12-30-2020

Distributed Load Testing by Modeling and Simulating User Behavior

Chester Ira Parrott
Louisiana State University and Agricultural and Mechanical College

DISTRIBUTED LOAD TESTING BY MODELING AND SIMULATING USER BEHAVIOR

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Division of Computer Science and Engineering

by
Chester Ira Parrott
B.S., Computer Science, Southeastern Louisiana University, 2013
May 2021

Dedicated to the strength, spirit, and sacrifices of the fierce women in my life.

Men have become the tools of their tools.

-- Henry David Thoreau

Walden

Acknowledgements

This work was partially funded by the Donald W. Clayton Endowment in tandem with the Division of Computer Science and Engineering. Any journey worth taking can only be completed through the generosity, help, and effort of many other people; as such, I have been given and owe much to many people. Thank you to Dr. Jianhua Chen, Dr. Nash Mahmoud, and Dr. Tao Jin for taking the time to serve on my committee and provide feedback on my research. To my advisor and committee chair, Dr. Doris Carver – thank you for your constancy and patience. You have been an inspiration and an invaluable font of wisdom.

Thanks to the faculty and staff of the Division of CSE for timely sparks of encouragement and help. Thanks also to my undergraduate professors: Dr. Theresa Beaubouef altered the course of my education with one conversation on the power of mathematics; Dr. Aron Culotta, Dr. Patrick McDowell, Dr. Ghassan Alkadi, and Dr. John Burris recommended me to LSU when I was searching for a graduate program. Thank you to Raghu Dodda, who exposed me to log-based metrics and was a soundboard for my ideas.

Special thanks to all my friends and family who have been there for me, even when I could not reciprocate. My grandparents gave up much for my education, although I was too young to appreciate it at the time. My parents imparted to me a stubborn tenacity in the face of adversity and an insatiable thirst for knowledge and understanding. My older brother Harlan dared me to re-imagine myself and to dream of what might be accomplished with discipline applied to my wild creative streak; my older sister June periodically pulled my head out of the clouds to remind me of the importance of family and to never give up.

Thank you to my wife, Robin, for pushing me to finish my bachelor's degree, for supporting my decision to pursue graduate studies, for honoring me with years of love and understanding, and for sharing this journey with me. Your resilient capacity for goodness amazes me every day. Thank you to our wonderful kids Georgia, Ian, Collin, and Molly-Lou – you did not start this journey with us, but you have given up so much time with me so that we could complete it, together. This would not have happened without each of you!

Table of Contents

List of Tables

List of Figures

Acronyms, Terms, and Definitions

Acronym	Term	Definition
$\mathcal{S}$	System	A goal-oriented collection of entities and procedures
N/A	Agent/Entity	An executor of some series of tasks within a given System in order to accomplish given (or discoverable) goals
N/A	Machine	A mechanical or electronic entity capable of executing routine tasks
N/A	Human/User	A biological entity capable of executing routine and higher-order tasks
HMS	Human-Machine System(s)	Systems which are comprised of humans and machines
MAS	Multi-Agent System(s)	Systems which are comprised of human Agents and machine Agents working toward a common goal
N/A	Model	A mathematical representation of an entity or a System
SUO, $\mathcal{S}_o$	System(s) under observation	A production HMS where realistic data for typical behaviors from Agents can be captured for modeling and mining
SUT, $\mathcal{S}_t$	System(s) under test	An HMS strictly reserved for pre-release testing of code and infrastructural changes to detect hidden flaws
LT	Load Testing	A series of procedures for identifying System-wide Load-related flaws within an SUT

Abstract

Modern human-machine systems such as microservices rely upon agile engineering practices which require changes to be tested and released more frequently than classically engineered systems. A critical step in the testing of such systems is the generation of realistic workloads or load testing. Generated workload emulates the expected behaviors of users and machines within a system under test in order to find potentially unknown failure states. Typical testing tools rely on static testing artifacts to generate realistic workload conditions. Such artifacts can be cumbersome and costly to maintain; however, even model-based alternatives can prevent adaptation to changes in a system or its usage. Lack of adaptation can prevent the integration of load testing into system quality assurance, leading to an incomplete evaluation of system quality.

The goal of this research is to improve the state of software engineering by addressing open challenges in load testing of human-machine systems with a novel process that a) models and classifies user behavior from streaming and aggregated log data, b) adapts to changes in system and user behavior, and c) generates distributed workload by realistically simulating user behavior. This research contributes a Learning, Online, Distributed Engine for Simulation and Testing based on the Operational Norms of Entities within a system (LODESTONE): a novel process to distributed load testing by modeling and simulating user behavior. We specify LODESTONE within the context of a human-machine system to illustrate distributed adaptation and execution in load testing processes. LODESTONE uses log data to generate and update user behavior models, cluster them into similar behavior profiles, and instantiate distributed workload on software systems. We analyze user behavioral data having differing characteristics to replicate human-machine interactions in a modern microservice environment. We discuss tools, algorithms, software design, and implementation in two different computational environments: client-server and cloud-based microservices. We illustrate the advantages of LODESTONE through a qualitative comparison of key feature parameters and experimentation based on shared data and models. LODESTONE continuously adapts to changes in the system to be tested which allows for the integration of load testing into the quality assurance process for cloud-based microservices.

Chapter 1. Introduction

My life seemed to be a series of events
and accidents. Yet when I look back, I
see a pattern.
-- Benoît Mandelbrot
A Fractal Life

With the proliferation of the human population, our interconnected devices, and the systems which now compose our infrastructure and every facet of modern civilization, our society has evolved into an ecosystem of humans and machines emitting a profound amount of data every day about how its denizens live and interact. One way in which our human-machine ecosystem records and echoes human behavior is through digital traces produced as a side effect of humans changing or interacting with other people, processes, and technology. From a human-centric perspective, such traces of behavior can be used to describe a population of users as well as the devices with which they regularly interact. However, the scale of data recording human-machine events can be overwhelming in even the simplest of scenarios; moreover, attempts to expand and use the knowledge inside streams of trace data can overwhelm the resources organizations are willing to dedicate toward analysis. As the amount of data produced by our human-machine ecosystem outpaces the ability to consume and understand it, a pressing need exists to change the computational approaches, tool-sets, and mind-sets we use to create and maintain the systems upon which our society relies.

A landmark stride toward improving the approaches associated with creation and analysis of systems is the advent of the cloud. The rapid propagation of cloud technology has sparked the widespread adoption of architectural patterns such as stateless microservices based on representational state transfer (REST) and similar protocols [56]. Microservices rely less frequently upon heavyweight, synchronous, stateful operations preferring lightweight, asynchronous, stateless operations built on serverless computing functionality such as Lambda Functions from Amazon Web Services (AWS) [3] or similar offerings from other commercial cloud-service providers [2, 8, 51, 61, 91].The decrease in cost for computational cycles juxtaposes the increase in availability of technological solutions and knowledge. Dynamically evolving software design choices replacing classical architectures motivate a rationale to change the manner and extent to which modern systems should be evaluated for quality.

1.1. Research Questions

The goal of this research is to improve the state of software engineering by addressing open challenges in load testing of human-machine systems with a novel approach that: a) models and classifies user behavior from streaming and aggregated log data, b) adapts to changes in system and user behavior, and c) generates distributed workload by realistically simulating user behavior. This research is motivated by the following questions:

RQ_1**:** Can a data-driven process for Markov Chain (MC) model-based load testing be developed that offers advantages over existing data-driven processes?

RQ_2**:** Can raw streaming behavioral data in a human-machine system be continuously and efficiently measured, modeled, and stored as MC models?

RQ_3**:** Can aggregated batched behavioral data in a human-machine system be systematically grouped into semantically related MC models?

RQ_4**:** Can MC models of a human-machine system be extended to adapt to changes in a system under observation?

1.2. Challenges of Markov Chain Model-based Load Testing

Current load testing (LT) processes are not capable of swiftly adapting to changes within a system under observation, requiring manual work by quality assurance engineer(s) (QA) for static and modeled processes [13]. Indeed, an automated load testing system capable of mimicking the capabilities of live human testers to adapt to the changes in a system under test does not exist [68]. Most classical automated testing systems are either strictly “record and play” based on system traces, randomized with a backing MC model, or use probabilistic timed automatons which extend the capabilities of a MC model by adding the dimension of thinktime to show how users might pause while moving from state to state to increase workload realism [68, 126].

However, even recent processes to LT are not adaptive or reactive to changes in the system to be tested [13, 77]; moreover, LT of modern microservice architectures provides additional challenges over classical architectures [56]. The current capabilities of cloud-native test systems can partially address the performance concerns listed in [56, 98] through ready availability, swift scale-out/scale-in, distributed computation spanning geographic and infrastructure boundaries to mimic actual users of cloud-based systems, and fine-grained component-level monitoring [88]. Thematic challenges from the works above are:

C_1: As users become more familiar with a system, usage patterns are likely to change which merits an update to a realistic-based testing load. [68]

C_2: As a system evolves over time, model building of fault-inducing loads might be improved through incremental analysis of the system internals. [68]

C_3: LT processes should align with agile development practices such as continuous integration, dynamic deployment, and test automation. [56]

C_4: LT processes should continuously update testing artifacts such as models and scenarios based on changes in the underlying infrastructure, system, and user behavior. [56]

The research suggests that viable processes addressing these challenges should be robust in differing properties; to wit, such processes should use multiple agents and be adaptive, realistic, efficient, and operationally based.

P_1*(Adaptive)*: A process for load testing which is adaptive should be able to automatically evolve, based on changes in the underlying system under observation (SUO).

P_2*(Realistic)*: A process for load testing which is realistic should produce or run load tests which are representative of how the SUO is used, or will be used.

P_3*(Efficient)*: A process for load testing which is efficient should not rely upon algorithms or data structures which preclude testing of arbitrarily large systems.

P_4*(Multiple Agents)*: A process for load testing which uses Multiple Agents should be agent-based and should rely on more than one operating agent.

P_5*(Operationally Based)*: A process for load testing which is operationally based should use data representative of the people, process, and technology of the SUO.

In summary, data-driven processes for realistic model-based load testing should continuously and automatically adapt to behavioral and infrastructural changes in a system to be tested. We examine how existing processes address these challenges.

1.3. Existing Processes

Previous attempts at automated load testing provide valuable insight and direction; however, no process exists which addresses challenges C_1-C_4 or exhibits properties P_1-P_5 listed in Section 1.2. A load testing process which exhibits these properties would exhibit advantages over existing load testing methodologies. We describe previous attempts to better frame the scope of this research.

(2002) Kant et al. [70] show the design of Geist as a means of showing how a MC model can drive a system for stress testing web servers. While Geist is 'dialable', it does not provide a mechanism for automatically adapting to changes in the SUO. Geist is based on operational traffic and provides a requisite level of realism, but does not provide a suitable means for efficiently scaling to 'large'-scale systems. Moreover, Geist does not allow for multiple running agents or modeling of user behavior; rather, it replays traffic which has previously been observed in the system similar to Selenium [19].

(2002) Menasce [84] shows methods which are not adaptive in nature, but rely upon manual changes to the load testing process. Menasce's process is based on virtual users which run in a load generating program. However, this process is not efficient, as the load generating program is not designed for distributed computing or for using multiple agents to generate the requisite workload. It also is not specifically designed for inducing faults; rather, it focuses on means to replay traffic behavior based on manually created customer/user models. As the load testing process is based on replaying of behavior which has been previously observed, it is realistic in nature.

(2005) Canfora et al. [23] use genetic algorithms in order to provide adaptive responses to change in the system under test (SUT). Their methods are not based in operational behavior or behavioral models, preferring a prescriptive rule-based process instead. Genetic algorithms are unconstrained [23], thus efficiency is not baked into this process, although the authors seek to constrain the boundaries of the genetic algorithms used. The genetic algorithms used do not use multiple agents due to performance limitations. However, the method is designed to elicit faults through the dynamic genetic algorithm process and will optimize toward finding such faults.

(2007) Barros et al. [9] show a process for load testing which is based on operational data and MC models to perform cached replaying of the behavior of the SUO. The process is not efficient in that models have the potential for becoming larger than is computationally feasible for many organizations, and the authors do not provide a means for compensating; moreover the process does not use multiple agents or a distributed framework for executing workload. The process is designed for benchmarking systems. However, it is operationally based as it uses logs from the SUO as input for the underlying MC models.

(2007) Penta et al. [37] use genetic algorithms to optimize against the Quality of Service (QoS) constraints given. Their algorithm is capable of changing and optimizing toward the goal of violating those constraints, thus the load testing process is adaptive. However, the process is not realistic in that it is focused on violating QoS and is not based on any realistic behavior patterns. Genetic algorithms are not inherently efficient due to the large state space they cover. The process by Penta et al. is not designed to use multiple agents or distributed computation. Finally, the process seeks to induce faults related to QoS and is not based on operational data, rather on states in the SUO.

(2009) Gu & Ge [53] use genetic algorithms as a means for load testing as they seek to have the process adapt to changes in the SUO. This process is not realistic as it searches over tests for suitable candidates rather than working with realistic behavior patterns. Genetic algorithms are not inherently efficient due to the large state space. The process is not designed for distribution

as it runs offline. Genetic algorithms are not by default agent-based; also, the process is not distributed. As with Penta et al.[37], the load testing process used in this process seeks to violate QoS restrictions. The process focuses on system modeling based explicitly on QoS, not on actual program execution or operational data.

(2012) Zhang et al. [143] have a compositional process for load testing which is adaptive in nature based on the analysis of system components. Their focus on the system components does not provide a realistic user-behavioral point of reference for load testing; however, it does lend itself gracefully to distributed workflows. However, the state space of the symbolic execution mechanism proposed can grow beyond the boundaries of what we would consider an 'efficient' load testing process. The compositional process is not inherently multi-agent based; however, each subcomponent could be analyzed by an agent with different goals and rewards. The process is not based on log data; rather, their method is based on system component information.

(2015) Cotreneo et al. [35] describe RELAI which does not adapt to changes in the SUO as it does not ingest additional data for behavior modeling after the first ingestion. RELAI supports usage of operational/usage profiles, but is structured for running on a single test machine, rather than being distributed. RELAI is not multi-agent based, neither is it designed to specifically find faults in the SUT; however, RELAI is based on operational data such as user profiles or machine profiles.

(2015) Schur et al. [105] do not provide a mechanism for automatically adapting to changes in the SUO. When defining ProCrawl, they specified challenges related to 'adapting' the underlying models to a new SUO-based operational profile. The process can be considered realistic as is based on user/system behavioral data to automate testing. ProCrawl does not work in a distributed fashion, thus scaling the process to a large-scale system is infeasible. ProCrawl uses operational data; however, it is not designed to work with multiple agents or in a distributed manner. ProCrawl focuses on extracting operational models from software in which multiple users are working through concurrent workflows.

Table 1.1. Qualitative Comparison of Existing Load Testing Processes

Author	P_1 Adaptive	P_2 Realistic	P_3 Efficient	$P_4$1 Multiple Agents	P_5 Operationally Based
Barros et al. [9]		✓			✓
Canfora et al. [23]	✓				
Cotreneo et al. [35]		✓			✓
Gu & Ge [53]	✓				
Kant et al. [70]		✓			✓
Menasce [84]		✓			✓
Penta et al. [37]	✓				
Schur et al. [105]		✓			✓
van Hoorn [58]		✓			✓
Vogele et al. [126]		✓			✓
Zhang et al. [143]	✓				

(2008, 2016) van Hoorn, Vogele et al. [58, 126] proposed Markov4JMeter as a MC method for realistic load testing; however, Markov4JMeter does not provide a mechanism for automatically adapting to changes in the SUO. Manual pre-processing is required to create the operationally based models used for executing LT in Markov4JMeter; however, these MC models are realistic when executing against the SUT. Markov4JMeter is based on JMeter [117] - a tool known in industry for having efficiency limitations [15] and operating on a single machine rather than multiple agents. This process was extended in 2016 with WESSBAS [126] which centers on a domain specific language for mining MC models from operational data. However, WESSBAS does not have the capability for continuous adaptation to the SUO, and still requires offline pre-processing and modeling of data.

A summary of these processes and how they relate to properties $P_1 - P_5$ defined in Section 1.2 is listed in Table 1.1. Upon review of the existing processes for automated load testing, a process to distribute load testing by modeling and simulating user behavior which successfully addresses all of the challenges C_{1-4} listed in Section 1.2 does not exist.

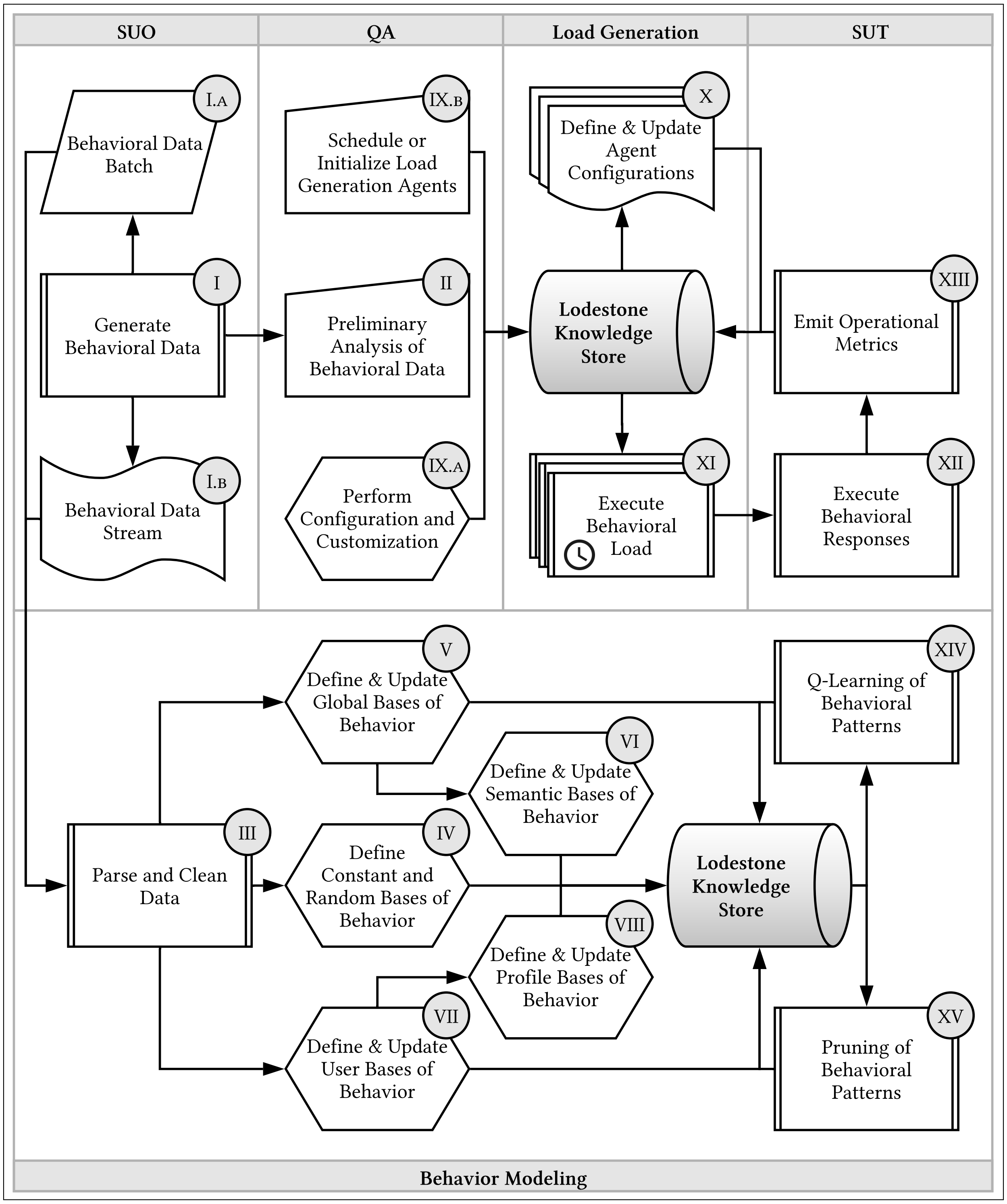

SUO
QA
Load Generation
SUT
I.A
Behavioral Data Batch
IX.B
Schedule or Initialize Load Generation Agents
X
Define & Update Agent Configurations
I
Generate Behavioral Data
II
Preliminary Analysis of Behavioral Data
Lodestone Knowledge Store
XIII
Emit Operational Metrics
I.B
Behavioral Data Stream
IX.A
Perform Configuration and Customization
XI
Execute Behavioral Load
XII
Execute Behavioral Responses
V
Define & Update Global Bases of Behavior
XIV
Q-Learning of Behavioral Patterns
VI
Define & Update Semantic Bases of Behavior
III
Parse and Clean Data
IV
Define Constant and Random Bases of Behavior
Lodestone Knowledge Store
VIII
Define & Update Profile Bases of Behavior
VII
Define & Update User Bases of Behavior
XV
Pruning of Behavioral Patterns
Behavior Modeling

Figure 1.1. Sequence Diagram of LODESTONE Process

1.4. LODESTONE Process

We present a Learning, Online, Distributed Engine for Simulation and Testing based on the Operational Norms of Entities within a system (LODESTONE). The process can be implemented in any distributed computing environment where there exists a SUT separated from a SUO and operational logs can be mined from the SUO. A more detailed overview of our process is illustrated by the diagram in Figure 1.1. LODESTONE requires the following end-to-end sequential steps continuously operating to generate workload against the SUT:

I. Generate Behavioral Data: Event data are produced by the interactions between people, process, and technology within the SUO. These data can be aggregated into a preliminary form or atomically encapsulated in a stream of events.

A. Behavioral Data Batch: A pre-aggregated set of events periodically emitted by the SUO.

B. Behavioral Data Stream: A raw set of events continuously emitted by the SUO.

II. Preliminary Analysis of Behavioral Data: Performed by subject matter expert to inform the parsing, cleaning, configuration, and initialization of load generation agents.

III. Parse and Clean Data: After preliminary analysis of input data is completed, an appropriate parsing model formats the data, and extraneous data points are cleaned from the input data.

IV. Define Constant and Random Bases of Behavior: Based on the possible set of states in the SUO, define MC models which may randomly transition between states or execute observed state transitions with equal likelihood.

V. Define & Update Global Bases of Behavior: Based on the possible set of states in the SUO, define MC models which may execute observed state transitions based on previously observed likelihood across all users.

VI. Define & Update Semantic Bases of Behavior: Based on the possible set of states in the SUO, define MC models which subdivide the global basis of behavior based on semantically linked portions of the SUO.

VII. Define & Update User Bases of Behavior: Based on the possible set of states in the SUO, define MC models which may execute observed state transitions based on previously observed likelihood for individual users.

VIII. Define & Update Profile Bases of Behavior: Based on the possible set of states in the SUO, define MC models which may execute observed state transitions based on previously observed likelihood for users with similar behavior patterns.

IX. Manage Load Generation: The QA may perform additional steps to manage load generation.

- *A. Perform Configuration and Customization*: The QA may perform configuration to improve the performance or operation of load generation.
- *B. Schedule or Initialize Load Generation Agents*: The QA may schedule or change load generation.

X. Define & Update Agent Configurations: Given the set of observable states in the SUO, and the MC models generated in IV-VIII, define an appropriately sized number of agents based on expected behavior for the SUO.

XI. Execute Behavioral Load: Multiple distributed agents interact with the SUT by executing behavior and volumetric state transitions previously recorded or customized by QA.

XII. Execute Behavioral Responses: The SUT provides responses for each state transition to the load generation agents, to include if the requested state transition resulted in failure.

XIII. Emit Operational Metrics: The observed number of responses, errors, and response time are recorded by the load generation agents for additional analysis and improvement.

XIV. Q-Learning of Behavioral Patterns: The metrics generated by the SUT are modeled according to the Q-Learning reinforcement learning algorithm for load generation agent usage.

XV. Pruning of Behavioral Patterns: As additional data come from the SUO, behavioral patterns and state transitions which are no longer in operation should be removed to conserve operational resources.

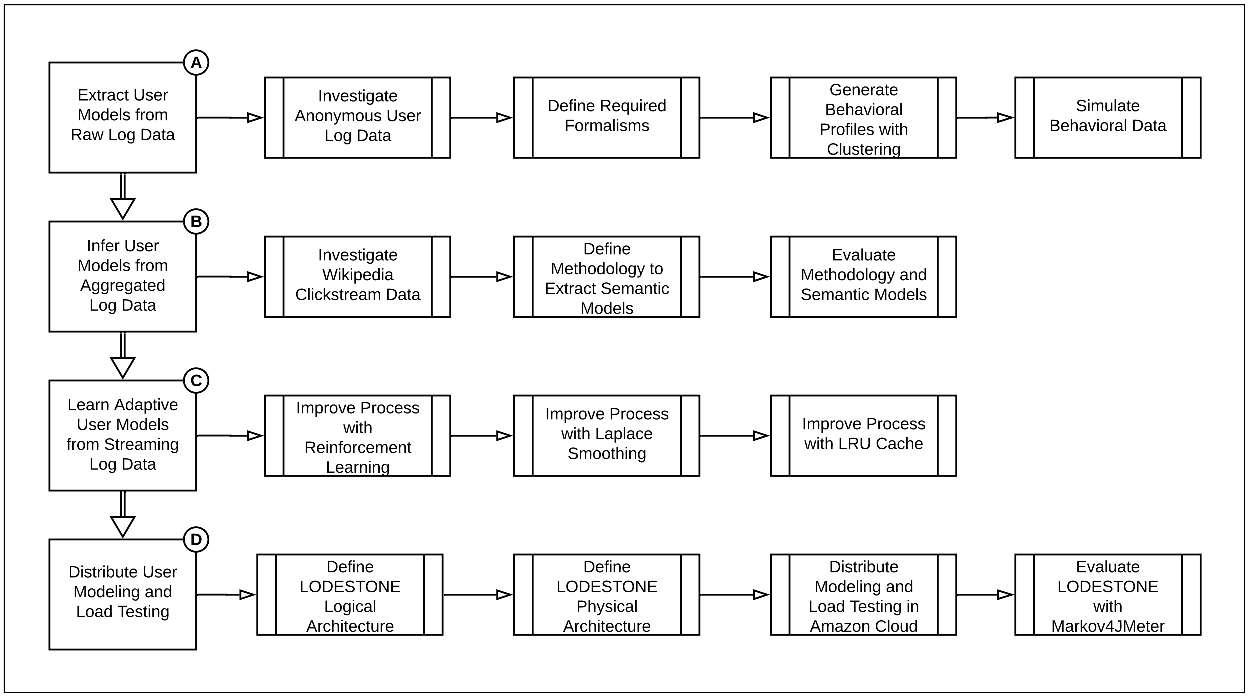

A
Extract User Models from Raw Log Data
Investigate Anonymous User Log Data
Define Required Formalisms
Generate Behavioral Profiles with Clustering
Simulate Behavioral Data
B
Infer User Models from Aggregated Log Data
Investigate Wikipedia Clickstream Data
Define Methodology to Extract Semantic Models
Evaluate Methodology and Semantic Models
C
Learn Adaptive User Models from Streaming Log Data
Improve Process with Reinforcement Learning
Improve Process with Laplace Smoothing
Improve Process with LRU Cache
D
Distribute User Modeling and Load Testing
Define LODESTONE Logical Architecture
Define LODESTONE Physical Architecture
Distribute Modeling and Load Testing in Amazon Cloud
Evaluate LODESTONE with Markov4JMeter

Figure 1.2. Research Roadmap

LODESTONE relies upon the usage of multiple distributed agents, operational data, and efficient algorithms to create a realistic and adaptive workload against the SUT. It combines methods for modeling user behavior from raw and aggregated data with algorithmic optimizations for adapting load generation as the SUO changes.

1.5. Contributions

The expense and difficulty associated with classical load testing methods have become outmoded by the ubiquity and speed with which software systems are developed and deployed. At its core, this research has followed the steps shown in Figure 1.2 to contribute LODESTONE: a novel process for testing systems based on the data generated through interactions between users and those systems that exhibits the properties P_{1-5} from Section 1.2.

A. Extract User Models from Raw Log Data: The inception of this research was an investigation of anonymous user data and a definition of formalisms supporting the overall process. Following this investigation was the creation of models from the anonymous data and showing the reduction of the quantity of models through clustering. In order to examine the quality of the clustered models, it was necessary to simulate behavioral data with an independently developed mechanism.

B. Infer User Models from Aggregated Log Data: The user and profile models created with the previous step of this research cannot be directly inferred from aggregated data due to the lack of individual identifiers for disambiguating observed behaviors. An investigation into Wikipedia's publicly available clickstream data was necessary to address those situations where only aggregated data are available. A methodology and algorithm were constructed to extract semantic models from such aggregated data and was evaluated as part of the research.

C. Learn Adaptive User Models from Streaming Log Data: In order to allow the models produced in the previous two steps of this research to adapt to changes in the SUO, improved data structures formed the core of the computational requirements. These data structures met the requirements of the formalisms previously described, and were augmented with reinforcement learning, Laplace smoothing, and a least-recently-used cache to solve for various scenarios where adaptation were necessary.

D. Distribute User Modeling and Load Testing: The final step of this research combined the results of the formalisms, analyses, algorithms, data structures, and tooling developed in the previous steps. A logical and physical architecture for how LODESTONE should operate and be structured preceded an implementation on Amazon Web Services and an evaluation against a contemporary tool widely used for load testing. [3]

In creating and evaluating LODESTONE, several directed aspects of the research were required for the process to be instantiated, completed, and evaluated; these aspects were as follows:

- A set of terminology and formalisms defining the parameters, data, and models to be amalgamated by LODESTONE and informed by an ad-hoc analysis of several anonymous public user behavioral datasets.
- A data ingestion process using the DBSCAN clustering algorithm to detect and model similar users by their behavior patterns.
- A functional microservice (TinyERP) and test oracle (Loki) written in a production-grade software framework (Spring Boot) for realistic evaluation of load testing processes, continuous generation of performance and behavioral data, generation of rule-based behavioral traffic, and controlled simulation of errors in the system.
- A collection of streaming statistics algorithms which provide dataset statistics in O(1) space and O(1) time.
- A Depth Constraint extension of Tarjan's STRONGCONNECT algorithm for efficient segmentation of aggregated log data into smaller subgraphs of related user behavior and executed on aggregated Wikipedia data.
- An extension of the formalisms of Chapters 2 and 3 through the Q-Learning reinforcement machine learning algorithm, supported by a proof of optimality.
- An extension of SparseVector and SparseMatrix data structures to address the "sunrise problem" of events which have never been observed but should have a non-zero probability of being simulated through Laplace smoothing.
- Addition of a least-recently-used event cache to allow for efficient reduction of inactive, invalid, or unlikely historical event transitions.
- Provision for the Kullback-Leibler Divergence for determining if clustered models provide the same level of informational representation as the raw data used by classical load testing processes.

- Requirement of a system to monitor and learn from in addition to the system to be tested; such a requirement, not appearing elsewhere in the literature, enables an implementation of LODESTONE to accept streamed data from a production environment for continuous adaptive testing of a system under test.
- The LODESTONE process as an implementation in Amazon Web Services with a logical architecture and a physical architecture.
- Qualitative and quantitative comparison against a ubiquitous load testing tool JMeter using identical models learned from rule-based behavioral data generated by TinyERP.
- A distributed load testing process which is adaptive, realistic, efficient, operationally based, and supports multiple-agent execution.

These aspects of the LODESTONE process delineate how raw system data may be collected in their various forms to efficiently learn and simulate interactions between the humans and machines in a real human-machine-system. LODESTONE may be used toward solving problems in fields such as cybersecurity, emergency response, product management, and user experience. However, with this research we tender theoretical concepts of and concrete evidence concerning the understanding of user behavior data to solve load testing challenges in software engineering.

Chapter 2. Background

The longer you can look back, the
farther you can look forward.
-- Winston Churchill
Road to Victory Speech, March 1944

We briefly overview the modeling of human-machine systems and the simulation of Agency within human-machine systems; additionally, we describe approaches to realistic generation of workload on software systems which use similar theories, mathematical structures, and computational structures as our framework.

2.1. The Modeling of Human-Machine Systems

We define a human-machine system (HMS) to be a collection of people, process, and technology working together within a larger organization of entities to accomplish a prescribed or emergent goal. Work has been done to model different entities within HMS in order to understand and predict how the HMS will operate under changing conditions. We posit that the growing co-dependency of humans and machines in our technological society allows for both sets of entities to be modeled and simulated as Agents within an HMS. Multi-Agent Systems (MAS) are detailed in texts such as the work on by Shoham and Leyton-Brown [109] as models with distributed Agency which are intended for the simulation and solving of problems. For more detailed exploration of mining for business processes and business roles, we refer the reader to the works by van der Aalst [120, 121] for process mining, and Colantonio et al. [30] for role mining. For additional background in load testing of software systems, we direct the reader toward Jiang and Hassan's systematic literature review on the subject [68]. We overview select works in modeling of people, process, and technology.

2.1.1. People

We discuss traditional methods of modeling user-specific Agents responsible for creating change within the SUO. One of the primary works on user profiling was done by Elaine Rich, who detailed a data-driven approach to modeling multiple users for clustering [102] resulting in a static model akin to a decision tree for user representation. Such user-profiling has been applied to security by the National Institute of Standards and Technology standard definition of Role-Based Access Control [44] which extended Ferraiolo's original work on the subject in [43]. Frank et. al. define two creating role-based profiles of users: the top-down prescriptive approach and the bottom-up descriptive approach [47]. The prescriptive approach is usually an effort-intensive manual process [54]; the bottom-up approach is usually data-driven, but suffers from over-generality or over-specific results. Middleton et. al. show a knowledge-based ontological method for profiling users within recommender systems such as used by Netflix [86].

2.1.2. Process

The collection of workflow steps, conditional rules, and reactions is another key element to movement of data and progress within an HMS which is done through process management and mining[36]. According to van der Aalst, such workflows are typically used for simulation purposes or general intelligence on the behavioral characteristics of an HMS for improvement purposes [121]. Pecarina described APSAT as a framework for mining and managing workflow processes from network traffic [95, 96]. Li, Kang, and Lv show a method [78] for dynamic business process mining from event data. Fei, Liqun, GuangYun, and Xiaolei describe an algorithm for detecting concept drift in process mining [42]. Such detection is useful for determining changes in existing business processes. Bose et. al. discuss a method for dealing with concept drifts in process mining meant to deal with changes over time [16]. As such, data-mining is a feasible means of modeling processes in an HMS; however, care must be taken to address changes in the HMS in order for simulation and studies to remain accurate after the mining process has completed.

2.1.3. Technology

Modeling technical components in an HMS is typically used for anomaly detection of system-level behaviors and performance characteristics[26, 57, 60, 69, 107, 111, 141]. Static means of using operational models have been used in software engineering practices such as Whittaker's approach to Cleanroom Software Engineering [128]. Tools such as ProCrawl can extract behavior models from web applications [105], and RELAI framework is able to use a probabilistic model based on operational profiles in order to improve software reliability [35]. Genetic algorithms are used to test Service Level Agreements and QoS through LT [23, 37, 53, 143]; however, these approaches are not directly using user-based behavior models to generate application workload. Anderson's novel approach using a MC representation of sequential instructions is described in [4] where instruction combinations are used as clustering features for categorizing various families of malware.

