8-2024

Game-Theory Application In Co-Resident Security Of Function-As-A-Service Cloud Environments

Damon L. Alsup

GAME-THEORY APPLICATION IN CO-RESIDENT SECURITY OF FUNCTION-AS-A-SERVICE CLOUD ENVIRONMENTS

A Dissertation

by

DAMON L. ALSUP

Submitted to the Graduate and Professional
School of Prairie View A&M University
in partial fulfillment of the requirements for the degree of

DOCTORATE OF PHILOSOPHY

August 2024

Major Subject: Electrical and Computer Engineering

GAME-THEORY APPLICATION IN CO-RESIDENT SECURITY OF FUNCTION-AS-A-SERVICE CLOUD ENVIRONMENTS

A Dissertation by

DAMON L. ALSUP

Submitted to the Graduate and Professional
School of Prairie View A&M University
in partial fulfillment of the requirements for the degree of

DOCTORATE OF PHILOSOPHY

Suxia Cui Chair of Committee,	Olusegun Odejide Member
Mohammad Chouikha Member	Chia-Che Tsai Member
Annamalai Head of Department	
Pamala Obiomon Dean of Engineering	Tyrone Tanner Dean of Graduate Studies

August 2024

Major Subject:
Electrical and Computer Engineering

ABSTRACT

Game-Theory Application in Co-Resident Security of Function-As-A-Service Cloud Environments

(August 2024)

Damon L. Alsup, B.A., University of Houston; B.S., M.S., Prairie View A&M University;

Chair of Advisory Committee: Dr. Suxia Cui

The cloud is a shared computing environment with a value beyond the sum of its parts. The number of customers that data-centers can serve, comparative advantages, and the ability to manage depreciation allow computing at economies of scale. The cloud allows for every element of factors of production to translate into goods and services. This shared environment spans across a vast clientele, introducing self-sustaining security risks. The vulnerabilities extend beyond the traditional gaps in computer security, through exploitation of the cloud's efficiency structures.

Shared computing resources enable the existence of co-resident attack vectors on cloud platforms. This study considered the result of modeling co-resident threats in simulation at the boundaries of game-theory using real-world workloads, scalable hardware specifications, and recognized attack parameters. Both attacker and benign user variables were adapted to an extended-time and geographically defined game-space and the results of the co-resident risk determined on an ecological scale.

This study sought to determine the applicability of this technique to emergent cloud structures. The current cloud trend is toward finer granularity programming of applications, where decoupling of data and algorithms into developer customized programming is ceded to by monolithic applications. This phasing into micro-service based limited purpose coding is called Functions-as-a-Service (FaaS). Supporting this feature is provider management, configuration, and patching which anchors FaaS in a serverless interface.

This cloud evolution of code, storage, and presentation into distinct sectors has altered the security environment into discrete sectors by reducing state, ephemeral hosting, and transient runtimes to enable the sought after economic efficiency. Where this increased the cloud dynamism, it also redistributed the cost to benefit analysis. The effective implementation of the game-theory principles required validation on this economic structure.

Index Terms – Cloud computing, data leakage, game-theory, mutli-tenancy, security

DEDICATION

To my grandparents.

ACKNOWLEDGMENTS

First, I would like to thank the committee for their time and advice throughout this study. The steps of academic work, article and literature review research, experimental and data analysis, and finally the writing process were immeasurably assisted by their guidance: Dr. Suxia Cui, Professor PVAMU, Dr. Olusegun Odejide, Adjunct Instructor PVAMU, Dr. Mohammad Chouikha, Professor PVAMU, Dr. Chia-Che Tsai, Assistant Professor TAMU, Dr. Annamalai, Professor PVAMU and Department Head.

I would also like to thank my cohorts and collaborators for helping me develop the ideas and planning that went into the experiments and data collection. And, I would be remiss to not express my gratitude for their keeping me on track when the ideas became ‘too good.’: Dr. Jack Zhang and Mohan Putluru, The University of Houston College of Technology and his graduate student. Ph.D. Candidate Aleksandr Earnest and Masters Student Moeez Akmal, and mechanical engineering student Cole M. Tipton, The Texas A&M University School of Engineering.

The Prairie View A&M University Graduate and Professional School and Texas A&M University Graduate and Professional School, David Fleeger and Dr. April Lovelady at Blinn Junior College, for their support. Dr. Dandrielle Lewis and Shayla DiBonaventura of the Prairie View Mathematics Department for helping me get through the last hundred yards.

Finally, and for all time. I would like to thank Geralyn Tipton. For all the new things we have shared and all the things to come...

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

CHAPTER I

INTRODUCTION

1.1 Economics of Cloud Computing

The computing cloud, 'the cloud,' is an economic model designed with the objective of providing information technology goods and services. 1 The cloud may serve academic, governmental, or private interests. With a widespread provider and consumer base, the cloud is a modern manifestation of managing scarcity in the interest of generating wealth [3]. The cloud is an advance and challenge, no less significant than the shift from mercantilism and agrarian economies to market and industrial models.

1.1.1) Factors of Production: The cloud provides widespread modern computational power by centering advanced information technology with maintenance, expertise, and logistic support. Networked distribution and data processing were tailored to specific requirements and packaging for distribution to cloud customers. These may be physical resources, such as power and network connectivity, or abstract, such as data structures. The cloud is modular, expansive, flexible, and has a high capacity that is sufficient to provide an economy of scale. The cloud provides this through the management of the factors of production: land, labor, and capital.

1.1.1.1) Land - The data-center: In the classical sense, land is the natural resources and geographical features used in production. Drawing from its characteristic

This dissertation follows the style of the IEEE Editorial Style Manual for Authors, 2022 Version of inexhaustibility, the cloud economic model considers the datacenter, its networking, and power sources. Emerging from and taking inspiration from grid computing, the data-center is designed as a collective housing of commodity-grade servers. This integration is more cohesive than peer-to-peer networks yet less monolithic than a mainframe configuration. The cloud business model is designed to market to the widest consumer base with the latitude to address the greatest range of requirements. It must do so in a fashion that meets the on-demand characteristic, with an agreed upon quality of service.

1.1.1.2) Labor - Technical and engineering staff: While surely an impressive achievement with multiple data-centers and hundreds of thousands of networked servers, these alone are not sufficient to achieve computing power greater than the sum of the parts. For the cloud to achieve that synergy, the cloud data-center requires the same factor that modernized agrarian economies: division of labor [4]. Labor costs consider the hierarchy of competencies, licenses, and management inherent in complex enterprises. What the cloud service provider gets in return is effective optimizations, longevity of tools and equipment, and sustainable task to purpose.

1.1.1.3) Capital – Virtualization: Capital outlays include payment for the load to the power grid, traffic on the leased backbone, and debts for infrastructure. Capital is also ‘those durable produced goods that are in turn used as productive inputs for further production’ [5]. More informally, capital is ‘the stuff that makes stuff.’

Central to the economy of the cloud is virtualization technology, a combination of hardware architecture features and software structures adapted to maximize the efficiency and utilization of data-center machines. Virtualization provides a multi-user platform by allowing the abstraction of resources coordinated with execution environments hosted on different physical machines (PMs). Through virtualization, the cloud can achieve scalability, which is an essential element of the 'pay-as-you-go' model. Taking advantage of the high speed of processors, multi-threading, memory management, and other lower-level architectures, an abstraction of the machine and operating system share time and space with other abstractions.

1.1.2) Services: Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data-centers that provide those services. The distinctions that separate any class of those services are aligned with the hardware and software that provide those services. Traditionally, these distinctions are separated into:

1.1.2.1) Infrastructure-as-a-Service (IaaS): IaaS requires the greatest upkeep from the customer, which is essentially a physical bare-metal or virtual machine. It requires the most system configuration, networking, and storage management. An example is Amazon EC2, which looks much like physical hardware, and users can control nearly the entire software stack from the kernel upwards.

1.1.2.2) Platform-as-a-Service (PaaS): PaaS reduces the necessary information technology tasks at some cost to the consumer. The focus of this model is providing a configured host on which those data-center customers can load commercial or in-house applications. The PaaS model forms the basis for a wide range of legacy and contemporary subsets, including Backend-as-a-Service (BaaS), which will be addressed in the second half of this work.

1.1.2.3) Software-as-a-Service (SaaS): SaaS providers abstract away the requirement for the customer to engage in the maintenance of IaaS or PaaS. In this model, packaged applications are presented for the consumer in the customized front-end. Although these may mirror non-cloud environments [6], they will likely be targeted to a different patronage.

Each of these services has an “on-demand” quality at different monetary scales, governed by how the provider adopts the three economic factors. They are developed from various levels of interconnectivity, and, in almost all cases, the services are offered in some form of abstraction.

1.1.3) The Entrepreneurial Spirit: The data-center is then capable of offering anything as a service (XaaS) targeted at both household consumers and commercial entities. Intellectual property, financial records, and industry operation software reside in the cloud, not just at the commercial data-center. More than the aforementioned services, the merger between the CSP and the user provides conditions for goods. These are developed from the final economic factor of production versions of the final factor, ‘the entrepreneurial spirit’ [7].

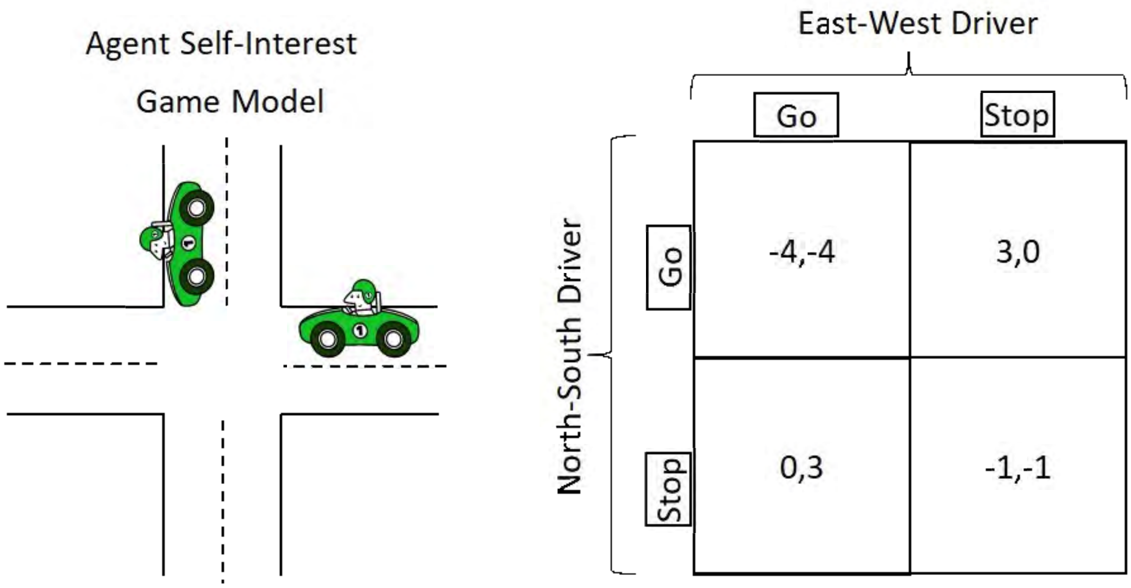
1.1.3.1) Locally Shared Entrepreneurship: Entrepreneurship, that is, the exploitation of previously non-commercialized knowledge and ideas) is essential to a regional economic system [8]. The exploitation can be either cooperative or competitive, and the ideas being applied may be either innovative or familiar. More generally, commercialization must only present a novel utility to an agent. This factor produces an ecosystem with its own life cycles and dynamics.

