

[8] BlazeMeter, "What's the Max Number of Users You Can Test on JMeter?." https://www.blazemeter.com/blog/whats-the-max-number-of-users-you-can-test-on-jmeter/, 2019. [Online; accessed: 2019-07-12].

[9] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance engineering for microservices: Research challenges and directions," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, (New York, NY, USA), pp. 223–226, ACM, 2017.

[10] Z. M. J. Jiang, "Load Testing Large-Scale Software Systems," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, pp. 955–956, IEEE, 2015.

[11] Z. M. Jiang and A. E. Hassan, "A Survey on Load Testing of Large-Scale Software Systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, 2015.

[12] P. Leitner and C.-P. Bezemer, "An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects," in *the 8th ACM/SPEC*, (New York, New York, USA), pp. 373–384, ACM Press, 2017.

[13] J. Chen and W. Shang, "An Exploratory Study of Performance Regression Introducing Code Changes.," *ICSME*, 2017.

[14] T.-H. Chen, M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, M. N. Nasser, and P. Flora, "Analytics-Driven Load Testing - An Industrial Experience Report on Load Testing of Large-Scale Systems.," *ICSE-SEIP*, 2017.

[15] M. Jiang, C. Wang, X. Luo, M. Miu, and T. Chen, "Characterizing the Impacts of Application Layer DDoS Attacks," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 500–507, IEEE, 2017.

[16] A. Rafael Lenz, A. Pozo, and S. Regina Vergilio, "Linking software testing results with a machine learning approach," *Engineering Applications of Artificial Intelligence*, vol. 26, pp. 1631–1640, May 2013.

[17] R. S. Shariffdeen, D. T. S. P. Munasinghe, H. S. Bhathiya, U. K. J. U. Bandara, and H. M. N. D. Bandara, "Adaptive workload prediction for proactive auto scaling in PaaS systems," in *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pp. 22–29, IEEE, 2016.

[18] W. Iqbal, A. Erradi, and A. Mahmood, "Dynamic workload patterns prediction for proactive auto-scaling of web applications," *Journal of Network and Computer Applications*, vol. 124, pp. 94–107, Dec. 2018.

[19] R. Gao and Z. M. J. Jiang, *An exploratory study on assessing the impact of environment variations on the results of load tests*. IEEE Press, May 2017.

[20] V. Apte, T. V. S. Viswanath, D. Gawali, A. Kommireddy, and A. Gupta, *AutoPerf: Automated Load Testing and Resource Usage Profiling of Multi-Tier Internet Applications*. Automated Load Testing and Resource Usage Profiling of Multi-Tier Internet Applications, New York, New York, USA: ACM, Apr. 2017.

[21] R. Ramakrishnan, V. Shrawan, and P. Singh, "Setting Realistic Think Times in Performance Testing," in *the 10th Innovations in Software Engineering Conference*, (New York, New York, USA), pp. 157–164, ACM Press, 2017.

[22] C. Vögele, A. Brunnert, A. Danciu, D. Tertilt, and H. Krcmar, *Using performance models to support load testing in a large SOA environment*. New York, New York, USA: ACM, Mar. 2014.

[23] A. van Hoorn, C. Vögele, E. Schulz, W. Hasselbring, and H. Krcmar, "Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems," in *8th International Conference on Performance Evaluation Methodologies and Tools*, ICST, 2015.

[24] C. Vögele, A. van Hoorn, and H. Krcmar, "Automatic Extraction of Session-Based Workload Specifications for Architecture-Level Performance Models," in *the 4th International Workshop*, (New York, New York, USA), pp. 5–8, ACM Press, 2015.

[25] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. Krcmar, "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems," *Software & Systems Modeling*, vol. 17, pp. 443–477, Oct. 2016.

[26] C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, "Exploiting load testing and profiling for Performance Antipattern Detection," *Information & Software Technology*, vol. 95, pp. 329–345, Mar. 2018.

[27] J. Wienke, D. Wigand, N. Köster, and S. Wrede, "Model-Based Performance Testing for Robotics Software Components," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pp. 25–32, IEEE, 2018.

[28] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864, IEEE, 2016.

[29] S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes, Second Edition*. Academic Press, 1975.

[30] W. Tinney, V. Brandwajn, and S. Chan, "Sparse vector methods," *IEEE transactions on power apparatus and systems*, no. 2, pp. 295–301, 1985.

[31] D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.