2.2. Markov Modeling for Load Testing of Systems

Markov Chain models are a subset of stochastic processes based on descriptive statistics; however, they can be additionally used in various real-life contexts such as anomaly detection modeling and Agent behavior. Karlin and Taylor define a stochastic Markov Process as:

> A stochastic process is a Markov process when
>
> $$Pr\{a < X_t \leq b | X_{t_1} = x_1, X_{t_2} = x_2, \ldots, X_{t_n} = x_n\}$$
> $$= Pr\{a < X_t \leq b | X_{t_n} = x_n\}$$
>
> where $t_i \in T$, and $t_1 < t_2 < \ldots < t_n < t$. A discrete time Markov Chain (MC) $\{X_n\}$ is a Markov process whose state space is a countable or finite set, and for which $T = (0, 1, 2, ...)$. It is customary to speak of X_n being in state i if $X_n = i$. The probability of X_{n+1} being in state j, given that X_n is in state i, is noted by $P_{ij}^{n,n+1}$, i.e.
>
> $$P_{ij}^{n,n+1} = Pr\{X_{n+1} = j | X_n = i\}.$$

> As such, $P_{ij}^{n,n+1} = P_{ij}$ is independent of n, and P_{ij} is the probability that the state value undergoes a transition from i to j in one trial. It is customary to arrange these probabilities as a matrix, that is, a square array. We further refer to $\mathbf{P} = ||P_{ij}||$ as the Markov matrix, or transition probability matrix of the process. As the measurable quantity of states is finite, the quantity of rows is equivalent to the quantity of states. [72]

Probabilistic methods such as shown by Menasce in [84], Barros et al. in [9], and Kant et al. in [70], use a MC model for generating background web traffic. However, in [70], we observe a looming fault in typical MC approaches: that individual users are modeled individually. Individual modeling of users cannot easily scale in modern web-based software LT. The inherent lack of scalability that such over-abundance of individualized models implies leads us to suggest means of clustering user behavior profiles, such as those shown by Barth to discover groups in wikis [10], Xiong et al. for clustering sequences of categorical data [137], Melnykov for clustering of clickstream data [83], and Benson for clustering of higher order mathematical models such as tensors [12]. Such clustering is performed to reduce the number of individual models being used and thus unnecessary performance overhead. The direction on MC clustering for model-based LT by Menasce [84] was followed by van Hoorn with Markov4JMeter [58] and WESSBAS [59].

The work by Barros et al. [9] shows that static LT methods must be able to react to changing user behaviors, and the underlying probabilistic models should be updated periodically to accurately represent the SUT. Such static methods provide valuable insight into how the system performs under certain types of load; however, dynamic interaction with the SUO provides additional insight into how the SUT reacts to different types of load which might not be accounted for by QA. While static methods might provide information on one particular part of the system, they do not always provide a broad view of how the system might holistically react under a realistic production-level load or react to change within an observed HMS. Change within an observed HMS is also studied in newer work by Bezemer et al. and Schulz et al. [13, 104] who state that the problem of stale data exists even in modern DevOps environments, preventing automated model-based LT from being adopted into DevOps and Agile development environments. The work by Schulz et al. represents a prescriptive effort such as the top-down approach in role mining to mapping the change delta C_δ within a continuous integration environment [104]. In order to provide a dynamic approach to addressing stale, missing, and incomplete data, we extend descriptive MC models with formalisms, efficient algorithms, and inferential methods such as those given by machine learning.

2.3. Concept Modeling

We show relevant methods and ideas from process mining, user behavior analysis and modeling, web application modeling, and conceptual models of knowledge (ontology). Additionally, we show previous work using Wikipedia data which is relevant to our methods and results.

The term 'ontology' found its mathematical underpinnings with the work of Wille, who established foundations for what he termed 'Concept Lattices' [131, 132, 133]. This series of works focused on providing an applications-centric bridge to the field of lattice theory with the key contribution being a formal definition of how concepts can be structured. The commonly used description of information-specific ontology came when Gruber listed [52] a formal process for the design and subsequent care of knowledge representation in the form of ontology. Gruber described ontology as "a specification of a conceptualization: in other words, a formal means of representing how knowledge and concepts are interrelated" [52]. With the creation of Wikipedia as an online knowledge store (ca. 2001), the global user community of the site has generated large amounts of domain-specific content. While Voss [127] states that Wikipedia articles do not have a strict semantic structure as do computer scientific ontologies (as described by Gruber in [52]), such structure can be inferred from the similarities between articles and categories. Since Wikipedia's inception, efforts have focused on extracting such formal concept specifications from the Wikipedia content. Punuru and Chen [97] show methods for extracting concepts from domain-specific documentation, and Schönhofen [103] details how cross-referencing Wikipedia titles and categories with document words can be used to classify the domain of documents. Similar efforts using natural language processing [63],[135],[134] have focused on extracting ontological structure from the site by the text inside the articles. However, other approaches such as Nakayama, Hara, and Nishio in [89] and Syed, Finin, and Joshi in [113] focus on the categories associated with each article, the resultant hierarchy of concepts on the site, and graphs of both the links and concepts extracted from the articles. One user study related to Wikipedia consumption by Yu, Thom, and Tam in [142] describes how users of the site tended to have exploratory patterns with consistent ‘backtracking’ (navigation from the current page to the previous one) when the navigational information of the site was not clear enough to complete various research-related tasks (assigned as part of the study). The results indicate that backtracking represents the single largest percentage of user behavior on the Wikipedia site: users locate a ‘gateway’ page (a topical anchor point for basing further searches) and iterate through the page links until the next ‘gateway’ can be located [142].

Wikipedia has released aggregated clickstream data to aid in deeper understanding of Wikipedia users’ behavior; this data may be used to improve upon previous Wikipedia-based ontological efforts. From [102], Rich proposes one of the original works on using a data-driven approach to modeling multiple users for clustering. This knowledge based approach results in a static model akin to a decision tree for user representation. Middleton, Shadbolt, and De Roure in [86] show a knowledge-based ontological method for profiling users in recommender systems. Anderson’s novel approach [4] using the MC representation of sequential actions is used for classification of malware. Instruction combinations are used as features for clustering malware families.

Static means of using such models of processes and usage have been used in software engineering practices such as Cleanroom Software Engineering [128]. Schur, Roth, and Zeller show how their tool ‘ProCrawl’ can extract behavior models from web applications [105], and Cotreneo et al. show that their ‘RELAI’ framework is able to use a probabilistic model based on operational profiles to improve software reliability [35]. Probabilistic methods proposed by Menasce in [84], Barros et al. in [9], Kant, Tewari, and Iyer in [70] use a MC model for generating behavioral traffic. However, in these approaches, each user’s behavior is represented by a unique model: an approach which does not scale when taking into account the volume of traffic for Wikipedia.

In order to reduce the number of individual models being used (and thus unnecessary performance overhead), many methods have been shown to cluster user behavior. Notable works include Barth in [10] who used social network analysis to identify groups of users in wikis (such as Wikipedia); Xiong, Wang, Jiang, and Huang in [137] showed how to use MC models to cluster sequences of categorical data (such as user behavioral trace data); Melnykov in [83] showed a method for clustering clickstream data; and Benson, Gleich, and Leskovec in [12] describe a means of segmenting network structures (such as our clickstream behavioral graph) using a tensor-based representation of the data and spectral clustering. Lee, Ellis, and Loui in [75]) show a MC model for clustering which is capable of detecting semantic concepts in audio data. Ahlers, Dirk and Mehrpoor show methods of semantically searching shared information in [1].

Community detection algorithms are another means of detecting subgroups or clusters inside of graphs. Yang et al. discuss different community detection algorithms and their performance in [140]. These algorithms have been implemented and tested in the R programming platform within the 'igraph' package.

As nomenclature for measuring algorithmic complexity, let E as the number of edges in the subject graph, and N to be the number of nodes, or vertices. The 'Edge betweenness' algorithm focuses on using Freeman's betweenness centrality as a means of filtering out edges which are highly shared between different communities in order to extract those highly shared edges from the larger graph. The algorithm runs in $O(E^2N)$[140]. 'Fastgreedy' runs in $O(Nlog^2(N))$ and starts with every node in the subject graph representing a single-member cluster or community then proceeds to pairwise match communities until a 'modularity' metric is no longer maximal[140]. 'Infomap' runs in $O(E)$ and uses decoders to reconstruct a view of the graph based on information collected from random walks on the network graph[140]. 'Label propagation' also runs in $O(E)$ and relies upon a majority heuristic by assigning tokens to each node in the graph and iterating until each node has the same token as the majority of its neighbors[140]. 'Leading eigenvector' runs in $O(N^2)$ on sparse graphs (where the number of edges is much lower than the number of nodes), and $O(N(N + E))$ otherwise; it relies upon using eigenvalues and eigenvectors of the modularity matrix to maximize modularity iteratively until there is less than an ϵ level of increase in modularity[140]. The ‘Multilevel’ algorithm is similar to ‘Fastgreedy’, but runs in $O(NlogN)$[140]. ‘Spinglass’ runs in $O(N^{3.2})$ and can use simulated annealing to find communities[140]. ‘Walktrap’ runs in $O(N^2log(N))$ for sparse graphs and $O(EN^2)$ otherwise; it also relies upon random walks of the network and works on edge degree distance between nodes to hierarchically merge communities pairwise[140].

Another effective algorithm to segment a graph is the STRONGCONNECT algorithm described in [116]; this algorithm works to extract strongly connected components from a directed graph, and weakly connected components from an undirected graph (all of which are further discussed in [38]). STRONGCONNECT runs in $O(N + E)$ and is deterministic in nature. Building on the approaches used in these algorithms, we show how the Wikipedia clickstream data are structured and how MC user behavior modeling and graph clustering can be used to extract semantic clusters from the Wikipedia.com website.

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 [67] as well as the collaboration between Jiang and Hassan [68]. In addition, we suggest a review of Leitner and Bezemer’s work [77] for a recent overview of industrial tooling. Chen and Shang [27] 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. [28] 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 [66]. It is important to be able to evaluate the performance and quality characteristics of the load and performance testing frameworks themselves.

2.4. Metrics and Validation

When performing automated performance or LT, there are a number of approaches for validating and comparing the specific sort of testing methods being executed and optimizing the variety and volume of the data being analyzed. Malik et. al [82] describe means of using Principal Component Analysis to reduce the total number of metrics required for analysis by performance quality analysts. Nikravesh et al. [90] show how various workload patterns can be used to determine the comparative accuracy of neural network and support vector machine approaches to optimizing scale in and scale out of computing resources. One means of comparing various LT mechanisms is proposed by Gao et al. [49] in order to meet large-scale performance characteristic goals in cloud based systems. Ferreira et al. [45] propose a tool to perform similar levels of optimization in Service Oriented Architectural (SOA) environments. Two additional approaches to validating and assessing LT frameworks are proposed by Avritzer et al. in [7] and Laaber et al. in [73]. Where Avritzer et al. focus on the approach, Laaber et al. show how various suites perform for continuous assessment. For additional context related to our work, we describe model-driven and adaptive methods.

2.5. Adaptive Load Testing

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. [99] 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. [108] describe adaptive auto-scaling strategies to meet performance goals; similarly, Iqbal et al. [62] 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).

2.6. Model-Based Load Testing

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 [50]. Apte et al. describe *AutoPerf* [5] for modeling performance metrics of a system under test while simultaneously driving the performance testing process. While *AutoPerf* requires profiling and testing to be two separate processes, LODESTONE allows for online statistical performance profiling. Kapoor et al. describe a statistical analysis and model of user behavior within a website [71] which is based on the hazard function and has been observed within various production systems (the authors have maintained) as a valid means of approximating the amount of time users spend interacting with a system. Ramakrishnan et al. [100] 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 [59, 122, 124, 125, 126]. Trubiani et al. in [119] 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 [129] for modeling performance testing in robotics.

In summary, the modeling of user, machine, operational, and performance characteristics of a system is a powerful means of improving that system's measurable quality and maintainability. For better results in a prospective automated performance testing process, adaptive methods should be included for improved speed in testing feedback.

This chapter has described various theoretical underpinnings, processes, and improvements to the core of system modeling for LT which inform and catalyze the techniques in LODESTONE for modeling user behavior from system logs.

Chapter 3. Extracting User Models from Log Data

In minor ways we differ, in major
we're the same.
-- Maya Angelou
Human Family

We define terminology for MC behavior models and supporting structures. With the exception of necessary clarification between the SUO and the SUT, the underlying systematic framework we describe below is similar to that used by Menasce [84] and van Hoorn [58]. However, the separation of systems is in the same spirit as the continuous integration pipelines described by Schulz et al. [104]. We recall the underlying mathematical structures for extension and show how they fit into a logical framework for learning and extending MC behavior models for LT from event data.

3.1. Systems Under Observation and Test

We employ the following formalisms to define how behaviors within a system $\mathcal{S}$ can be represented in the context of this work. Simply put, $\mathcal{S}$ should consist at the minimum of a set of actions, a set of states, and a transition function in order to match a minimalistic finite state machine definition. We expect that operation of $\mathcal{S}$ produces output logs representing the current operational status and Agent behaviors. We extend $\mathcal{S}$ with additional formalisms to support streaming log-based behavior profiling.

Actions (A) = $\{Browse(B), Read(R), Add(A), Edit(E), Destroy(D)\}$
States (S) = $\{External, Login, Logout, Home, Account, Configuration\}$
Roles (R) = $\{Auditor, User, Business\ Administrator,$
$Technical\ Administrator, Super\ Administrator\}$
Proficiency (P) = $\{Untrained, Novice, Competent, Expert, Guru\}$

Figure 3.1. Sample System Definition

For an example of how a sample system can be structured, refer to Figure 3.1. The sample system represents an enterprise resource planning website and contains six states which may be accessed by users interacting with the system. The users are subdivided into several roles which represent how they might need to access different states within the system. Each user has a level of proficiency which represents how familiar they are with the system based on the amount of time they need to perform the different actions in the system. The actions allowed to a user enables them to interact with the site by changing the underlying data storage mechanisms while maneuvering between the states of the system.

With the example from Figure 3.1 in mind, we define the system $\mathcal{S}$ such that:

$$\mathcal{S} = \{A, S, R, U, P, \mathcal{E}_k, \mathcal{E}_u, \phi_p, \phi_l, \phi_\mu, \phi_k, \phi_u\}.$$

Let A be the set of all actions which can be taken within $\mathcal{S}$, S be the set of all observable states inside $\mathcal{S}$, R be the set of all behavioral roles in $\mathcal{S}$, U be the set of users which interact with $\mathcal{S}$, and P be the proficiency that each user $u \in U$ can assume while interacting with $\mathcal{S}$. Let $\mathcal{E}$ be the set of all possible errors in $\mathcal{S}$. We classify such errors into two disjoint subsets of $\mathcal{E}$: known errors $\mathcal{E}_k$, and unknown errors $\mathcal{E}_u$.

The following functions represent how each of the above sets are used to model the behaviors of entities in $\mathcal{S}$.

Transition Function

Let $\phi_p : S \times S \times A \times R \rightarrow \mathbb{R}_{[0,1]}$ be a probability function such that, given a source state, a target state, an action, and a role, returns a probability between 0 and 1 that the action will take place.

Proficiency Function

Let $\phi_\mu : S \times S \times A \times R \times P \rightarrow \mathbb{N}_{[0,\infty)}$ be a probability function such that, given a source state, a target state, an action, a role, and a proficiency, returns the discrete mean transition time between the states for the given input tuple in seconds between 0 and ∞.

Role Function

Let $\phi_l : S \times S \times A \times R \rightarrow \{0, 1\}$ be a binary function such that, given a source state, a target state, an action, and a role, returns a 1 if the input tuple represents a legal transition and a 0 if the input tuple represents an illegal transition. Such roles can be visualized as in Figure 3.2 and Figure 3.3 using the sample system defined in Figure 3.1. In Figure 3.2, the 'Super Administration User' is capable of performing every action while in states ϕ_2, ϕ_4, and ϕ_5; in contrast, Figure 3.3 shows how a sample 'User' is capable of reading and browsing state ϕ_5, but cannot even access ϕ_4.

Known Error Function

Let $\phi_k : S \times S \times A \times R \rightarrow \{0, 1\}$ be a binary function such that, given a source state, a target state, an action, and a role, returns a 1 if the input tuple represents a known error transition and a 0 otherwise. A known error transition represents a static set of discoverable or pre-defined logical flaws in $\mathcal{S}$.

Unknown Error Function

Let $\phi_u : S \times S \times A \times R \rightarrow \{0, 1\}$ be a binary function such that, given a source state, a target state, an action, and a role, returns a 1 if the input tuple represents an unknown error transition and a 0 otherwise. An unknown error transition represents a flaw which is not statically defined in $\mathcal{S}$ through the use of ϕ_k.

The change delta C_δ between $\mathcal{S}_t$, and $\mathcal{S}_o$ may affect any of the sub-components of S, including the functions. It is of note that ϕ_u, ϕ_k are not explicitly defined in $\mathcal{S}_o$, but they implicitly exist if the rest of the conditions above are followed; however, in $\mathcal{S}_t$, the error functions ϕ_u, ϕ_k may specify failure logic for testing Agents to discover as part of evaluating an LT approach. There may also exist a testing-error oracle separate from $\mathcal{S}$ which is capable of alerting observers of

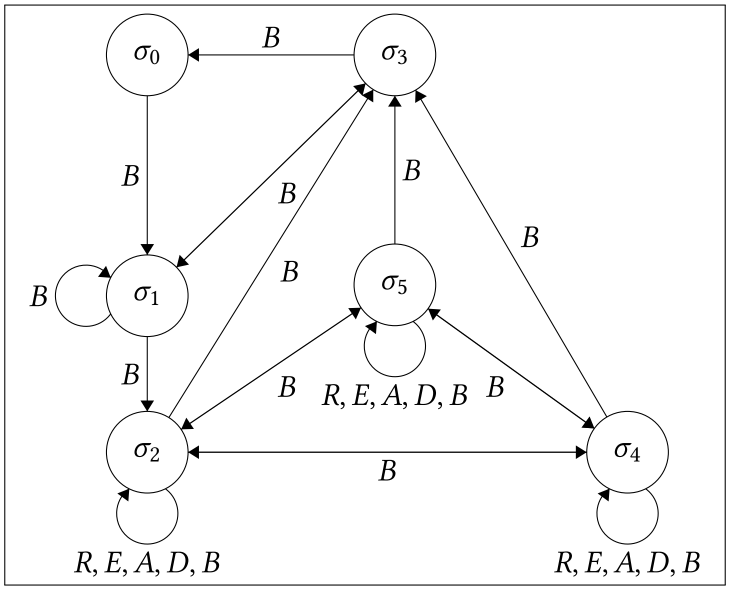


Figure 3.2. Legal Application State Space for User Profile: ‘Super Administrator User’

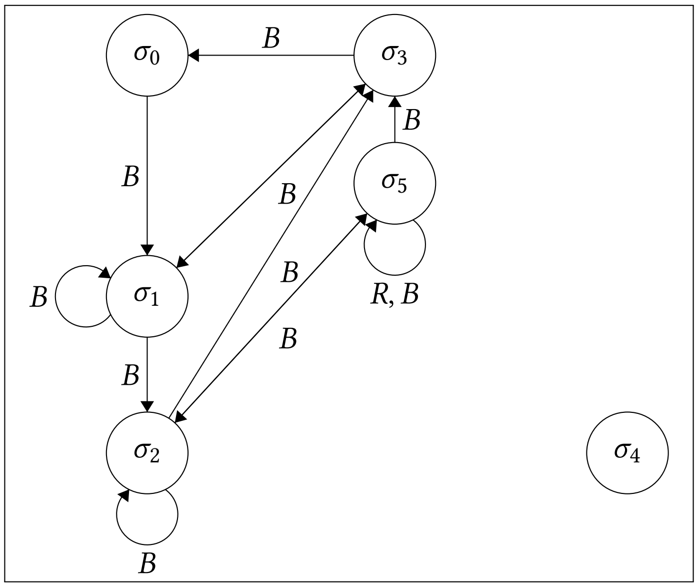


Figure 3.3. Legal Application State Space for User Profile: ‘User’

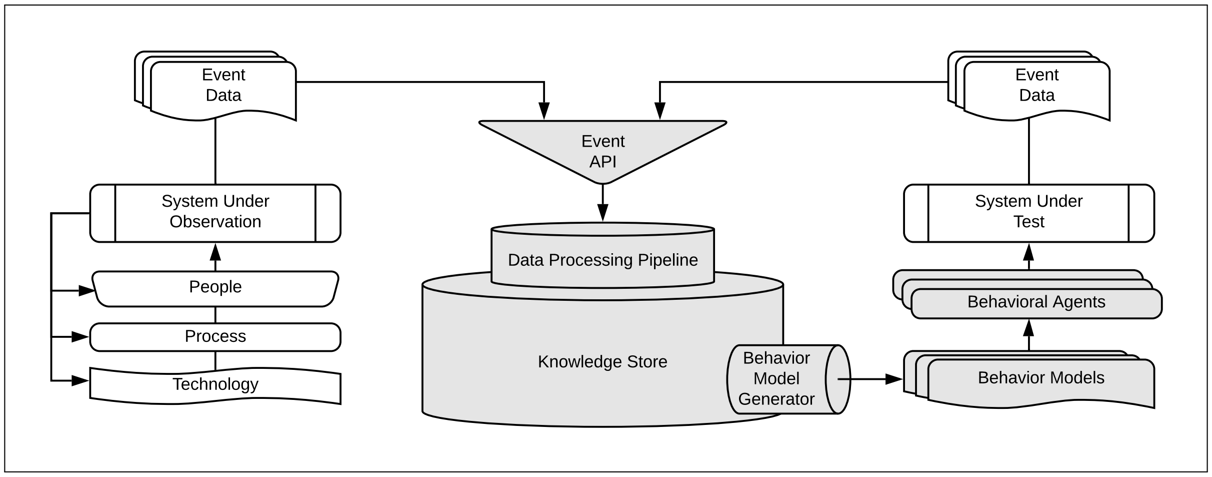


Figure 3.4. Framework for Simultaneous Modeling and Simulation of User Behavior

errors through additional event data. For systems such as $\mathcal{S}_o$, oracles are useful for showing operational metrics and failures in production. However, for systems such as $\mathcal{S}_t$, oracles are useful for training and detecting issues in the SUT. The data simulation testbed we describe in Section 3.7 provides more detail about testing oracles and their place in the load testing process.

3.2. Event Data Collection

The interactions between people, process, and technology within an HMS result in the generation of system logs or streams of event data. The difficulty of extracting events from the SUO and deciphering information from such events can vary; however, we can assume some basic structure to event data streams. An event is a five-tuple $(\sigma_s, \sigma_t, \tau, u, e, s)$ such that $\sigma_s \in S$ is a source state, $\sigma_t \in S$ is a target state, $\tau \in T$ is a time-stamp, $u \in U$ is a unique identifier for each user or some distinct entity in $\mathcal{S}_o$, $e \in \mathcal{E}$ represents a detected error in $\mathcal{S}_o$, and s is additional event information. For $\mathcal{S}_o$ in Figure 3.4, the Event API represents a mechanism for collecting events from raw web responses, filesystems, or databases. The Event API is used to collect data for the Data Processing Pipeline to parse and pass event tuples into the Knowledge Store. The Knowledge Store is a central location used for storing transition probabilities within the ϕ_p function, mean times in the ϕ_μ function, and any additional behavioral data for generating behavior profiles. For $\mathcal{S}_t$ in

Figure 3.4, the same Event API used for the SUO represents a mechanism for collecting events from the SUT. Where the data from $\mathcal{S}_o$ are used to model behavior patterns, the data collected from $\mathcal{S}_t$ are used to calculate the effectiveness of an LT approach and reward testing Agents for successfully detecting flaws in the SUT.

3.3. Descriptive Models for Agent Behavior

We conceptualize users and machines within $\mathcal{S}, \mathcal{S}_o$, and $\mathcal{S}_t$ as Agents, a simulated collection of which can operate within a MAS. Individual agents and groups of agents can follow various behavior models or bases. It is important for application stakeholders, technologists, and researchers to understand how behaviors of systems and users of systems [11] interrelate; to that end, operational and simulation models should be understandable by specialists and non-specialists alike. We define the Descriptive Model for Agent Behavior (DMAB) as any statistically descriptive model of behavior used to govern the actions of an Agent.

For example, let two Agents A, and B, be responsible for generating random numbers from a distribution D. Let us say that Agent A operates using the five number summary statistics of D : $\{D_{\min}, D_{Q1}, D_{\mu}, D_{Q3}, D_{\max}\}$. As such, Agent A operates using a DMAB. In contrast, let us say Agent B operates using an opaque deep-learning model which cannot be easily described. As such, B does not operate using a DMAB. We assert that a DMAB is more likely to be acceptable in an enterprise MAS due to its descriptive nature. Although less descriptive models have the potential of representing complex behavior patterns, the overhead of maintaining user confidence in such models can outweigh any perceived value of increased accuracy. The cost analysis between a model and how understandable it is should be performed in any situation where there is a question of justifying errant MAS behaviors. Such an exercise is a crucial step in the development life-cycle of any automated MAS.

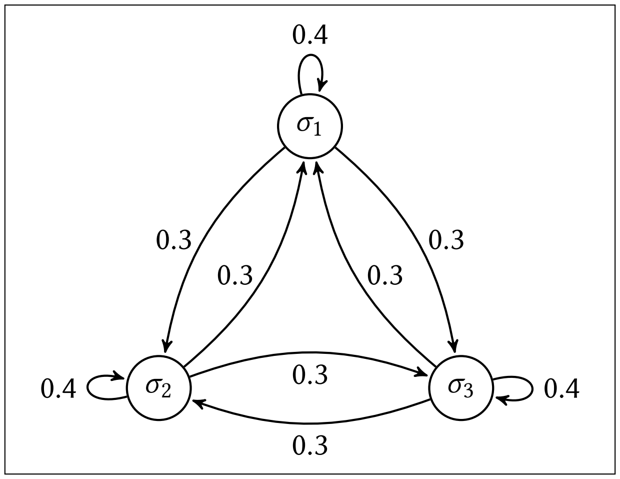


Figure 3.5. Markov Chain Directed Graph for Example User Behavior Model

Table 3.1. Conditional Probability Matrix for Example User Behavior Model

	σ_1	σ_2	σ_3
σ_1	0.4	0.3	0.3
σ_2	0.3	0.4	0.3
σ_3	0.3	0.3	0.4

The MC is a DMAB, due to the cause-effect nature of its mathematical structure; as such it has been used in many MAS, including the LT tool Markov4JMeter [58]. Markov4JMeter, although not built on fully distributed and independent operating models, does model users or Agents as Threads within JMeter [58]. A DMAB based on MC logic can be categorized into different bases of behavior, depending on available data from an SUO.

3.4. Behavior Bases

Within an automated MAS, MC based Agents can operate according to a prescriptive or descriptive basis of behavior. It is important to remember that although we use an underlying MC, there are many different approaches to storing behavioral profiles in memory based on event data. Additional methods involve constructing hidden Markov models, Bayesian networks, decision trees/forests, neural networks, Petri nets, as well as many other abstractions. It is conceivable then that our methods can be ported to those models as well; however, the main behavior profile structure we explore here is the MC. The MC can be equivalently represented as a connected graph as in Figure 3.5 or an adjacency matrix as in Table 3.1. We define and focus on six of many possible distinct categories of the MC, which we term: constant basis, random basis, user basis, profile basis, global basis, and semantic basis. We will use model and basis interchangeably throughout the rest of this document.

3.4.1. Constant Basis

The Constant Basis of Behavior (CBB) is a prescriptive DMAB representing a fixed set of sequential behaviors in a HMS with static probability of transition between the individual states. The CBB is representative of rule-based approaches to LT such as in JMeter [117], or in code-based LT mechanisms.

3.4.2. Random Basis

The Random Basis of Behavior (RBB) is a prescriptive DMAB representing random transitions between states in a HMS. The RBB is representative of randomized approaches to LT such as in JMeter [117].

3.4.3. User Basis

The User Basis of Behavior (UBB) is a DMAB representing how an individual user may behave in a HMS. These bases may be learned by observing individual users and may be used to build a MC model which represents how each individual user transitions between states in a given system.

3.4.4. Profile Basis

The Profile Basis of Behavior (PBB) is a DMAB which comprises a set of similar UBB. The PBB represents the behavior patterns present within a UBB without requiring the additional storage space or computational costs necessary for an agent using the PBB to operate.

3.4.5. Global Basis

The Global Basis of Behavior (GBB) is a DMAB which comprises all UBB. The GBB is calculated similarly to the UBB with the exception of user identity distinction being removed. In essence, the GBB is that PBB which comprises the aggregated behavior patterns of all users observed in an SUO.

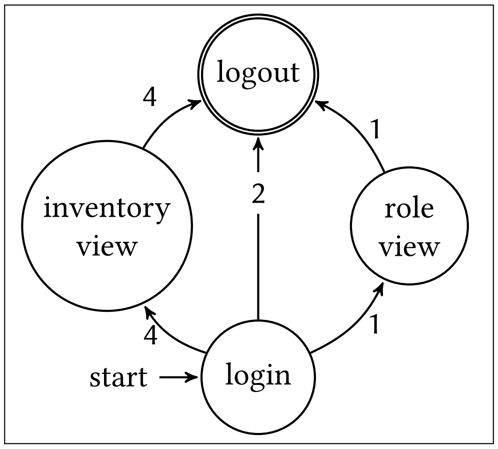


Figure 3.6. Aggregated User Behavior Count Graph (©2020 IEEE)

3.4.6. Semantic Basis

The Semantic Basis of Behavior (SBB) is a DMAB which comprises a set of logically divided subgraphs within a GBB. The SBB is calculated through the use of graph partitioning or clustering methods such as Tarjan's algorithm [116] and represents behavior patterns which are connected through semantic means. Semantic connections can be procedure calls within an application library, or API calls within the same module. When only aggregated data such as the GBB are available from the SUO, the SBB can divide the GBB such that Agent behavior in an automated MAS may be controlled and observed with finer grained detail.