Unfortunately, the entrepreneurial spirit can elicit selfish conduct. The virtualization methods which are meant to be shared are then subject to a user's control. In other words, the capital expenditures that were intended to become a good or service are instead detracting from each other. This contention need not be a directly aggressive act. However, each party may be inclined to disrupt such an effort or engage in their own efforts to overuse the platform. These objectives, regardless of the number of competing users, still occur in the self-contained vitalized environment.

The shared computing environment, which is dynamic, economically advantageous, and accessible, created conditions ripe for resource competition. Although these may not be directly in opposition, each cloud user is serving their own best interest. Where possible, users will seek to maximize return on investment through programmatic means that seize processor time, memory channel exploits, or network access. They will employ peremptory programmatic methods, take advantage of cloud service provider (CSP) policy, and deploy loads on resources that may have designs on them by another user.

An analog to this can be found in two drivers approaching a malfunctioning traffic light at right angles (see Fig. 1.1). Either may stop while letting the other one pass, depending on their sense of what the other will do. Of course, neither may stop, depending on their sense of urgency. Or they both may stop, apprehensive as to the other driver's caution.

As a virtualized system, users share time and space on cloud resources. Without having a direct conflict, like the drivers at the intersection, they will be seeking to maximize their payoffs through expeditious execution of their own code. If an execution is too selfish, it will result in a mutually destructive outcome.



		East-West Driver	
		Go	Stop
North-South Driver	Go	-4,-4	3,0
	Stop	0,3	-1,-1

Fig. 1.1: A game theory representation of two drivers meeting at an uncontrolled intersection from perpendicular directions. The drivers assume the role of players acting in their best interests, but not necessarily in opposition. The driver strategies determine the outcome for the other, their payoffs determined by the matrix. Payoff's for the N-S driver and E-W driver are an ordered pair, in respective order. The players, strategies, and payoffs form the essential elements of an uncontrolled intersection game.

1.1.3.2) Entrepreneurs as Players: The uncontrolled intersection example above is not the only parallel or similarity to circumstances that arise in the cloud. It is, however, an appropriate model to adopt agents in the cloud as players in a game, perhaps requiring the same branch target buffer, attempting to make use of speculative instruction hardware, or in the queue at a fat-tree network hub. Game theory can be applied to many aspects of the cloud to include power usage, networking, scheduling, and security.

A discussion of game theory in a broad sense can be found in Appendix A. Additional development is included for other cloud computing parallels. These may be mixed strategy games (see Appendix A.11.) or extended form (see Appendix A.1.3.) games.

Models may be cooperative or competitive. However, each considers an uncontrolled localized environment. For instance, one may consider the circumstances where not just one intersection with a broken traffic light models the access to a host computer, but the case of an entire neighborhood without traffic lights. See the development of this model in Appendix A.5. The result would produce queuing issues, a traffic wave, and cyclical congestion in the network. Such cyclical patterns will emerge throughout the development of the game-theory applied in Chapter 2. Within these cyclical models, pockets of stability can emerge.

1.2. Study Threat Model

The ability to generate wealth makes the cloud an attractive target, and its broad network footprint presents a wide attack surface. Data in transition risks compromise where any breach presents an exploitable entry point. While the cloud evolves, its own dynamism sustains competing interests. The threat evaluated in this study emerges from the co-residence of these interests in the virtualized cloud environment.

1.2.1) Co-Resident Threats: Exploitation of shared resources emerges from the system features that provide the ability to affect co-resident users. These may entirely obstruct their capabilities or do so only minimally. Effects may be asymmetric or perhaps place the offending party at an equal risk.

1.2.1.1) Denial-of-Service (DoS): A denial of service attack disallows a co-resident user the ability to utilize the cloud environment. Sophisticated manipulation of the virtualization software through user-available controls can be adapted to limit the ability of other users to execute their programs [9]. Generally a resource heavy attack, DoS may entirely consume the attacker's assets.

1.2.1.2) Preemption: Nuanced interaction with the virtualization management software provides a means to acquire computing resources and hold onto them, delaying the execution of co-resident users. These may be billed or unbilled executions, nor may it be possible for the affected user to ascertain the preemption [10]. A more subtle implementation than the DoS attack, preemption cheats the CSP as well.

1.2.1.3) Covert Side-Channels: A side-channel is established through unintended signal leakage, allowing for the exchange of information without the proper awareness of the CSP. This may be done to avoid billing, clandestine communication, or as a lead-in to an advanced persistent threat. These are generally attained through some hardware manipulation [11] and archive varying levels of bandwidth.

1.2.2) Co-Tenant Data Leakage: The ability to infer information or to outright steal information from a co-tenant user has been a continuous concern for CSPs and users. An unforeseen algorithmic combination, system vulnerability, or hardware features are all potential gaps that might be exploited. A more hostile version of the side-channel attack, the co-tenant data leakage, presents a particular threat to cloud users.

1.2.2.1) Leakage Objectives: The objective of the attacker is to infer data through manipulation of the virtualized environment. The ability to do so in a surreptitious manner can catch the victim entirely unaware that they have been affected. The attacker may be seeking a cryptographic key, raw data, or process information. Because the victim user may be entirely dependent on the cloud for their business model, the loss of this data can be an existential threat. What data obtained either singularly or over an extended period of time has been shown to depend on the sophistication of the attack [12].

1.2.2.2) Excluded Scope: In modern cloud systems, many attacks have been thoroughly explored. These may be virus or Trojan-type software intrusions, which have existed for decades before the cloud and are not a focus of the study. Those co-resident threats which are listed in Subsection 1.2.1 are similarly not considered. Isolation escapes from neither the traditional cloud virtual machines nor the containerized virtualization in Chapter F is considered. These topics are considered in other works.

1.3 Contributions

This study evaluated the ability of game-theory to model mitigation strategies for co-tenant data leakages. The ability to extract scoring and utility in traditional virtual machine environments is conducted in simulation as a method of providing complements to the strategic and player components of game-theory. The influence of game-theory on the cloud evolution to the serverless model is explored and addressed in its technical aspects. Finally, an examination of whether the strategic component of game-theory is present in modern user-programmed platforms. This is conducted with projected high-use programmatic algorithms in prospective high-value/high-risk applications.

CHAPTER II

LITERATURE REVIEW

2.1 The Tension of Cloud Virtualization:

Virtualization is the factor that allows the cloud economy. The data-center hosts have capabilities beyond allowing a single user execution and can share resources in such a way that they provide multiple users the illusion of being the only one on the machine. With roots dating back to the 1960s, the development of virtualization has existed on large-scale mainframes and can now be achieved on small-scale microprocessors [13]. In the cloud economy, virtual machines (VMs) are the "stuff that makes stuff."

Virtualization can take many forms and has been implemented in single machine and cluster formats. However, it is the shared characteristics that exploit the hardware and software optimizations that open the risk to cloud users. So, while allowing for the efficient use of data-center resources, virtualization gives rise to conflicts, which can be more antagonistic than those arising at a traffic intersection.

2.2) Virtual Machines: The primary means of exploiting the high performance of modern networked servers in multitenant environments is the virtual machine (VM). The ultimate goal of VMs has been to provide the individual user with the appearance of being the sole operator with isolation, native utility, and a guest operating system [14]. This is achieved through several architectural elements.

2.2.1) Central Processor Unit Sharing: Modern central processing units (CPU) are multi-core and multi-threading. This allows not only the distribution of workloads but also the multi-processing/multi-tasking for different operational, functional, and administrative tasks. This capacity can be divided up into space and time between several users with varying degrees of effects on performance [15]. Core level and CPU level sharing are managed by the hypervisor or a virtual machine manager (VMM). Along with branch predictors, pipelining, memory addressing, and other components [16], the CPU can process multiple guest machines concurrently.

2.2.2) Memory Sharing: The memory architecture of modern computers is arranged into multiple levels to provide a high-capacity channel for digital information. This is usually accomplished in three levels, L1 through L3 [17]. This hierarchy provides proximity to the core for data and instructions that can be accessed at a rate on the same order that the CPU can process them. This allows larger capacity storage for multiple users that might be three orders of magnitude slower in data retrieval time [18]. This memory hierarchy has a cached structure which is based on locality principles [19]:

- Temporal Locality: Reuse of specific data within a small-time frame. In other words, if some piece of data or instructions is used by a program, it will likely be used again within a shorter period of time than it would take to retrieve it from a different level of the memory hierarchy.

- Spatial Locality: The use of data objects within storage locations that are close to each other. In this case, it is likely that programmatic and data elements are grouped sufficiently that gathering adjacent information will save time in future calls.

Various methods of prioritizing access affect each level of the cache hierarchy [20]. These are built on a model of set associative addressing [21]. Additional algorithms based on pre-fetching [22] and memory control [23] have been implemented to optimize the access speed and utility of the cache. In keeping with the economic impetus for cloud development, memory optimization will remain a feature in systems due to the large expense of greater capacity closer to the CPU.

2.2.3) Power Sharing: A second power source may be an added feature on some servers. However, it is generally a redundancy measure. The cores and memory of the machine will be fed from the same source, cooled from the same heat sink arrangement, and operated under the same distribution network [24]. Sensing modules in a system monitor and adjust the flow as necessary on some architectures. This protects the system and can, like other features, affect the execution of programs.

2.2.4) Virtual Machine Coordination: The coordinating unit of VMs is the cloud hypervisor. The hypervisor, or virtual machine manager (VMM), coordinates VM placement and life cycle. A primary example of this is Xen [25], although many others exist [26]. The cloud is a multi-tenant environment of hosts, networks, and data storage that requires consideration for security at the coordination level [27]. Although the hypervisor can become an attack target on its own [28], it will not be the primary focus of this study.