3.5. Behavior Model Generation and Clustering

In Figure 3.4, we show that underlying event data is collected by the Event API; each MC basis is constructed from these events. We construct each MC DMAB by tracking and aggregating the temporal deltas and frequencies of input-state to output-state transitions for each uniquely identifiable data stream and creating a set of UBB. For example, in Figure 3.6, the running counts of transitions between each state σ_i are represented as weighted edges in the Knowledge Store as the adjacency matrix illustrated with the heatmap in Figure 3.7. By dividing each of the weights of all outgoing edges for each source state by the total number of events originating from the same state, we compute the resultant MC shown in Figure 3.8– the UBB for the observed behavior stream. We can visualize such a UBB as the heatmap in Figure 3.9 with columns representing the

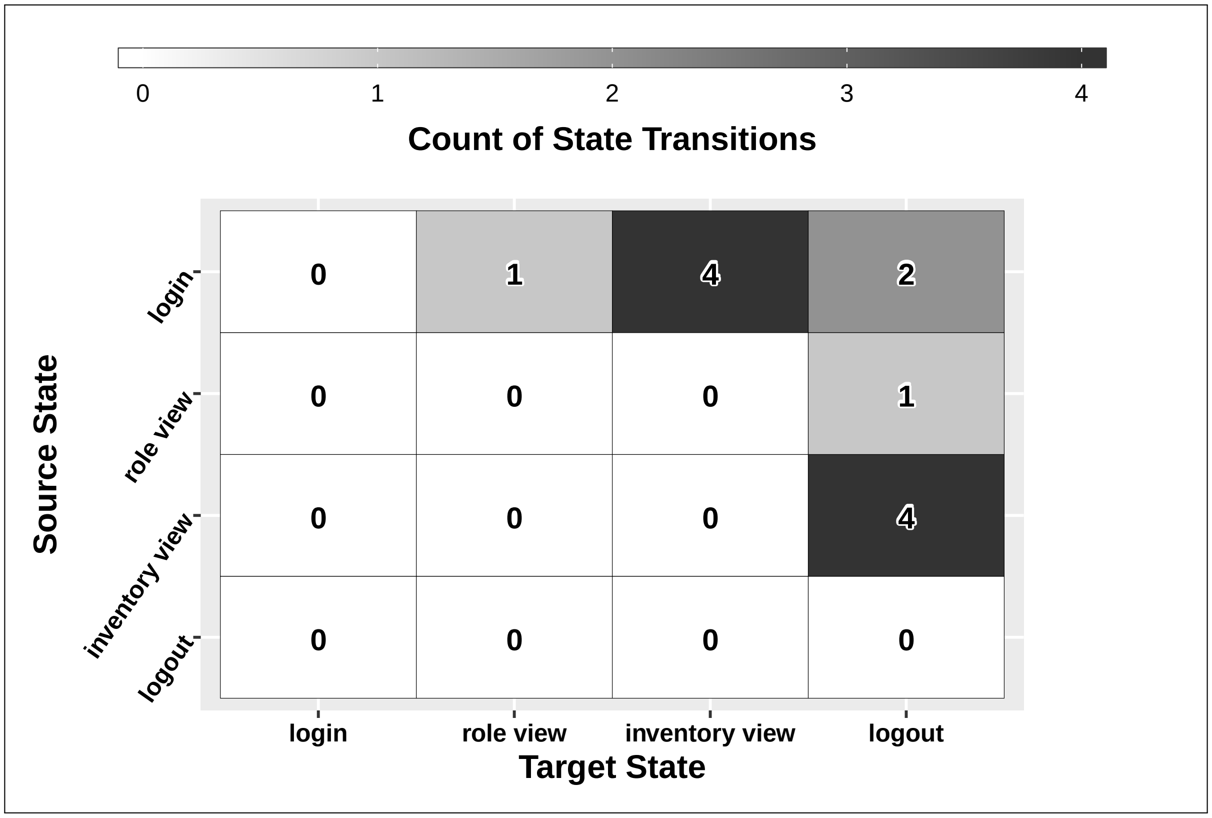


Figure 3.7. Heatmap of Transition Count Matrix for Example Users

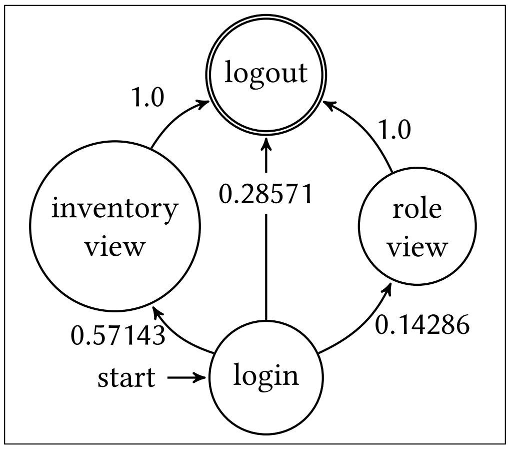


Figure 3.8. Markov Chain Directed Graph for User Behavior

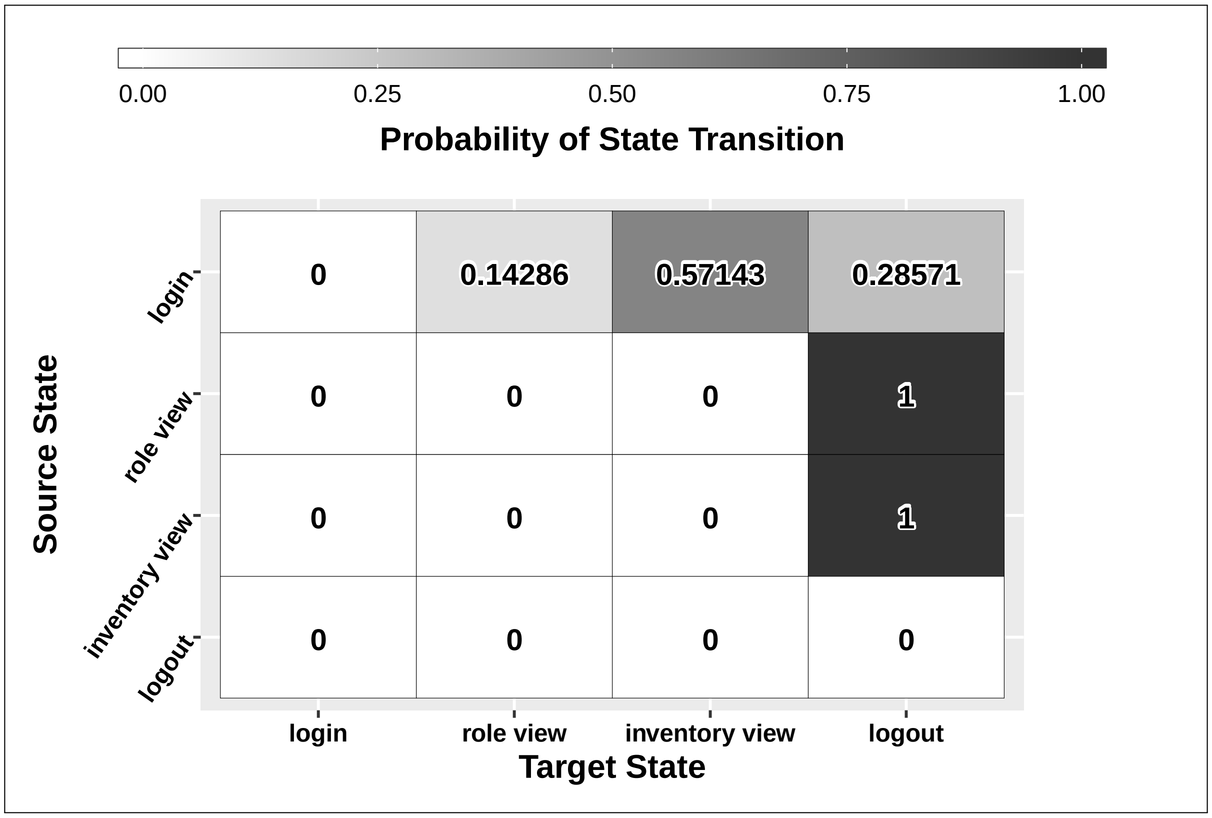


Figure 3.9. Heatmap of Conditional Probability Matrix for Example Users

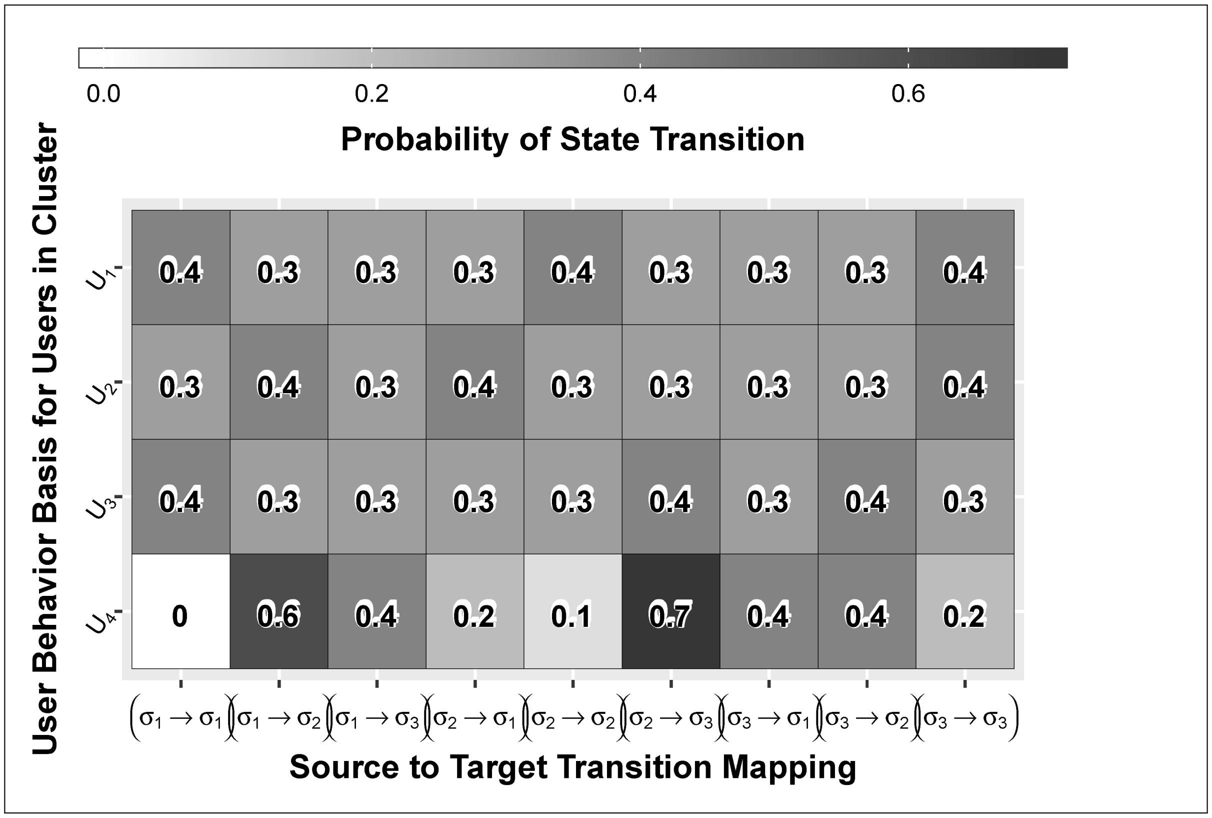
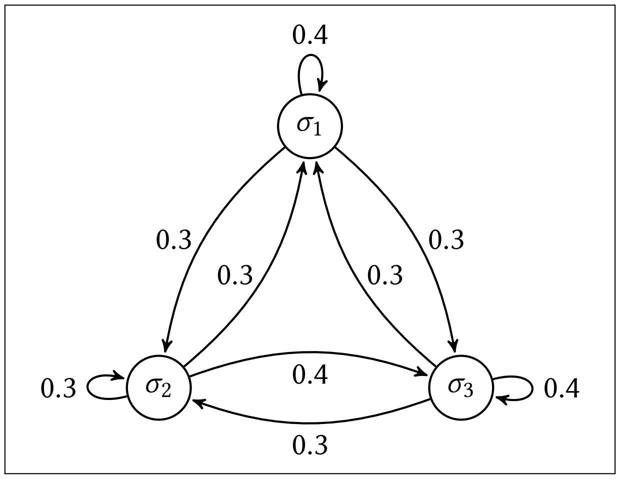


Figure 3.10. Conditional Probability for Clustered UBBs to Transition from σ_i to σ_j

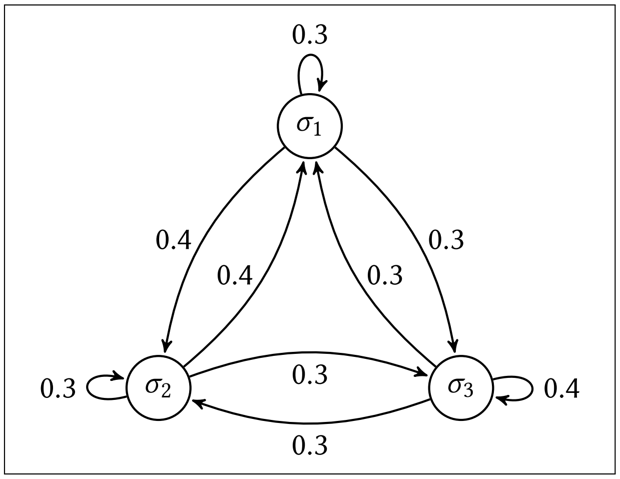
start state and row indexes representing the target state, with each shade showing the relative importance of state transitions within the recorded UBB. If users are not uniquely identifiable in the data stream with identifiers such as session ids, MAC addresses, or microservice API keys, we may still capture a UBB for each uniquely identified subset of events existing within the stream of data.

Calculating the GBB requires similar aggregation of such frequencies for all inbound event streams rather than per user as the UBB. When there exist too many UBB representations to economically store or describe how Agents in the MAS interact we may extract SBBs or PBBs to reduce the complexity of the bases.

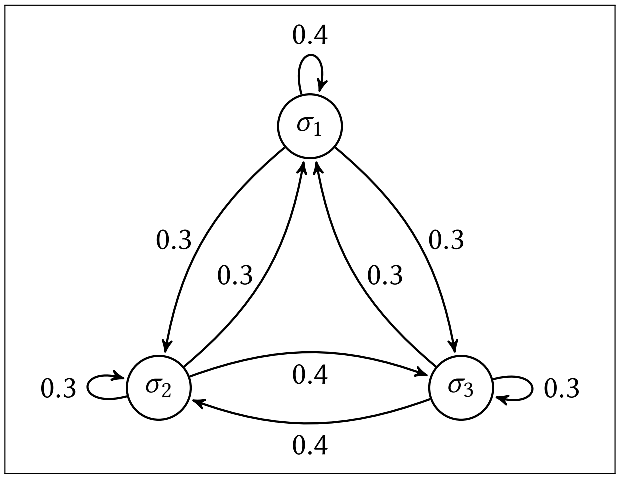
We use unsupervised machine learning in the form of clustering to extract PBBs from the UBBs. While clustering can be run against the raw observed frequency adjacency matrices, this approach creates more noise, as it favors the weights associated with frequent state transition volumes over conditional transition probabilities. We may use methods such as DBSCAN, K-



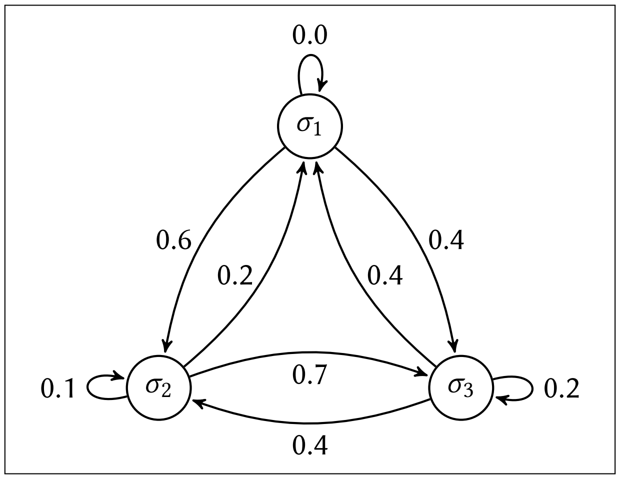
(a) UBB for Example User U_1



(b) UBB for Example User U_2



(c) UBB for Example User U_3



(d) UBB for Example User U_4

Figure 3.11. Markov Chain Directed Graph for Example User Behavior Model

Means, X-Means, or KNearestNeighbor to determine which of the MC are of similar structure. Our approach relies upon DBSCAN due to the fact that an expected resultant number of clusters does not need to be specified nor require constant re-computation as is required in the X-Means approach outlined by van Hoorn [58]. As such, DBSCAN can be continuously run with minimal pre-configuration rather than requiring manual intervention. Each output cluster represents a different behavior profile in P – to be executed by Agents for LT. An example set of UBBs can be seen in Figure 3.11 and represented as the heatmap in Figure3.10. By extracting PBBs instead of only recording UBB and storing those, we reduce the amount of data required to represent the operational variance present and observed in the SUO. The SBBs are extracted from the GBB with the constrained depth-first Tarjan's algorithm to effectively separate aggregated behavioral data from Wikipedia.[1] The approach LODESTONE uses for storing modeled user behavior patterns is similar in implementation to how tools such as Markov4JMeter [58] and WESSBAS [59] achieve MC modeling. As several anonymized user behavioral data sets of different formations are publicly available, we examine them to inform the direction of LODESTONE.

3.6. Experimental Data

To confirm the regularity of behavioral patterns occurring within external, non-simulated datasets, we analyze various anonymous and publicly available user behavioral datasets from UCI Machine Learning Repository [79] and Mozilla Labs [74]. Each data set represents the actions of distinct sets of users interacting with a system. The behaviors of each user are logged and anonymized / symbolized in order to preserve user privacy. This section describes each dataset, places them within the context of this research, and uses them to describe specifics of the formalisms and data models of our approach.

1 See Chapter 5 for details on depth-constrained Tarjan's algorithm.

3.6.1. Week In the Life of a Browser (WITL)

The ‘Week In the Life of a Browser’ data (WITL) [74] represent the recorded interactions between the Mozilla Firefox web browser and a consenting set of users. These data were recorded across several versions of the product; the users also consented to responding to surveys about how the software was typically employed. The data represent seven days of events recorded for about 27,000 users (as well as the survey responses.) Three tables of information are provided: a list of users (Table 3.2), a list of recorded events (Table 3.3), and a list of survey responses (Table 3.4).

The Table 3.2 provides (for each user) a randomized user_id (for event and survey response correlation), the operating system and Firefox versions, and the number of extensions installed. A few other fields are listed in the data dictionary; however, these were sparsely populated or not cogent to this study. The Table 3.3, provides the user id generating the event, the code associated with an event in the Firefox browser, string data used to further describe the event (these fields are not consistently populated), a Unix epoch-based time-stamp, and an un-populated session id. The Table 3.4 shows the written responses to survey questions asked of each user participating in the study. These questions can be used for further clustering of the data (e.g. correlation between survey responses and event sequences); however, these data are not part of our further analysis but included here for the sake of completion.

As data are recorded at the user level, the prevalent behavior base for the WITL data set is the *User Basis*. Each event is correlated to an individual user (as opposed to a group of people.) Each user represents a distinct user/machine combination using the Firefox browser; this uniqueness is enforced by the os, fx, and version attributes shown in Table 3.2. Thus, for each user, it is necessary to collect a descriptive model as a basis for behavior of that user. There are a number of events in the WITL data set which users and the Firefox browser can initiate which are fully detailed in [74]. There is a direct mapping from the states of the software and the states in the DMAB. Edges in the UBB represent the transition between states in the software, and the weights on the edges represent the frequency the transition was observed in the sample data for the given user.

Table 3.2. Week In the Life of a Browser - Users Table Sample

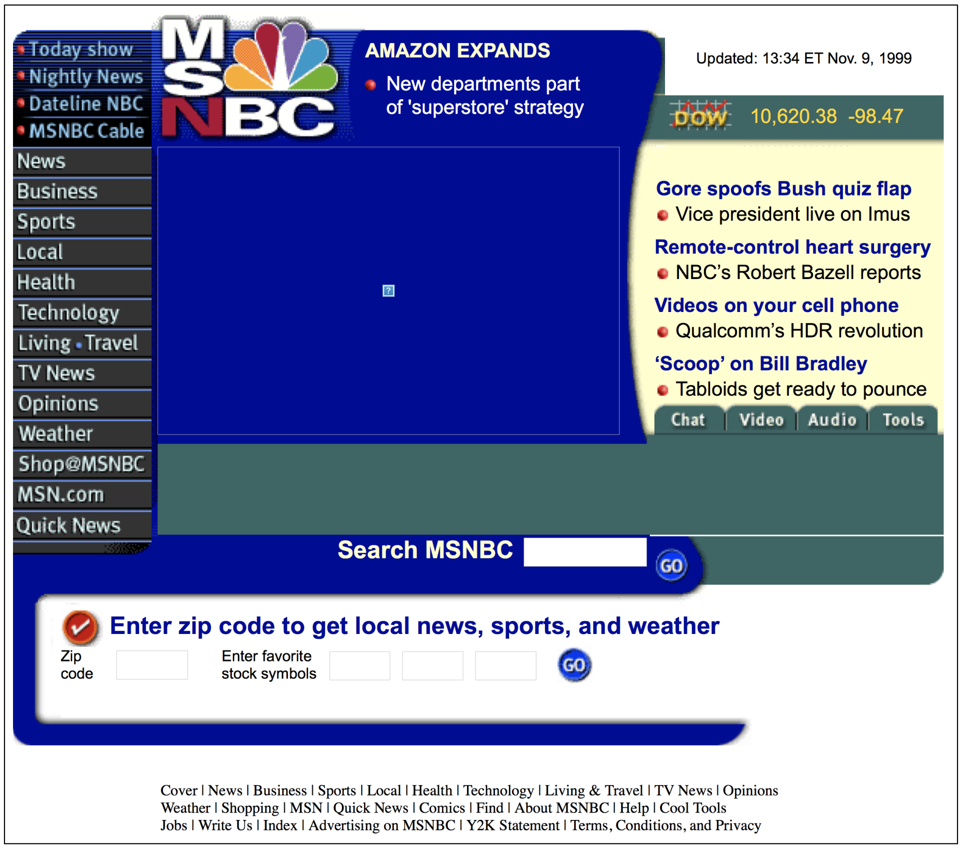
id	location	fx_version	os	version	survey_answers	number_extensions
0		4.0b6	WINNT Windows NT 6.1	1.0.3		3
1		4.0b6	WINNT Windows NT 5.1	1.0.3		4
2		4.0b6	WINNT Windows NT 5.1	1.0.3		5
3		4.0b6	WINNT Windows NT 6.1	1.0.3		5
4		4.0b4	WINNT Windows NT 5.1	1.0.2		3
5		4.0b6	WINNT Windows NT 5.1	1.0.3		9
6		4.0b6	WINNT Windows NT 5.1	1.0.3		8
7		4.0b6	WINNT Windows NT 6.1; WOW64	1.0.3		2
8		4.0b6	WINNT Windows NT 5.1	1.0.3		3

Table 3.3. Week In the Life of a Browser - Events Table Sample

user_id	event_code	data1	data2	data3	timestamp	session_id
0	26	1 windows	4 tabs		1288742315024	
0	0	5			1288742315158	
0	22	npGoogleOneClick8.dll	1.2.183.39	Google Update	1288742315246	
0	22	NPSWF32.dll	10.1.85.3	Shockwave Flash	1288742315302	
0	22	npYState.dll	1.0.0.7	Yahoo Application State Plugin	1288742315369	
0	22	npVeetle.dll	0.9.18.0	Veetle TV Core	1288742315490	
0	22	npvlc.dll	0.9.18.0	Veetle TV Player	1288742315491	
0	22	nppdf32.dll	9.4.0.195	Adobe Acrobat	1288742315491	
0	23	13510			1288742315707	

Table 3.4. Week In the Life of a Browser - Survey Table Sample

user_id	**q1**	**q2**	**q3**	**q4**	**q5**	**q6**	**q7**	**q8**	**q9**	**q10**	**q11**	**q12**	**q13**	**q14**
7	6	0			0	1	2	10	0|1|3	1	0|1|3|4|6	0|8	0|1|3	
48	6	0		0	0	1	2	7	0|1|2|3	7	2|3|4|5|6	0|1|2|3|5|6|8	0|1|2	<freeform-text removed>
51	4	1	0|3	1	0	2	2	7	0	7	1|2|3|4|5|6	0|1|3|5|8|10		
56	5	0		0	0	1	3	7	0	4	2|3|4|5|6	1|2|3|5|7|8|10	0|1|3|4|5	
59	6	0		0	0	1	2	7	0	1	1|4|5|6	0|1|3|5|6|7|9|10	0|1	
68	6	0		0	0	1	3	7	0|3	0	2|3|4|5	0|2|3|4	3	
69	6	0		0	0	3	3	10	0|1|3	0	1|3|4|5|6	1|2|3|5|8|9	0|1|2|3|5	<freeform-text removed>
71	6	1	0	1	0	2	4	9	0|1|3	0	0|1|4|6	0|2|3	0|1|5	
73	3	1	3	1	0	1	2	8	0|2|3	0	2|3|4|5|6	0|1|3|7|8|9	0|1|3|4|5	<freeform-text removed>



AMAZON EXPANDS
New departments part of 'superstore' strategy

Updated: 13:34 ET Nov. 9, 1999

DOW 10,620.38 -98.47

Gore spoofs Bush quiz flap
Vice president live on Imus

Remote-control heart surgery
NBC's Robert Bazell reports

Videos on your cell phone
Qualcomm's HDR revolution

'Scoop' on Bill Bradley
Tabloids get ready to pounce

Search MSNBC

Enter zip code to get local news, sports, and weather

Zip code   Enter favorite stock symbols

Cover | News | Business | Sports | Local | Health | Technology | Living & Travel | TV News | Opinions
Weather | Shopping | MSN | Quick News | Comics | Find | About MSNBC | Help | Cool Tools
Jobs | Write Us | Index | Advertising on MSNBC | Y2K Statement | Terms, Conditions, and Privacy

Figure 3.12. MSNBC.com on November 9, 1999

3.6.2. MSNBC.com Anonymous Web Data

The 'MSNBC.com Anonymous Web Data' data (MSNBC) [55][20] represent the user browsing patterns on the MSNBC.com website on September 28, 1999. For a view of the organization of the site homepage, refer to Figure 3.12. Individual users who browsed the site were observed, and the categories of articles they browsed were recorded. The sequences of article categories comprise this dataset and provide insight into how users interacted with the site. A sample of the category to key mapping is listed in Table 3.5 with a sample of sequences from the dataset.

Each Sequence row in Table 3.5 (b) represents a unique user of the site on the observed day. As such, the prevalent basis is the *User Basis*, and a UBB can be constructed from each sequence having $n >= 2$. We further stipulate for this dataset that $n <= 16$, as the maximum number of items per sequence can be about 14,000. Intuitively, the length of a sequence is indicative of the behavioral patterns a web crawler might exhibit, indexing the site and traversing each link as the

Table 3.5. MSNBC.com Anonymous Web Data

(a) Categories

Key	Name
1	Frontpage
2	News
3	Tech
4	Local
5	Opinion
6	On-Air
7	Misc
8	Weather

(b) Sequences

Sequence
1 1
2
3 2 2 4 2 2 2 3 3
5
1
6
1 1
6

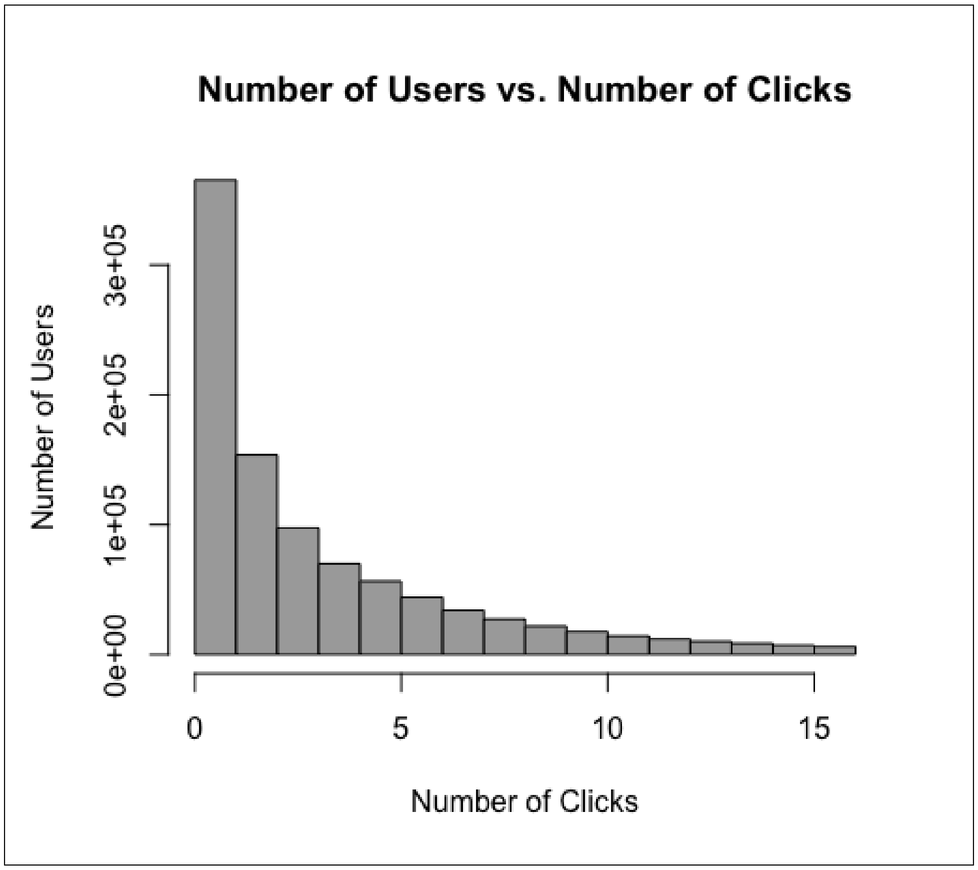


Figure 3.13. MSNBC.com Number of Users vs. Number of Clicks

Table 3.6. Anonymous Microsoft Web Data

(a) Categories

Key	Name
1287	International AutoRoute
1288	library
1289	Master Chef Product Information
1297	Central America
1215	For Developers Only Info
1279	Multimedia Golf
1239	Microsoft Consulting
1282	home

(b) Sequences

Type	Key
C	10016
V	1025
V	1026
C	10017
V	1004
C	10018
V	1008
V	1058

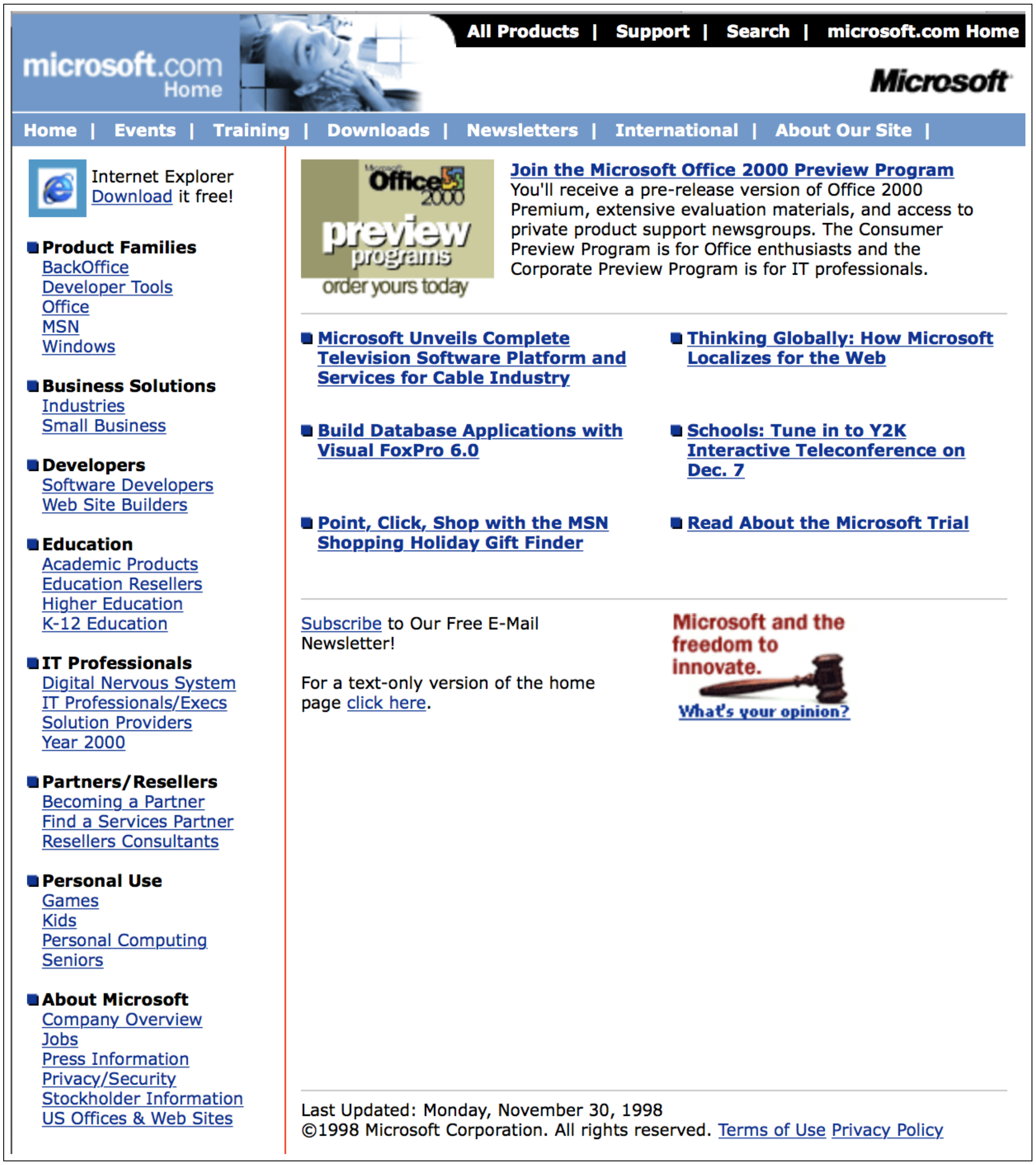
(c) Transposed

User	Sequence
10016	1025 1026
10017	1004
10018	1008 1058

crawler discovers it – such sequences can grow to be quite long due to the automated nature of web crawlers. As we are only interested in the behavior of users, the typical user's attention span on MSNBC.com can be modeled using the general exponential hazard function as illustrated in Figure 3.13. Figure 3.13, represents 95% of the user sequences observed.

3.6.3. Anonymous Microsoft Web Data

The 'Anonymous Microsoft Web Data' data (AMSWD) [17][18] represent behavior patterns of 38,000 Microsoft.com website users from a week in February of 1998. We show with Figure 3.14 the organization of the site in that epoch. The data represent a list of 'vroots', or categories for articles, a list of numbered users who interacted with the site, and a sequence of votes each user put for different articles on the site (see Table 3.6). The user and sequence data may be transposed (see Table 3.6) to provide a similar format to the data in Table 3.5. As each sequence of data can be compiled back to individual users, the *User Basis* is most prevalent here. There are more than 38,000 different users, so automating the process of browsing the Microsoft.com would require an equivalent number of UBBs to represent the user population.


All Products | Support | Search | microsoft.com Home
microsoft.com Home
Microsoft
Home | Events | Training | Downloads | Newsletters | International | About Our Site |
Office 2000 preview programs order yours today
Microsoft and the freedom to innovate.

Internet Explorer
Download it free!

Product Families
BackOffice
Developer Tools
Office
MSN
Windows

Business Solutions
Industries
Small Business

Developers
Software Developers
Web Site Builders

Education
Academic Products
Education Resellers
Higher Education
K-12 Education

IT Professionals
Digital Nervous System
IT Professionals/Execs
Solution Providers
Year 2000

Partners/Resellers
Becoming a Partner
Find a Services Partner
Resellers Consultants

Personal Use
Games
Kids
Personal Computing
Seniors

About Microsoft
Company Overview
Jobs
Press Information
Privacy/Security
Stockholder Information
US Offices & Web Sites

Join the Microsoft Office 2000 Preview Program
You'll receive a pre-release version of Office 2000 Premium, extensive evaluation materials, and access to private product support newsgroups. The Consumer Preview Program is for Office enthusiasts and the Corporate Preview Program is for IT professionals.

- **Microsoft Unveils Complete Television Software Platform and Services for Cable Industry**
- **Thinking Globally: How Microsoft Localizes for the Web**
- **Build Database Applications with Visual FoxPro 6.0**
- **Schools: Tune in to Y2K Interactive Teleconference on Dec. 7**
- **Point, Click, Shop with the MSN Shopping Holiday Gift Finder**
- **Read About the Microsoft Trial**

Subscribe to Our Free E-Mail Newsletter!

For a text-only version of the home page click here.

What's your opinion?

Last Updated: Monday, November 30, 1998
©1998 Microsoft Corporation. All rights reserved. Terms of Use Privacy Policy

Figure 3.14. Microsoft.com in November, 1998

3.7. Simulating Behavioral Data

A significant challenge addressed by this research was the lack of freely available user behavior data recording interactions within a HMS. The optimal data set would provide sufficient richness of detail for accurate study; in addition, those data would be necessarily obscured in order to protect the privacy and anonymity of users being modeled. Moreover, there was no standard example of a controlled environment for comparing and contrasting the effectiveness of two analogous load testing processes in a microservice architecture environment [68]. Sample microservices such as Sock Shop were developed for load testing as well as microservice software emulation [6]; yet when framed within the goals of this research, sufficiently described or prescribed testing information are largely not available for these systems. The absence of such data as well as the need for a testing oracle prompted the need for a solution which would be capable of repeatably generating error information. To solve this challenge, we designed, tested, engineered, and evaluated a multi-faceted microservice application simulation testbed. In addition, this test application needed to be capable of generating log data in the format required for this research while amalgamating the classical scripted approach of load testing tools such as JMeter [117] with the observations garnered through the experimental analysis of Section 3.6.