2.3 Cloud Threats

Like other systems with multiple clients, the cloud is subject to attacks from hackers and other malicious users. Some of these are the malicious effects introduced by replicating code (trojans, viruses, worms) carried over from legacy systems [29]. Also, the interactive nature of the cloud leaves it vulnerable to directed vectors [30]. These have been studied in great detail and remain topics of considerable industry activity, sometimes with different implications than traditional vectors [31].

The Open Web Application Security Project (OWASP) acts as a recognized advocate for industry security. The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. Periodically, the most significant threats are publicized as a touchstone for the expansion of threat awareness. They are listed in Table 2.1 and are further described in Appendix B.1.

TABLE 2.1
THE OWASP TOP 10 WEB APPLICATION SECURITY RISKS. THE OPEN WEB APPLICATION SECURITY PROJECT (OWASP) PUBLISHES THE GREATEST THREATS TO WEB APPLICATIONS EVERY TWO YEARS. THIS IS THE LIST HERE IS DESCRIBED IN FURTHER DETAIL IN APPENDIX B.1.

Number	Threat
1	Broken Access Control
2	Cryptographic Failures
3	Injection
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable and Outdated Components
7	Identification and Authentication Failures
8	Software and Data Integrity Failures
9	Security Logging and Monitoring Failures
10	Server-Side Request Forgery

2.3.1) The Co-Resident Risk: In addition to traditional attacks, side-channels emerge due to virtual machines sharing a host. Multiple users trading the processing resources introduce these risks. The danger of collocating such virtualization was recognized even as it was being implemented [32, 14]. Various attacks were theorized and determined to be realizable through low-level system operations [33] and atomic timing of the memory [34].

Perhaps the most simple [35] of these is the 'Prime and Probe' attack. Through a series of memory loads and reads, an attacker can infer the region of the last-level cache (LLC) being utilized by the victim. This attack takes place in three steps:

- (1) Prime – the attacker fills one or more cache sets by reading from a specific memory region, ideally one which they have control of and exact knowledge;
- 2) Idle – the attacker waits for a period of time which allows the cache to be used by other tenants, specifically a target which they are wish to determine co-residence;
- (3) Probe – the attacker refills the cache sets by reading from the same memory region.

During the third stage, the attacker times the memory access and load time. Activity in the cache by the target will likely cause eviction of the attacker's cache entries. Compared to unaccessed cache, the read time in the last step will be higher. This will alert the attacker as to which region of memory the victim is using. Such an attack can be used to force memory relocation [11] and leak cryptographic information when incorporated with other attacks [36].

2.3.2) Side-Channel Communication: Through trace and timing attacks, hackers may infer knowledge of the victim through collateral signaling/emissions. Beyond the coarse-grained prime-and-probe attack, higher-resolution data leakage can also take advantage of memory operations. These generally occur in conjunction with other architectural features [37, 38]. Power monitoring [39], electromagnetic attack, acoustic cryptanalysis [40], and data-bus locking [23] enabled denial of service and risk exposure of encryption keys. These occur, in theory or practice, in a variety of methods and are difficult to avert [12].

2.3.3) Advanced and Coordinated Co-Resident Threats: Although it requires specific conditions [36], a potentially devastating attack is flush and reload. In this shared cache attack, ElGamel and AES key can be exposed with effective noise reduction. The attack happens in three steps:

- (1) Flush - The attacker removes data containing instructions located in a memory page;
- (2) Idle – the attacker waits for a period of time which allows the cache to be used by other tenants that they are able to determine co-residence;
- (3) Reload - The attacker times the reload of the same data into the processor.

An attacker can infer from a faster reload that this instruction was executed from the cache by the victim. Such an attack is problematic as it requires cohesion from the entire cache hierarchy and free reign of clflush instruction. System-level instructions are essential to the applicability of these kinds of attacks [33].

2.4 Co-Resident Threat Mitigation

The very characteristic that made the cloud attractive is economic computation, which requires a degree of risk from co-tenancy. Hardware changes that would eliminate the collocation attacks cannot be implemented immediately [41], are costly [42, 43], and difficult to implement. The alternative is to eliminate the co-residence necessary for a side-channel attack [44].

Such solutions must balance with the other objectives, such as task scheduling [45], energy consumption [46], load balancing [47], and consolidation [48]. Identifying and attempting to mitigate a potential attack relies on various stochastic methods and balancing schema that quantify characteristic outcomes associated with benign or hostile behavior [41, 49], environmental opportunity, or identity [50].

2.4.1) Hardware Mitigation: In modern computer systems, user processes are isolated by the operating system and the hardware. Additionally, in a cloud scenario, it is crucial that the hypervisor isolates tenants from other tenants that are co-located on the same physical machine [51]. However, the hypervisor does not protect tenants against the cloud provider supplying the operating system and hardware.

Intel SGX provides a mechanism that addresses this scenario [52]. It aims to protect user-level software from attacks from other processes, the operating system, and even physical attackers. This provides a trusted execution environment for both monolithic and distributed applications.

It was demonstrated in previous research that fine-grained software-based side-channel attacks can form on an SGX enclave targeting co-located enclaves [53]. This was malware running on real SGX hardware, abusing SGX protection features to conceal itself. The attack was significant to this research both in a native environment and across multiple Docker containers. A Prime+Probe cache side-channel attack took place against an RSA enabled collocated SGX enclave. The attack took advantage of a constant-time multiplication primitive working in SGX enclaves, where there are no timers, no large pages, no physical addresses, and no shared memory. Within 5 minutes an automated attack from 11 traces was able to extract the full RSA private key.

2.4.2) Migration Mitigation: As co-tenancy is a requisite condition for co-resident attacks, the simple relocation of a virtual machine to another core or host seems like a ready cure. This is, in essence, the same thought process as the 'air-gap' solution applied to network threats in early internet applications. Taking this mitigation to its logical conclusion, however, overrides the economic advantages that come with having a virtual machine in the first place because an air-gapped solution necessarily requires one of the most expensive components: maintaining personnel.

Migration solutions require a more nuanced approach to be effective in a cloud environment. First, the determination of whether a co-resident threat exists sufficient due to another VM achieving co-residency with a benign user is different from a benign user landing on a system where they must determine the threat of all the users already present [54]. Second, the implications for a live migration are different from the process of stopping and starting, particularly in the attacker attempting to re-acquire the target [44, 55]. In a sense, even the enclave means of security is a migration.

All co-tenancy issues are not created equally. Mere side-channel issues suggest a different information leakage than attempts to steal encryption keys [56, 34]. The determination of whether migration is effective is a strategic decision considering factors as intricate as the attack itself [57].

2.5 Game Theory Application In The Cloud

A cloud data center is a shared and highly dynamic environment. Activity occurring over space and time is constrained by the provider and the limitations of the resources. User agents seek to act in their own best interests, possibly exploitative of the effects of other's actions. Competing interests in the cloud reflect co-dependence modeled in game theory.

2.5.1) Applicable Game Theory: Any applicable game-theory model depends on the aspect of the cloud that is being developed. Models for network contention differ from those for power delivery, which, in turn, differ from security. For instance, power resource sharing has been proposed as a cooperative game [58]. Resource contention, on the other hand, has been modeled as a coalition game [59]. Security has been modeled in the Nash equilibrium game and as a bin-packing problem [48], as well as many others. Each of these brings its own characteristics and degree of applicability.

2.5.2) Challenges to Cloud Security Game Theory Models: Difficulty in obtaining metrics to evaluate co-resident risk, user information, and scoring for game theory applications remains a challenge [60, 61]. A rudimentary Nash equilibrium has been applied where the actions of system-sharing users must be considered for enhanced security. However, probabilities of co-location do not provide expenditures and utilities.

Applying what information is available in cloud computing, modern implementations invoke machine learning and artificial intelligence [62] and intricate models exist to deduce the utility associated with these strategies [63]. Previous works also recognize the idealization of no risk but hue to the defense [64].

It is the quantitative aspect of utility that provides strategic evaluation. Evolutionary models of game-theory exist that evaluate these payoffs differently than the Nash equilibrium, and these will be applied in the second section of this study. However, the extensive form games with multiple moves, mixed strategies, and elimination of dominated strategies are dependent on measurable determinations of success and failure in an economic context. A development of this concept is in Appendix C.

2.6 Serverless Computing and Functions-as-a-Service

The success of cloud computing showed that the collocation of computational power with technical and logistical support allows information technology at an economy of scale. The IaaS, PaaS, and SaaS offerings provide variety and a multi-tier system with gradient pricing. Following this, an ambiguous and noncommittal proposition of the Internet of 'Things' (IoT) was enough to invite outliers into the cloud [65]. 'Cloud' becoming 'fog' [66], further obscuring 'Things' at 'The Edge' [67, 68].

Breaking out of 'Something-as-a-service' (XaaS) [6] categorized the cloud components, expanded to become 'Anything-as-a-service' (XaaS) [69], and was modeled as 'Everything-as-a-service' (XaaS) [70]. Since the boundaries between different types of 'as-a-Service' may be disappearing [71], it was clear that another acronym was not the solution.

A structural change was occurring in the cloud, interfacing with a wider and deeper customer base. These were supported by an IaaS analog called backend-as-a-service (BaaS), although the demarcation is not well defined [72]. A modular and finer-grained paradigm emerged [73], one which would require its own security evaluation [34]. These came to be called serverless computing and functions-as-a-service.

2.7 Serverless Cloud

“Serverless computing refers to the concept of building and running applications that do not require server management. It describes a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and bill needed at the moment.” For providers, the application of serverless has attracted more consumers and increased their competitive advantage [74].

In spite of, or perhaps because of, the large number of offerings, cloud service providers (CSP) found that they had a high degree of under-utilization. By some estimates, 90% of the data-center capacity was idle at any given time [75]. Over-subscription was a contingency for which the providers had to provide idle space because cloud utilization was viewed as a bin-packing problem. Notably, this provided an incentive to scale down to micro-services, which answered the market demand for greater utility per cost [76].

Granularity and specialization simply made economic sense. IaaS was a macroeconomic term suitable for large-scale applications, but a one-size-fits-all model was never the intent of the cloud [77]. Micro-services became a means to decouple monolithic applications into independent components [78]. So, the term ‘serverless’ is somewhat of a misnomer [79] because servers remain an integral part of cloud computing data-centers [80]. Micro-services was meant as a technical term, while serverless was a marketing term [81] [82].