We developed TinyERP to be a microservice which emulates the critical functions necessary to support an enterprise resource planning business unit. In addition to the TinyERP microservice, we designed a testing oracle called Loki was designed to be paired with the microservice so that errors could be randomly generated by sequences of actions within the stream of events sent to TinyERP. These errors are randomly generated based on a seed value in the application configuration in addition to the percentage of such 'known' and 'unknown' error transitions to be generated. As TinyERP is a simulated environment as well as a fully functional application, being able to control the existence of such errors is important for validating the effectiveness of a testing tool. The principal components required to successfully implement TinyERP were:

- A software component for fluently developing templates used for generating and executing the scripted behavior patterns in TinyERP

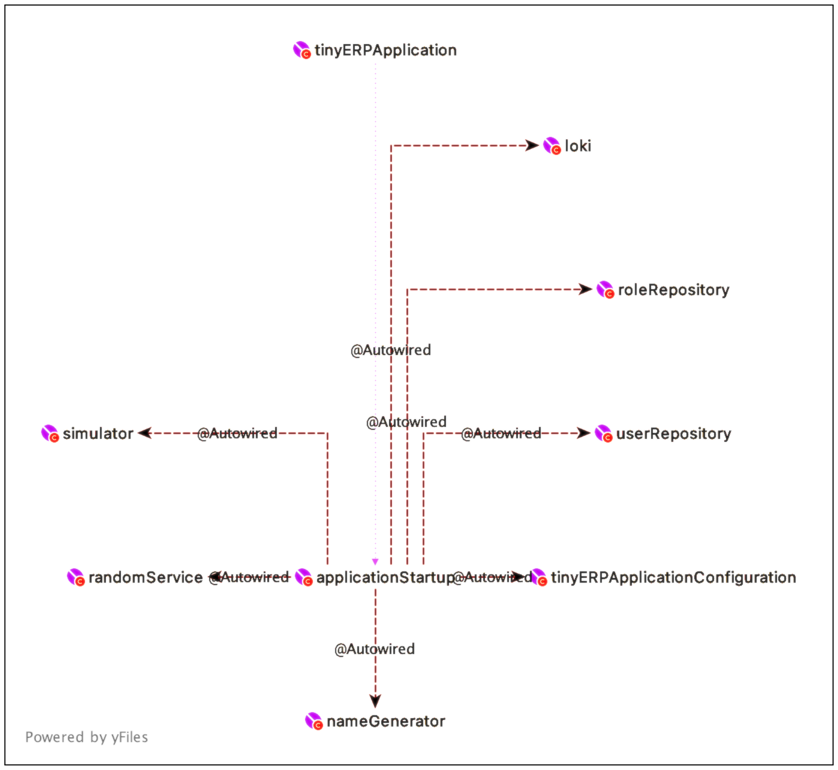


Figure 3.15. Class Diagram for TinyERP Spring Boot Services

- A configuration process which can swiftly generate various behavior data sets conforming to test script analogues in other load testing processes as well as parameterize the boundaries for TinyERP
- A simulation engine which executes the behavior scripts generated by TinyERP with web request level details which would be present in a production environment and meets our definition of S
- A functional multi-tiered microservice which operates within a production-quality software environment to facilitate diagnostic-level stimulus and response metrics capture and recording
- A testing oracle which generates critical operational metrics and error information for TinyERP as well as accurately simulating and recording ϕ_k and ϕ_u
- A metrics and statistics generation component which is transparently lightweight and does not significantly impact the run-time characteristics of TinyERP.

```
# DATABASE CONFIGURATION
spring.security.user.password=****************
spring.h2.console.enabled=true
h2.tcp.enabled=true

#LOKI CONFIGURATION
loki.seed=42
loki.logFilePath=logs
loki.numberOfUsers=100
loki.generateLogs=true
loki.startTime=1500000000000
loki.failureRate=0.10
loki.timeIncrement=120000
#three separate conditions for
finalizing the number of generated logs.
loki.endTime=1600000000000
loki.maxNumberOfLogs=1000000
loki.maxRunTime=300000
```

Figure 3.16. TinyERP and Loki Runtime Configuration

The implementation of TinyERP grew from a small set of data generation and testing scripts with a static web service to its culmination in the development of a fully functional microservice in Spring Boot [123]. Spring Boot is an industry standard framework for developing microservices which standardizes much of the underlying boilerplate functionality such as logging, documentation, deployment, dependency management, dependency injection, in addition to many other features [123]. As a fully featured framework, it is also designed to seamlessly operate on development desktops in addition to client-server models and more modern cloud-based environments such as AWS [3, 123]. Due to the abstraction of complexities, Spring Boot was an optimal framework for engineering TinyERP so that operational inconsistencies would be within a largely controlled environment. The premise of such a controlled environment was to ensure that errors and variations could be more accurately simulated and measured.

The connections between the major underlying components of TinyERP can be seen within the class diagram of the Spring Boot application shown in Figure 3.15. TinyERP is the main microservice application, relying up a User and Role Repository to emulate the principal access control mechanisms of the microservice. The core initialization platform of TinyERP operates via an ApplicationConfiguration which may be modified by a markup configuration file without requiring TinyERP be recompiled or reinterpreted. The required data elements can be seen in Figure 3.16, containing database configuration elements as well as boundary conditions for simulated data generation. One of the primary parameters stored within this configuration file is the seed value for initializing the RandomService which provides randomly generated values throughout the execution of TinyERP. While the RandomService uses the configured seed value to ensure randomness where it is required, the RandomService also ensures consistency in the output between executions of the Simulator component. While outputs may be vastly different between executions having differently configured seed values, executions sharing the same seed value contain very little significant variance. The NameGenerator uses the most frequently assigned female, male, and surnames available by the Social Security Administration [110] for generation of sample users and usernames to simulate with generated load testing scripts.

Operation of TinyERP follows a sequence of steps denoted in Figure 3.17 which begins with (I) the creation and configuration of behavior parameters. These parameters consist of (II) configuration data and script templates written with a Java library component developed during the course of this research. The execution (III) of TinyERP has two primary modes, microservice only, or microservice and simulation execution (IV). The rationale for the two modes of operation follows the need for TinyERP to be capable of independently running on multiple machines to support a distributed testing scenario. In (V), the RandomService, NameGenerator, configuration random seed information, and behavior script templates are used to define a set of logical agents (VI) which execute behavioral load.

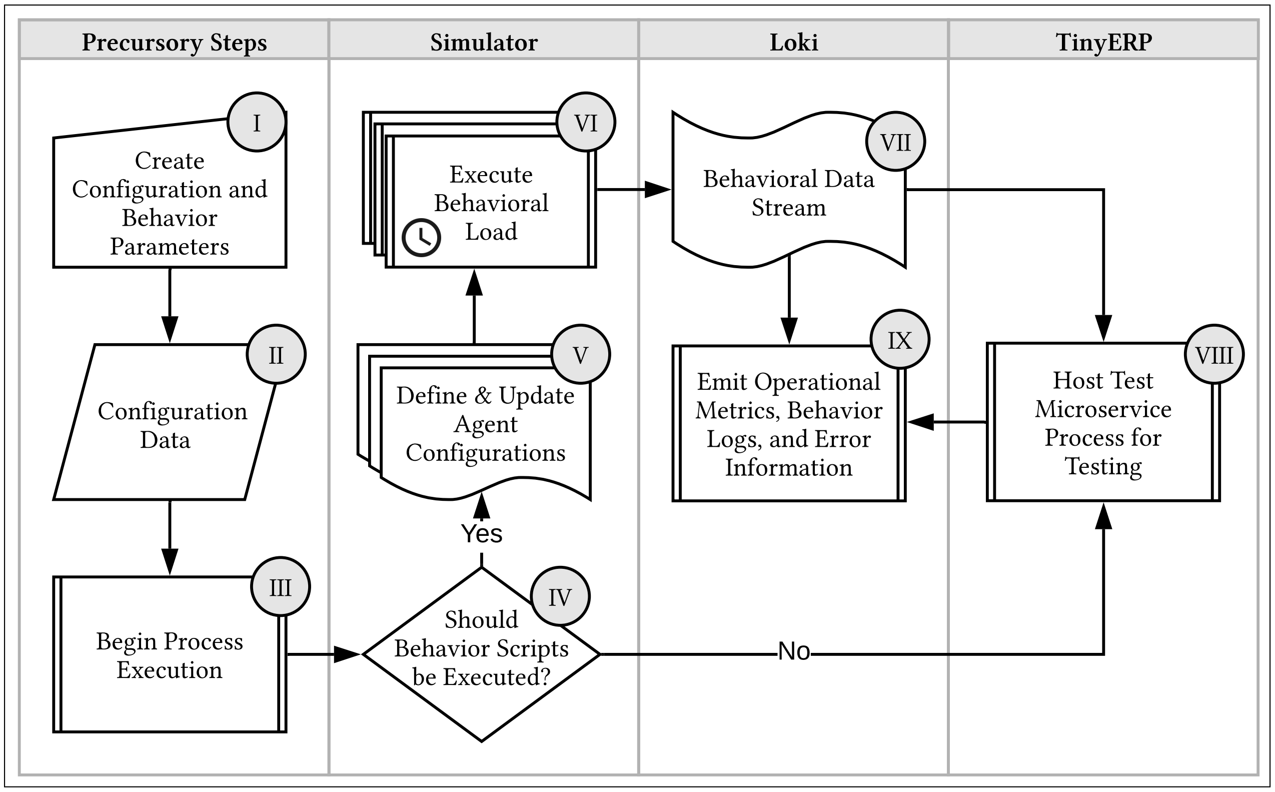

Precursory Steps
Simulator
Loki
TinyERP
I
Create Configuration and Behavior Parameters
II
Configuration Data
III
Begin Process Execution
IV
Should Behavior Scripts be Executed?
Yes
No
V
Define & Update Agent Configurations
VI
Execute Behavioral Load
VII
Behavioral Data Stream
VIII
Host Test Microservice Process for Testing
IX
Emit Operational Metrics, Behavior Logs, and Error Information

Figure 3.17. Sequence Diagram of Simulated Data Generation Process

These agents follow the sequential set of behaviors prescribed by the script templates. An example of such a template using the fluent script template generator is shown in Figure 3.18. The template follows the format of a prescriptive probabilistic state machine, implicitly beginning with a LOGIN action and ending with a LOGOUT action. The steps in between may be bounded and repeated in sequence or individually, depending on the script outline configuration. The set of generated agents emit a behavioral data stream (VII) monitored by Loki which logs the behaviors to the system user log files as in Figure 3.19. Loki can also generate simulated errors if the configured random seed determines an input state to be considered faulty. The error is logged in the behavior data as well as the state data.

The event stream also actuate the TinyERP microservice - TinyERP (VIII) to emulate the actual usage of the microservice by a population of users by emitting realistic response times as well as non-functional architectural deficiencies such as server configuration specifications. Loki also captures operational metrics such as CPU usage, memory usage, behavior logs, and all simulated

```
ScriptBuilder.getBuilder()
    .addScript(
        Script.getBuilder()
            .setRepeatProbability(0.2)
            .setUsedProbability(0.2)
            .addStep(
                ScriptAction.getBuilder()
                    .setModule(ACCOUNT)
                    .setAction(VIEW)
                    .setRepeatProbability(0.50)
                    .setMaxRepeats(5)
                    .build()
            )
            .addStep(
                ScriptAction.getBuilder()
                    .setModule(ACCOUNT)
                    .setAction(ADD)
                    .setRepeatProbability(0.50)
                    .setMaxRepeats(5)
                    .build()
            )
            .addStep(
                ScriptAction.getBuilder()
                    .setModule(ACCOUNT)
                    .setAction(MODIFY)
                    .setRepeatProbability(0.50)
                    .setMaxRepeats(10)
                    .build()
            ).build()
    )
```

Figure 3.18. TinyERP User Simulation Script Template

```
SESSIONID,USERNAME,SESSIONEVENTINDEX,PREVIOUSTIME,CURRENTTIME,
PREVIOUSMODULE.PREVIOUSACTION(PREVIOUSMESSAGE),
CURRENTMODULE.CURRENTACTION(CURRENTMESSAGE),TIMEDIFFERENCE(seconds)
"49e62b01","slaude",1,1500000005465,1500000015499,"GLOBAL.LOGIN(+)","USER.VIEW(error)",10.034
"7040d78c","hwilli",1,1500000035864,1500000039614,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",3.75
"789e0c48","fguzma",1,1500000010990,1500000058867,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",47.877
"4f7370f2","dphern",1,1500000037905,1500000041655,"GLOBAL.LOGIN(+)","ACCOUNT.VIEW(+)",3.75
"47d88d9a","chorst",1,1500000015927,1500000022550,"GLOBAL.LOGIN(+)","ORDER.VIEW(+)",6.623
"971b16fa","balani",1,1500000007556,1500000031822,"GLOBAL.LOGIN(+)","INVENTORY.VIEW(+)",24.266
"53538245","dboyki",1,1500000025267,1500000029017,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",3.75
"19f8db21","hwolfe",1,1500000009848,1500000016634,"GLOBAL.LOGIN(+)","INVENTORY.VIEW(+)",6.786
"8412c7c1","escott",1,1500000033230,1500000060455,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",27.225
"e2780a66","jcosen",1,1500000047962,1500000051712,"GLOBAL.LOGIN(+)","USER.ADD(+)",3.75
"00681822","jgilli",1,1500000023221,1500000030512,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",7.291
"fa18f255","jjohso",1,1500000007526,1500000011276,"GLOBAL.LOGIN(+)","ROLE.ADD(+)",3.75
"ab241434","mbradl",1,1500000028996,1500000032746,"GLOBAL.LOGIN(+)","ACCOUNT.VIEW(+)",3.75
"54c9038d","itanne",1,1500000039166,1500000046666,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",7.5
"68304f82","mkaram",1,1500000039655,1500000050017,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",10.362
"1ecc1840","mmurdo",1,1500000036414,1500000052461,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",16.047
"997eb791","sbooth",1,1500000014281,1500000022092,"GLOBAL.LOGIN(+)","USER.VIEW(error)",7.811
"aef4bd84","wmoore",1,1500000010978,1500000019375,"GLOBAL.LOGIN(+)","INVOICE.VIEW(+)",8.397
"2089005e","thart",1,1500000006598,1500000022014,"GLOBAL.LOGIN(+)","ACCOUNT.VIEW(+)",15.416
"d11a219d","civan",1,1500000016580,1500000027944,"GLOBAL.LOGIN(+)","INVENTORY.VIEW(+)",11.364
"3fc775c4","cbare",1,1500000013915,1500000017665,"GLOBAL.LOGIN(+)","ROLE.VIEW(+)",3.75
"73ba4004","rrober",1,1500000024570,1500000027685,"GLOBAL.LOGIN(+)","USER.VIEW(error)",3.115
"92557412","ccowen",1,1500000032892,1500000040392,"GLOBAL.LOGIN(+)","INVOICE.ADD(error)",7.5
```

Figure 3.19. Sample User Data Stream

```
TIME
DELTATODETECTFAILURE
DELTATODETECTNOVELFAILURE
DISKSPACE
EVENTCOUNT
EVENTRATE
FAILURESFOUND
FREEMEMORY
MEMORYUSAGE
NOVELFAILURESFOUND
SESSIONS
THREADCOUNT
```

Figure 3.20. System Metrics captured for TinyERP

error information from TinyERP for system level approximation of the effectiveness of the load testing tool. The full set of system-level metrics that are captured by Loki is enumerated in Figure 3.20. For each run of TinyERP, the metrics in Figure 3.20 are streamed to the TinyStats component where various statistics are calculated for evaluating the effectiveness of the load testing system being used.

The TinyStats component is a powerful contribution in its own right because each time a number is added to the set of metrics, the statistics may be called in O(1) time minus the operating cost of system level math operations such as log_n. For example, the naive mean calculation operation on a set $N = \{n_1, n_2, n_3, ..., n_j\}$ of measured data points requires an O(1) operation to insert a new element into the set, an O(n) operation to provide the sum of the contained elements, and either a constant time operation to provide the number of elements in the set, or another O(n) operation to count the number of elements. To calculate the mean, the following formula applies:

$$\mu_{naive} = |N|^{-1} \cdot \sum_{i=1}^{j} n_i.$$

By amortizing the precursory statistical operations in constant time, the metric statistics can also be called with constant time in addition to removing the need to store the numbers in memory. For a contrasting example, the TinyStats version of the mean operation costs O(1) to update the current sum of elements $\Sigma_{current}$ and O(1) to update the current count of elements N_n with no additional space requirement. Then, to calculate the mean, the following formula applies:

$$\mu_{TinyStats} = \frac{\Sigma_{current}}{N_n}.$$

Additional statistics calculations such as first, second, third mean, skewness, kurtosis, min, and max may be similarly calculated. TinyStats is a lightweight statistics library which may accept very large streams of numbers and provide statistics with very little required processing time. Refer to Appendix B. for more details on the TinyStats statistics component developed to support this research.

The simulated behavioral logs generated by TinyERP emulates concepts and patterns observed in Section 3.6, and the scripted approach used to generate behavior-based traffic is similar to the steps used to generate load tests as on the Sock Store microservice [6] as well as JMeter [117]. However, one of the main catalysts affecting this research was the ability of the Simulator to quickly generate different data patterns with minimal configuration and execute it against different endpoints - allowing for local testing on TinyERP in addition to cloud testing on an implementation of TinyERP on AWS (see Chapter 6). Also of note is the test oracle Loki's consistently thorough logging, simulated error generation, comprehensive metrics, and lightweight statistics via TinyStats.

3.8. Summary

We have defined the formalisms, data structures, and processes required to ingest data with LODESTONE. The Descriptive Model of Agent Behavior represents the bases used for all agent operations throughout the approach. We have defined the agent-based structure required to execute the various bases of behavior used to generate realistic workload. By clustering behavior profiles, we have shown how to reduce the data storage footprint required to operate the realistic load generating agents. Our experimental analysis of three anonymous behavioral web datasets informed how the pattern-based behavior of users can be modeled; the analysis illustrated the type of data produced by systems upon which the LODESTONE process can generate workload. The implementation of a scripting template component and simulation engine enabled the creation of various datasets based on rule-based patterns common to classical load testing processes. The microservice testbed, behavior logging, metrics calculations, and our testing oracle Loki provided a way to locally compare LODESTONE with other tools as well as a template for replicating the research within a cloud-based environment.

As described in this chapter, we extend formalisms of traditional state machines and stochastic models for data-driven user modeling which provide additional details over historical methods (RQ_1 from Chapter 1). In addition, we describe how various forms of behavioral data can be analyzed, clustered into more efficient models, and how TinyERP can be used to simulate the generation of such data (RQ_1, RQ_2 from Chapter 1). When behavioral data cannot be simulated, scripted, or extracted at the necessary level of atomicity described within this chapter, other methods for extracting profiles of user behavior may be extracted from aggregated data. One particular method is through the mining of semantic clusters from aggregated data and is described in Chapter 4.

Chapter 4. Inferring User Models from Aggregated Log Data

The whole is greater than the sum of
its parts.
-- Aristotle
Metaphysics, Book 8

Wikipedia is a rich source of information for researchers who find value from not only the content, but also the structure and organization of the articles in this open-source repository. We contribute a means of using aggregated user behavioral clickstream data from Wikipedia to derive meaningful semantic clusters. We apply community detection algorithms and modified graph algorithms to extract semantic clusters from graphs of user behavior. We derive a set of connected components which represent related articles within a shared subject domain. These resulting knowledge communities have application in numerous domains where understanding user behavior would be relevant, including software engineering, healthcare, and security domains [93].

4.1. Introduction

The Internet has become the primary mechanism through which human knowledge is produced, stored, and consumed. We direct our attention to means through which users of this 'system of systems' extract knowledge. One particular source, the website 'Wikipedia.com', has become a rich source of information for users and researchers. As an open-source knowledge repository, Wikipedia hosts millions of articles on a broad range of topics. Researchers find value from not only the content but also the structure and organization of these articles. Since this site has been available to the public, the number of articles has increased from 0GB in 2001 to 12GB in 2014, cementing Wikipedia as one of the major repositories of human knowledge (see Figure 4.1). With the release of monthly aggregated clickstream data, Wikipedia has provided a new perspective for researchers to study how users interact with the site's contents. When a set of Wikipeda articles share topically related content, we refer to these articles as a knowledge community, or semantic cluster[1]. From the released behavioral data, we investigate how such data can be mod-

1 The terms *knowledge community* and *semantic cluster* are used interchangeably in this chapter.

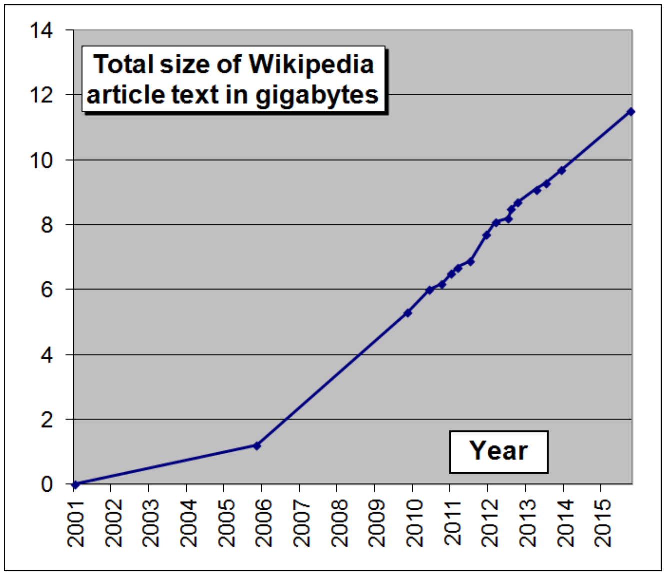


Figure 4.1. Compressed Text Size of All Wikipedia Articles Over Time[130]

eled, knowledge communities can be extracted through various community detection algorithms, and how these extracted communities can model behavior of the Wikipedia website users. Our methods can be extended to any website where similar behaviors occur, such as corporate wiki sites, user forums, and social media sites.

Previous work concerning data from Wikipedia.com has shown how the data can be used to construct ontologies, human language models (through natural language processing), and improvements to the process of requirements engineering[81]. The data used in these works have come from the content of the Wikipedia website itself: ontologies of domain knowledge based on how content is organized on the site [112], natural language models based on the text within the articles [31], and extraction of verbiage for software requirements engineering based on the text within the articles[81]. The form, structure, content, and links present within a Wikipedia article reflect the authors' understanding of the semantic relations between the authored content and other content on the site. In contrast, the aggregated clickstream data reveal how users understand semantic relations between different articles on the site. We posit that research efforts using content from Wikipedia as source material can benefit from analysis of the aggregated clickstream data.

In particular, as researchers use information about how the content of the site is structured to form ontologies of domain knowledge, understanding how users interact with the links which are present can provide valuable perspective into how the content is related. This assertion applies to other knowledge stores as well (such as corporate wiki sites [113]). Understanding how consumers of a site interact with its content is to understand the content from the eyes of those consumers. Collectively, the activity patterns of Wikipedia users can provide insight into the 'relatedness' of articles on the site. We loosely define relatedness as the degree to which two or more articles share similar content, subject matter, and domain knowledge. Relatedness is the heart of semantic structure in a knowledge community. The core of a topic listed on Wikipedia is usually not stored in just one article. Meaning must be extracted from several articles (which might be linked together by contributors to that article). When links between several articles are regularly traversed by readers, logically, traversed links are more germane to the shared topicality of those articles than are those links which might be present but minimally traversed (or not traversed at all). Links listed in an article echo how the author(s) intended the content to be consumed; however, user behavior shows how articles are actually related in a knowledge community.

Existing methods for analyzing these data rely on the content. We extend prior methods with new analysis that includes user behavior. Mahmoud and Carver state [81] the necessity of Wikipedia-based knowledge extraction efforts to be able to focus on individual domains rather than the entirety of the site's contents.

We describe the clickstream data (a small portion of which is illustrated in Figure 4.2) to be a weighted directed graph, the Wikipedia Behavioral Clickstream Graph, where the distinct articles are represented by vertices, the links between articles are represented by edges, and the amount of web traffic between two articles are represented by the edge weights. Based on the behavioral clickstream data however, one immediate observation is that not all links in each article are actually used or represented as edges. Thus, we can truncate the number of links/edges in the Wikipedia Link Graph (a small portion of which is illustrated in Figure 4.3) to be only those links which have been traversed by the users, as observed in Figure 4.2.

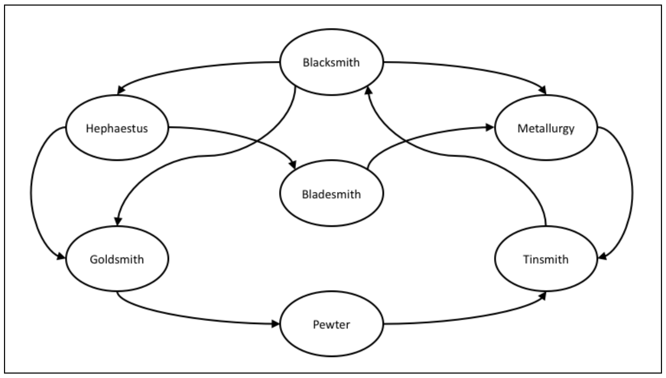


Figure 4.2. Example Wikipedia Behavioral Clickstream Graph

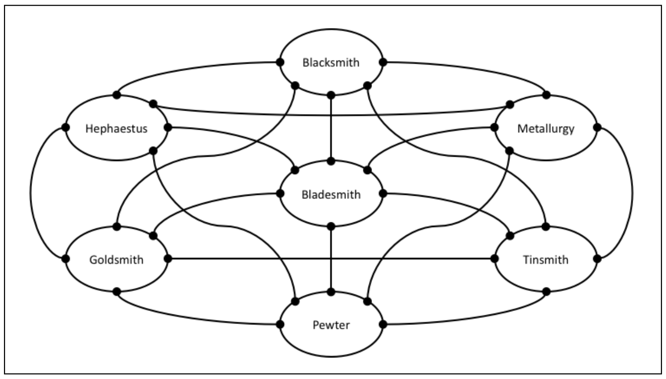


Figure 4.3. Example Wikipedia Link Graph

Table 4.1. Example Wikipedia Clickstream Data

Previous	Current	Type	Count
Blacksmith	Hephaestus	link	1000
Hephaestus	Goldsmith	link	100
Goldsmith	Pewter	link	1000
Pewter	Tinsmith	link	1000
Tinsmith	Blacksmith	link	100
Hephaestus	Bladesmith	link	100
Blacksmith	Goldsmith	link	100
Blacksmith	Metallurgy	link	30000
Metallurgy	Tinsmith	link	100
Bladesmith	Metallurgy	link	100

Classic link-based approaches to building ontologies from Wikipedia use the links in each article (which might be on the order of several hundred, depending on the article). We propose that when constructing a Wikipedia-based domain ontology, the linked articles to which users navigate are more relevant to the domain than those linked articles which are ignored. In addition, articles which are in strongly connected components of the Wikipedia behavioral clickstream graph are topically more representative of the encompassing domain than articles which are only weakly connected. Strongly connected components require components which are constructed of outbound directed edges only, where weakly connected components can be constructed using inbound edges as well. In effect, a weakly connected component can be constructed which contains a set of articles not satisfying Tarjan's classic Connected Component algorithm [116], as discussed in this chapter.

4.2. Wikipedia Clickstream Graph

The first clickstream dataset for Wikipedia was released by the Wikimedia Foundation for the month of January 2015 [136]. Sporadic monthly releases have since followed which represent the aggregated traffic of all articles for the observed month. Table 4.1 shows a simplified example of how the data are arranged; although these transitions exist in the data, the exact frequencies have been simplified for the sake of our example. These data contain a list of links as tuples containing information on the 'Previous' or 'source'/'referrer' article from which the browsing behavior is initiated, the 'Current' or 'target' article which is being navigated to, the volume of traffic associated with the browsing behavior ('N'), and additional classification information on the link itself ('Type'). Individual records of user behavior are obscured by a monthly aggregation of the traffic. The clickstream data provide more information about how the site is navigated other than internal link traffic (traffic from one article on Wikipedia to another); however, as this research focuses on internal traffic, the additional classification information is useful for filtering out external link traffic.[2] The data provides one-hop frequency information. As shown in Figure 4.2, each record in the data provides the previously observed page as well as the currently observed page, for example from Bladesmith to Metallurgy. By constructing a graph of the clickstream, additional analysis of the data provides multi-hop information for base level statistical analysis; moreover, such analysis provides the data to analyze the semantic relationships between the links on the site.

Our clickstream graph $G = (V, E)$ where $V = \{v | v \text{ is a distinct article on Wikipedia}\}$, and $E = \{e = (v_1, v_2, w) | v_1, v_2 \in V, w \in \mathbb{N}, w > 0\}$ is the set of all edges between any two articles on Wikipedia where users have navigated from v_1 to v_2, and where w represents the directed volume of traffic flowing in the direction: $v_1 \rightarrow v_2$. The clickstream graph is similar to the data structure used in the PageRank algorithm [92] employed by Google's search engine framework. The clickstream graph as shown in Figure 4.2 is a sub-graph of the link graph shown in Figure 4.3; however, we see that traffic in the clickstream graph represents search behavior on the site as well. A link to an article might not be present on the site and thus in the link graph, yet it might be present in the clickstream graph. Conversely, links which are part of a source article might never be clicked by users of the site; thus, these links will not be present within the clickstream graph, but present within the link graph. A minor modification to the clickstream graph calculates total transition volume from each source article and represents all edge weights as probabilities of transitioning from the source article of the edge to the target article of the edge, by dividing edge weight by total source volume. The resultant data structure is a MC representing the internal

2 Traffic from Wikipedia to external sites or from external sites to Wikipedia is not useful to this research.

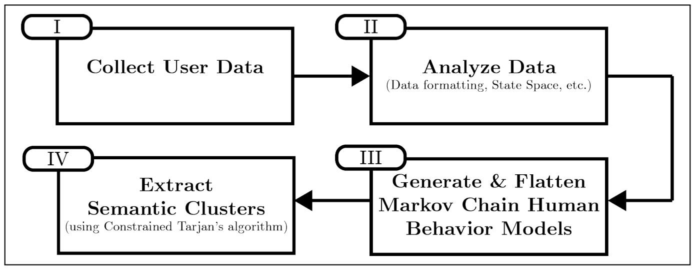

I
Collect User Data
II
Analyze Data
(Data formatting, State Space, etc.)
III
Generate & Flatten Markov Chain Human Behavior Models
IV
Extract Semantic Clusters
(using Constrained Tarjan's algorithm)

Figure 4.4. Process to Extract Knowledge Communities

traffic at Wikipedia as a memoryless stochastic process. However, this MC clickstream graph is large and represents the traffic for all users across all articles on the Wikipedia website. We now focus on how to extract semantic clusters of interrelated articles on specific domains from this clickstream graph.

4.3. Methodology to Extract Semantic Clusters

Semantic clusters in the context of Wikipedia Clickstream Graph are knowledge communities of articles which are topically similar and thus 'semantically related'. The high level process we describe for extracting these communities is shown in Figure 4.4. The steps range from (I) collection and (II) analysis of user data, to (III) flattening and aggregation, and finally (IV) the extraction of semantic clusters. Many methods exist for extracting semantic clusters, communities, and connected components from graphs; however, as described in Section 4.2, each algorithm comes with computational cost to consider. As a MC representation of the Wikipedia Clickstream Graph is simply a graph with additional data associated, we use efficient graph-based algorithms for extracting communities. Tarjan's algorithm[116] is a means of discovering connected components of nodes within a connected graph. A connected component is considered to be "strongly connected" if this algorithm is run on a directed graph by only following the direction of the edges. A connected component is defined to be "weakly connected" if the algorithm is run on a directed graph and follows the edges disregarding the direction (thus as an undirected graph). More formally, Tarjan defines [116] strongly connected components with the following lemma, and provides a Depth First Search (DFS)-based algorithm for discovering the components shown in Figure 4.5.

```
 StrongConnect:
begin:
  index <- 0
  indices <- {}
  lowlink <- {}
  on_stack <- {}
  S <- Stack()
  G <- Graph()

  for node in G.nodes:
    if node not in indices:
      ConnectComponent(
        G.nodes[node]
      )
end
```

```
 ConnectComponent(n):
begin:
  v <- n.value
  indices[v] <- index
  lowlink[v] <- index
  index <- index + 1
  S.append(v)
  on_stack[v] <- True

  for e in n.edges:
    v <- e.left
    w <- e.right

    if w not in indices:
      ConnectComponent(
        G.nodes[w]
      )
      lowlink[v] <- min(
        lowlink[v], lowlink[w]
      )
    elif on_stack[e.right]:
      lowlink[v] <- min(
        lowlink[v], indices[w]
      )

  if lowlink[v] == indices[v]:
    cc <- Component()
    while len(S) > 0:
      w <- S.pop()
      on_stack[w] <- False
      cc.add_node(G.nodes[w])
      if w is v:
        break
    G.components.append(cc)
end
```

Figure 4.5. Tarjan's StrongConnect Algorithm and ConnectComponent SubRoutine [116]

> *Lemma 8:* Let $G = (V, E)$ be a directed graph. We may define an equivalence relation on the set of vertices as follows: two vertices v and w are equivalent if there is a closed path $p : v \Longrightarrow v$ which contains w. Let the distinct equivalence classes under this relation be $V_i, 1 < i < n$. Let $G_i = (V_i, E_i)$ where $E_i = \{(v, w) \in E | v, w \in V_i\}$. Then each G_i is strongly connected [...] the sub-graphs of each G_i are the strongly connected components of G.[3] [116]

Extracting strongly connected components from the clickstream graph is equivalent to finding sets of articles which are circularly linked together through behavioral traffic patterns. Thus, if link traffic in our clickstream graph of Figure 4.2 shows that an article on 'Blacksmith' has traffic to an article on 'Metallurgy', the article on 'Metallurgy' has traffic to 'Tinsmith', and the article on 'Tinsmith' has traffic back to 'Blacksmith', we consider these articles and any others in the path from the source article back to itself as being in one strongly connected component. The weaker condition - weakly connected components - only requires an undirected path between these articles.

When random and systematically iterative behaviors appear in the data, such as those emitted by web crawling programs, it is the case that unrelated topics may become behaviorally related by the data. The resultant behavioral chains then extend through the graph to connect topics which are unrelated. When such long chains of unrelated data appear, we must constrain the depth of the search through these vertices in order to reduce the size of the resultant strongly connected components and increase the semantic relatedness of each component's articles. We introduce a Depth Constraint to Tarjan's algorithm - the recursion through which the STRONGCONNECT algorithm performs its search is halted when the constraint has been met.

Users of the Wikipedia site might not choose to browse hundreds of articles in a single reading or research session; it is necessary to prevent the creation of strongly connected components which might have a deep transitive relationship, but not as deep of a semantic connection between the content of the two articles. An example illustrated with Figure 4.6 is 'Blacksmith'$\longrightarrow$ 'Hephaestus' $\longrightarrow$ 'Mythology' $\longrightarrow$ 'Zeus' $\longrightarrow$ ... $\longrightarrow$ 'Blacksmith'. Using the original STRONGCONNECT algorithm results in a set of articles represented by Figure 4.6. The shaded nodes are the set

3 It is of note here that while the exact definition of G, V, and E are different here from how they are defined in Section 4.3, our definition extends the definition being used in this Lemma.

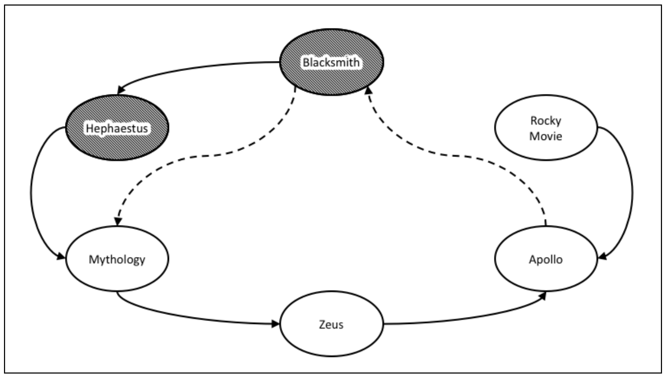


Figure 4.6. Full-Depth Connected Component

of articles most related by their content to ‘Blacksmith’. Where the transitive relationship chain might be thousands of links deep and create a strongly connected component, the resultant connected component will also have thousands of articles (such as ‘Zeus’ and ‘Apollo’) which are not directly relevant to the content of the original article (‘Blacksmith’). Our goal is to reduce the number of articles which have this transitive relationship. We increase the depth of the constraint by powers of 10 to simulate the level of depth the entire knowledge community might browse in extending the content of an article. Constraining the maximum recursion depth to ten links deep reduces the inclusion of tertiary transitive articles in our semantic clusters which are not topically related to the domain being described by the other articles in the cluster.

We modify the STRONGCONNECT algorithm as depicted in Figure Figure 4.7 with bold text and adding a recursion counter which increases with each level of recursion. If a max depth of recursion has been reached and we have traversed out to a maximum distance from the current search node, the recursion is broken and the next child of the search node is processed. Extracting connected components from the clickstream graph with the additional Depth Constraint reduces the number of extraneous nodes in each of these connected components. By reducing extraneous nodes, we are able to increase the semantic relatedness of each connected component. We illustrate via Figure 4.6 through the subset of shaded nodes versus the set of all nodes in the Figure. The more semantically related nodes are shaded where all nodes depicted in Figure 4.6 are only transitively related. Again, semantic relatedness is analogous to cohesion in that

```
 ConstrainedConnect(k):
begin:
  index <- 0
  indices <- {}
  lowlink <- {}
  on_stack <- {}
  S <- Stack()
  G <- Graph()
  max_depth <- k

  for node in G.nodes:
    if node not in indices:
      ConstrainedComponent(
        G.nodes[node],0
      )
end
```

```
 ConstrainedComponent(n,depth):
begin:
  v <- n.value
  indices[v] <- index
  lowlink[v] <- index
  index <- index + 1
  S.append(v)
  on_stack[v] <- True

  for e in n.edges:
    v <- e.left
    w <- e.right
    if depth >= max_depth:
      break

    if w not in indices:
      ConstrainedComponent(
        G.nodes[w],
        depth + 1
      )
      lowlink[v] <- min(
        lowlink[v], lowlink[w]
      )
    elif on_stack[e.right]:
      lowlink[v] <- min(
        lowlink[v], indices[w]
      )

  if lowlink[v] == indices[v]:
    cc <- Component()
    while len(S) > 0:
      w <- S.pop()
      on_stack[w] <- False
      cc.add_node(G.nodes[w])
      if w is v:
        break
    G.components.append(cc)
end
```

Figure 4.7. Constrained StrongConnect Algorithm and ConnectComponent SubRoutine [116]

the articles are cohesive in their meaning. The articles discuss content which is similar in nature and topicality; without the Depth Constraint, the number of unrelated articles produced by the original STRONGCONNECT algorithm is untenable as a model for representing knowledge communities.

4.4. Evaluation

The process for extracting semantic clusters from previously aggregated clickstream data produced results which we present in this section. In addition, we discuss the results and how they impact the process of segmenting communities of related subjects on Wikipedia in a way which capitalizes upon the ways in which users interact with the site as well as being computationally friendly.

4.4.1. Results

The behavioral clickstream data are structured such that the graphical approach subject to the depth constraint is able to reveal information about the articles on the Wikipedia site which might not be readily apparent from the links listed in each article. Where the Wikipedia Links Graph shown in Figure 4.3 could represent several articles which are connected by links, these articles could only be partially connected by content. Specifically, Figure 4.8, Figure 4.9, and Figure 4.10 illustrate how the depth constraint filters noisy traversals between articles. In Figure 4.8, we show how the maximum recursion is set to 10,000 nodes. For each search node, the DFS tree is traversed in the STRONGCONNECT algorithm for at most 10,000 hops until a cycle back to the search node is reached; this is represented by the dashed lines in Figure 4.6.

To traverse this tree without optimization is computationally expensive; moreover, it can be problematic as resultant connected components include many articles which are not cohesively relevant to the domain of the DFS tree's search root. The upper limit of 10,000 is used to illustrate the scale of larger, unconstrained executions of Tarjan's STRONGCONNECT Algorithm. By ex-

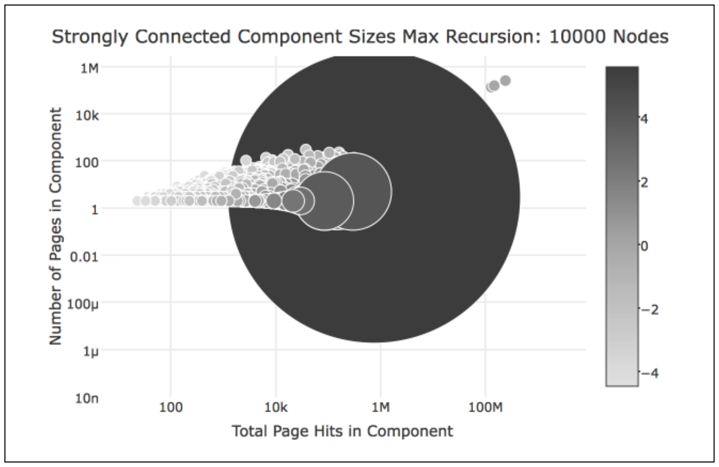


Figure 4.8. Strongly Connected Components With Depth Constraint: 10000

ample of what STRONGCONNECT can produce without suitable Depth Constraint, Figure 4.8, represents the results of the algorithm running with a Depth Constraint of 10,000. There are components which have an inordinate amount of traffic across all the articles inside each component. Similarly, these components have an inordinate amount of articles in each component.

Each circle in Figures 4.8, 4.9, and 4.10 represents a connected component extracted using Depth Constraint. The connected component contains a set of articles which we expect to share content within a shared domain. The size of the circles represent the number of articles in logarithmic scale where their position on the x-axis shows the total amount of traffic for each component. While we might not care as much about the volume of traffic being larger for one component than another, it is problematic to have that many pages in only a few components. Such disproportionately sized clusters are undesirable due to the natural preference for smaller cohesive clusters of topically related articles.

Similarly, for weakly connected components with a maximum recursion depth of 10,000 nodes such as those depicted in Figure 4.9, even more of these components have an overly large amount of pages. Both inbound edges and outbound edges are computed for the weakly connected condition to be satisfied, thus, the weakly connected components algorithm uses even more computational resources than the strongly connected version of the algorithm. By reducing the depth

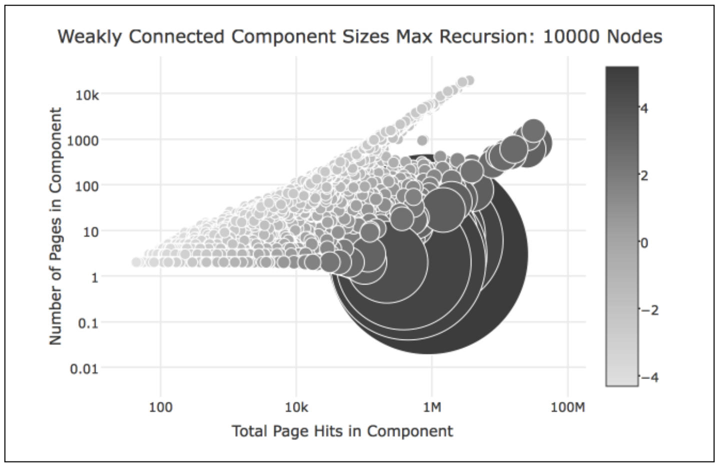


Figure 4.9. Weakly Connected Components With Depth Constraint: 10000

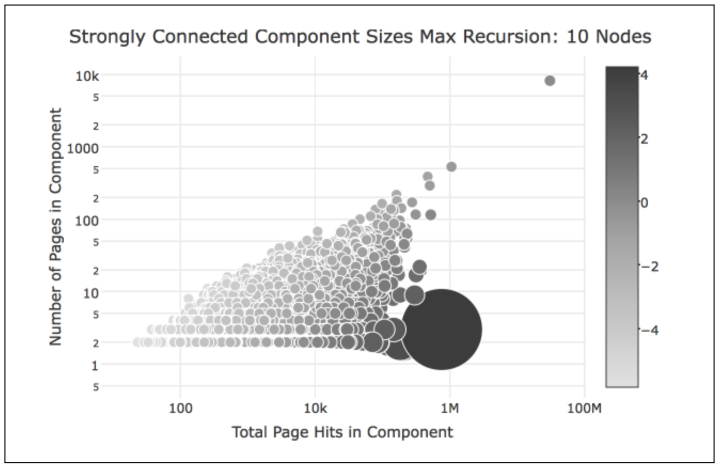


Figure 4.10. Strongly Connected Components With Depth Constraint: 10

of recursion as in Figure 4.10 to 10 nodes (Depth Constraint: 10), a drastic size reduction in the scale of the outliers occurred. With the progression from Figure 4.8 and Figure 4.9 to Figure 4.10, we observe how reducing the Depth Constraint simultaneously reduces the average size of the resultant connected components.

With the classical Tarjan algorithm, the output consists of many tiny connected components and few extremely large connected components as shown in Figure 4.8 and Figure 4.9. An example of a tiny connected component is the set containing the shaded nodes shown in Figure 4.6; in contrast, a large, semantically unrelated component would be the set of all nodes depicted in Figure 4.6 such that the dashed lines represent hundreds or thousands more nodes. Inspection of the large connected components revealed that the articles represented had many differing unrelated topics and thus did not share a holistic semantic relationship. By constraining the depth to which maximum recursion was able to execute to a smaller number of traversed links, we reduced the size of the outlier components by orders of magnitude which can bee seen in Figure 4.10. This change is made observable by the smaller circles shown in Figures 4.8, 4.9, and 4.10 results in the production of components having a more consistent size. The large connected components represent long chains of links in articles which are semantically related to each other but not to articles further down the chain. The chained relations between Wikipedia articles require any DFS-based community detection algorithm to make similar adjustments as we have shown with Tarjan's algorithm. Such adjustments increase the cohesiveness of the articles in each knowledge community.

4.4.2. Discussion

We have shown that using community detection algorithms and adapting graph algorithms toward the goal of detecting knowledge communities can be used to extract semantically related articles from Wikipedia. By constraining the depth to which such community detection algorithms execute, we are able to reduce the number of extraneously related articles in the resultant knowledge communities. The articles maintained by such knowledge communities are semantically related because users navigate the site by clicking links on each source page to reach the next target page and the community-based content-management processes governing the site reflect that related and relevant links are typically present on each page [136]. Modeling knowledge communities as individual behavioral MC models will enable researchers to extend previous efforts at constructing ontologies, requirements engineering, knowledge mapping, and semantic mapping of the articles on Wikipedia. In addition, using Wikipedia's behavioral data requires orders of magnitude less space and memory as compared to the raw article text. If we include the amount of data related to users editing pages, the costs of space can rise by an additional four or five orders of magnitude. For researchers on a constrained budget, these behavioral data could provide insight previously not available at a space/computational cost reduction. In addition, using Depth Constraint to extract knowledge communities from the data simplifies the process of analyzing the semantic relatedness of the Wikipedia.com website content.

These results show how clickstream data can be used to extract semantic clusters from Wikipedia data. Previous methods using only the link graph to construct Wikipedia-based domain ontologies result in groups of articles which include unrelated pages, as index pages and other unrelated links appear in every article on the site. Inclusion of unrelated pages is due to extraneous transitive links; the 'backtracking' observed in the study by [142] could provide additional insight into or partially explain how user behavior with the Wikipedia site could cause such results. The depth constraints presented reduce the inclusion of such unrelated transitive links in semantic clusters. The connected component approach is used by Janik and Kochut [63] for text categorization and classification of non-Wikipedia documents; however, our usage for the connected components algorithm is focused on extracting semantic clusters from Wikipedia. Janik and Kochut create a graph, extract connected components, and use Wikipedia to classify documents according to a similar graph constructed from Wikipedia data. However, Janik and Kochut do not use clickstream data; rather, they focus on using the natural language from the documents they seek to classify and the natural language from the articles on Wikipedia. While

Janik and Kochut use similar methods, the output and focus of their work is different from our research, in that they seek to classify documents through features extracted from the natural language of documents, not create semantic clusters of related documents through the behavioral analysis of the users interacting with those documents.

By reducing the recursive depth of the Depth First Search steps in the STRONGCONNECT algorithm, the nodes in the resulting components are more tightly connected than those which can be extracted from the links graph. This 'tight-connectedness' property filters extraneous edges from components of large graphs such as social networks or computer networks. We have contributed the Depth Constraint extension of the STRONGCONNECT algorithm; however, an additional way in which the data can be constrained is the Pareto Constraint, formed by looking at the weight associated with the edges of the graph. While the edge weight represents traffic volume, it is not immediately relevant for the construction of the Depth-Constrained connected components within our clickstream graph; moreover, these edge weights could be used for pruning the size of the connected components by removing links which only have transitive relatedness to the overall component. We define this procedure to be a Pareto Constraint; in other words, if the inbound and outbound traffic volume associated with a vertex is below an observed or predefined Pareto frontier, we disclude that vertex from the connected component. If the vertex is discluded from the connected component, all of that vertex's outbound neighbors must thus be discluded as well unless they can be visited by a transitive neighbor of the originating node of the STRONGCONNECT algorithm's current iteration. We plan future inquiry on Pareto Constraint to restrict the size, breadth, and depth of connected components in large network graphs.

4.5. Summary

We have shown a modified version of Tarjan's STRONGCONNECT algorithm which we use to extract semantic clusters from a monthly aggregation of Wikipedia.com user browsing patterns. We represent the aggregated browsing patterns as a graph. Previous community detection algorithms do not run in the same order of efficiency as Tarjan's STRONGCONNECT algorithm, and Tarjan's STRONGCONNECT algorithm is not typically used for extracting communities from graphs, or network structures. The changes we have made to Tarjan's algorithm capitalize upon the compact structure of Wikipedia browsing data, the efficient structure of Tarjan's Algorithm, and the relatedness of the sites to extract meaning from data which previously has not been explored. Our modified version of the algorithm reduces the depth to which the embedded search algorithm in STRONGCONNECT traverses the graph. The output of the algorithm is a set of connected components which represent related articles within a shared subject domain. This depth constrained graph-based approach to extracting knowledge communities uses aggregated browsing data to capitalize on user behavior to produce representative clusters of semantically related articles on Wikipedia.com.

With this chapter, we demonstrated that aggregated batched behavioral data in a human-machine system can be systematically grouped into semantically related MC models, which we wished to show with RQ_2 from Chapter 1. One of the key ways of understanding a set of users in an online community is by understanding how those users interact with the systems also in that community. Such interactions most often are captured in various forms and can be stored and modeled as a large interconnected graph of nodes and edges. With the Depth Constraint this chapter contributes, we have shown a general means of extracting clusters of nodes within a graph based on the weight and direction of the edges in that graph. Where Tarjan's STRONGCONNECT algorithm provides an efficient means of segmenting large graphs such as the Wikipedia Clickstream graph, the imposition of a Depth Constraint to the STRONGCONNECT algorithm allows for larger clusters of nodes to be further segmented. Our Depth Constraint contribution to Tarjan's STRONGCONNECT is useful in the research efforts related to graph segmentation, data clustering, and general knowledge extraction from large data sets. By segmenting such large sets of interrelated data into smaller segments, we reduce the amount of computation and time it takes for analysts and scientists to understand the data. Smaller succinct models are also optimal for automation efforts such as those used in the generation of realistic workloads on a system under test. We discuss how such models may be alternatively learned and automated in the following chapter.

Chapter 5. Learning Adaptive User Models from Streaming Data

The consequences of an act affect the
probability of its occurring again.
-- B.F. Skinner
Wade & Tavris, Psychology

A critical step in the testing of modern software is the generation of realistic workloads on a system under test before that system under test is released to production, also known as load testing. Appropriately generated workload adequately emulates the expected behaviors of users and machines within the system under test to find potential failure states. Typical testing tools rely on static testing artifacts such as scripts or recorded packet streams to generate realistic workload conditions. Such artifacts can be cumbersome and costly to maintain; however, model-based alternatives to static artifacts do not allow for reactive adaptation to changes in the system under test or its currently observed usage. Model-based alternatives do not address the sunrise problem - when failure states exist in the system under test but are not empirically modeled due to an incomplete amount of data.

We describe the formalisms of Markov Chain behavior models which are useful for load testing and capable of being extended. We detail a novel approach to modeling and executing workload which provides greater flexibility over traditional static and model-based load testing tools in three ways. The first improvement is by evolving observed Markov Chain behavior models to Markov Decision Process behavior models through reinforcement learning. The second is by efficiently solving the sunrise problem with amended sparse data structures to facilitate Laplace smoothing over the sample state space. The third is by providing our models with the streaming ability to learn and forget previously observed behavior probabilities in the system under test such as new states or pruned execution paths via a least-recently-used cache.

5.1. Introduction

The spontaneous growth of human-machine systems over the past century has provided an expanding treasure trove of benefits to society; however, with such benefits comes the responsibility of ensuring that technology does not adversely affect those who use it. One of the critical roles that data scientists can fill on system development teams is the determination of features and aspects most used and beloved by a system's users [11]. Understanding which features are most used is an important part of knowing which code paths are necessary to upgrade, test, and support between and after software releases. Previous work by van Hoorn has shown that the application of data scientific methods and models to system logs to create operational profiles of the system can extend the capabilities of classical tools used for testing [58]. Testing software based on operational profiles, behavior and process models is not new; neither is the process of discovering operational profiles and models from behavioral data. However, to the best of our knowledge, the application of operational profiles toward an evolving, realistic, online load testing system based on Markov Chains (MC) has not been fully explored.

Automated testing tools extend the capabilities of traditional human quality assurance engineers (QA) through the use of data captured from a system under observation (SUO, or $\mathcal{S}_o$). These data are a vehicle for modeling behavior and simulating the various types of behaviors which might elicit potential faults within the system under test (SUT, or $\mathcal{S}_t$). It is important to delineate these two system environments: the SUO is a production system where realistic data can be captured for modeling and mining, where the SUT is a staging system mirroring the SUO, but strictly reserved for pre-release testing of code and infrastructural changes to detect hidden faults in the system change delta. Data streams from the SUO are typically put through an extensive process where they are queried, exported, defined, mined, and modified to become monolithic testing artifacts for usage in tools such as JMeter [58] and Selenium [19]. As the SUO evolves with time to meet the needs of the human-machine system in which it is placed, the testing artifacts must also change to meet those same needs. The consequence is that accurate and consistent LT of a SUT is a cumbersome undertaking requiring significant re-work by QA [104].

Model-based approaches to LT such as Markov4JMeter by van Hoorn [58] have been proposed to address the required manual update of static artifacts, but have a shared susceptibility to adaptation issues when the underlying system undergoes significant changes between releases [13]. In addition, sufficient data from the SUO between releases may not exist to accurately update testing artifacts or models for testing [13]. Model-based approaches have been shown to suffer from adaptation challenges brought about by the proliferation of continuous integration practices such as DevOps and Agile Programming [13]. To help address such challenges within the scope of the challenges and problems (C_{1-4}, P_{1-5}) listed in Chapter 1, we list the following problem statements addressed by the contributions of this chapter:

Problem Statement 5.1*:* Existing approaches to LT currently require significantly repetitive manual effort to integrate into a living testing and DevOps environment. (See C_1, C_2, C_3, C_4, P_1, and P_3 in Chapter 1.)

Problem Statement 5.2*:* Existing approaches to LT do not learn to actively search the SUT for flaws while still following the previously observed behavioral distributions. (See C_4, P_1, P_2, P_3 and P_4 in Chapter 1.)

Problem Statement 5.3*:* Existing approaches to LT do not allow for stale, inactive, or dead activity patterns to be filtered when generating testing artifacts, testing, or modeling. (See C_1, C_2, C_3, C_4, P_1, and P_3 listed in Chapter 1.)

Problem Statement 5.4*:* Existing approaches to LT do not account for missing or incomplete data when generating testing artifacts, testing, or modeling. (See C_2, C_3, P_1, P_2, and P_5 listed in Chapter 1.)

Problem Statement 5.5*:* Existing approaches to LT do not provide a comprehensive set of formalisms for modeling a human-machine system. (See C_4 in Chapter 1.)

In addressing these problems, we show how logs from a mirrored production environment can facilitate the automatic maintenance of an online model of usage behavior (Problem Statement 5.1) for realistic testing against an SUT. Recall Figure 3.4 for the interplay between the SUO and the SUT. We discuss an approach for describing the various behavioral functions in a SUO

(Problem Statement 5.5). In addition, we define an extension to MC "behavior mixes" [58] with Q-Learning, allowing for the model to evolve to find faults within the SUT (Problem Statement 5.2). We apply streaming versions of data structures, statistical methods, and caching to allow for operational changes in the SUO to be reflected in the testing patterns applied to the SUT (Problem Statement 5.3). Such improvements also reduce the computational cost and space required to operate, store, and maintain operational models. Finally, we apply Laplace smoothing to our MC behavioral models to account for faults which might exist within the SUT, but not reachable through data-modeling (Problem Statement 5.4).

5.2. Q-Learning Behavior Models

We developed the process of evolving MC models to Q-Learning Behavior Models (MCQL).

Q-Learning is a machine learning technique useful for emulating emergence within a HMS with which learning Agents are interacting. Due to the mathematical similarities in the underlying mechanisms of Q-Learning and MC models, an intuitive mapping exists toward a MAS built on these mechanisms. Q-Learning allows Agents to follow state transitions which have previously been marked as more fruitful than others in the process of optimizing toward the goals of the MAS. We provide the following definition of Q-Learning by Mitchell for the reader [87]:

> The optimal action in state s is the action a which maximizes the sum of the immediate reward $r(s,a)$ plus the value of V^* (equivalent to V^{π^*}) of the immediate successor state, discounted by γ :
>
> $$V^{\pi} \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+1} \tag{5.1}$$
>
> $$V^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s) \tag{5.2}$$
>
> $$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \tag{5.3}$$
>
> where $\delta(s,a)$ denotes the state resulting from applying action a to state s. An Agent can acquire the optimal policy by learning V^*, provided it has perfect knowledge of the immediate reward function r and the state transition function δ. In instances where we do not have access to such reward or state transition functions, we resort

to the Q-function for evaluation of the optimal state transition.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \tag{5.4}$$

It follows that we can rewrite equation 5.2 as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a) \tag{5.5}$$

Maximum Likely Reward is a greedy way of determining the most optimal path for generating an error in $\mathcal{S}_t$. In calculating the most likely path and the most likely error action based on a MC behavior model, we can filter out testing scenarios in which typical system behavior is expected. By using Maximum Likely Reward, we provide a mechanism for our learning Agents to pursue those errors which might have been previously un-observable within the SUT.

Theorem 1*:* MCQL provides the Maximum Likely Reward in a stochastic process where rewards are given for Agents reaching specific states.

Proof. Assume functions $\delta : S \times A \longrightarrow S$, the transition function, and $r : S \times A \longrightarrow \mathbb{R}$ the reward function. Further assume we are given $\mathbb{M}$, a Markov Chain represented in its matrix $P_{i,j}$ form to represent the probability that $\delta(s,a)$ can occur for all $s \in S, a \in A$. As the state space of $\delta(s,a)$ and $\mathbb{M}$ are equivalent, we can define a function $\delta_{\mathbb{M}} : \delta(s,a) \longrightarrow \mathbb{R}$ which denotes the probability that $\delta(s,a)$ will occur. Then, for all $s \in S, a \in A$, we can calculate the MCQL function as follows:

$$Q_{\mathbb{M}}(s,a) \equiv r(s,a)\delta_{\mathbb{M}}(\delta(s,a)) + \gamma V^*(\delta_{\mathbb{M}}(\delta(s,a))) \tag{5.6}$$

As all terms are weighted by the probability that event an can occur, we force the argmax to return the Maximum Likely Reward. *Q.E.D.*

In the case of this work, the Maximum Likely Reward is the chain of transitions through $\mathcal{S}$ such that the Q-Learning reward and ϕ_p are both maximized. In order to address the situations in which ϕ_p results in zero, we must solve the sunrise problem: namely, determining the conditional probability that an event will occur even though it has not been observed in the sample data.

Table 5.1. Conditional Probability Feature Vectors for Example Users

	(σ_1, σ_1)	(σ_1, σ_2)	(σ_1, σ_3)	(σ_2, σ_1)	(σ_2, σ_2)	(σ_2, σ_3)	(σ_3, σ_1)	(σ_3, σ_2)	(σ_3, σ_3)
U_1	0.4	0.3	0.3	0.3	0.4	0.3	0.3	0.3	0.4
U_2	0.3	0.4	0.3	0.4	0.3	0.3	0.3	0.3	0.4
U_3	0.4	0.3	0.3	0.3	0.3	0.4	0.3	0.4	0.3
U_4	0.0	0.6	0.4	0.2	0.1	0.7	0.4	0.4	0.2

5.3. Streamlining MCQL Data Modeling

We describe the process to efficiently mine and store MC behavior models in order to solve the sunrise problem with amended sparse data structures to facilitate Laplace smoothing over S. We imbue our models with the ability to "forget" previously observed behavior probabilities in the SUO through a least-recently-used (LRU) cache.

As $|S| \in \mathcal{S}$ can grow quite large, causing the number of transitions in ϕ_p to grow large as well, it is important to efficiently implement MC behavior models so that they may be used in an operational environment. We rely upon sparse vectors and sparse matrices [118] to reduce the operational memory and computational footprint. Sparse vectors are similar to hashed map data structures where the entry key is the vector index, the value is the value stored at the vector index, and most importantly, zero-values such as $U_4 \rightarrow (\sigma_1, \sigma_1)$ depicted in Table 5.1 are not stored. Sparse vectors allow for efficient computation of matrix and vector operations by skipping zero-valued indexes in the structure. While our computational examples provide non-sparse values for convenience, it is of note that most operational profiles will be extremely sparse in nature.

Such sparseness within data from the SUO presents a problem when dealing with novel states and transitions in an SUT. As such, we must present a solution to the so-called sunrise problem while leaving our approach computationally efficient. The sunrise problem is a classical way of representing the difficulty that descriptive statistics has with attempting to model the probability of an event occurring when prior examples do not exist in the observed sample data. Colloquially, we want to know the likelihood that the sun will not rise tomorrow. As the event in question has never occurred, we do not have a rough estimate of the requested likelihood. Such an estimate may be calculated with Laplace smoothing which is the inclusion of a non-zero smoothing value to a priori distributions in order to account for the occurrence of rare events.

We thus extend the sparse vector and sparse matrix algorithm with a smoothing factor l such that $SparseVector_l$ is a $SparseVector$ where the zero values are replaced with l but not stored in memory, and the non-zero values are discounted by a fraction of l, such that $||SparseVector_l|| = ||SparseVector||$. We similarly extend the definition of a sparse matrix, such that non-zero values are discounted according to the l value and the row-major sums still equal to one. By smoothing our internal data structures while maintaining the computational and storage efficiency associated with sparse vectors, our approach allows for potential of changes existing in C_δ when representative sample data are not available for behavior modeling.

However, when sample data are available through historical means, but a series of applied system changes $C_{\delta_n}, n \in \mathbb{N}_{[1,\infty]}$ renders those data stale or inaccurate, we must also be able to dynamically adjust our MC models accordingly. Such updates should optimally occur without the need for major interactions from QA [104], in order to increase the usefulness of our approach. We recall the User (UBB), Profile (PBB), Semantic (SBB), and Global (GBB) Behavior Bases we introduced in Section 3.4. Each frequency-based adjacency matrix which is used to generate UBB,PBB,SBB, and GBB is adjusted to account for any attributed staleness in the data by providing a configurable range of time beyond which data are truncated, or forgotten. This is similar to a Least-Recently-Used Cache in systems development, and based on a priority queue with the priority of the stored events within the queue being the time τ when each event occurred. When a truncation filter range is reached, all elements beyond the priority range of the queue are purged, thus removing stale data from our operational models. By converting our behavior bases to use smoothed sparse data structures, the possibility of those events occurring are still non-zero, so the truncation of the data is non-destructive.

To evaluate our clustered models against a given dataset, it is necessary to show the relative divergence between models in a means other than the distance metric used for clustering. Various statistical tests, metrics, and pre-metrics exist to provide the required comparative power. However, in many instances such as Fisher's Test of Independence, χ^2, G-Test, and Kullback-Leibler Divergence (D_{KL}), non-zero frequencies, probabilities, categorical observances must exist for appropriate comparative behavior to occur. To overcome situations where the data are sparse, we use Laplace smoothing with $\alpha = 1$ to ensure that event frequency matrices are non-zero, overcoming the sunrise problem. Frequency-based Laplace smoothing over a categorical distribution is $Pr(i) = \frac{x_i+\alpha}{N+\alpha d}$ where d is the number of known categories, x_i is the number of observations in the i^{th} category, and N is the total number of observations in all categories. After the data are smoothed, we use D_{KL} to measure the accuracy of our clustered models.

The Kullback-Leibler Divergence is defined as

$$D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} = -\sum_i P(i) \log \frac{Q(i)}{P(i)}$$

and is undefined where $P(i) = 0$ or where $Q(i) = 0$. This divergence is roughly a measure of the amount of 'surprise' associated with model Q when the expectation is to see results based on the parameters (or prior probabilities observed) from model P. The undefined nature of this function over zero observed probabilities in P or Q has forced an ergodicity requirement for Markov Chain comparisons in previous work.

To determine if the information conferred by the UBBs is represented by the clustered set of PBBs, we let O_m be a trace observed from model m. We then wish to see if

$$D_{KL}(O_{UBB}\|Data) \leq D_{KL}(O_{PBB}\|Data). \tag{5.7}$$

The information conferred by the UBBs is represented by the clustered set of PBBs when the inequality in Equation 5.7 holds. The process of studying and simulating the MAS begins when the UBBs and PBBs have been evaluated and we know if the MAS should consist principally of the UBBs or if we may use the PBBs instead. If system resources allow, the reduced PBB models can be executed in parallel with the UBB models or with the GBB model.

5.4. Summary

With the large amount of data in systems, it is critical to be able to adjust to changes in the underlying data in addition to inferring probabilities from data which may not be present. The impact of understanding, modeling, testing, and simulating human-machine systems is crucial to obtain for increasing the efficiency of our societal co-evolution with technology. Data scientists must provide materials and methods toward such ends – addressing computational and storage problems in non-traditional ways. With this work, we developed a set of formalisms for merging Markov Chains with Q-Learning (MCQL) toward the goal of harnessing the emergent behavior typically associated with Agents in a MAS. We proved that MCQL provides the Maximum Likely Reward in a stochastic process where rewards are given for Agents reaching specific states. We detailed the steps for streaming sample data into MCQL and how MCQL uses Laplace Smoothing with Sparse Vectors and Sparse Matrices to efficiently model expected behavior as well as to allow for the exhibition of unexpected behaviors by MCQL. MCQL is capable of simultaneously mining and modeling event data for model-based load testing of software systems and is based on the data structures and methods required for improving previous approaches.

We presented five problem statements and addressed them with a series of formalisms and implementation mechanisms which can extend the capabilities of a workload modeling and simulation engine. The first problem (Problem Statement 5.1) dealt with the manual effort involved with scrubbing and maintaining operational models for LT. We addressed Problem Statement 5.1 with a tandem SUO/SUT environment, having event data automatically being mined and modeled through the pipeline depicted in Figure 3.4. The second problem (Problem Statement 5.2) reflects that current model-based LT approaches do not actively search for flaws in an SUT; rather, they simply execute the operational models provided to them. We addressed (Problem Statement 5.2) with the combination of Markov Chains and Q-Learning (MCQL) which opens up the prospects of various types of simulations in system fault testing, performance testing, security testing, as well as functional and non-functional requirements testing.

Principally, MCQL allows for human-like variation in testing procedures to occur. By introducing MCQL into a testing approach, we enable additional testing paths previously unavailable through rote scripting, test cases defined by hand, and even model-based LT such as Markov4JMeter. The third and fourth problems (Problem Statements 5.3 and 5.4) describe incomplete, inactive, or dead operational patterns existing in stale operational data – models derived from which are not able to adjust to changes in the SUO. Our approach includes formalisms and data structures necessary for efficiently addressing the sunrise problem and adapting to changes in a SUT. Such formalisms were missing from previous approaches (Problem Statement 5.5) and their inclusion here allows for future development and research to advance in the area of pre-emptive workload-related error detection in a SUT. MCQL and the computational optimizations provide an avenue for QA to introduce intelligent variation into LT tools.

With this chapter, we demonstrated that raw streaming behavioral data can be continuously and efficiently measured, modeled, and stored as MC models (RQ_2 from Chapter 1); moreover, such MC models of a human-machine system can be extended to adapt to changes in a system under observation (RQ_4 from Chapter 1). In order to meet several challenges in the field of automated systems testing, we contributed the pairing of stochastic processes in the form of Markov Chains with reinforcement learning with the Q-Learning algorithm for learning of adaptive user behavior models from streamed data. We also contributed modifications to the SparseVector and SparseMatrix data structures which allow for simulation of novel behaviors with Laplace Smoothing. Our modified data structures also can exhibit the ability to forget unused behaviors through a least-recently-used cache. We showed how the Kullback-Leibler Divergence can be used as a checkpoint to determine if PBBs may be used for user behavior simulation as an alternative or complement to UBBs. The contributions of this chapter address problems imposed upon traditional testing platforms such as JMeter [117] by the evolution in systems development practices. The formalisms, algorithms, data structures, and general concepts specified in this chapter catalyzed the development and refinement of a real-time approach for generating workload in software systems, detailed in Chapter 6.

Chapter 6. Distributed User Modeling and Load Testing

Handfuls make a load.
-- Irish Proverb
Routledge Book of World Proverbs

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. Tools and processes exist 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 [94].

We developed LODESTONE: a real-time process for generating workload in software systems. This real-time process 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 [94].

6.1. Introduction

[1]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 processes to automatically evaluate the quality of a software system are continuously increasing; however, such processes do not readily incorporate real-time understanding of and feedback from a SUT. SUO allow for logs and metrics to be captured. Figure 6.1 shows an example log from a SUO. However, such logs and

1 Contents of this chapter adapted, with permission, from:
C. Parrott, and D. Carver, "Lodestone: A Streaming Approach to Behavior Modeling and Load Testing," 2020 3rd International Conference on Data Intelligence and Security (ICDIS), ©2020 IEEE.

```
{
  "source_state": "login",
  "target_state":"role_view",
  "transition_time":1500000087,
  "user":"bbicke",
  "session":"9474dec4-f3f8",
  "errors":null,
  "ResponseMetadata": {
    "HTTPHeaders": {
      "connection": "keep-alive",
      "content-length": "2",
      "content-type":
         "application/json"
    },
    "HTTPStatusCode": 200,
    "RetryAttempts": 0
  }
}
```

Figure 6.1. Recorded Log Data from SUO (©2020 IEEE)

metrics are not always brought into an analytically modeled or executable form for performance or quality testing due to their complexity and the associated cost to properly analyze and utilize. In systems which must continuously be evaluated for quality, various factors must be accounted for - from functional requirements such as consistent user experience, to non-functional requirements such as system up-time and security concerns. Where sophisticated tools and processes can potentially ensure quality within an SUT or an SUO, such tools and processes are anecdotally eschewed in favor of other quality-based metrics such as: estimates from engineers, projected timelines, agile sprint burndown, analysis and reporting, as well as the fiscal costs associated with completed project delivery.

The inter-dependency between people, process, and technology portents a paradigm shift in how our organizations and systems should be structured, viewed, and modeled.