2.8 Serverless Becoming Function-as-a-Service

Serverless shifted requirements to provision memory, configure networking, and update software from the customer to the CSP [83]. Decoupled elements of applications redistributed the lanes of responsibility in the cloud, redefined the operational environment, and reduced bloat[84]. While IaaS, PaaS, and SaaS remain viable, a localized sub-genre evolved in what had already become a competitive economy.

Like many other aspects of cloud computing (See Section 2.5.), the serverless model can be examined with game theory. One such method is to evaluate the evolution of software using the extended form [85]. See Appendix A.1.3. Alternatively, due to the granular nature of the serverless environment, evolution can be modeled in terms of evolutionarily stable strategies that resist elimination through proliferation. A game-theoretic examination of this effect in Appendix F describes the cloud conditions that were fertile for the Function-as-a-Service model.

Function-as-a-Service (FaaS) implements user-defined logic as stateless functions. These functions are written by the client business activity on a serverless run-time platform in an appropriate high-level language. As institution-developed business logic, FaaS is tailored to formats and modularity more specific than those packaged cloud products. The co-resident security of the FaaS model is considered in the remainder of the study.

2.9 Serverless FaaS Virtualization Candidates

Virtual machines provide a single-owner appearance to multiple users' applications that might be running on the same host or even the same core. The VM was expected to remain 'on' and even in an optimized migration scenario, might take in the range of minutes to start. Disencumbering the end user from managing this overhead is the basis for the serverless name, giving user-designed logic its agility. As the capital element of the cloud economy (See Section 1.1.) the proper virtualization method for serverless considers several candidates.

2.9.1) Virtual Machines (VM): Virtual machines of traditional cloud computing (IaaS, PaaS, and SaaS) have secured their place in the data-center. They are able to implement the complete software stack, migrate under the control of a VMM, and take advantage of a vast array of security measures. Virtual machines are a very mature technology that can be applied in any modern computing environment.

The overhead associated with this method of virtualization, however, is too cumbersome for serverless computing. With start times that can extend into several minutes, they are simply not responsive enough. While code that executes in a discrete function execution environment is possibly only a few lines long, the VM is bulky, slow, and non-portable.

They are provisioned from the abstraction of a bare-metal server from which an operating system supports an additional software application when minimization is the objective in serverless. The VM is the provider's method of rationing computing power in the data-center with the standard security methods. However, the serverless customer will not have direct access to this level of abstraction.

2.9.2) Micro VM: Micro-VMs reduce the overhead of a VM by merging the Linux kernel with a reduced hardware emulator [86]. In this method, they customize the Linux kernel and shrink peripheral device support. The micro-vm implements a few of the robust security measures of the complete VM stack. However, this is a result of deliberate trade-offs [87]. The micro-VM occupies the position of being a middle-ground solution, implementing a set of optimizations but requiring specific security enhancements.

2.9.3) Uni-Kernels: Light-weight, narrow utility options exist. A uni-kernel merges the minimal OS components and application binaries with both small memory footprints and fast startup times [88]. Auxiliary components to the uni-kernel require additional overhead, which further limits their applicability in environments where programmers customize logic [89]. Developing a broad security package is anti-ethical to this method of virtualization. A generalized solution can be made fast but lacks robust, definitive security [90].

2.10 Containers

For a large base of the serverless model, an alternate, finer-grained form of virtualization is employed: containers. Rather than hardware virtualization in VMs, micro-services run in an operation system virtualization platform. As a structure running on a shared environment, it is nimble and portable, has low startup latency, and is scalable. Package-dependent granularity permits containers (the serverless run-time environment) to be the economic analog to the virtual machine.

By design, FaaS executes a discrete piece of code that may be only a few lines. When applying high-level languages, such as Python or node.js, the execution time may be in the order of tens of milliseconds. A previously compiled, meaning, assembled, and linked object file is able to run on an appropriate operation system. However, such an arrangement is not characteristic of a serverless environment. Under these circumstances, a balance must exist between a running environment set-up time and the actual clocking of code execution.

Providing a virtualization of the operating system, Linux within this research, the container deviates from the VM hardware abstraction in both its composition and capabilities. Structures in the Linux kernel remove selected interface methods present in the VM model. Where the virtual machine requires the VMM, a kernel-based method of resource management is distributed to containers. Such streamlining results in a lighter-weight implementation but, at the same, time imposes a different security model.

2.10.1) Name Spaces: The container environment is constructed with isolation ingrained into its component threads. Virtualization is achieved by processes that are walled off from other shared system users at the Linux kernel level [91]. A name-space sequesters the processes that form a container run-time environment by blinding them to those not in that name-space.

Although far from perfect [92] or even universal in methodology [89], the ability to uniquely identify and associate the essential functional components of a host resource is the foundation of container security. Originating with the Plan 9 Linux version from Bell Labs, name-spaces were limited to a single isolation measure. In order of release, flag, and functionality, name-spaced resources are in Table 2.2, and descriptions are in Appendix D.1.

2.10.2) Control Groups: In the Linux kernel, control groups (cgroups) manage system resources or a set of tasks or processes. This structure forms a hierarchy that applies to all of the process' children. The cgroup mechanism partitions groups of processes into rationed resources. Child processes also inherit attributes from their parent processes.

TABLE 2.2:
NAME-SPACES. LINUX NAME-SPACES UTILIZED IN CONTAINER FORMATION. THESE ISOLATE RESOURCES BY OBSCURING PROCESSES FROM VISIBILITY TO OTHER USERS ON THE SAME PLATFORM.

Number	Name-space
1	Mount for file system isolation
2	UTS for hostname and domain name isolation
3	IPC for IPC and message queue isolation
4	PID for process ID isolation
5	Network for network resource isolation
6	User for UID/GID isolation
7	Cgroup for control group isolation
8	Time for clock time isolation

Linux containers place resource limits on each container within the name-spaces and prevent a single container from overworking the host [93]. cgroups can also assign parallel resources to each container and gauge their usage. Through file editing (See

Section 2.10.3.), the resources listed in the cgroups can be managed for parsing out to users [91]. cgroups are listed in Table 2.3, and a description is in Appendix D.2.

2.10.3) Images: Containers are developed from portable scripts. A container engine interprets this script to construct multi-level system tools, libraries, and templates with access to necessary dependencies in an image format [94]. Then, the engine creates the container in which the serverless functions run. These generally become active faster than a VM run-time environment, complete a task, and will shut down thereafter.

Containers can run as normal system processes, though features such as overlay file systems can add performance overheads. A hierarchical sandboxing structure results from a parent process spawning child processes that have restricted resource views, for example virtual network devices or remapped root file systems [95].

TABLE 2.3:
CONTROL GROUPS (CGROUPS). CGROUPS MANAGE THE RESOURCE RATIONING TO USERS SHARING THE SAME PLATFORM. THE RESOURCE PROVISIONING ASSUMES A HIERARCHICAL STRUCTURE WHICH PARSES OUT COMPUTING POWER TO SUBORDINATE PROCESSES.

Number	Control Group
1	blkio
2	cpu
3	cpuacct
4	cpuset
5	device
6	freezer
7	hugetlb
8	memory
9	net_cls
10	net_prio
11	ns perf_event
12	pid
13	rdma

2.10.4) Container Scanners: Image scanning for recognized software, configuration, and composition vulnerabilities is the first tier of security for containers. Scan logic is based on the published vulnerabilities in several open-source and market offerings [96]. Images are tested at creation time for any of the published package or dependency vulnerabilities.

However, the system of adopting scanner utilities into the container environment is challenging in practice [94]. CVEs (See below in Subsection 2.10.5.) must span multiple types of containers and serverless platforms [97] and expand to the FaaS environment [98]. The use of scanning tools is an improvement over making no effort to detect vulnerabilities. However, they are at best 65 % accurate in the detection rate, leaving over a third of the vulnerabilities undetected by any tool that can perform integrated static and dynamic scans. [96]

2.10.5) Common Vulnerabilities and Exposures Updates: The Common Vulnerabilities and Exposures (CVE), which has identified threats to computing systems in the thousands, has hundreds of recognized container weaknesses [99]. When vulnerability within a computer system or an architecture that risks exposure of data or code is discovered, a CVE numbering authority (CNA) will list and describe the CVE. While these may apply to the system as a whole, there are entries particular to the FaaS run-time environment.

CVEs are a product of a coalition of computer security organizations managed through the MITRE Corporation, which includes the National Institute of Standards and Technology (NIST) and partnered with the U.S. Cybersecurity and Infrastructure Security

Agency (CISA). The CVE maintains several entries for vulnerabilities and exposures of containers as well as remedies [100] [93]. The availability to implement the CVE within vulnerability scanners provides for an automated security posture, depending on the available source of scanners [101]. However, the scope of the assignment and its consequences present conflicted interests [102].

2.11 Serverless Threats

Serverless computing, a cornerstone element of the Cloud Native Computing Foundation (CNCF), has vulnerabilities that are an industry-level concern. The threats to security in the cloud are cataloged by the industry organization, The Open Worldwide Application Security Project (OWASP). The OWASP Top Ten security threats to the cloud are published biannually, and are considered the standard priorities for vulnerability mitigation. They are listed and described in Appendix B.1.

The serverless environment considers a variation of these vulnerabilities that reorders and adjusts them for serverless. These have a similar composition and are in a different order than the base version (see Table 2.4). A fuller description is in Appendix E.1. Adjustments and cautions made to the serverless run-time environments to address these vulnerabilities have provided a more robust level of protection.

2.11.1) Serverless FaaS Threats: The transition of management and configuration of the serverless platform to the provider and reduction of end-user interaction with the operating system seemingly reduces the risk of vulnerabilities [103]. Minimizing run-time also notionally reduces the attack surface [104]. Experiments have shown that this is not the case [105]. At best, transition to container environments is a security/performance trade-off [106] and, though the attack surface may be reduced, it has been brought closer to the attacker.

TABLE 2.4
THE OWASP TOP TEN INTERPRETATION FOR SERVERLESS. DETAILS AND EXPLANATIONS ARE IN APPENDIX E.1.

Threat
Injection
Broken Authentication Cryptographic Failures
Sensitive Data Exposure
XML External Entities (XXE)
Broken Access Control
Security Misconfiguration
Cross-Site Scripting (XSS)
Insecure Deserialization
Using Components with Known Vulnerabilities
Insufficient Logging and Monitoring

Limitation of privileges is a ready solution for serverless in general and FaaS in particular [2]. However, the centralization of security protocols under the single umbrella of role-based access control (RBAC) and attribute-based access control (ABAC) the risk from a single point of failure [107]. Also, containers cannot be the exclusive method of meeting the threats to serverless environments. Their organic isolation methods are only an effective measure so long as other resource conflicts can be averted [92].