6.2. Modeling a Human-Machine System

The migration toward automated and computational processing of units of work in organizations has led to an easily observable co-dependence between human and machine entities teaming to fulfill organizational goals. Conway's Law [32] states that organizations create systems of machines which are roughly reflective of the organization in which those systems are created. As such systems are reflective of the organizations, so too are the organizations reflective of the requisite systems. Even casual study of traces or logs produced by systems of humans and machines can cause an observer to lend credence to the attribution of 'Agent-like' properties to the various entities which compose a human-machine system [138]. While machines are not sentient, their composition imbues within them a set of 'Agent-like' properties analogous to the 'Agent-like' properties of their human counterparts [106, 109]. Capable of sharing similar properties, actions, and goals, users and machines may both be equivalently modeled in a human-machine system as Agents [21, 24, 64, 109]. We define an Agent as an entity capable of executing a set of actions to accomplish one or many given or discoverable goals [106]. In the computational sense, Multi-Agent System(s) (MAS) are an instance of distributed computing in which each computational node executes actions as commanded by one or more Agents [21, 109]. From a semantic perspective, a MAS is simply a collection of Agents working toward a larger purpose which is either pre-defined or emergent, by design [34, 106, 109]. There exists utility for MAS to inform various aspects of modeling, simulating, and testing of realistic events within the context of a larger system [40, 65, 139].

Effort has been dedicated to modeling and understanding how people, process, and technology come together to form a cohesive and productive human-machine system [32, 76, 101, 114]. User behavior mining uses event data from systems to prescribe or describe sets of behaviors for improving user experience or controlling user behaviors in a human-machine system [29, 30, 33, 39]. Process mining relies upon event data from human-machine systems to derive repeatable models representing how such events are typically sequenced within a human-machine system [121]. Several main data models are used by these modeling approaches, such as Petri Nets [25], Causal Nets [121], and Markov Chains [114]; however, the MC is the model most ubiquitously extending to other areas of behavior modeling and research [72]. Simpler models such as the MC can provide much more context toward the goal of improving a human-machine system [46] by allowing for feedback and approval from a wider audience such as business management professionals, software engineers, and QA, all of whom are instrumental in designing, building, and maintaining the systems critical to the functional well-being of our society [105]. Human-machine systems are developed by software engineers through a systematic process known as software engineering and evaluated by QA in the testing process.

6.3. Testing a Human-Machine System

It is a well known problem that the cost of maintaining a system after it has been deployed is much higher than the cost associated with producing the system itself. The software engineering process - whether agile, waterfall, or some other development process is used - usually begins with an inception phase in which the software is conceptualized [115]. Inception is followed by a requirements elicitation phase, in which the desired behaviors of the system are documented [80]. Requirements documentation is a time-consuming and expensive process which sets the standards of quality for the rest of the software engineering process, when done correctly [81].

The requirements process results in a detailed description of how a project should be designed, implemented, tested, and maintained [81]. If an unaddressed error or oversight in the requirements process occurs, a flaw in the system may be the result [115]. Due to human error, such flaws are not always detected in the testing process as they are not accounted for in requirements documentation [115]. It is quite typical for non-functional system specifications to be under-documented as compared to functional requirements [80]. It follows that the process of testing non-functional requirements must tend to be less thorough than that of testing functional requirements. Incomplete evaluation of a system under test may have long-reaching consequences in the maintenance phase of the software development lifecycle when it becomes a system under observation [80]. A system under test $\mathcal{S}_t$ is roughly equivalent to a system under observation $\mathcal{S}_o$ plus a change delta C_δ, i.e.

$$\mathcal{S}_t + C_\delta \ \approxeq \ \mathcal{S}_o.$$

Undiscovered faults in a system under observation can result in failures ranging in severity from loss of productivity to loss of life and limb, depending on the level of risk associated with usage [22]. 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 updates from a system under test. One type of software testing is the generation of workload on a system under test to ensure that system meets expected performance requirements [68].

To preempt and reduce the monetary cost associated with system maintenance, a QA uses manual processes and automated tools to validate an acceptable level of quality within a system under test [115]. One set of tools and processes, load testing (LT), is a subset of automated testing which focuses upon the non-functional qualities of a system under test such as security, stability, and scalability [68, 80]. LT is used to evaluate the quality of the system under test through exposure to expected workload [68]. A successful LT process evaluates a system under test through repetitive and lengthy exposure to various patterns of system usage [105]. A properly maintained and developed LT process and infrastructure are critical for reducing the cost associated with maintaining production systems by exposing failures inconspicuous to other automated or manual testing efforts. We posit that LT processes should also: create a realistic test-bed for evaluating a system under test, be cost-efficient as well as computationally efficient, be straightforward for QA to employ, and be unambiguous for non-QA to evaluate. Widely available tools such as JMeter [117] and Selenium [19] are well-established as an effective means of LT in both the industrial and research communities. Although JMeter has begun migrating toward large-scale distributed execution of test cases, it is still a stand-alone product, lacking the capabilities of cloud-native testing systems [15]. Tools like JMeter primarily rely upon recorded or configured scenarios to submit a stream of statically defined stimuli to a system under test [105].

Static scenarios can be a powerful tool for discovering flaws in a system under test 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 [105]. As artifacts, static scenarios must be regularly pruned, updated, and maintained by QA in order to combat the risk of diminished expected value [105]. The bulky nature of testing artifacts can emit financial and productivity costs such as: storage, transmission, security, execution, maintenance, and modification [105]. Due to the cost overhead associated with monolithic testing artifacts, simplified models of such scenarios have been shown to be alternative to static LT scenarios [48, 105]. Model-based LT processes draw from various bodies of research (data-mining, statistics, process mining, and machine learning) to reduce the operational and computational cost associated with static scenarios. However, model-based LT processes can also suffer from the same cost overhead as static scenarios: execution, maintenance, and modification [105]. Model-based LT processes typically trade thoroughness for simplicity, so they must be carefully validated by non-QA as well; toward this end, simple statistical models such as the MC have been used to mature the research in LT [105]. Maturation in processes and tooling is required for QA to adapt to modern architectural choices and system development strategies such as DevOps, continuous integration, and microservices [13].

6.3.1. Research Direction

To the best of our knowledge, a streaming cloud-based process to LT does not exist which is designed to meet the accelerating modeling and responsiveness requirements of the agile development practices and uses a Markov process, such as in WESSBAS and the Markov4JMeter extension to JMeter. To help address these limitations, we developed 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 process to ingesting system logs, modeling and simulating human-machine behaviors, and executing realistic LT upon a human-machine system. We recall RQ_1 from Chapter 1 in order to focus the direction and intent of our investigation.

RQ_1: Can a data-driven process for MC model-based load testing be developed that offers advantages over existing data-driven processes?

We want to show that if such a data-driven process for MC model-based load testing is developed, then it offers advantages over an existing data-driven process. The following two assertions establish the sufficiency of LODESTONE as it pertains to RQ_1:

A_1: LODESTONE extends the features of an open-source tool for MC-based LT such as JMeter.

A_2: When compared to an open-source tool for MC-based LT such as JMeter, LODESTONE provides measurable performance benefits as the scale of LT increases.

In this chapter, we substantiate the validity of Assertions A_1 and A_2. Sections 6.2 and 6.3 contain relevant research related to MC model-based load testing. In order to provide additional context and terminology, we discuss User Behavior Modeling for LT in Section 6.4. We describe the overall process and the architecture of LODESTONE in Section 6.5. Section 6.6 contains an evaluation describing the quantitative and qualitative guidelines used for our study and a discussion of our results. We conclude with a Summary and potential future directions in Section 6.7.

6.4. User Behavior Modeling for Load Testing

Throughout the ongoing effort of synthesizing and evolving tools and processes to measure and improve the operational quality of software systems, it is imperative to account for the human factor exemplified by the practices of QA. We describe microservices, system logs, and behavior modeling as related to LT. A data-driven model-based process to LT allows for a certain degree of human facility to be imposed on the process.

6.4.1. 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 L to be a sextuple

$$L = (\sigma_s \in S, \sigma_t \in S, \tau, u, e, s)$$

such that σ_s is a source state and σ_t is a target state 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 will also refer to such sextuples as logs. The process of discovering and modeling through log structures is described by Menasce et al. [85] as Customer Behavior Model Graphs and extended by Menasce in [84]; further, a streaming log processing process is discussed by Du and Li [41]. In order for the process of modeling logs to commence, a mechanism must exist for collecting a stream of logs from raw web responses, filesystems, and databases; moreover, such a mechanism must also parse, clean, and pass sextuples into a readily-available

```
{
  "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"
}
```

Figure 6.2. Recorded Session Data from SUO (©2020 IEEE)

location for reporting and analyses purposes, such as extracting behavior profiles. A cleaning, filtering, and modeling mechanism is not built into Markov4JMeter [58] as JMeter is a LT tool not a data-mining tool; however, WESSBAS [126] approaches the problem of modeling the data required for Markov model-based LT with a domain-specific language.

6.4.2. Modeling User Behavior

A simple process for modeling user behavior as proposed by Whittaker [128] involves the observed relative activation frequency of each state $s \in S$ as recorded in system logs or other data. See the login, role_view, logout counts as shown in Figure 6.2 for an example of how this frequency may be recorded and stored. In contrast, a Markov approach to behavior modeling is based on the observed relative activation frequency of each transition between states, as shown in the `event_chain` attribute in Figure 6.2. We briefly outline the critical points; a more thorough description on modeling from log data can be found in the work of Menasce [85]. Karlin and Taylor define a Markov Process as:

Table 6.1. Observed State Transitions for Example User Behavior Model (©2020 IEEE)

	σ_1	σ_2	σ_3
σ_1	320	240	240
σ_2	18	24	18
σ_3	300	300	400

> "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 additional knowledge concerning its past behavior." [72]

Karlin and Taylor describe a Markov Process as being a MC if it is composed of a distinct set of states which are countable, and finite [72]. We can visualize the MC as the directed graph in Figure 3.6, where nodes of the graph are states of the chain, and edges represent the transition between states. 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.

In Table 6.1, we show a sample user's behavioral statistics with observed state transitions between $\sigma_i \longrightarrow \sigma_j$. By dividing each row vector by the row sum, we produce the MC-based user behavior model shown in Table 3.1, which can be visualized as the graph in Figure 3.5. Similarly, the aggregation of user sessions can be illustrated as the directed graph in Figure 3.6 and condensed to the MC shown in Figure 3.8. An example of how these data can be stored as shown in Figure 6.3. As is typical in modern web-based software systems, if the size of S is large it is more computationally and spatially efficient to represent the counts and frequencies of extracted behavior models as sparse vectors, as defined by Tinney et al. [118]. Sparse vectors can be structured from session data as represented by the `transition_counts`, `user_behavior`, and `clustered_behavior_profile` attributes in Figure 6.3 and Figure 6.4. By assuming

```
{
  "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
  }
}
```

Figure 6.3. Aggregated User Behavior Model (©2020 IEEE)

```
{
  "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
}
```

Figure 6.4. Clustered Profile Behavior Model (©2020 IEEE)

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 Table 6.2. Note that the $\$$ symbol in the matrix represents the final accepting state of the MC, such as logout; moreover, 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 [85], 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. The proficiency metric can also inform the hazard function process for modeling transition wait times as in [71].

6.4.3. Modeling Behavior Profiles

Generating behavior profiles from users relies on machine learning algorithms such as clustering [126]. LODESTONE executes the DBSCAN [14] clustering algorithm on aggregated user profiles, as illustrated by `user_behavior_profile` attribute in Figure 6.3.

Table 6.2. Sample Learned Markov4JMeter Model (©2020 IEEE)

	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

In Table 5.1, we see four example profiles which represent four different users having similar behavioral tendencies. The similarities can be illustrated with the heatmap in Figure 3.10. The members of this cluster do not have identical behavior patterns, but enough similarity exists such that a replication of their individual behavior patterns in study or in simulation is not needed. 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 6.4. After the learning process is completed, the Euclidean centroid of the set of user profiles is calculated to become the "representative" matrix or representative profile of the cluster. Various distance metrics can be used depending on how well the distance metrics subdivide the data; however, in instances with large sets of possible events, the curse of dimensionality can threaten the completion of this process. 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 [126]. 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[58] 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.

6.5. LODESTONE Architecture

We present the logical and physical architecture of LODESTONE.

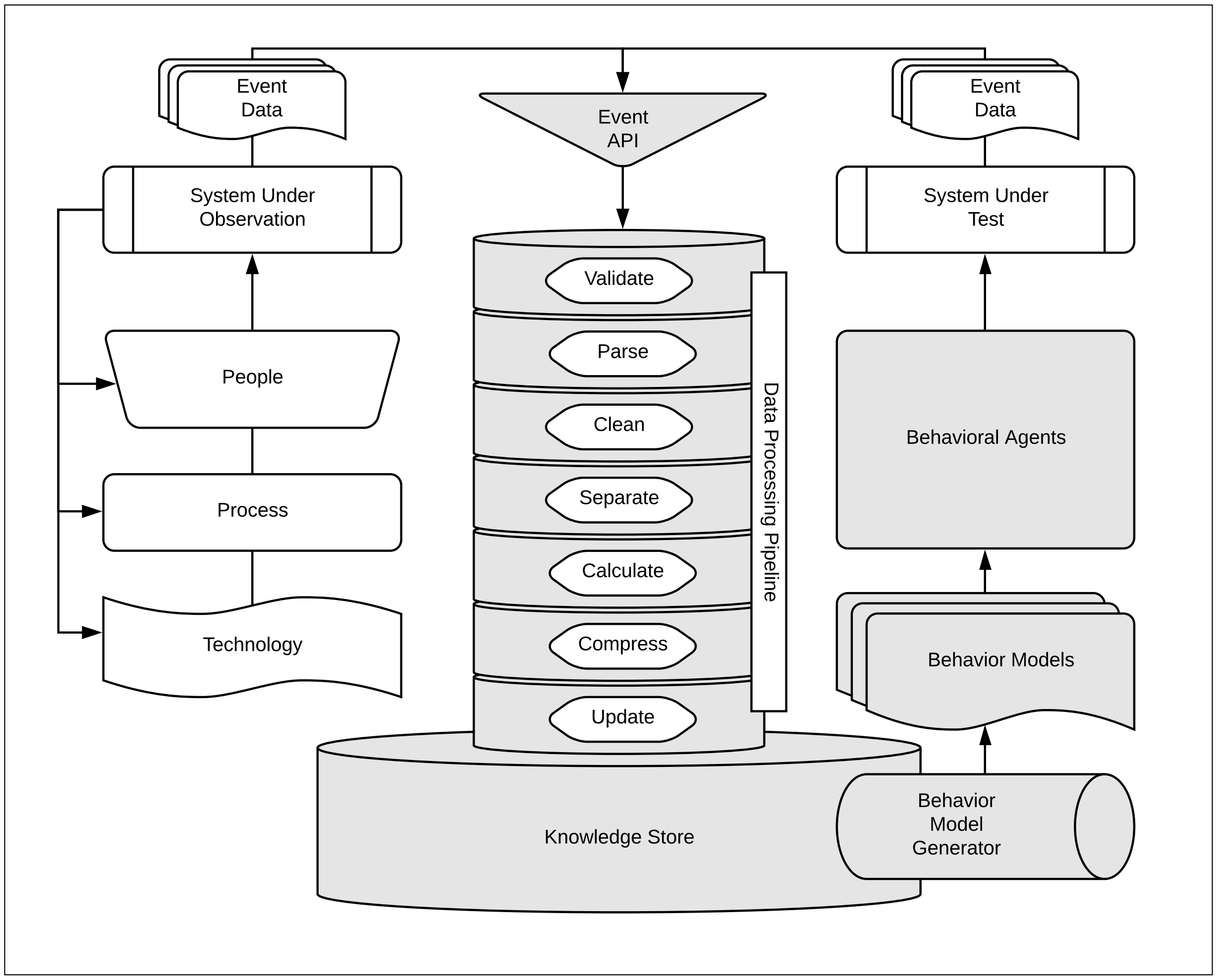


Figure 6.5. LODESTONE Logical Architecture (©2020 IEEE)

6.5.1. Logical Architecture

The logical architecture of LODESTONE shown in Figure 6.5 is composed of data ingestion, processing, storage, modeling, simulation, and training. Generally, the SUO will closely replicate the SUT to the point that scenarios, events, and processes will generally not deviate drastically between the two systems. As human and machine agents interact within the SUO, resultant event data are then passed to the LODESTONE REST or TCP-based Event API, which must be fault tolerant enough to scale with the workload of the SUO, the SUT, and the variations of good and bad data which may be sent through the API.

For storage, modeling, and value to be captured from ingested event data, LODESTONE must provide a consistent processing pipeline as shown in Figure 6.5. This pipeline must be capable of adjusting to the variations in data that can arise as a result of errant processes, failed connections, as well as functional and non-functional bugs in the SUO. The validation, parsing, and cleaning steps accept various types of data which can be produced by the SUO, and converts them into a consistent event format. The event is then separated if the inbound data are aggregated, frequency statistics as described in Section 6.4.3 are calculated or updated, and the data are compressed for updating existing knowledge stores. Like the rest of LODESTONE, the knowledge store must be capable of scaling with the size of relevant behavioral information required for simulation and testing of the SUT. Behavior modeling in LODESTONE consists of a number of statistics calculated to form a MC which describes the first order approximation of behavior patterns characterizing individual agents, groups of agents, as well as the whole population of agents.

Think time metrics as described in Section 6.4.2 are then calculated for the amount of time spent transitioning between states as a model for agents waiting to perform the necessary action. We then calculate, update, and store user behavior models as an initial step. Additional work can be then done to define and describe profile behavior models as from the user behavior models. Simulating realistic load in the SUT consists of running a scalable set of agents following the behavior models previously modeled. Outputs from the simulation are then fed back into LODESTONE as a means of recording discovered errors with the requisite replication scenarios. With enough agents simulating realistic behaviors, a realistic load is produced upon the SUT in order to allow functional and non-functional performance issues to be discovered. Additional data from the SUT into LODESTONE produce feedback from the behavior models; however, it provides an additional means for ensuring the performance quality characteristics are better measured and tested. Namely, the feedback loop can be used to strengthen the behaviors of the agents to test those areas of the SUT which might not necessarily be receiving enough attention; moreover, to evolve the testing patterns associated with the agents.

The learning capability of LODESTONE is based on the ability of the underlying MC or behavior models to be updated as additional information is received either from the SUO, or the SUT and appropriately shuttled through the Data Processing Pipeline. It is not merely sufficient to serialize and store individual users, actions, transitions, counts, and frequencies in a series of database tables; rather, such database objects must also be kept up-to-date through a streaming calculation of the necessary statistics. The models themselves are kept in the principal knowledge store; but the raw data are aggregated in memory to prevent an overwhelming amount of repetitive bits being stored and queried as the amount of inbound data increases with the observed and expected system load. By maintaining the models instead of the data, we enforce the ability of LODESTONE to scale in response to the data demands; in addition, the ability to adapt to changing application states allows for a testing system to remain online without needing to update the underlying configuration or modeling parameters, unlike other commercial and open source systems.

6.5.2. Physical Architecture

In Figure 6.6, 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`; in addition, they contain actions such as `add`, `modify`, `view`[2]. As the microservice SUO is used, the event data are collected into (2) a triggered Data Processor Lambda which cleans, processes, and writes to (3) DynamoDB, the Knowledge Store - capturing live and completed session information represented as in Figure 6.2. 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 6.3 within the Behavioral Model Cache which is implemented also in DynamoDB. As these user models are updated, the model builder runs

2 See Chapter 3 and Appendix A for the basic structure of the SUO and SUT.

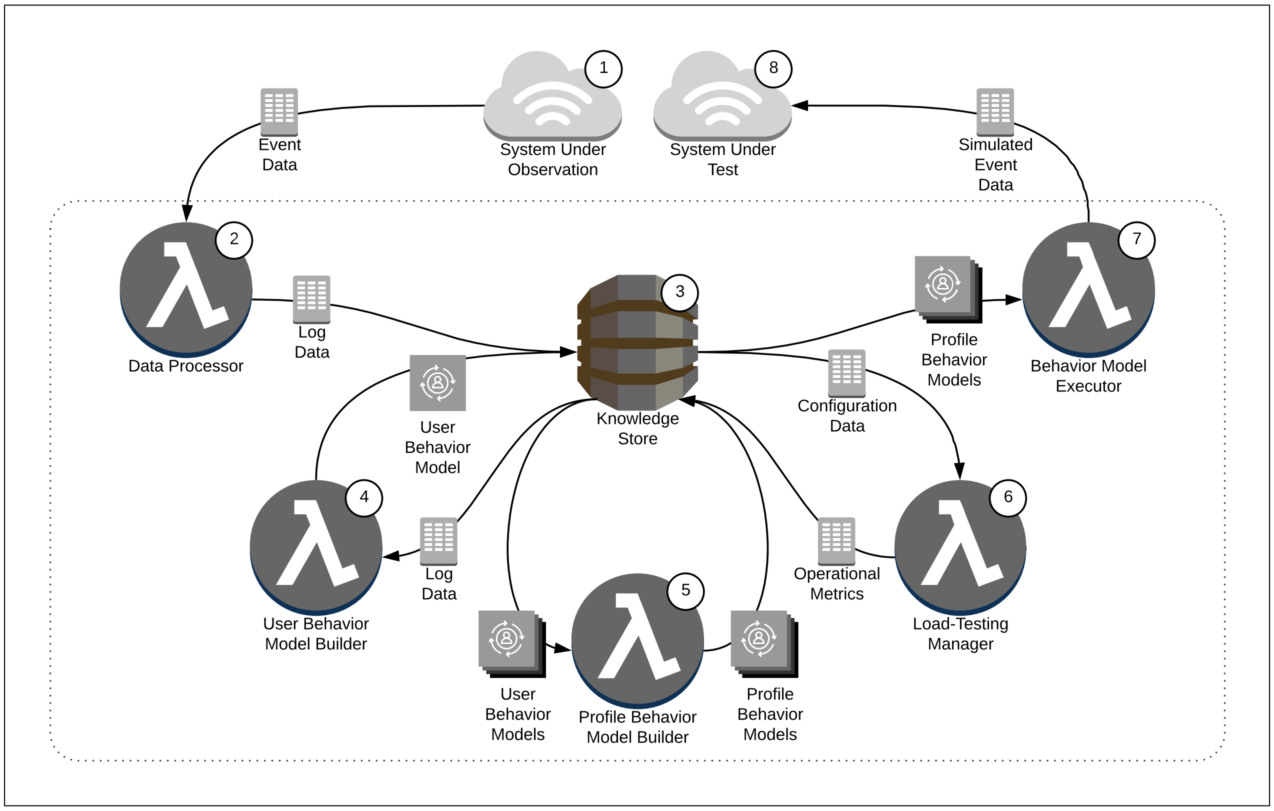


Figure 6.6. LODESTONE Physical Architecture and Data Flow (©2020 IEEE)

another processing step (5), the Profile Behavior Model Builder, to cluster the user models into profile models as represented in Figure 6.4. 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) will scale out the requisite number of Lambda operations when triggered; these Lambda instances represent 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 the cloud ecosystem being used. Through the analysis and log-metrics platforms provided by AWS, 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.

6.6. Evaluation

We defined a methodology used for evaluating LT systems through their measurable run-time performance, satisfying Assertion A_1, and selected desirable key features satisfying Assertion A_2. We show the results on LODESTONE and Markov4JMeter as well as provide discussion within the boundaries of our methodology.

6.6.1. 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. Our qualitative methodology reflects the properties P_{1-5} listed in Chapter 1.

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. This is related to Problem P_5 listed in Chapter 1.

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. This is related to Problems P_2, P_3 and P_4 in Chapter 1.

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. This is related to Problems P_3 and P_4 in Chapter 1.

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. This is related to Problem P_3 in Chapter 1.

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. This is related to Problems P_1 and P_3 in Chapter 1.

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. This is related to Problems P_1 and P_5 in Chapter 1.

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. This is related to Problems P_1, P_3, and P_5 in Chapter 1.

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. This is related to Problems P_2 and P_4 in Chapter 1.

6.6.2. Quantitative Methodology:

We describe the process 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.

For creation of the models, we generated a baseline of rule-based simulated behavioral data using the TinyERP system as defined in Chapter 3 and Appendix A. With the rule-based approach that TinyERP uses to define user behavior, we defined behavioral patterns similar to those exhibited in the anonymous datasets analyzed in Chapter 3 and matched those with the states of the

Table 6.3. Feature Matrix of Select Load Testing Tools (©2020 IEEE)

	JMeter[117]	Markov4JMeter[58, 126]	LODESTONE
Behavioral	✓	✓	✓
Modeled	✗	✓	✓
Distributed	✗	✗	✓
Compressed	✗	✗	✓
Streaming	✗	✗	✓
Scalable	✗	✗	✓
Cloud-based	✗	✗	✓
Time-variant	✗	✗	✓

TinyERP system. For more information, recall Figure 3.18 as an illustration of the rules generating the baseline of sample traffic. The TinyERP LODESTONE implementation was then directed toward the AWS implementation of LODESTONE for live data capture. The AWS implementation of LODESTONE captured the behavioral data and performed modeling internally.

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 such as 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.

6.6.3. Results and Discussion

We compare the features between Markov4JMeter and LODESTONE, as shown in Table 6.3. JMeter [117] 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

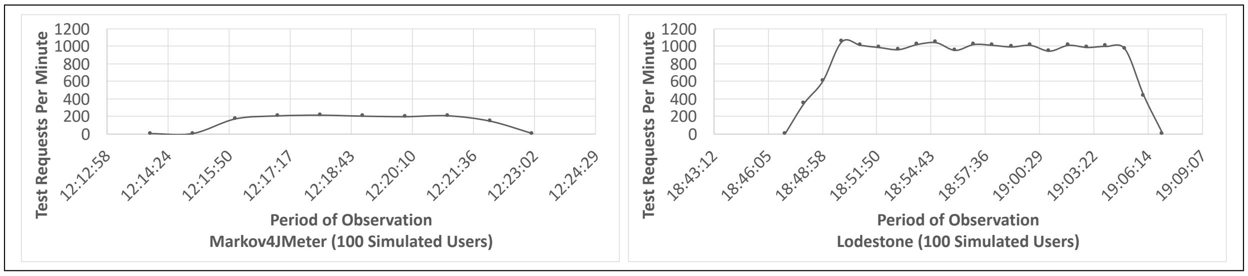


Figure 6.7. Test Per Minute Volume (UV=100) (©2020 IEEE)

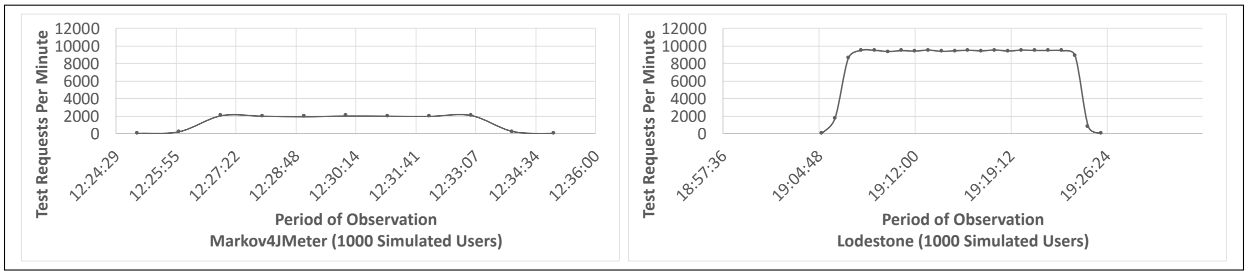


Figure 6.8. Test Per Minute Volume (UV=1000) (©2020 IEEE)

if explicitly configured from static artifacts. Markov4JMeter extends JMeter with the capability to configure user-behavior Markov models within the LT system [58]. 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 [126], 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 process is both distributed and scalable. In addition, LODESTONE provides analysis of think-time per user as well as per profile, where Markov4JMeter's documentation [58] shows a standard Gaussian think-time per behavior test iteration.

For the first set of tests, we used UV=100 users on both LT engines. In Figure 6.7a, we observed the SUT warming up to around 200 SRPM while running Markov4JMeter with 100 users; by comparison, in Figure 6.7b, we observed the SUT warming up to 1000 SRPM while running LODESTONE with 100 simulated users. For the second set of tests, we used UV=1000 users on both LT engines observing 2000 SRPM in Figure 6.8a for Markov4JMeter; for LODESTONE, we observed around 9500 SRPM in Figure 6.8b. 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 cause an operational limitation or non-functional fault that Markov4JMeter would not capable of detecting unless on a testing machine having CPU and RAM beyond the capabilities of the testing machine that was used. Any additionally perceived slowdown in Markov4JMeter is a well known limitation of the product [15] due to the JMeter product being primarily single-client based. While the Markov4JMeter product could achieve better performance than displayed here through iterative tuning of Java parameters [15], the underlying limitations of JMeter being single-client based is insurmountable compared to a cloud-based process for LT. 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 we considered were 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 and environmental factors to clear. Throughout the multiple iterations of the experiment, the SRPM of LODESTONE remained an order of magnitude higher than the SRPM of Markov4JMeter.

Our results show that our cloud-based implementation of LODESTONE can extend the features of an open-source tool for MC-based LT such as JMeter by exhibiting all of the qualitative parameters in Section 6.4.1 where JMeter only exhibits the Behavioral parameter, per our Assertion A_1. In addition, our quantitative analysis shows that LODESTONE consistently quantitatively outperforms JMeter by an order of magnitude, per Assertion A_2.

6.7. Summary

We have described the processes and architecture used to implement LODESTONE- a real-time process to ingest event 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 process 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 process 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.

The research described in this chapter demonstrates that a data-driven process for MC model-based load testing can be developed that offers advantages over existing data-driven processes per RQ_1 from Chapter 1; moreover, raw streaming behavioral data in a human-machine system can be continuously and efficiently measured, modeled, and stored as MC models per RQ_2 from Chapter 1. The specific advantages that LODESTONE provides are verified with our quantitative and qualitative comparison of a cloud-based implementation with a ubiquitous load testing tool JMeter - extended with the Markov4JMeter plugin.

Chapter 7. Conclusion

If a conclusion is not poetically balanced, it cannot be scientifically true.
-- Isaac Asimov
The Robots of Dawn

7.1. Summary

Modern systems must be able to withstand sudden catastrophic changes in their environment and usage; however, the rapidly changing set of tools and processes associated with software and systems development continuously solves technological difficulties at the inevitable cost of adding an untold number of additional challenges. The challenge that this research stands to solve is the process of generating realistic workload on a software system when such systems regularly change. The traditional approach to load testing includes hand-crafted scripts, web traffic recordings of test users, and the rote execution of tools and processes which are replayed by quality assurance professionals against a system to be tested. Such artifacts must necessarily co-evolve with the system being tested or risk becoming stale and uninformative or potentially destructive. The manual update process for these testing artifacts is time-consuming, error prone, expensive, and can undermine the efforts of technologists to adequately evaluate changes to systems before those changes are implemented. As our reliance upon technology increases, so too does its importance in our society due to the widespread distribution and intensity of demand. As such, it is not expedient to decry regular system evaluation due to the perceived urgency of desired change or an expected cost to evaluate a system.

This research details LODESTONE: a novel process to distributed load testing by modeling and simulating user behavior. We specify LODESTONE within the context of a human-machine system in order to illustrate the necessity for including distributed adaptation and execution in existing load testing processes. This research contributes and stands on a bottom-up process to understanding how users interact with systems by an examination of several anonymized or aggregated user behavior datasets. This research also presents a formal system of modeling individual users or profiling populations of users in addition to algorithms and data structures for improving the computational and storage efficiency required to mine user behavior datasets. Likewise this research contributes a fluent software application programming interface for quickly developing user behavior script templates, a software component for constant-time amortized calculation of streaming summary statistics, and a test oracle paired with a microservice which is developed on an industry-standard framework for the purpose of evaluating load testing processes. We further extend this research with machine learning for profiling user behaviors from atomic system logs, a heuristic algorithm for profiling semantic user behaviors from aggregated logs, and an adaptation of reinforcement learning to user behavior profiles so that they co-evolve with a system under test.

The research details how the microservice and test oracle can be effectively replicated within a cloud-based environment on Amazon Web Services. With the cloud implementation we contribute a qualitative and quantitative methodology for comparing load testing processes. We empirically show how even using an entirely different technology stack the lodestone process can outperform industry-standard load-testing processes and tools. This research rises to the challenge of reducing the cost and time to evaluate rapidly changing systems with LODESTONE, which continuously adapts to changes in the system to be tested which allows for load testing to be integrated into the quality assurance process for cloud-based microservices.

7.2. Research Questions

With this research, we have addressed the problem that data-driven processes to realistic model-based load testing should continuously and automatically adapt to behavioral and infrastructural changes in a system to be tested. We have demonstrated that LODESTONE provides features which outperform and outmode industry-grade tools by extending the current research in load testing. The contribution of this research subsumes the following research questions:

RQ_1: Can a data-driven process to MC model-based load testing be developed that offers advantages over existing processes?

We have presented LODESTONE as a data-driven MC model-based process for load testing. In Chapter 3, we described how data can be analyzed and simulated through the TinyERP implementation of LODESTONE. In Chapter 6, we showed how a cloud-based implementation of LODESTONE has both qualitative and quantitative advantages over an open-source load testing tool through load-testing experiments and a comparison of features. In Table 7.1, we showed how Lodestone compares against current research efforts in load testing through several parameters (P_{1-5}) defined in Chapter 1. Through the diagrams and analyses presented, LODESTONE thus demonstrates that data-driven process to MC model-based load testing can be developed that offers advantages over existing processes by merging formalisms, data structures, models and features which do not exist together in current LT processes in order to outperform current LT processes.

RQ_2: Can raw streaming behavioral data in a human-machine system be continuously and efficiently measured, modeled, and stored as MC models?

Load testing tools such as JMeter as well as the research listed in Table 7.1 relies upon data or models which are statically engineered and maintained by QA. By comparison, LODESTONE can accept raw streaming behavioral data from a human-machine system, efficiently measure, model, and store those data as MC models. We demonstrate this capability with our experimentation in Chapter 6, and outline the necessary improvements to a classic data extraction, transfer, load process required by such tools as JMeter in Chapter 5. With DBSCAN clustering (described in Chapter 3), we can extract Profile Behavior Bases in lieu of User Behavior Bases from these raw data - other tools such as JMeter rely upon the raw data which can be orders of magnitude more substantial in size. Where Markov4JMeter can extend the capabilities of JMeter by using clustered models such as our PBBs, it is still not capable of streaming those data directly from the SUO or learning from those data in a streaming fashion. LODESTONE thus demonstrates how raw streaming behavioral data in a human-machine system can be continuously and efficiently measured, modeled, and stored as MC models.

RQ_3: Can aggregated batched behavioral data in a human-machine system be systematically grouped into semantically related MC models?

One characteristic of anonymized data that the research listed in Table 7.1 and tools such as JMeter with Markov4JMeter do not address is when such data are aggregated beyond the point of deciphering the actions and intents of individual users. As we illustrated in Chapter 3, all such state-transitional data can be represented as a graph. With Chapter 4, we showed how such a graph can be segmented with an extension to Tarjan's STRONGCONNECT algorithm that reduces the depth the graph is traversed for new neighbor nodes removing the false trails issued by web crawlers and search engine indexing software. By segmenting the aggregated behavioral data of Wikipedia users with the Depth Constrained STRONGCONNECT algorithm, we showed that aggregated batched behavioral data in a human-machine system can be systematically grouped into semantically related MC models.