Serverless systems may display the effects of attacks differently than those resulting from common attack vectors and may exhibit results differently than those intended by the attacker [108]. The depth of change bought about by serverless in general and the spectrum of FaaS participants in specific would suggest that security will be increasingly audience driven. These may be corporate, individuals, or even government actors.

2.11.2) Container Co-Resident Threat: The risks that come with container co-residence have long been a concern complementary to various container vulnerabilities:

- Container compromise: compromise by means of illegitimate data access, Man-in-the-Middle (MitM) attacks or by affecting the control flow of instructions executed in other containers.
- Denial of Service: disturb normal operation of the host or other containers
- Privilege escalation: obtain a privilege not originally granted to a container.

If a process inside the container compromises the Linux kernel, the isolation provided by the container mechanism becomes invalid [99]. Even prior to the implementation of a serverless environment, these threats were explored in early versions of OS virtualization. These include [13]:

- FreeBSD Jails
- Linux-VServer
- Solaris Zones
- OpenVZ
- Cells/Cellrox

At the time of these sandboxing methods, the most pertinent to the current standards in Linux containers was LXC. The security of such systems is intricate and nuanced. This applies to the mounting of file systems [13], group IDs [94], and processes. These concerns are present until today.

2.12 Serverless Mitigations

2.12.1) Hardware Security: Containers may implement one of the enclave protections that were applied to VMs. These are protections from the ‘honest but curious’ provider of virtualized systems [109]. The goal is to create a secure container mechanism, even from ‘friendly’ entities.

SGX is a hardware security feature that uses on-chip memory encryption/decryption of cache lines. This is an intermediate process on lines in the EPC written to and fetched from DRAM. Memory modifications and rollbacks are detected in the enclave to enhance memory integrity. SGX stores each thread’s enclave execution state in a 4 KB thread control structure (TCS) to support multi-threaded execution inside enclaves. The host OS, however, must include a Linux SGX driver and, to boost performance, a SCONE kernel module. SCONE does not support the system call fork [95].

Before making a system call, an enclave thread must copy memory-based arguments and leave the enclave [95]. Software guard extensions (SGX) protects the confidentiality and integrity of a Linux process’ memory, and code, and external file and network I/O from unauthorized and potentially privileged attackers. Memory pages belonging to an enclave reside in the enclave page cache (EPC), which cannot be accessed by code outside of the enclave.

Additional CPU protection mechanisms also prevent attacks on the Linux kernel, for example, Supervisor Mode Execution Prevention [99]. These are intricate protection methods that require configuration at or above the level necessary to manage systems that serverless was meant to reduce.

2.12.2) Linux Security Modules: Various additive components can be integrated into the Linux kernel to fortify the connections between the host components. These are supplemental to name-space, cgroup, and privilege resource isolation and loadable as an modular component of the kernel. These are generally designed around process-limiting logic:

- SELinx, for security enhanced Linux, stands between those processes which use root process enablers to illegitimately access objects outside of the kernel. SELinux is a central part of Centos, Red Hat, and Fedora distributions. This mandatory access control (MAC) policy setting applies to the applications, processes, and files in a container.
- AppArmor, or application armor, is integrated into Debian/Ubuntu as opposed to SELinux. While SELinux applies as the strong point around files, AppArmor imposes constraints on file paths [96]. This methodology contributes to simpler implementation of security policies and the ability to analyze the application during run-time [110]. These alternatives are allowed by the enforcement and complain settings in AppArmor.

SELinux and AppArmor are MAC mechanisms adopted by containers [99]. This is an industry term defined by the National Institute of Standards (NIST) SP 800-192:

> “A means of restricting access to system resources based on the sensitivity (as represented by a label) of the information contained in the system resource and the formal authorization (i.e., clearance) of users to access information of such sensitivity.”

Additional optional protections include the module MemGuard, which can limit the CPU access to the memory through Linux memory bandwidth management to prevent a DoS attack on the memory. [96]

2.12.3) Secure Computing Mode (SecComp): In addition to the MAC modules above, discretionary access control (DAC) implements transferable management tools. These are applicable to the containers on the basis of users or groups. In Docker, this occurs when SecComp denies the available system calls for a container through a SecComp profile. [96]

SecComp constrains the system calls, which a process can invoke by either filtering them to the kernel from the container [99] or allowing only read, write, and exit in strict mode [111]. These process-limiting structures are built into the Berkeley packet filter (BPF), constraining the ability of any user function to affect the kernel.

2.12.4) Capabilities: Capabilities are a list of privileges that can be enabled or disabled for a process. Like the SELinx, they limit a root-enabled processes to the minimum permissions required for it to perform its function. A component of POSIX, capabilities provides a mechanism for partitioning traditional superuser privileges and assigning them to particular processes. [112] Capabilities divided superuser privileges into 38 distinct tracks representing permission to process a given kernel resources.[99]

2.13 Serverless and FaaS Platforms

Whether a version of micro-services is implemented for commercial purposes or for hobbyist projects, the choice of serverless venues is voluminous. The development of serverless platforms presents challenges [71] and requires its own set of terms [113].

However, a well tread path of addressing these challenges [83] has garnered a structural refinement.

2.13.1) Commercial Platforms: Amazon Lambda is generally credited with being the genesis of FaaS computing [81].[1] But there are multiple CSP's currently offering FaaS environments:

- AWS Lambda
- Microsoft Azure
- Google Cloud Functions (GCF)
- IBM Cloud Functions (Open Whisk)

Each of these CSP offerings has stylistic and functional features. They can generally be expected to have an event triggering model [115], some means of limiting execution time [116] , and possibly a means to specify computing power [117]. In all cases, the billing is in short term granularity [74].

Users share platforms with other users simultaneously. For instance, one FaaS client may be utilizing 50% of a system's CPU resources, while two others divide up the remaining 50%. Each may be running the same functions or different functions in their computational space. However, the means that most commercial users employ to ration this space is generally not published.

[1] Note: The first reference to serverless computing found by this research was for a public start up in 2010 [114]. The platform utilized the Amazon cloud as a back-end. No further documentation of this provider was available, although the article describes a Functions-as-a-Service platform. Amazon Lambda has the character of industry standard naming, and holds to proprietary methods. Various models of how FaaS environments arose are in Appendix F.

2.13.2) Open Source Platforms: Amazon Web Services holds the majority of the FaaS commercial market, but several open source FaaS platforms exist. To varying degree these fit the same characteristics of the commercial variants. At least one, Open Whisk, is an open sourced variation on a commercial product, in this case from IBM. These are suitable for internal company networks and for academic study but are limited by necessary supporting infrastructure for widespread use. So, while they may have the essential components of commercial systems (See Table 2.5.) each has unique features [118].

TABLE 2.5
OPEN SOURCE CHARACTERISTICS.

Platform	Orchestrator	Scaling Model
OpenFaaS[2]	Docker Swarm, Kubernetes	Prometheus
Kubeless[3]	Kubernetes	Native
Open Whisk[4]	None Required	Kafka
Fission	Kubernetes	Keda
Nuclio	Kubernetes	Native

2.13.3) Orchestration Considerations: Linux containers are the isolation mechanism in many multi-tenant environments running FaaS applications. They provide performance and are readily adaptable. The deployment, or process of running 2 container from an image, management, scaling, and destruction of containers is accomplished by a container orchestrator. Orchestration is an essential element of sequentially activating functions in separate containers to form FaaS application tapestries.

[2] Community Version
[3] FaaS Framework compliant
[4] O/S Variant

Serving as an industry standard, Docker is prominent for the packaging of the containers and Kubernetes for their deployment [95]. While alternative orchestration software exists, such as Docker Swarm, the vast majority of serverless platforms use Kubernetes. See Table 2.5.

Security is also an inherent concern in the orchestrator engineering [106]. Spe3ific threat models for orchestration have been explored [119] and their place in the container management hierarchy has been cataloged [111]. However, the vulnerability use cases center on the containers themselves [93]. This study retains the 'black box' characteristic of orchestrators due to the significant features that containers, access control mechanisms, and the Linux environment provide.

2.13.4) Testing Platform Evaluation:

Testing platforms were selected from three candidates discussed in Chapter 2.15, Table 2.5.

The preliminary evaluation included:

- OpenFaaS
- Open Whisk
- vHive[5]

Each was evaluated for its suitability as a testing platform. OpenFaaS was rejected due to inconsistency between its open-source and commercial variants as well as

[5] vHive was considered as an alternative due to its suitability to academic study [120]. This platform was not a member of the original evaluation. Its inclusion here recognizes its implementation of the base platform: Kubernetes.

challenges identified by cohort testers [121]. The Open Whisk platform required an overhead in configuration and re-install, which was prohibitive of its adoption. vHive, as well as the other candidate platforms, highlighted the prevalence of the Kubernetes cluster as a component of the majority of open-source platforms.

2.14 Serverless Applications

The serverless model has been adopted for a wide range of applications and for various purposes. Cost savings, reduction of idle time, and appropriateness for event-driven programs comprise a large subset of these transitions. In some cases, a basis for the transition cannot be identified [122]. The most broadly associated benefit of using serverless computing is the promise of scalability.

At the same time, long-running applications are less suitable for the framework than bursty loads. Automated attempts to deconstruct monolithic applications have inconsistent results [123]. Trust in the serverless model suffers from a lack of control in performance [124], double billing [125], and lock-in [71].

Scientific computing and simulations [126] [127] have shown particular promise in serverless platforms. The current architecture and programming models have been sought after to adapt to high-performance computing (HPC) [128], industrial applications [129], and artificial intelligence [130]. Given the ubiquitous interest in the platform, security has become a natural concern.

Transient function execution places the demand for short startup overhead in juxtaposition to deliberately quick execution. Accordingly, scaling back robust security barriers to achieve high availability presents an inherent dilemma. The lack of state

[131], provider-mandated timeouts [132], and platform trigger availability [133] factor into the selection of virtualization methods appropriate for FaaS.

2.15 The FaaS Co-Resident Threat Model

Several threat models in the FaaS environment have been discussed in the literature. These may extend only to the conceptual descriptions [134] with largely abstract proposals of vulnerability, while other studies examine very specific cases in the program flow [135]. Direct links to the vulnerabilities proposed by the OWASP Serverless Top Ten (see Appendix E.1.), e.g. server-side forgery and insufficient logging, along with mass parallelism, are particular to the FaaS environment [122]. Still, the large selection of serverless platforms, data management methods, and provider architectures has frustrated a generally applicable study [136].

A general study of memory threats may be possible in serverless environments, just as in traditional VM-based clouds; speeding up access to remote data and enabling faster data sharing between functions provides a solution for local in-memory caching. The primary impetus for FaaS popularity is that managing such tasks no longer falls to users [137]. However, caches are not entirely transparent to users, either because providers require explicit provisioning or because they provide a separate API for users to access the cache.

An alternative is that co-resident attacks can be considered in their collective forms by taking advantage of FaaS parallelism. The distribution of workloads in the FaaS environment skews the possibility of repeated instances of functions running on a single host. Targeting functions with particular attributes becomes advantageous, where less than 20% of functions are responsible for 80 % of invocations [80]. This study exploited this characteristic and considers the effects of function logic, data passing, and susceptibility to shared infrastructure during application execution [98].

CHAPTER 3

METHODOLOGY

3.1 Co-Resident Game Theory Boundaries

A competitor's viability may rely entirely on the cloud, which suggests a survival of the fittest paradigm. Given the game's risks, rules, and alternatives to players, characteristics of utility and rationality could be assessed for zero-sum payoffs and strategies allowing play. This allowed determination of measurements at the boundaries comparable to a predator-prey model and a game of "keep-away" assessed with the baseline models for strategy development.

3.1.1) Game Theory Boundaries: The zero-sum (predation), zero-trust model, and inter-quartile range-minimal usage (IQR-MU) optimization were compared. The boundary application permitted model behavior at the edges of what allowed a game to be played. See Table 3.1 Attackers and victims were assigned co-resident attack metrics from a review of the current requirements for a successful attack. The simulation was allowed to run until mitigations required additional support. Other metrics in the L-V model were kept consistent.

3.1.2) Test Conditions: In the absence of reliable scoring (trustworthy measures of payoffs) for the users of the cloud, measures were found in simulations aligned with games at the boundaries of the three characteristic features of a game:

- Predator-Prey (Zero-Sum)
- Single Objective (Zero-Strategy)

- Zero-Trust (Zero-Player)

TABLE 3.1
GAME THEORY BOUNDARY CONDITIONS. THE BOUNDARY CONDITIONS ARE VALID ONLY AS LONG AS THE PARTICULAR COMPONENT OF GAME THEORY IS ABLE TO ADHERE TO THE OTHER TWO COMPONENTS. AN INSTANCE OF A GAME WITH A UTILITY MEASURE INSIDE OF THE BOUNDARY IS EXPRESSED WITH THE DASHED LINE.

Component	Boundary Condition	Explanation
Players	A zero-trust relationship.	Irrationality: In this case no information exchange is possible due to meeting minimum time for channel formation. Migration of virtual machines is omni-present in the entire data-center and no subsequent co-location is possible. The only common knowledge is ignorance awareness.
Strategies	A zero-balance objective.	Non-Strategic: Complete concern for fastest possible and resource preservation. There are no concerns for interactive considerations.
Payoffs	A zero-sum game.	In order for one to succeed, another must fail. A closed system without community expansion.

These models are at the boundaries of conditions that allow the use of game theory methods. Essential elements that allow the hacker and victims to have payoffs, behave rationally, and have strategies will form the game space. An evaluation of risks for the individual will be rated in terms of the system health through measurements simulated over a 3.5-day period where no distinct advantage is recognizable to either type of user from the outside. The method will only consider the defining characteristics at the boundaries (see Table 3.2).

TABLE 3.2
GAME THEORY CLOUD SECURITY CHARACTERISTICS. THE SECURITY GAME CHARACTERISTICS THAT ARE IMPLEMENTED ON THE SIMULATED CLOUD PLATFORM.

Characteristic	Boundary Behavior
Rational players	At the boundary of playing a dominated strategy in zero-trust
Strategies	At the boundary of no competing interests in IQR-MU optimization
Payoffs	At the boundary of zero-sum in the predator-prey model

A baseline model will be used to develop metrics that do not consider security but rather focus on optimizing system performance through VM migration. This optimized performance model will form one leg of the game space at the strategy boundary. This boundary will intersect with the zero-trust model, where any VM may be benign or malicious, unilaterally eliminating the possibility of predation. This forms the boundary of player rationality. A model exhibiting the L-V predation characteristic will be developed, implementing a zero-sum game at the boundary of payoffs (see Fig. 3.1).

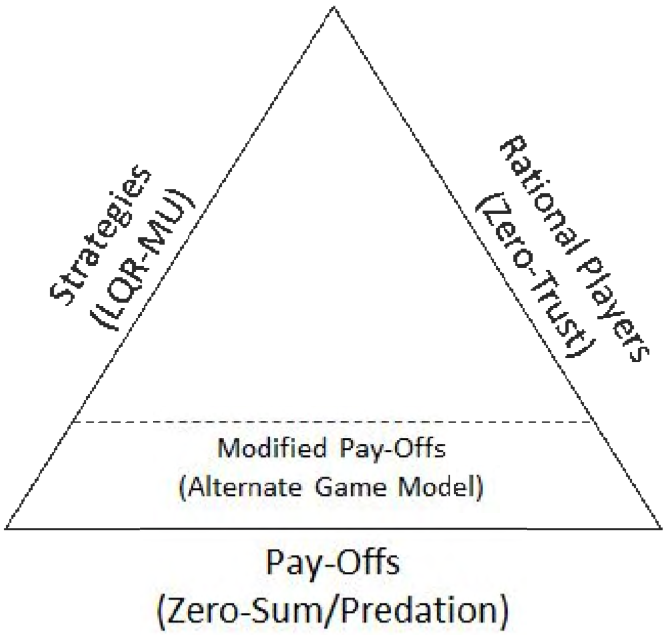


Fig. 3.1: Graphical representation of the boundaries of game-space. The essential elements of rational players, payoffs, and strategies are constructed to the limits of environment. Correlating characteristics for the data-center VM placement method is in parenthesis. Modified Payoffs represent a change in security strategy that moves inside the game-space, parallel to the Lotka-Voltarra model, and alteration to the security environment.

3.2 Building Game Theory Scores In Simulation

The game theory security model is evaluated at the boundary of the three game characteristics. These were introduced as a Lotka-Voltarra (L-V) predator-prey model (zero-sum). See Equations 3.2. Positive parameters a_{11} and a_{22} represent species characteristics and a_{21} and a_{12} model the result of species interaction, which are defined in Table 2.

3.2.1) Ecological and Economic Coupling: Within limitations, economic and ecological model parallels are well recognized [138] and based on the same life cycle [8]. The execution of a business model depends on its productivity, paralleling an animal's foraging characteristics. The expansion and dilation of time within the phases are in accordance with the species/habitat interaction.

A behavioral aligned sequence follows Fig. 3.2. Where virtual machines are a transient construction, they migrate both within or across data-centers under the policies of the provider [72]. This matching of phases in the two domains is the foundation of applying the ecological model to the economic model in the scoring of co-resident cloud game-theory.

3.2.2) Life Cycle Parallels: An animal life cycle generally exhibits four stages with a wide variation in duration and proportion to each other [139]. Birthing, growth, reproduction, and death have a spectrum of characteristic activities. Metabolic [140] and environmental factors [141] influence the foraging and migration that accompanies reproduction [142]. These are characteristics found in a range of habitats and behaviors of both predator [143] and prey animals [144]. These are:

- Birth. The process of natal to adolescent stages.
- Migration. Expansion of domain.
- Foraging. Nesting, cultivation, and sustenance.
- Multiplication. Courting, procreation, gestation.
- Death. Expiration, elimination, ingestion.

The predator-prey relationship within a landscape greatly affects the distribution and activities of these species throughout the life cycle and the habitat [141]. Similarly, none of the above stages is mutually exclusive of any other, given the circumstances. As an economic model, the use of cloud services exhibits parallel phases.

3.2.3) Predator-Prey Model: To implement the zero-sum game, a predator-prey relationship was established [145]. The fundamental predator-prey model is the Lotka-Voltara system of differential equations. The solution of this model depicts a phased bi-species cyclical variation in population. The parameters that govern the model quantify animal characteristics for efficiency in hunting, consumption and digestive efficiency, reproduction, and lifespan. Where such factors are realizable in simulation, it is possible to align the essential parallels across ecological and cloud domains.

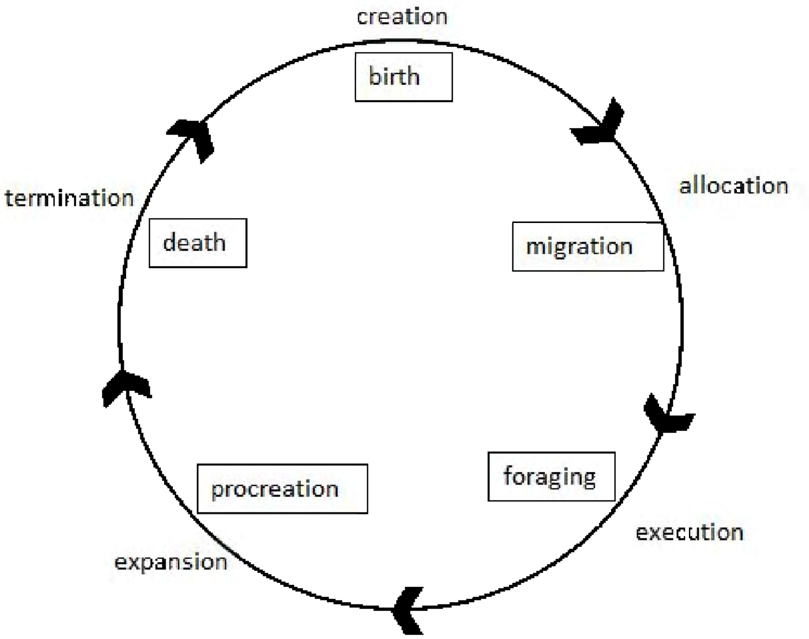


Fig. 3.2: Ecological/economic life cycle parallels. Phases in the economic cycle are matched with the phases in ecological cycles of an organism. These match the virtual machine in cloud data-center with the in the life cycle of species in an appropriate habitat.

The model of interaction between a predator and a prey is an interdependent relationship. It exists throughout a given space and time and is expressed through the Lotka-Voltarra system of differential equations.

$$dx/dt = a_{11}x - a_{12}xy \qquad (3.1)$$

$$dy/dt = -a_{22}y + a_{21}xy \qquad (3.2)$$

Positive parameters a_{11} and a_{22} are factors quantifying species characteristics and a_{21} and a_{12} are species interaction constants, which are defined in Table 3.3. The solution to the Lotka-Voltarra system of differential equations creates a cyclical, out-of-phase population for both the predators and the prey. For a continuous solution, this can be found over any given interval using an appropriate solver (see Fig. 3.3).