RQ_4: Can MC models of a human-machine system be extended to adapt to changes in a system under observation?

The speed at which software changes and the software development lifecycle evolves has become a liability for software system testers who rely on manual processes and static artifacts for testing. Such static artifacts introduce additional security risk if they contain sensitive data, and the amount of time associated with updating them means that the cost to continuously evaluate software for quality increases to the point where some levels of testing are no longer readily applied to new releases. Teams using processes such as continuous integration may release software several times per hour, depending on the processes enacted. It is imperative for testing tools to be able to adapt to changes in the software to be tested and observed. With Chapter 5, we showed how the combination of Markov Chains with Q-Learning and Laplace smoothing allows for new behaviors to occur which are not in the observed data. In addition, the least-recently-used cache in Chapter 5 shows how outdated transitions which were previously learned can be pruned from the testing models and LODESTONE can adapt to the SUO in that way. Through the various improvements in Chapter 5, we showed that MC models of a human-machine system can be extended to adapt to changes in a system under observation.

Table 7.1. Qualitative Comparison of LODESTONE with Existing Load Testing Processes

Author	Adaptive	Realistic	Efficient	Multiple Agents	Operationally Based
BARROS ET AL. [9]		✓			✓
CANFORA ET AL. [23]	✓				
COTRENEO ET AL. [35]		✓			✓
GU & GE [53]	✓				
KANT ET AL. [70]		✓			✓
MENASCE [84]		✓			✓
PENTA ET AL. [37]	✓				
SCHUR ET AL. [105]		✓			✓
VAN HOORN [58]		✓			✓
VOGELE ET AL. [126]		✓			✓
ZHANG ET AL. [143]	✓				
LODESTONE	✓	✓	✓	✓	✓

We have shown answers to each of the research questions listed in Chapter 1. The LODESTONE process in this research contains the processes, data structures, algorithms, architectural designs, and implementation results to support an affirmative conclusion to each of the research questions in Chapter 1. The primary contribution of this research can be subdivided into several aspects.

7.3. Contribution and Significance

The expense and difficulty associated with classical load testing methods have become outmoded by the ubiquity and speed with which software systems are developed and deployed. At its core, this research contributes LODESTONE: a novel process for testing systems based on the data generated through interactions between users and those systems that exhibits the properties P_{1-5} from Section 1.4 by: a) modeling and classifying user behavior from streaming and aggregated log data, b) adapting to changes in system and user behavior, c) generating distributed workload by realistically simulating user behavior. Toward the goal of creating and evaluating LODESTONE, several directed aspects of the research are required for the process to be instantiated, completed, and evaluated; these aspects are as follows:

- A set of terminology and formalisms defining the parameters, data, and models to be amalgamated by LODESTONE and informed by an ad-hoc analysis of several anonymous public user behavioral datasets.

- A data ingestion process using the DBSCAN clustering algorithm to detect and model similar users by their behavior patterns.
- A functional microservice (TinyERP) and test oracle (Loki) written in a production-grade software framework (Spring Boot) for realistic evaluation of load testing processes, continuous generation of performance and behavioral data, generation of rule-based behavioral traffic, and controlled simulation of errors in the system.
- A collection of streaming statistics algorithms which provide dataset statistics in O(1) space and O(1) time.
- A Depth Constraint extension of Tarjan's STRONGCONNECT algorithm for efficient segmentation of aggregated log data into smaller subgraphs of related user behavior and executed on aggregated Wikipedia data.
- An extension of the formalisms of Chapters 2 and 3 through the Q-Learning reinforcement machine learning algorithm, supported by a proof of optimality.
- An extension of SparseVector and SparseMatrix data structures to address the "sunrise problem" of events which have never been observed but should have a non-zero probability of being simulated through Laplace smoothing.
- Addition of a least-recently-used event cache to allow for efficient reduction of inactive, invalid, or unlikely historical event transitions.
- Provision for the Kullback-Leibler Divergence for determining if clustered models provide the same level of informational representation as the raw data used by classical load testing processes.
- Requirement of a system to monitor and learn from in addition to the system to be tested; such a requirement, not appearing elsewhere in the literature, enables an implementation of LODESTONE to accept streamed data from a production environment for continuous adaptive testing of a system under test.
- The LODESTONE process as an implementation in Amazon Web Services with a logical architecture and a physical architecture.

- Qualitative and quantitative comparison against a ubiquitous load testing tool JMeter using identical models learned from rule-based behavioral data generated by TinyERP.
- A distributed load testing process which is adaptive, realistic, efficient, operationally based, and supports multiple-agent execution.

These aspects of the LODESTONE process delineate how raw system data may be collected in their various forms to efficiently learn and simulate interactions between the humans and machines in a real human-machine-system. LODESTONE may be used toward solving problems in fields such as cybersecurity, emergency response, product management, and user experience. However, with this research we tender theoretical concepts of and concrete evidence concerning the understanding of user behavior data to solve load testing challenges in software engineering.

A dynamic process to addressing stale, missing, and incomplete data must extend descriptive MC models with formalisms, efficient algorithms, and inferential methods (such as those given by machine learning). We showed processes to extracting and modeling user behavior within the context of a human-machine system in order to illustrate qualitative and quantitative advantages over existing load testing methodologies.

7.4. Future Work

Future work to improve LODESTONE can focus on various facets of the process. Of particular note is the future role that testing oracles can play in LT of continuous integration environments, and how testing oracles can be improved for continuous integration environments. Cleaning and parsing data from the SUO is one of the major preliminary steps given in Figure 1.1; of particular importance is how machine learning might be used such that raw event data can be automatically parsed and cleaned into a usable form without needing additional interaction from system engineers. As the maturity of the process improves, error data from the SUT could be used even further for operational modeling for future development of system enhancements as well as for testing purposes. As we have discussed behavior modeling of users in depth, we also would like to further understand how DMABs could be used for adding business value, operational value, and intelligence purposes in software development organizations.

Appendix A. TinyERP Table Data Definitions

The TinyERP microservice consists of various modules, each of which support multiple actions, depending on the authorization permissions granted to the user calling the service. Below is the data definition language defining the various modules required to operate TinyERP. Each "CREATE" statement represents the initialization of the in-memory database used for managing online transaction processing for user behavior. The entity tables are ROLE, USER, ACCOUNT, INVOICE, ORDERS, and WIDGET. The join tables are USER_ROLE and WIDGET_ORDER.

```
create table ROLE
(
        ID BIGINT default auto_increment
                primary key,
        DESCRIPTION VARCHAR(255),
        MAXIMUM_NUMBER_OF_USERS BIGINT,
        MINIMUM_NUMBER_OF_USERS INTEGER,
        MODULE INTEGER,
        NAME VARCHAR(255),
        PERCENT DOUBLE,
        USER_TYPE INTEGER
);
```

Figure A.1. Role Table DDL for TinyERP Application

```
create table USER
(
        ID BIGINT default auto_increment
                primary key,
        FIRST_NAME VARCHAR(255),
        LAST_NAME VARCHAR(255)
);
```

Figure A.2. User Table DDL for TinyERP Application

```
create table ACCOUNT
(
        ID BIGINT default auto_increment
                primary key,
        NAME VARCHAR(255),
        OWNER_USER_ID BIGINT,
        constraint FKB4JVTN40PEM0OTJ1GA6R7T852
                foreign key (OWNER_USER_ID)
                ref. USER
);
```

Figure A.3. Account Table DDL for TinyERP Application

```
create table INVOICE
(
        ID BIGINT default auto_increment
                primary key,
        END_TIME TIME,
        START_TIME TIME,
        ACCOUNT_ID BIGINT,
        constraint FKOEVV8H8T2QGYM9S0CN7OH069B
                foreign key (ACCOUNT_ID)
                ref. ACCOUNT
);
```

Figure A.4. Invoice Table DDL for TinyERP Application

```
create table ORDERS
(
        ID BIGINT default auto_increment
                primary key,
        AMOUNT DOUBLE,
        ORDER_TIME TIME,
        ACCOUNT_ID BIGINT,
        INVOICE_ID BIGINT,
        constraint FK3C7GBSFAWN58R27CF5B2KM72F
                foreign key (ACCOUNT_ID)
                ref. ACCOUNT,
        constraint FKJA77CITBXEILGQCHN5VVBI55J
                foreign key (INVOICE_ID)
                ref. INVOICE
);
```

Figure A.5. Orders Table DDL for TinyERP Application

```
create table USER_ROLE
(
        USER_ID BIGINT not null,
        ROLE_ID BIGINT not null,
        primary key (USER_ID, ROLE_ID),
        constraint FK859N2JVI8IVHUI0RL0ESWS6O
                foreign key (USER_ID)
                ref. USER,
        constraint FKA68196081FVOVJHKEK5M97N3Y
                foreign key (ROLE_ID)
                ref. ROLE
);
```

Figure A.6. User_Role Table DDL for TinyERP Application

```
create table WIDGET
(
        ID BIGINT default auto_increment
                primary key,
        DESCRIPTION VARCHAR(255),
        NAME VARCHAR(255),
        PRICE DOUBLE
);
```

Figure A.7. User_Role Table DDL for TinyERP Application

```
create table WIDGET_ORDER
(
        WIDGET_ID BIGINT not null,
        ORDER_ID BIGINT not null,
        primary key (WIDGET_ID, ORDER_ID),
        constraint FK4D3877UHT4K9QX2CQ9K6GQQJU
                foreign key (WIDGET_ID)
                ref. WIDGET,
        constraint FKCWQDL0NQGIJU04D5FLVDY5C50
                foreign key (ORDER_ID)
                ref. ORDERS
);
```

Figure A.8. Widget_Order Table DDL for TinyERP Application

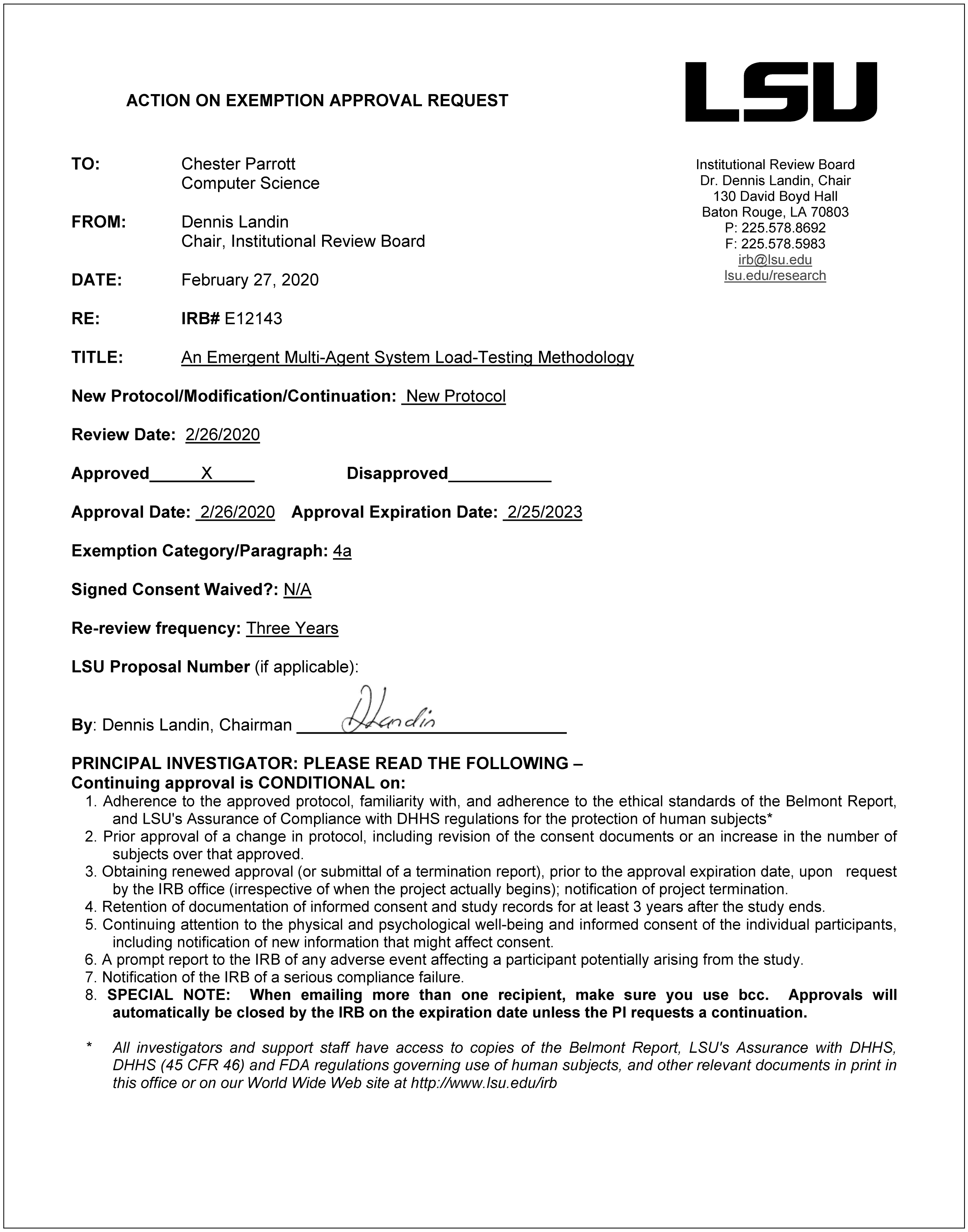
Appendix B. TinyStats Definition

We refer the reader to the code at

`https://github.com/parrottsquawk/TinyStats/blob/master/TinyStats.java` for a more mathematically readable version of this class.

```
        public class TinyStats implements StatisticalSummary {
            private double n, n-1, n-2, n-3; // number of items,
                // decremented versions to save processing time.
            private double sum; // Sum
            private double mu_1, mu_2, mu_3, mu_4;
            //first, second, third, and fourth moments.
            private double sumxlogx; // sum of x* log_2(x)
            private double max = Double.MIN_VALUE,
                           min = Double.MAX_VALUE;

            public TinyStats(double[] data) {
                for (double d : data) {
                    this.put(d);
                }
            }

            public TinyStats() {
            }

            public void put(double x) {
                n-3 = n-2;
                n-2 = n-1;
                n-1 = n;
                n++;

                max = x > max ? x : max;
                min = x < min ? x : min;

                sum += x;
                sumxlogx += x * Math.log(x) / Math.log(2);

                double std_dev = x - mu_1;
                double std_dev_divided_by_n =
                    std_dev / n;
                double std_dev_squared_divided_by_n =
                    std_dev * std_dev_divided_by_n;
                double std_dev_cubed_divided_by_n_squared =
            std_dev_divided_by_n *
            std_dev_squared_divided_by_n;
        double t1 = std_dev_divided_by_n * 3 * mu_2;

        mu_4 +=
            (std_dev_divided_by_n *
            std_dev_cubed_divided_by_n_squared) *
            n-1 * (n * n-3 + 3) +
            std_dev_divided_by_n * t1 * 2 -
            std_dev_divided_by_n * 4 * mu_3;
        mu_3 += std_dev_cubed_divided_by_n_squared *
                n-2 * n-1 - t1;
        mu_2 += std_dev_squared_divided_by_n * n-1;
        mu_1 += std_dev_divided_by_n;

    }

    public double getMean() {
        return mu_1;
    }

    public double getSum() {
        return sum;
    }

    public double getVariance() {
        return n < 2 ? Double.NaN : mu_2 / n-1;
    }

    public double getStandardDeviation() {
        return Math.sqrt(getVariance());
    }

    @Override
    public double getMax() {
        return max;
    }

    @Override
    public double getMin() {
        return min;
    }

    @Override
    public long getN() {
        return (long) n;
    }

    public double getKurtosis() {
        return ((n * mu_4) / (mu_2 * mu_2));
    }

    public double getSkewness() {
        return (Math.sqrt(n) * mu_3) /
                Math.sqrt(mu_2 * mu_2 * mu_2);
    }

    public double getEntropy() {
        return (-sumxlogx * Math.log(n) /
                (n * Math.log(2)));
    }

    public double getExactHistogramBinSize() {
        return 3.49 * getStandardDeviation() *
                Math.pow(getN(), -(1.0 / 3.0));
    }
}
```

Appendix C. Institutional Review Board Exemption Approval



ACTION ON EXEMPTION APPROVAL REQUEST

LSU

Institutional Review Board
Dr. Dennis Landin, Chair
130 David Boyd Hall
Baton Rouge, LA 70803
P: 225.578.8692
F: 225.578.5983
irb@lsu.edu
lsu.edu/research

TO: Chester Parrott
Computer Science

FROM: Dennis Landin
Chair, Institutional Review Board

DATE: February 27, 2020

RE: **IRB#** E12143

TITLE: An Emergent Multi-Agent System Load-Testing Methodology

New Protocol/Modification/Continuation: New Protocol

Review Date: 2/26/2020

Approved X **Disapproved** ___________

Approval Date: 2/26/2020 **Approval Expiration Date:** 2/25/2023

Exemption Category/Paragraph: 4a

Signed Consent Waived?: N/A

Re-review frequency: Three Years

LSU Proposal Number (if applicable):

By: Dennis Landin, Chairman [illegible]

PRINCIPAL INVESTIGATOR: PLEASE READ THE FOLLOWING –
Continuing approval is CONDITIONAL on:

1. Adherence to the approved protocol, familiarity with, and adherence to the ethical standards of the Belmont Report, and LSU's Assurance of Compliance with DHHS regulations for the protection of human subjects*
2. Prior approval of a change in protocol, including revision of the consent documents or an increase in the number of subjects over that approved.
3. Obtaining renewed approval (or submittal of a termination report), prior to the approval expiration date, upon request by the IRB office (irrespective of when the project actually begins); notification of project termination.
4. Retention of documentation of informed consent and study records for at least 3 years after the study ends.
5. Continuing attention to the physical and psychological well-being and informed consent of the individual participants, including notification of new information that might affect consent.
6. A prompt report to the IRB of any adverse event affecting a participant potentially arising from the study.
7. Notification of the IRB of a serious compliance failure.
8. **SPECIAL NOTE: When emailing more than one recipient, make sure you use bcc. Approvals will automatically be closed by the IRB on the expiration date unless the PI requests a continuation.**

* *All investigators and support staff have access to copies of the Belmont Report, LSU's Assurance with DHHS, DHHS (45 CFR 46) and FDA regulations governing use of human subjects, and other relevant documents in print in this office or on our World Wide Web site at http://www.lsu.edu/irb*

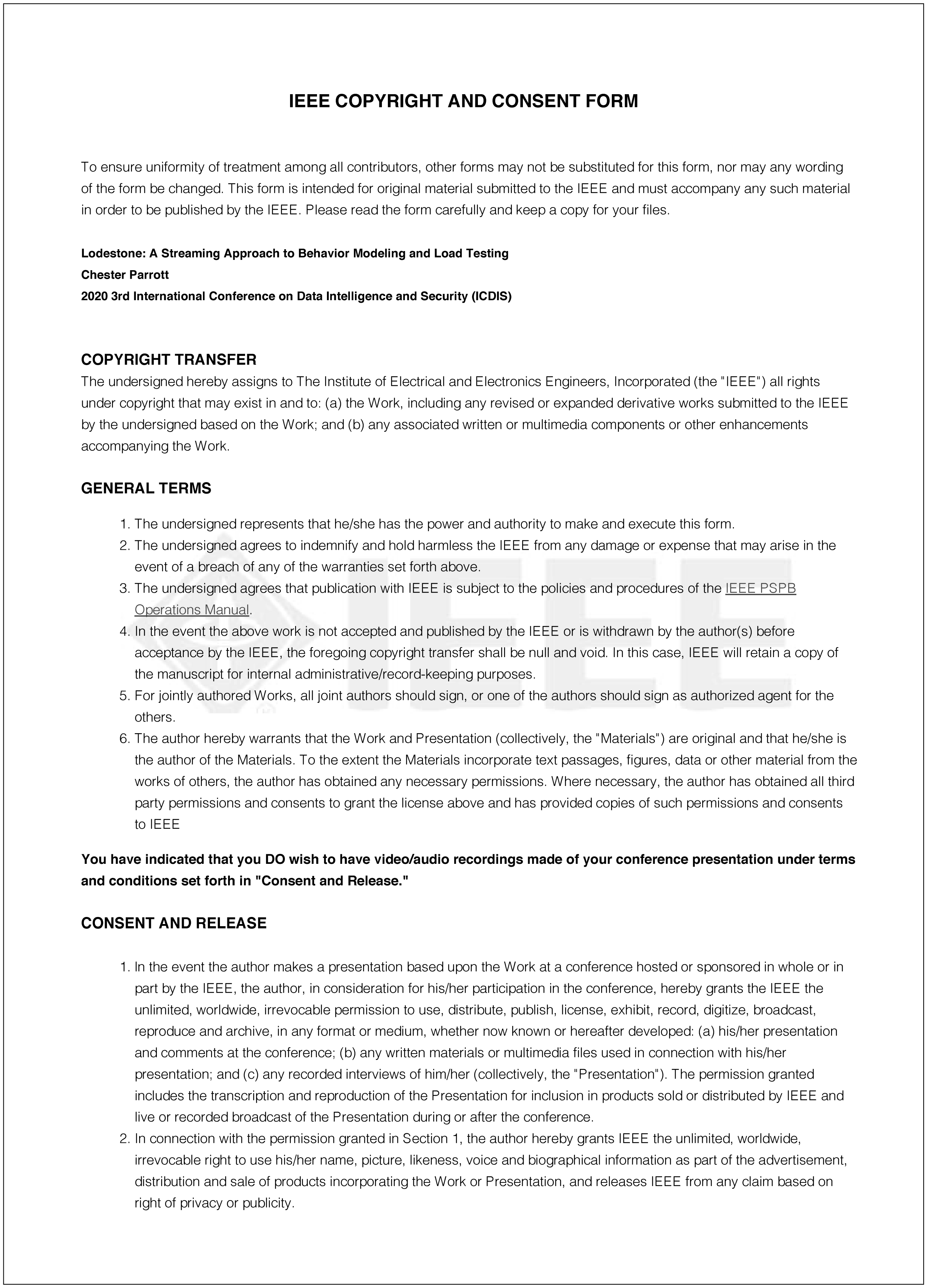
Appendix D. IEEE Copyright Information

D.5. IEEE Disclaimer

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Louisiana State University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to https://www.ieee.org/publications/rights/rights-link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

D.6. IEEE Copyright and Consent Form

Full content follows on the next page.



IEEE COPYRIGHT AND CONSENT FORM

Lodestone: A Streaming Approach to Behavior Modeling and Load Testing
Chester Parrott
2020 3rd International Conference on Data Intelligence and Security (ICDIS)

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

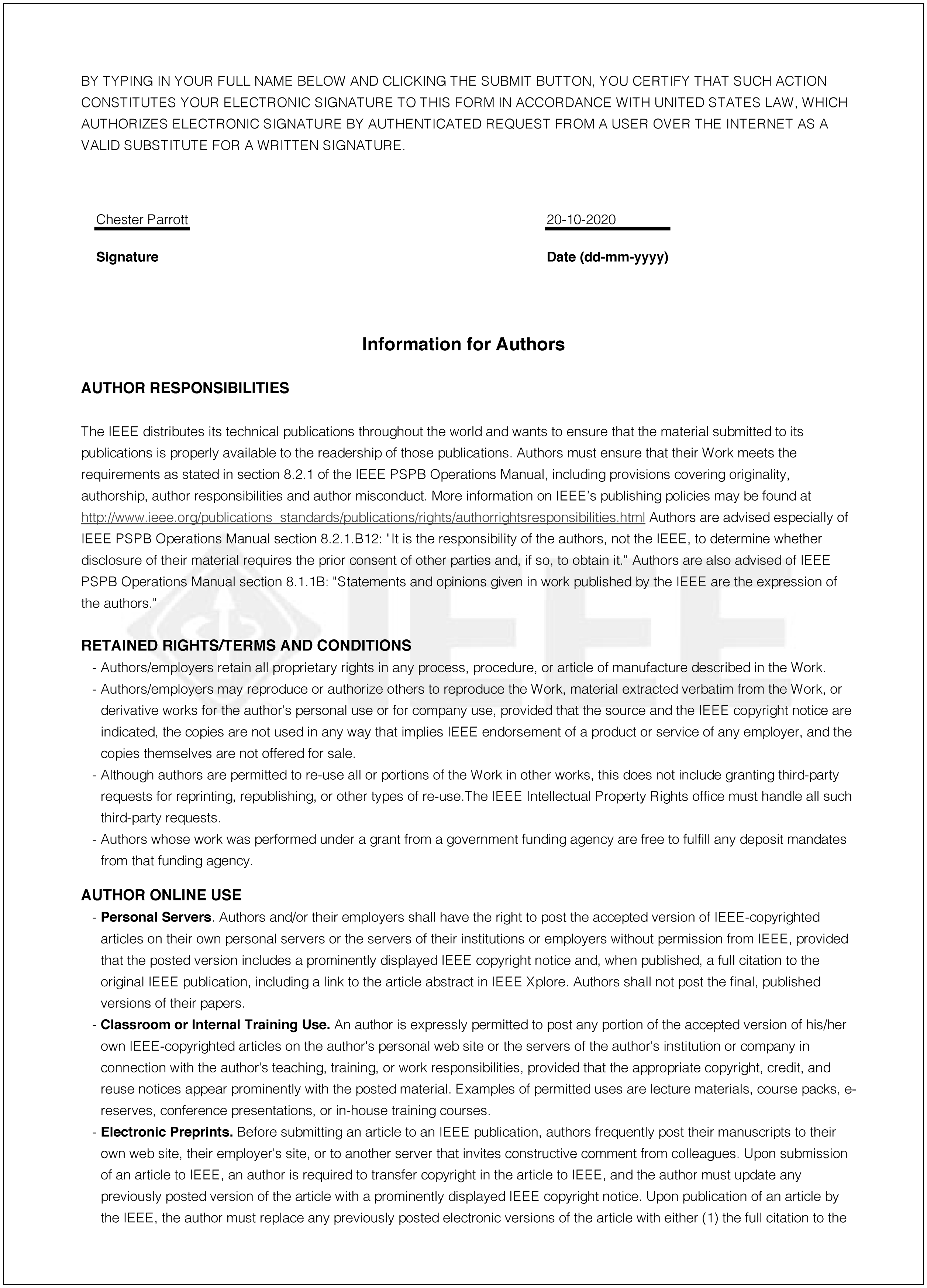
GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.



BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Chester Parrott	20-10-2020
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the

Appendix E. Further Reading

J. P. Achara, M. M. Maaz, W. Saab, R. Rudnik, and J.-Y. Le Boudec. *T-RECS: A Software Testbed for Multi-Agent Real-Time Control of Electric Grids.* Tech. rep. EPFL, 2017.

D. Ahlers, M. Mehrpoor, K. Kristensen, and J. Krogstie. "Challenges for information access in multi-disciplinary product design and engineering settings". In: *Digital Information Management (ICDIM), 2015 Tenth International Conference on.* Oct. 2015, pp. 109–114. DOI: 10.1109/ICDIM.2015.7381865.

B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane. "Graph-based malware detection using dynamic analysis". In: *Journal in Computer Virology* 7.4 (2011), pp. 247–258.

J. B. Ard, M. Bishop, C. Gates, and M. X. Sun. "Information behaving badly". In: *Proceedings of the 2013 workshop on New security paradigms workshop.* ACM. 2013, pp. 107–118.

M. M. Arif, W. Shang, and E. Shihab. "Empirical study on the discrepancy between performance testing results from virtual and physical environments". In: *the 40th International Conference.* New York, New York, USA: ACM Press, 2018, pp. 822–822.

Q. Bai, M. Zhang, and H. Zhang. "A colored Petri net based strategy for multi-agent scheduling". In: *Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems, 2005* (2005).

M. Ben Salem and S. J. Stolfo. "Modeling User Search Behavior for Masquerade Detection. " In: *RAID* 6961.Chapter 10 (2011), pp. 181–200.

T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext transfer protocol–HTTP/1.0.* Tech. rep. HTTP Working Group, 1996.

O. Brdiczka, J. Liu, B. Price, J. Shen, A. Patil, R. Chow, E. Bart, and N. Ducheneaut. "Proactive Insider Threat Detection through Graph Learning and Psychological Context". In: *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on.* May 2012, pp. 142–149. DOI: 10.1109/SPW.2012.29.

G. Cai, J. Gao, and Y. Huang. "Modeling electronic institutions with extended colored Petri net". In: *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on.* 2007.

D. Cappelli, A. Moore, and R. Trzeciak. *The CERT Guide to Insider Threats: How to Detect, Prevent, and Respond to Information Technology Crimes.* 2012.

X. Congqi, L. Tianmei, D. Runnan, and J. Guizhi. "Research on Energy-Hub Control Method of Micro-grid Based on Multi-agent & Petri Nets". In: *Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on.* 2013.

G. Creech and J. Hu. "Generation of a new IDS test dataset: Time to retire the KDD collection". In: *2013 IEEE Wireless Communications and Networking Conference (WCNC).* IEEE, 2013, pp. 4487–4492.

O. M. Dahl and S. D. Wolthusen. “Modeling and execution of complex attack scenarios using interval timed colored Petri nets”. In: *Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on* (2006).

V. Di Gesu, G. Lo Bosco, and J. H. Friedman. “Intruders pattern identification”. In: *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on.* IEEE, 2008, pp. 1–4.

T. F. Düllmann, R. Heinrich, A. van Hoorn, T. Pitakrat, J. Walter, and F. Willnecker. “CASPA - A Platform for Comparability of Architecture-Based Software Performance Engineering Approaches.” In: *ICSA Workshops* (2017).

Y. F. Eddy, H. B. Gooi, and S. X. Chen. “Multi-agent system for distributed management of microgrids”. In: *IEEE Transactions on power systems* 30.1 (2015), pp. 24–34.

E. Egho, C. Raïssi, T. Calders, N. Jay, and A. Napoli. “On measuring similarity for sequences of itemsets”. In: *Data Mining and Knowledge Discovery* 29.3 (2012), pp. 732–764.

P. Estraillier and F. Kordon. “Structuration of large scale Petri nets: an association with higher level formalisms for the design of multi-agent systems”. In: *Systems, Man, and Cybernetics, 1996., IEEE International Conference on.* 1996.

F. M. Facca and P. L. Lanzi. “Mining interesting knowledge from weblogs: a survey.” In: *Data Knowl. Eng. ()* 53.3 (2005), pp. 225–241.

G. Fan, H. Yu, L. Chen, and D. Liu. “A Game Theoretic Method to Model and Evaluate Attack-Defense Strategy in Cloud Computing”. In: *Services Computing (SCC), 2013 IEEE International Conference on.* 2013.

D. Ferreira and J. P. Ferreira. “A workflow management system for coordinating distributed information-based business processes”. In: *Agent-Based Systems in the Business Context. Papers from the AAAI Workshop.* Technical Report WS-99-02. AAAI Press. 1999.

D. R. Ferreira, S. Alves, and L. H. Thom. “Ontology-Based Discovery of Workflow Activity Patterns”. In: *Business Process Management Workshops.* pp.314 - 325. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

D. R. Ferreira, F. Szimanski, and C. G. Ralha. “Improving process models by mining mappings of low-level events to high-level activities.” In: *J. Intell. Inf. Syst. ()* 43.2 (2014), pp. 379–407.

H. Fiorino and C. Tessier. “Agent cooperation: a Petri net based model”. In: *Multi Agent Systems, 1998. Proceedings. International Conference on.* 1998.

M. Flores-Badillo and E. Lopez-Mellado. “Mobile agent based automation of distributed workflow processes”. In: *System of Systems Engineering, 2008. SoSE ’08. IEEE International Conference on.* 2008.

V. Frias-Martinez, J. Sherrick, S. J. Stolfo, and A. D. Keromytis. “A network access control mechanism based on behavior profiles”. In: *Computer Security Applications Conference, 2009. ACSAC’09. Annual.* IEEE. 2009, pp. 3–12.

M. P. Gallaher, A. C. O'Connor, and B. Kropp. "The economic impact of role-based access control". In: *Planning report* (2002), pp. 02–1.

J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. "A survey on concept drift adaptation". In: *Computing Surveys (CSUR* 46.4 (Apr. 2014).

D. Giraldo, A. Herrera, M. E. Sánchez, and J. Villalobos. "Analysis of ICT services by observing "fit for use" attributes." In: *CONF-IRM*. 2017.

J. Glasser and B. Lindauer. "Bridging the Gap: A Pragmatic Approach to Generating Insider Threat Data". In: *Security and Privacy Workshops (SPW), 2013 IEEE*. May 2013, pp. 98–104. DOI: 10.1109/SPW.2013.37.

M. Gupta, J. Gao, C. Aggarwal, and J. Han. *Outlier Detection for Temporal Data*. Vol. 5. Morgan & Claypool Publishers, Mar. 2014.

F. He, Q. Miao, Y. Li, F.-Y. Wang, and S. Tang. "Modeling and analysis of artificial transportation system based on multi-agent technology". In: *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*. 2006.

C. Herrero and J. Oliver. "Extended cooperating automata". In: *Systems, Man and Cybernetics, 2003. IEEE International Conference on*. 2003.

L. Hodge and M. Kamel. "An agent-based approach to multisensor coordination". In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 33.5 (2003), pp. 648–661.

D. of Homeland Security Science and T. Directorate. *A Roadmap for Cybersecurity Research*. Department of Homeland Security Science and Technology Directorate. http://www.cyber.st.dhs.gov/documents.html, Nov. 2009.

T. Hospedales, S. Gong, and T. Xiang. "A Markov Clustering Topic Model for mining behaviour in video". In: *IEEE International Conference on Computer Vision. Proceedings* (Jan. 2009), pp. 1165–1172.

F.-S. Hsieh. "Collaborative timed Petri net for holonic process planning". In: *American Control Conference, 2003. Proceedings of the 2003*. 2003.

F.-S. Hsieh and J.-B. Lin. "A multiagent approach for managing collaborative workflows in supply chains". In: *Computer Supported Cooperative Work in Design, Proceedings of the 2014 IEEE 18th International Conference on*. 2014.

X. Huang, J. Yong, J. Li, and J. Gao. "Prediction of student actions using weighted Markov models". In: *2008 IEEE International Symposium on IT in Medicine and Education (ITME)*. IEEE, 2008, pp. 154–159.

Z. Jian, H. Shirai, I. Takahashi, J. Kuroiwa, T. Odaka, and H. Ogura. "Masquerade detection by boosting decision stumps using UNIX commands". In: *Computers and Security* 26.4 (Jan. 2007), pp. 311–318.

P. Jun. “Application Research of the Game Theory in the Teaching of College Physical”. In: *Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on.* 2013.

F. Khalil, J. Li, and H. Wang. “Integrating recommendation models for improved web page prediction accuracy”. In: *ACSC ’08: Proceedings of the thirty-first Australasian conference on Computer science.* Australian Computer Society, Inc, Jan. 2008.

G.-W. Kim, S. H. Lee, J. H. Kim, and J. H. Son. “An Effective Algorithm for Business Process Mining Based on Modified FP-Tree Algorithm”. In: *Communication Software and Networks, 2010. ICCSN ’10. Second International Conference on.* Feb. 2010, pp. 119–123. DOI: 10.1109/ICCSN.2010.77.