TABLE 3.3
LOTKA-VOLTARRA MODEL PARAMETERS. X:PREY POPULATION, Y:PREDATOR POPULATION

L-V Parameter	Definition
a_{11}	prey birth rate
a_{12}	rate of prey death due to predator
a_{22}	predator death rate
a_{21}	rate of predatory birth due to prey

A discrete solution is obtainable from a difference equation solution but with the aid of a limiting constant [146]. Due to the discontinuous solution on a grid space, the result is less smooth and non-cyclical (see Fig. 3.4). Given an adequate coupling of the predator-prey relationship to a cloud-co-resident environment, this is the expected characteristic of the graphical relationship. The projection of the cloud will be onto an analogous grid to a discrete L-V solution. An evaluation of the relationship between the relationship between users of the cloud and the predator and prey requires an equally considered approach.

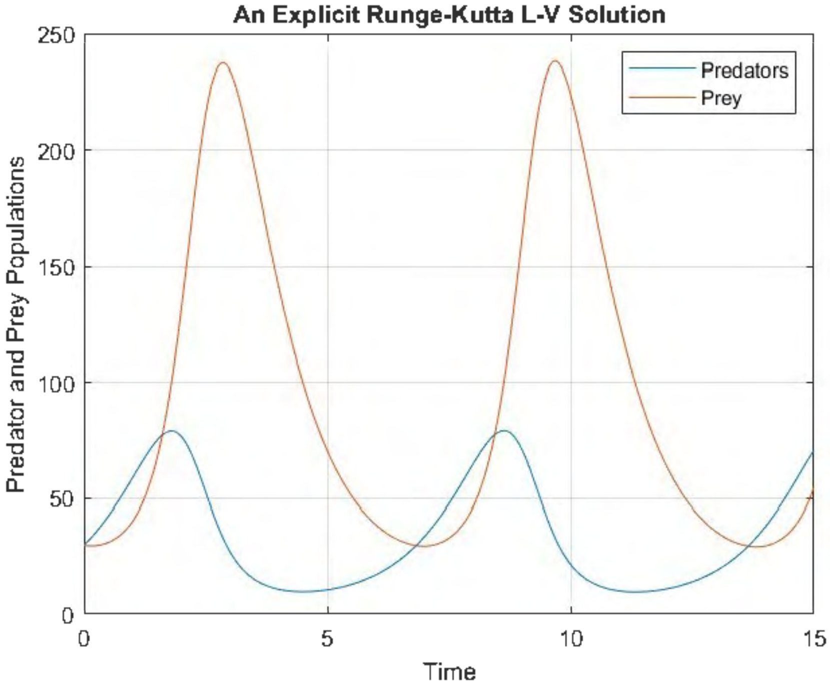


Fig. 3.3: A Runge-Kutta solution to the Lotka-Voltarra differential equations. These biphasic solutions were adopted from an ODE45 MATLAB code.

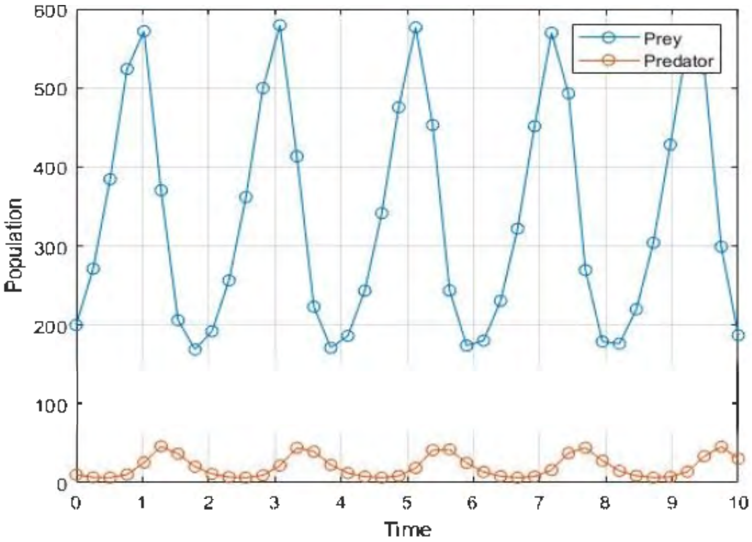


Fig. 3.4: A difference equation Lotka-Voltarra solution

These models demonstrated the characteristic out-of-phase population variation of the LotkaVoltarra differential equations (see Equations 3.2). By adding additional constraints on virtual machine/animal migration, a population fluctuation variance could be developed by bringing the model inside the envelope of the boundaries. Job completion rates could be developed for both models. However, the applicability of the model was limited by the accuracy of the temporal measurements that were available for attacks and defense. Such information is characteristically course-grained, if accurate at all.

3.3 FaaS Testing

Serverless and functions-as-a-service tests to evaluate the threat of information leakage due to multi-user parallelism in a ‘black box’ FaaS environment. Characteristics inherent to a hosted, user-defined function run-time, with representative user visible metrics, were ingrained in a managed analog to commercially available and open-source FaaS environments. Elements that could be reduced, such as stylistic packaging, were eliminated, and sequential elements were sectioned off to maintain data integrity and manage causes to effects.

3.4 Predicting Co-Resident Threats in Functions-as-a-Service

FaaS-appropriate functions executing as a serverless workload may exhibit timing variations resulting from internal tensions between function logic and data manipulation interfacing with the platform, in this case, Kubernetes, management processes. As the parallelism of these functions increases, bin-packing optimizations are tested for the transfer of these perturbations to parallel running functions. Adapting data leakage objectives on a VM [147], resource contention, bus de-confliction, scheduling, and other optimizations can be evaluated indirectly. Orchestration management software will act as the communication medium to evaluate the effects of function, library, and data selection.

3.4.1) Scope and Provisioning: Rather than integrate packaged applications available through commercial and open-source registries (Docker, OpenFaaS, etc.) or applications associated with academic journals, atomic functions were developed as test cases on a FaaS analog. Container-based images were selected for their general applicability, size, and adaptability to image builds. The platform for orchestration was selected for its availability, scalability, and modularity.

3.4.2) Reproducible and Pertinent Objectives: Based on a study that found unreliable repeatability in Kubernetes [148], a discretely measured process of development informed the experiment's format. Encumbering or non-viable courses of action were re-evaluated and adjusted with specific attention to reproducibility. In particular, the previous study suggested that the Kubernetes platform would provide a noisy channel, requiring multiple iterations of testing. Proceeding from Chapter 2, characteristics suitable for the FaaS model inherent to the platform, run-time environment, and selective coding were constrained to those representative of real use cases.

3.5 Experiment Function Selection and Development

While additive modules provide for easy grouping of programming capabilities, a containerized environment may be weighed down by unnecessary libraries. Redundant abilities, such as functions that are available in multiple modules, induce container bloat. On the other hand, optional libraries may present a wider attack surface by leaving gaps in input validation algorithms [94]. The specific language, algorithms, and container environments were considered.

3.5.1) Programming Language and Algorithms: Multiple cross-sections exist to evaluate a proper test language. Studies comparing the compiled versus the interpreted programming environments in FaaS consider the available benchmarking for comparison [149]. Library adoption of FaaS applications and their effects on networking are debated for commercial and open-source offerings [150]. Lower-level language adaptations in the FaaS context are further evaluated for their security [151].

Where options are available for any given platform, six programming environments were evaluated for the FaaS representative test. Languages that were considered for the test cases are in Table 3.4. The JavaScript language is native to the Open Whisk platform, which is found to be suited for asynchronous calls [152]. Alternatively, Python supports asynchronous calls [153]. While both are suitable for extended applications, this study focuses on the local effects of platform co-residence.

The affinity of some languages to a given platform, while others have little use in FaaS, filtered the Java and JavaScript options. C++ was rejected due to its greater suitability to other attack methods. Studies in FaaS saw Web Assembly rejected for similar reasons to C++ [154]. Of the remaining languages, Rust and Python, Python was selected due to its widespread adoption.

TABLE 3.4
CANDIDATE PROGRAMMING LANGUAGES. PROSPECTIVE LANGUAGES FOR EXPERIMENTAL PURPOSES SUGGESTED BY THE APPLICABLE LITERATURE

Language	FaaS Applicability	Benefits / Costs or Risks	Compiled or Interpreted
C++	Widespread adoption in programming	Powerful addressing methods / risk of crash	Compiled
Java	Rarely implemented	Object oriented. Bulky to adopt	Byte-code compiled, JVM interpreted
JavaScript	Native to Open-Whisk	Web-Enabled API, Asynchronous Capable / Implementation overhead	Interpreted, JIT compilation
Rust	Web native	Niche in audience	Compiled
Web Assembly	Suitable for integrated, FaaS uses, Sandboxed	Portable, Fast, Effective at hardware specific use-cases	Compiled
Python	Adapted in most platforms	Multiple libraries / Difficult to optimize in containers	Interpreted

The Python language codes of the candidate routines (see Subsection 3.5.3.) serve as examples of what may be used in engineering, high-performance computing, and financial applications. Python has a significant knowledge base in the FaaS and serverless environment; it is easily portable, and modularity contributes to the method of risk mitigation proposed in this study. It was selected, in part, for its slim container run-time; however, the ability to reduce container size is limited by multi-stage build techniques.

3.5.2) Algorithm Classes: Classes of algorithms were considered for their application in multiple fields [155, 122] and common usage in larger applications [127]. Common usage may be in the form of modules, remote procedure calls, integrated coding, or sub-routines. These algorithm classes were selected for their utility in applications that are being targeted for future FaaS applications [130, 156] as evaluated by their presence in academic literature. See Table 3.5.

Algorithms were selected under criteria that align with fine-grain computations, functions, and routines found in industrial, technical, or financial markets [126, 155, 157]. Their presence in prior FaaS bench-marking literature, availability as a component in modular routines, and capacity to integrate into the baseline structure of applications were factors in selection. The ability to stress architectural units, platform data structures, and system features were also considered [158].

The final criterion was the presence of alternative sub-routines to accomplish the main task of the algorithms. For example, a random walk may be a niche test, but the routines expressing nodes and edges are utilized in various applications, that is, graph theory, optimization. The process of this development is described in the following sections, Table 3.6, and Appendix G.4.

3.5.3) Candidate Functions: Sixteen candidate Python functions were considered for platform experimentation. See Table 3.6. Basic functions and alternate routines were developed to adjust execution time with a single input parameter. The target execution time for a single execution on a stand-alone host running command line Python3 was in the order of 0.5 seconds. This was measured on a single test platform with time distributions shown in Section 3.7.1.