H.-S. Kim and S.-D. Cha. “Empirical evaluation of SVM-based masquerade detection using UNIX commands”. In: *Computers and Security* 24.2 (Jan. 2005), pp. 160–168.

A. Kobsa. “Generic User Modeling Systems”. In: *User Modeling and User-Adapted Interaction* 11.1 (Mar. 2001), pp. 49–63.

C.-H. Kuo. “Development of distributed agent-oriented Petri net simulation and control environment for discrete event dynamic systems”. In: *2004 IEEE International Conference on Systems, Man and Cybernetics* 5 (2004), 5001–5006 vol.5.

C.-H. Kuo and T.-S. Chen. “Modeling and control of autonomous soccer robots using high-level Petri nets”. In: *SICE Annual Conference 2010, Proceedings of.* 2010.

C.-H. Kuo and C.-S. Huang. “Distributed modeling and simulation of 300 mm fab intrabay automation systems using distributed agent oriented Petri nets”. In: *Robotics and Automation, 2003. Proceedings. ICRA ’03. IEEE International Conference on.* 2003.

C.-H. Kuo and I.-H. Lin. “Modeling and Control of Autonomous Soccer Robots Using Distributed Agent Oriented Petri Nets”. In: *Systems, Man and Cybernetics, 2006. SMC ’06. IEEE International Conference on.* 2006.

C.-H. Kuo and I.-H. Lin. *Modeling and Control of Autonomous Soccer Robots Using Distributed Agent Oriented Petri Nets.* Vol. 5. IEEE, 2006.

C.-H. Kuo, C.-H. Wang, and K.-W. Huang. “Behavior modeling and control of 300 mm fab intrabays using distributed agent oriented Petri net”. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 5 (2003).

J. Lejeune, L. Arantes, J. Sopena, and P. Sens. “A fair starvation-free prioritized mutual exclusion algorithm for distributed systems”. In: *Journal of Parallel and Distributed Computing* 83 (Sept. 2015), pp. 13–29.

C. Liu, Y. Ge, H. Xiong, K. Xiao, W. Geng, and M. Perkins. “Proactive workflow modeling by stochastic processes with application to healthcare operation and management”. In: *KDD ’14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, New York, USA: ACM Request Permissions, Aug. 2014, pp. 1593–1602.

Y. Liu. “Game theory semantics for PCTL model checking label-extended probabilistic Petri net”. In: *Computer and Information Science (ICIS), 2014 IEEE/ACIS 13th International Conference on.* 2014.

T. Lux and M. Marchesi. “Scaling and criticality in a stochastic multi-agent model of a financial market”. In: *Nature* (1999).

Maksim. https://commons.wikimedia.org/wiki/File:Scc.png.

E. Manavoglu, D. Pavlov, and C. L. Giles. “Probabilistic user behavior models”. In: *Audio, Transactions of the IRE Professional Group on* (Nov. 2003), pp. 203–210.

S. Mei-hong, J. Shou-shan, G. Yong-gang, C. Liang, and C. Kai-duan. “A Method of Adaptive Process Mining Based on Time-Varying Sliding Window and Relation of Adjacent Event Dependency”. In: *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on.* Jan. 2012, pp. 24–31. DOI: `10.1109/ISdea.2012.536`.

K. Montanez. *Amazon Access Samples Data Set.* 2011. URL: `https://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples`.

J. N. Mordeson and P. S. Nair. *Fuzzy Mathematics: An Introduction for Engineers and Scientists.* Physica, 2001. ISBN: 3790814202.

J. Murphy, V. Berk, and I. Gregorio-de Souza. “Decision Support Procedure in the Insider Threat Domain”. In: *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on.* May 2012, pp. 159–163. DOI: `10.1109/SPW.2012.17`.

C. Pascal and D. Panescu. “On resource allocation in a holonic manufacturing execution system”. In: *System Theory, Control, and Computing (ICSTCC), 2011 15th International Conference on.* 2011.

A. A. Pouyan and S. Reeves. “Behavioral modeling for mobile agent systems using Petri nets”. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on.* 2004.

A. Prodromidis, P. Chan, and S. Stolfo. “Meta-learning in distributed data mining systems: Issues and approaches”. In: *Advances in distributed and parallel knowledge discovery* 3 (2000), pp. 81–114.

R. Ramakrishnan and A. Kaur. “Little’s law based validation framework for load testing”. In: *Information & Software Technology* (Nov. 2018).

A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith. “Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms”. In: *Journal of Mathematical Modelling and Algorithms* 5.4 (2006), pp. 475–504. ISSN: 1572-9214. DOI: `10.1007/s10852-005-9022-1`. URL: `http://dx.doi.org/10.1007/s10852-005-9022-1`.

J. R. Rice. “The algorithm selection problem”. In: *Advances in computers* 15 (1976), pp. 65–118.

M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. “A middleware infrastructure for active spaces”. In: *IEEE pervasive computing* 1.4 (2002), pp. 74–83.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition).* Pearson, 2009. ISBN: 0136042597.

B. S and S. Saravanan. “Survey of Network Anomaly Detection Using Markov Chain”. In: *International Journal of Computer Science, Engineering and Information Technology* 4.1 (Feb. 2014), pp. 49–55.

S. B. Sanjabi and F. Pommereau. “Modelling, verification, and formal analysis of security properties in a P2P system”. In: *Collaborative Technologies and Systems (CTS), 2010 International Symposium on* (2010).

O. Shehory and S. Kraus. “Methods for task allocation via agent coalition formation”. In: *Artificial intelligence* 101.1-2 (May 1998), pp. 165–200.

J. Shen, J. Luo, and G. Gu. “An object-oriented net graph model for agent group-based network management”. In: *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 31. Proceedings.* 1999.

K. M. Sim, S. C. K. Shiu, and M. L. Bun. “Simulation of a multi-agent protocol for task allocation in cooperative design”. In: *Systems, Man, and Cybernetics, 1999. IEEE SMC ’99 Conference Proceedings. 1999 IEEE International Conference on.* 1999.

K. A. Smith-Miles. “Cross-disciplinary perspectives on meta-learning for algorithm selection”. In: *ACM Computing Surveys (CSUR)* 41.1 (2009), p. 6.

H.-J. Song, Z.-Q. Shen, C.-Y. Miao, A.-H. Tan, and G.-P. Zhao. “The Multi-Agent Data Collection in HLA-Based Simulation System”. In: *Principles of Advanced and Distributed Simulation, 2007. PADS ’07. 21st International Workshop on* (2007).

M. Sorba, M. Ghenima, and H. H. Ben Ghezala. “Towards a hybrid approach for a predictive modeling of user navigational behaviors: State of the art”. In: *2013 3rd International Symposium ISKO-Maghreb.* IEEE, 2013, pp. 1–7.

A. C. Squicciarini, G. Petracca, W. G. Horne, and A. Nath. “Situational awareness through reasoning on network incidents”. In: *CODASPY ’14: Proceedings of the 4th ACM conference on Data and application security and privacy.* New York, New York, USA: ACM Request Permissions, Mar. 2014, pp. 111–122.

X. Tan and H. Xi. “Hidden semi-Markov model for anomaly detection”. In: *Applied Mathematics and Computation* 205.2 (Jan. 2008), pp. 562–567.

A. J. C. Trappey, D. W. Hsiao, and L. Ma. “Maintenance Chain Integration Using Petri-Net Enabled Multiagent System Modeling and Implementation Approach”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 3 (2011).

A. J. C. Trappey, D. W. Hsiao, L. Ma, and Y.-L. Chung. “Maintenance chain integration using Petri-net enabled Prometheus MAS modeling methodology”. In: *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on.* 2009.

D. Verdegem and P. Harrison. *Tarjan's algorithm and topological sorting implementation in Python.* Nov. 2015. URL: `http://logarithmic.net/pfh/blog/01208083168`.

W. Wang, X. Zhang, and S. Gombault. “Constructing attribute weights from computer audit data for effective intrusion detection”. In: *Journal of Systems and Software* 82.12 (2009), pp. 1974–1981.

G. I. Webb, M. J. Pazzani, and D. Billsus. “Machine learning for user modeling”. In: *User modeling and user-adapted interaction* 11.1-2 (2001), pp. 19–29.

R. Weiss and C. Steger. “Design and implementation of a real-time multi-agent system”. In: *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean.* 1998.

G. Wen, Z. Duan, G. Chen, and W. Yu. “Consensus tracking of multi-agent systems with Lipschitz-type node dynamics and switching topologies”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.2 (2014), pp. 499–511.

R. J. Witty, A. Allan, J. Enck, and R. Wagner. “Identity and access management defined”. In: *Research Study SPA-21-3430, Gartner* (2003).

B. Xian-hua and X. Wei. “An Approach of Workflow Optimization of Global Supply Chain Based on Object Oriented Petri Net Paper Title”. In: *Information Technology and Computer Science, 2009. ITCS 2009. International Conference on.* 2009.

Y. X. Y. Xie and S.-Z. Y. S.-Z. Yu. “A Large-Scale Hidden Semi-Markov Model for Anomaly Detection on User Browsing Behaviors”. In: *Networking, IEEE/ACM Transactions on* 17.1 (Feb. 2009), pp. 54–65.

N. Ye and T. Farley. “A scientific approach to cyberattack detection”. In: *Computer* 38.11 (2005), pp. 55–61.

N. Y. N. Ye, Y. Z. Y. Zhang, and C. M. Borror. “Robustness of the Markov-chain model for cyber-attack detection”. In: *Reliability, IEEE Transactions on* 53.1 (Mar. 2004), pp. 116–123.

W. Ye, R. Li, and H. Li. “Role Mining Using Boolean Matrix Decomposition with Hierarchy”. In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on.* July 2013, pp. 805–812. DOI: `10.1109/TrustCom.2013.98`.

W. T. Young, H. G. Goldberg, A. Memory, J. F. Sartain, and T. E. Senator. *Use of Domain Knowledge to Detect Insider Threats in Computer Activities.* IEEE, 2013.

A. N. Zakrzewska and E. M. Ferragut. "Modeling cyber conflicts using an extended Petri Net formalism". In: *Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium on* (2011).

L. Zhang, Y. Zhang, P. Jamshidi, L. Xu, and C. Pahl. "Workload Patterns for Quality-Driven Dynamic Cloud Service Configuration and Auto-Scaling". In: *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC).* IEEE, 2014, pp. 156–165.

Cited Work

[1] D. Ahlers and M. Mehrpoor. “Everything is Filed under ‘File’: Conceptual Challenges in Applying Semantic Search to Network Shares for Collaborative Work”. In: *Proceedings of the 26th ACM Conference on Hypertext & Social Media*. ACM. 2015, pp. 327–328.

[2] Alibaba Cloud. *Alibaba Cloud Function Compute*. `https://www.alibabacloud.com/help/doc-detail/52895.htm`. [Online; accessed: 2019-05-17]. 2019.

[3] Amazon Web Services. *AWS Lambda - Serverless Compute - Amazon Web Services*. `https://aws.amazon.com/lambda`. [Online; accessed: 2019-05-17]. 2019.

[4] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane. “Graph-based malware detection using dynamic analysis”. In: *Journal in Computer Virology* 7.4 (2011), pp. 247–258.

[5] 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.

[6] M. Autili, A. Perucci, and L. De Lauretis. “A hybrid approach to microservices load balancing”. In: *Microservices*. Springer, 2020, pp. 249–269.

[7] A. Avritzer, V. Ferme, A. Janes, B. Russo, H. Schulz, and A. van Hoorn. “A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing”. In: *Software Architecture*. Cham: Springer International Publishing, Sept. 2018, pp. 159–174.

[8] Azure Cloud. *Azure Functions*. [Online; accessed: 2019-05-17]. 2019. URL: `https://azure.microsoft.com/en-us/services/functions`.

[9] M. D. Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann. “Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services”. In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. June 2007, pp. 612–617. DOI: `10.1109/DSN.2007.102`.

[10] F. J. Barth. “Using Social Network Analysis and Hierarchical Clustering to Identify Groups in Wikis”. In: *2010 Brazilian Symposium of Collaborative Systems II - Simposio Brasileiro de Sistemas Colaborativos (SBSC-II)*. IEEE, 2010, pp. 8–11.

[11] A. Begel and T. Zimmermann. “Analyze this! 145 questions for data scientists in software engineering”. In: *Proceedings of the 36th International Conference on Software Engineering*. 2014, pp. 12–23.

[12] A. R. Benson, D. F. Gleich, and J. Leskovec. “Tensor Spectral Clustering for Partitioning Higher-order Network Structures”. In: *arXiv preprint arXiv:1502.05058* (2015).

[13] C.-P. Bezemer et al. “How is Performance Addressed in DevOps?” In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ICPE ’19. Mumbai, India: Association for Computing Machinery, 2019, pp. 45–50. ISBN: 9781450362399. DOI: 10.1145/3297663.3309672. URL: https://doi.org/10.1145/3297663.3309672.

[14] D. Birant and A. Kut. “ST-DBSCAN: An algorithm for clustering spatial–temporal data”. In: *Data & Knowledge Engineering* 60.1 (2007), pp. 208–221.

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

[16] R. Bose, W. van der Aalst, I. Zliobaite, and M. Pechenizkiy. “Dealing With Concept Drifts in Process Mining”. In: *Neural Networks and Learning Systems, IEEE Transactions on* 25.1 (Jan. 2014), pp. 154–171. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2278313.

[17] J. S. Breese, D. Heckerman, and C. M. Kadie. *Anonymous Microsoft Web Data Data Set*. 1998. URL: https://archive.ics.uci.edu/ml/datasets/Anonymous%20Microsoft%20Web%20Data.

[18] J. S. Breese, D. Heckerman, and C. Kadie. “Empirical analysis of predictive algorithms for collaborative filtering”. In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1998, pp. 43–52.

[19] A. Bruns, A. Kornstadt, and D. Wichmann. “Web Application Tests with Selenium”. In: *IEEE Software* 26.5 (Sept. 2009), pp. 88–91. ISSN: 1937-4194. DOI: 10.1109/MS.2009.144.

[20] I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. “Visualization of navigation patterns on a Web site using model-based clustering”. In: *KDD*. 2000, p. 280.

[21] L. Cai, C. K. Chang, and J. Cleland-Huang. “Supporting agent-based distributed software development through modeling and simulation”. In: *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*. 2003.

[22] K. Campbell, L. A. Gordon, M. P. Loeb, and L. Zhou. “The economic cost of publicly announced information security breaches: empirical evidence from the stock market”. In: *Journal of Computer Security* 11.3 (2003), pp. 431–448.

[23] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. “An approach for QoS-aware service composition based on genetic algorithms”. In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM. 2005, pp. 1069–1075.

[24] J. R. Celaya, A. A. Desrochers, and R. J. Graves. “Modeling and analysis of multi-agent systems using petri nets”. In: *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*. 2007.

[25] W. Chainbi. “Multi-agent systems: a Petri net with objects based approach”. In: *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on.* 2004.

[26] V. Chandola, A. Banerjee, and V. Kumar. “Anomaly Detection for Discrete Sequences: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839.

[27] J. Chen and W. Shang. “An Exploratory Study of Performance Regression Introducing Code Changes.” In: *ICSME* (2017).

[28] 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.” In: *ICSE-SEIP* (2017).

[29] B. Chikhaoui, S. Wang, and H. Pigot. “Causality-Based Model for User Profile Construction from Behavior Sequences”. In: *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on.* Mar. 2013, pp. 461–468. DOI: 10.1109/AINA.2013.109.

[30] A. Colantonio, R. Di Pietro, and A. Ocello. *Role Mining in Business: Taming Role-Based Access Control Administration.* World Scientific, 2012.

[31] R. Collobert and J. Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning.* ACM. 2008, pp. 160–167.

[32] M. E. Conway. “How do committees invent”. In: *Datamation* (1968).

[33] D. J. Cook and N. C. Krishnan. *Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data.* John Wiley & Sons, 2015.

[34] P. A. Corning. “The re-emergence of “emergence”: A venerable concept in search of a theory”. In: *Complexity* 7.6 (2002), pp. 18–30. ISSN: 1099-0526. DOI: 10.1002/cplx.10043. URL: http://dx.doi.org/10.1002/cplx.10043.

[35] D. Cotroneo, R. Pietrantuono, and S. Russo. “RELAI testing: a technique to assess and improve software reliability”. In: *IEEE Transactions on Software Engineering* PP.99 (2015), pp. 1–1. ISSN: 0098-5589. DOI: 10.1109/TSE.2015.2491931.

[36] B. De Carolis, S. Ferilli, and D. Redavid. “Incremental Learning of Daily Routines as Workflows in a Smart Home Environment”. In: *Transactions on Interactive Intelligent Systems (TiiS* 4.4 (Jan. 2015).

[37] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno. “Search-based testing of service level agreements”. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation.* ACM. 2007, pp. 1090–1097.

[38] R. Diestel. *Graph Theory (Graduate Texts in Mathematics, 173)*. Springer-Verlag, 1997. ISBN: 0387982108.

[39] V. Dimitrova, T. Kuflik, D. Chin, F. Ricci, P. Dolog, and G.-J. Houben. *User Modeling, Adaptation and Personalization*. Vol. 8538. 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings. Cham: Springer, June 2014.

[40] W. Dong. "Multi-agent test environment for BPEL-based web service composition". In: *Cybernetics and Intelligent Systems, 2008 IEEE Conference on*. 2008.

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

[42] D. Fei, Z. Liqun, N. GuangYun, and X. Xiaolei. "A Algorithm for Detecting Concept Drift Based on Context in Process Mining". In: *Digital Manufacturing and Automation (ICDMA), 2013 Fourth International Conference on*. June 2013, pp. 5–8. DOI: 10.1109/ICDMA.2013.2.

[43] D. Ferraiolo and D. Kuhn. "Role Based Access Control". In: *15th National Computer Security Conf.* Oct. 1992, pp. 554–563.

[44] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. "Proposed NIST standard for role-based access control". In: *ACM Transactions on Information and System Security (TISSEC)* 4.3 (2001), pp. 224–274.

[45] C. H. G. Ferreira, L. H. Nunes, L. A. Pereira, L. H. V. Nakamura, J. C. Estrella, and S. Reiff-Marganiec. "PEESOS-Cloud: A Workload-Aware Architecture for Performance Evaluation in Service-Oriented Systems". In: *2016 IEEE World Congress on Services Computing (SERVICES)*. IEEE, 2016, pp. 118–125.

[46] D. R. Ferreira, F. Szimanski, and C. G. Ralha. "A Hierarchical Markov Model to Understand the Behaviour of Agents in Business Processes". In: *Business Process Management Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 150–161.

[47] M. Frank, J. M. Buhmann, and D. Basin. "On the Definition of Role Mining". In: *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*. SACMAT '10. Pittsburgh, Pennsylvania, USA: ACM, 2010, pp. 35–44. ISBN: 978-1-4503-0049-0. DOI: 10.1145/1809842.1809851. URL: http://doi.acm.org/10.1145/1809842.1809851.

[48] H. Funke. "Model Based Test Specifications: Developing of Test Specifications in a Semi Automatic Model Based Way". In: *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2011, pp. 496–500.

[49] R. Gao, Z. M. Jiang, C. Barna, and M. Litoiu. *A Framework to Evaluate the Effectiveness of Different Load Testing Analysis Techniques*. IEEE, Apr. 2016.

[50] 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.

[51] Google Cloud. *Google Cloud Functions.* [Online; accessed: 2019-05-17]. 2019. URL: https://cloud.google.com/functions.

[52] T. R. Gruber. "Toward principles for the design of ontologies used for knowledge sharing?" In: *International journal of human-computer studies* 43.5 (1995), pp. 907–928.

[53] Y. Gu and Y. Ge. "Search-based performance testing of applications with composite services". In: *Web Information Systems and Mining, 2009. WISM 2009. International Conference on.* IEEE. 2009, pp. 320–324.

[54] P. Harika, M. Nagajyothi, J. John, S. Sural, J. Vaidya, and V. Atluri. "Meeting Cardinality Constraints in Role Mining". In: *Dependable and Secure Computing, IEEE Transactions on* PP.99 (2014), pp. 1–1. ISSN: 1545-5971. DOI: 10.1109/TDSC.2014.2309117.

[55] D. Heckerman. *MSNBC.com Anonymous Web Data Data Set.* 2000. URL: http://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data.

[56] 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. L'Aquila, Italy: ACM, 2017, pp. 223–226. ISBN: 978-1-4503-4899-7. DOI: 10.1145/3053600.3053653. URL: http://doi.acm.org/10.1145/3053600.3053653.

[57] X. A. Hoang and J. Hu. "An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls". In: *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic* 2 (Nov. 2004), pp. 470–472.

[58] 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. [Online; accessed: 2019-05-19]. 2008.

[59] 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.

[60] J. H. J. Hu, X. Y. X. Yu, D. Qiu, and H.-H. C. H.-H. Chen. "A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection". In: *Network, IEEE* 23.1 (Feb. 2009), pp. 42–47.

[61] IBM Cloud. *IBM Cloud Functions.* [Online; accessed: 2019-05-17]. 2019. URL: `https://www.ibm.com/cloud/functions`.

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

[63] M. Janik and K. J. Kochut. "Wikipedia in action: Ontological knowledge in text categorization". In: *Semantic Computing, 2008 IEEE International Conference on.* IEEE. 2008, pp. 268–275.

[64] N. R. Jennings. "On agent-based software engineering". In: *Artificial intelligence* 117.2 (Mar. 2000), pp. 277–296.

[65] N. R. Jennings, K. Sycara, and M. Wooldridge. "A Roadmap of Agent Research and Development". In: *Autonomous Agents and Multi-Agent Systems* 1.1 (Jan. 1998), pp. 7–38.

[66] 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).* IEEE, 2017, pp. 500–507.

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

[68] Z. M. Jiang and A. E. Hassan. "A Survey on Load Testing of Large-Scale Software Systems". In: *IEEE Transactions on Software Engineering* 41.11 (Nov. 2015), pp. 1091–1118. ISSN: 0098-5589. DOI: `10.1109/TSE.2015.2445340`.

[69] S. S. Joshi and V. V. Phoha. "Investigating hidden Markov models capabilities in anomaly detection". In: *the 43rd annual southeast regional conference.* New York, New York, USA: ACM Press, 2005, p. 98.

[70] K. Kant, V. Tewari, and R. Iyer. "Geist: A web traffic generation tool". In: *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation.* Springer. 2002, pp. 227–232.

[71] K. Kapoor, M. Sun, J. Srivastava, and T. Ye. "A hazard based approach to user return time prediction". In: *KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Request Permissions, Aug. 2014.

[72] S. Karlin and H. M. Taylor. *A First Course in Stochastic Processes, Second Edition.* Academic Press, 1975. ISBN: 0123985528.

[73] C. Laaber and P. Leitner. "An evaluation of open-source software microbenchmark suites for continuous performance assessment". In: *the 15th International Conference.* New York, New York, USA: ACM Press, 2018, pp. 119–130.

[74] M. Labs. *A Week in the Life of a Browser - Version 2: Aggregated Data Samples.* July 2011. URL: https://testpilot.mozillalabs.com/testcases/a-week-life-2/aggregated-data.html.

[75] K. L. K. Lee, D. Ellis, and A. C. Loui. "Detecting local semantic concepts in environmental sounds using Markov model based clustering". In: *IEEE International Conference on Acoustics, Speech and Signal Processing. Proceedings* (Mar. 2010), pp. 2278–2281.

[76] P. Leitao and A. W. Colombo. "Petri net based Methodology for the Development of Collaborative Production Systems". In: *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on.* 2006.

[77] 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: ACM Press, 2017, pp. 373–384.

[78] N. Li, J. Kang, and W. Lv. "A hybrid approach for dynamic business process mining based on reconfigurable nets and event types". In: *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on.* Oct. 2005, pp. 289–294. DOI: 10.1109/ICEBE.2005.5.

[79] M. Lichman. *UCI Machine Learning Repository.* 2013. URL: http://archive.ics.uci.edu/ml.

[80] A. Mahmoud. "An information theoretic approach for extracting and tracing non-functional requirements". In: *2015 IEEE 23rd International Requirements Engineering Conference (RE).* Aug. 2015, pp. 36–45. DOI: 10.1109/RE.2015.7320406.

[81] A. Mahmoud and D. Carver. "Exploiting online human knowledge in Requirements Engineering". In: *2015 IEEE 23rd International Requirements Engineering Conference (RE).* IEEE. 2015, pp. 262–267.

[82] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora, and G. Hamann. "Automatic Comparison of Load Tests to Support the Performance Analysis of Large Enterprise Systems". In: *14th European Conference on Software Maintenance and Reengineering (CSMR 2010).* IEEE, 2010, pp. 222–231.

[83] V. Melnykov et al. "Model-based biclustering of clickstream data". In: *Computational Statistics & Data Analysis* 93.C (2016), pp. 31–45.

[84] D. A. Menascé. "Load testing of Web sites". In: *IEEE Internet Computing* 6.4 (June 2002), pp. 70–74. ISSN: 1089-7801. DOI: 10.1109/MIC.2002.1020328.

[85] 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.* 1999, pp. 119–128.

[86] S. E. Middleton, N. R. Shadbolt, and D. C. De Roure. "Ontological user profiling in recommender systems". In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 54–88.

[87] T. M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. ISBN: 0070428077.

[88] S. Nachiyappan and S. Justus. "Cloud testing tools and its challenges: A comparative study". In: *procedia computer Science* 50 (2015), pp. 482–489.

[89] K. Nakayama, T. Hara, and S. Nishio. "Wikipedia mining for an association web thesaurus construction". In: *International Conference on Web Information Systems Engineering*. Springer. 2007, pp. 322–334.

[90] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung. "Evaluating Sensitivity of Auto-Scaling Decisions in an Environment with Different Workload Patterns". In: *2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2015, pp. 415–420.

[91] Oracle Cloud. *Oracle Functions*. `https://docs.cloud.oracle.com/iaas/Content/Functions/Concepts/functionsoverview.htm`. [Online; accessed: 2019-05-17]. 2019.

[92] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank citation ranking: bringing order to the web." In: *Stanford InfoLab* (1999).

[93] C. Parrott and D. Carver. "A Model of Wikipedia Knowledge Communities: Learning from User Behavior". In: *2020 22nd International Conference of the Society for Design and Process Science (SDPS)*. SDPS, 2020, pp. 108–117.

[94] C. Parrott and D. Carver. "Lodestone: A Streaming Approach to Behavior Modeling and Load Testing". In: *2020 3nd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2020.

[95] J. Pecarina and J.-c. Liu. "APSAT: A framework for modeling and analysis of workflow dynamics". In: *in msn-centric Syst., in Collaboration Tech. and Syst. (CTS), 2012 Int. Conf. on.* 2012, pp. 575–582.

[96] J. Pecarina and J.-c. Liu. "Behavior instance extraction for risk aware control in mission centric systems". In: *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conference on.* Feb. 2013, pp. 51–58. DOI: `10.1109/CogSIMA.2013.6523823`.

[97] J. Punuru and J. Chen. "Automatic acquisition of concepts from domain texts". In: *2006 IEEE International Conference on Granular Computing*. IEEE. 2006, pp. 424–427.

[98] 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*. ACM. 2019, p. 15.

[99] A. Rafael Lenz, A. Pozo, and S. Regina Vergilio. “Linking software testing results with a machine learning approach”. In: *Engineering Applications of Artificial Intelligence* 26.5-6 (May 2013), pp. 1631–1640.

[100] 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: ACM Press, 2017, pp. 157–164.

[101] A. S. Rao and M. P. Georgeff. “Modeling rational agents within a BDI-architecture”. In: *KR* (1991).

[102] E. Rich. “User modeling via stereotypes*”. In: *Cognitive science* 3.4 (1979), pp. 329–354.

[103] P. Schönhofen. “Identifying document topics using the Wikipedia category network”. In: *Web Intelligence and Agent Systems: An International Journal* 7.2 (2009), pp. 195–207.

[104] H. Schulz, A. van Hoorn, and A. Wert. “Reducing the maintenance effort for parameterization of representative load tests using annotations”. In: *Software Testing, Verification and Reliability* 30.1 (2020). e1712 stvr.1712, e1712. DOI: 10.1002/stvr.1712. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1712. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1712.

[105] M. Schur, A. Roth, and A. Zeller. “Mining Workflow Models from Web Applications”. In: *IEEE Transactions on Software Engineering* 41.12 (Dec. 2015), pp. 1184–1201. ISSN: 0098-5589. DOI: 10.1109/TSE.2015.2461542.

[106] H. M. Schwartz. *Multi-Agent Machine Learning: A Reinforcement Approach*. Wiley, Jan. 2014.

[107] W. Sha, Y. Zhu, T. Huang, M. Qiu, Y. Zhu, and Q. Zhang. “A Multi-order Markov Chain Based Scheme for Anomaly Detection”. In: *2013 IEEE 37th International Computer Software and Applications Conference Workshops (COMPSACW)*. IEEE, 2013, pp. 83–88.

[108] 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)*. IEEE, 2016, pp. 22–29.

[109] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008. ISBN: 9781139475242.

[110] Social Security Administration. *Beyond the Top 1000 Names*. URL: https://www.ssa.gov/oact/babynames/limits.html. (accessed: 05.29.2017).

[111] Y. Song, A. D. Keromytis, and S. J. Stolfo. “Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic.” In: *NDSS 2009* (2009).

[112] F. M. Suchanek, G. Kasneci, and G. Weikum. "Yago: A large ontology from wikipedia and wordnet". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6.3 (2008), pp. 203–217.

[113] Z. S. Syed, T. Finin, and A. Joshi. "Wikipedia as an Ontology for Describing Documents." In: *ICWSM*. 2008.

[114] H. A. Taha. *Operations Research: An Introduction.* Collier Macmillan Ltd, 1977. ISBN: 0024188204.

[115] M. A. Talib, A. Khelifi, A. Abran, and O. Ormandjieva. "Techniques for quantitative analysis of software quality throughout the SDLC: The SWEBOK Guide Coverage". In: *2010 Eighth ACIS International Conference on Software Engineering Research, Management and Applications.* May 2010, pp. 321–328. DOI: `10.1109/SERA.2010.47`.

[116] R. Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[117] The Apache Software Foundation. *Apache JMeter.* `https://jmeter.apache.org`. [Online; accessed: 2019-05-19]. 2019.

[118] W. Tinney, V. Brandwajn, and S. Chan. "Sparse vector methods". In: *IEEE transactions on power apparatus and systems* 2 (1985), pp. 295–301.

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

[120] W. Van Der Aalst. "On the Representational Bias in Process Mining". In: *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on.* June 2011, pp. 2–7. DOI: `10.1109/WETICE.2011.64`.

[121] W. Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes.* Vol. 2. Springer, 2011.

[122] 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.* Springer. 2008, pp. 124–143.

[123] VMware, Inc. *Spring Framework.* URL: `https://spring.io/microservices`. (accessed: 01.12.2018).

[124] 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.

[125] 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: ACM Press, 2015, pp. 5–8.

[126] 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”. In: *Software & Systems Modeling* 17.2 (Oct. 2016), pp. 443–477.

[127] J. Voss. “Collaborative thesaurus tagging the Wikipedia way”. In: *arXiv preprint cs/0604036* (2006).

[128] J. A. Whittaker and J. H. Poore. “Statistical testing for cleanroom software engineering”. In: *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on.* Vol. ii. Jan. 1992, 428–436 vol.2. DOI: `10.1109/HICSS.1992.183256`.

[129] 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)*. IEEE, 2018, pp. 25–32.

[130] Wikipedia.com. *Wikipedia: Size of Wikipedia.* URL: `%7Bhttps://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia%7D`.

[131] R. Wille. “Concept lattices and conceptual knowledge systems”. In: *Computers & Mathematics with Applications* 23.6 (1992), pp. 493–515. ISSN: 0898-1221. DOI: `http://dx.doi.org/10.1016/0898-1221(92)90120-7`.

[132] R. Wille. “Restructuring lattice theory: an approach based on hierarchies of concepts”. In: *Ordered sets.* Springer, 1982, pp. 445–470.

[133] R. Wille. “Subdirect decomposition of concept lattices”. In: *Algebra Universalis* 17.1 (1983), pp. 275–287.

[134] F. Wu and D. S. Weld. “Automatically refining the wikipedia infobox ontology”. In: *Proceedings of the 17th international conference on World Wide Web.* ACM. 2008, pp. 635–644.

[135] Y. Wu, S. Yang, and X. Yan. “Ontology-based subgraph querying”. In: *Data Engineering (ICDE), 2013 IEEE 29th International Conference on.* IEEE. 2013, pp. 697–708.

[136] E. Wulczyn and D. Taraborelli. *Wikipedia Clickstream.* http://dx.doi.org/10.6084/m9.figshare.1305770. Feb. 2015.

[137] T. Xiong, S. Wang, Q. Jiang, and J. Huang. “A New Markov Model for Clustering Categorical Sequences”. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on.* Dec. 2011, pp. 854–863. DOI: `10.1109/ICDM.2011.13`.

[138] H. Xu and S. M. Shatz. “A framework for model-based design of agent-oriented software”. In: *Software Engineering, IEEE Transactions on* 1 (2003).

[139] H. Xu and S. M. Shatz. “A framework for modeling agent-oriented software”. In: *Distributed Computing Systems, 2001. 21st International Conference on.* 2001.

[140] Z. Yang, R. Algesheimer, and C. J. Tessone. “A Comparative Analysis of Community Detection Algorithms on Artificial Networks”. In: *Scientific Reports* 6 (Aug.). URL: `http://dx.doi.org/10.1038/srep30750`.

[141] N. Ye. “A markov chain model of temporal behavior for anomaly detection”. In: *Proceedings of the 2000 IEEE Systems.* 2000.

[142] J. Yu, J. A. Thom, and A. Tam. “Ontology evaluation using wikipedia categories for browsing”. In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management.* ACM. 2007, pp. 223–232.

[143] P. Zhang, S. Elbaum, and M. B. Dwyer. “Compositional load test generation for software pipelines”. In: *Proceedings of the 2012 International Symposium on Software Testing and Analysis.* ACM. 2012, pp. 89–99.

Vita

Chester Ira Parrott was born in Tennessee and raised in Baton Rouge, Louisiana. His focus has always been on forging creative solutions to technological problems through data-driven tools and analysis. A non-traditional background provided him with stubborn tenacity, a solid practical foundation, and a driving curiosity for his higher education in computer science and mathematics. Before completing his undergraduate degree he provided cyber-security, access management, consultation, and software development services to Fortune 500 companies, as well as to state, local, and federal governmental agencies.

He worked full-time while completing his undergraduate studies in computer science and mathematics and graduated from Southeastern Louisiana University. Upon graduation, he came to Louisiana State University to pursue graduate studies in computer science. Even as his family grew from two to six throughout his graduate studies, he continued to work full-time as a data scientist and technology leader. He has presented original research at industrial and academic conferences and currently holds one patent. He plans to continue in the field of data scientific user behavior analysis toward a better understanding of the patterns through which people and machines interact to improve the current-state tools and practices of software engineering.