Each function imports a system module to take the input parameter and a time module to attain the beginning of execution, starting at the input command, and the function finishing time. Functions may contain more than one additional module, but no more than four imported modules. The same analysis for distribution that evaluated the preliminary test (see Section 3.7.) functions was undertaken with results in Appendix G.5.

TABLE 3.5
CANDIDATE ROUTINES AND LITERATURE SUPPORT

Routine	Description	Reference
Matrix Operations	Creation an inverse matrix capable of being multiplied with input matrix to form identity matrix	[159] [160]
Sequences	Finding elements of a limited general term capable of creating a convergent or divergent series	[161]
Sorting	Arrange a vector of values in ascending or descending order	[77]
Floating Point Operations	Calculations for irrational numbers or using irrational numbers and operations to a small epsilon	[158]
Monte Carlo Simulations	Calculation for an approximate solution using probabilistic methods	[162]
Simplex Methods	Optimization routines for multi parameter systems	
Prime Number Operations	Assessing whether a number or set of numbers is prime or developing primes	[86]
Modular Arithmetic	Calculations with a modulus component	[9]
Integration	Determine area under a curve or volume within a shell	[163]

TABLE 3.6
CANDIDATE FUNCTIONS

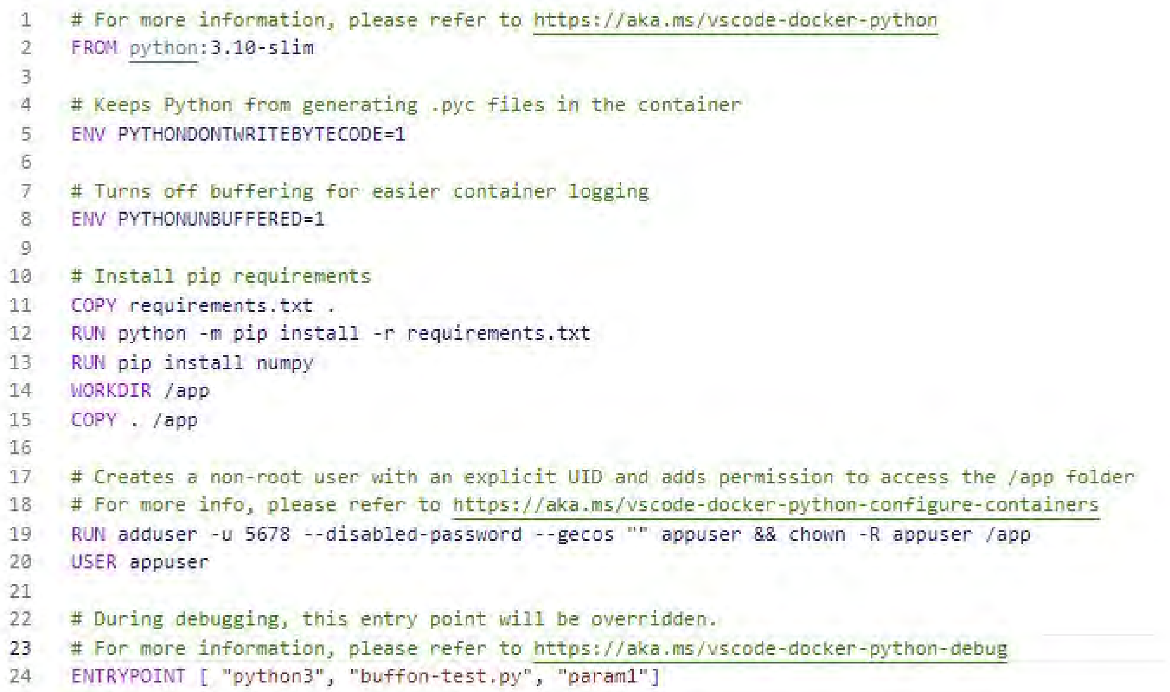
Description	Input Parameter / Target Condition	Utilization Targets / Unique Libraries
Buffon's Needle test using Numpy randomization	n = 2750 / n^{th} iteration estimation of pi.	Randomizing hardware / Numpy.
Buffon's Needle test: LCG, Taylor Series, and recursion	n = 2750 / n^{th} iteration estimation of pi.	Recursion / None.
Buffon's Needle test:mRandom module randomization	n = 2750 / n^{th} iteration estimation of pi.	Randomizing hardware, iterated loops / Random.
Factorial Computation	n = 150 / Factorial of n	Recursion function calls / None
Fibonacci Sequence	n = 25 / Finding the nth number	Function calls / None
Integration under a polynomial	n = [5 - 0] / Find area under an n^{th} order polynomial to epsilon	ALU / None
Large number FP division using math module as convergence check	n = 8 / Iterative division of a number to a quotient less than 2 ^n	Floating point unit, cache / Math
Large number FP division using Numpy module as convergence check	n = 8 / Iterative division of a number to a quotient less than 2 ^n.	Floating point unit / Numpy
Matrix inversion using LU decomposition	n = 147 / n order of a square matrix)	Floating point unit, cache storage / None.
Matrix inversion using Numpy	n = 77 / n = order of a square matrix)	FP Unit / Numpy
Random Walk using Random	n = 41 / n = number of steps	Randomizer / Random
Random Walks using Time	n = 41 / n = number of steps	Timing Hardware / Time
Sorting	n = 16 / n = range of vector item length	ALU, Memory / None
Sorting using itemgetter iteration module	n = 16 / n = range of vector item length	ALU, Memory / Operator
Twin primes calculation	n = 10×10^9 / Twin primes below the value n	ALU / None

3.5.4) Container Builds and Images: A significant factor in container build routines was the imported Python libraries. Baseline import of system and time modules allowed for simple input and output as well as execution time determination, respectively. Additional modules included commands that could be compared with similar functionality in other modules or *ad hoc* algorithms. String handling routines were non-prioritized in favor of numeric modules.

Algorithms that adopted *ad hoc* subroutines were constrained to operate on the same data type, variable number, and configuration as the imported modules. However, the exact numeric values of internal parameters could not be guaranteed, particularly in random tests. Multiple imports from the same modules were avoided.

Adhering to these objectives, the containers for this test were built on the Python:3.10-slim baseline image. Due to the limited utility of staged Python builds, a single stage process was programmed into the Dockerfile. An alternative virtualization software could serve to reduce the Python container size by using multi-stage builds. This was rejected due to the additional layer of abstraction.

A typical build file is in Fig. 3.5. The successful build process creates an image which is saved to a local Docker registry. The Docker image build begins with a download process of any files that are not present on the local registry (see Fig. G.1 in Appendix G.1). An image build process is necessary for each candidate function.



```
1   # For more information, please refer to https://aka.ms/vscode-docker-python
2   FROM python:3.10-slim
3
4   # Keeps Python from generating .pyc files in the container
5   ENV PYTHONDONTWRITEBYTECODE=1
6
7   # Turns off buffering for easier container logging
8   ENV PYTHONUNBUFFERED=1
9
10  # Install pip requirements
11  COPY requirements.txt .
12  RUN python -m pip install -r requirements.txt
13  RUN pip install numpy
14  WORKDIR /app
15  COPY . /app
16
17  # Creates a non-root user with an explicit UID and adds permission to access the /app folder
18  # For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
19  RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
20  USER appuser
21
22  # During debugging, this entry point will be overridden.
23  # For more information, please refer to https://aka.ms/vscode-docker-python-debug
24  ENTRYPOINT [ "python3", "buffon-test.py", "param1"]
```

Fig. 3.5 A typical Dockerfile for container build. A single stage Dockerfile for Buffon's needle test. The build requires the import of a Numpy module (See line 13.) and takes a single input parameter (See line 24.)

3.6. Orchestration and User Interface

Kubernetes was selected for the experiment platform for its versatility, availability, and modularity. Its command line interface was amenable to scripting, and the modularity of its run-time environment was suitable for the scaling characteristic of FaaS. A single host variant, Minikube, was chosen due to its mirror characteristics and its developmental capabilities. No alterations to the default structure were integrated in order to maintain the 'black box' characteristic of the platform.

3.6.1) Cluster Environment: Kubernetes incorporates the Docker container environment as a native run-time and offers performance metrics at the FaaS user level. The container run-time utilized in this project is Docker open-source for Linux distributions. Docker container images are stored on a local repository in the experimental host operating an Ubuntu 22.04 LTS install. Once created and placed in the repository, the host is disconnected from the network to avoid interference from updates.

3.6.2) Pod Structure: The pod is the primary unit of code run-time in the Kubernetes cluster. Although Docker containers do not require the use of pods, in order to have the co-resident risk on a Kubernetes cluster, a level of native isolation had to be adopted. The pods in this experiment have a one-to-one relationship with the containers therein.

3.6.3) Data Extraction: Function execution timing data is extracted by internal commands within the function code. These are generally the time of start and time of finish. As a basic feature, these require the import of the Python time module. The code to do so is demonstrated in the Python codes for this study in Appendix G.4.

The default configuration of Kubernetes is inadequate for extracting scheduling data and container spin-up, start time, and stop time. These require the addition of verbosity flags during cluster activation. The accompanying Figs. are in Appendix G.3. Without the flags being set, timing data is a null value for all event data in the pod structure (see Fig. G.3. By including the «–extraconfiguration=scheduler.v=4» flag on startup, timing data on pod scheduling was available to the micro-second. See Fig. G.4. This metric served to order pods within name-spaces (see Section G.3.1 in Appendix G). The gap between the scheduling of pods and the beginning of function execution is significant.

This proved to be the case in multiple platforms (see Figs. in G.7). This required the addition of an additional intervening metric with the «–extra-configuration=api-server.v=4» flag on startup. This permitted the extraction of the container start up within a pod as well as its shutdown after code execution. See Appendix G.3.2. These data extraction methods were applied as in the following sections.

3.7 Function Preliminary Tests

Preliminary tests of functions were undertaken to determine characteristics of execution times through multiple invocations. These tests were designed to determine aggregate execution time characteristics that indicate whether any temporal phenomena distinguish the stand-alone and shared platform or can be recognized as mutated in a co-resident platform.

Four test functions were selected from those in Section 4.3 and coded as in Appendix G.4. The selections were:

- Buffon’s needle test using a random module for test and Numpy module for calculation.
- Large number floating point division using the math module for convergence testing.
- A Fibonacci sequence calculation algorithm with no imported libraries.
- Matrix inversion using Numpy randomization for matrix creation and matrix inversion.

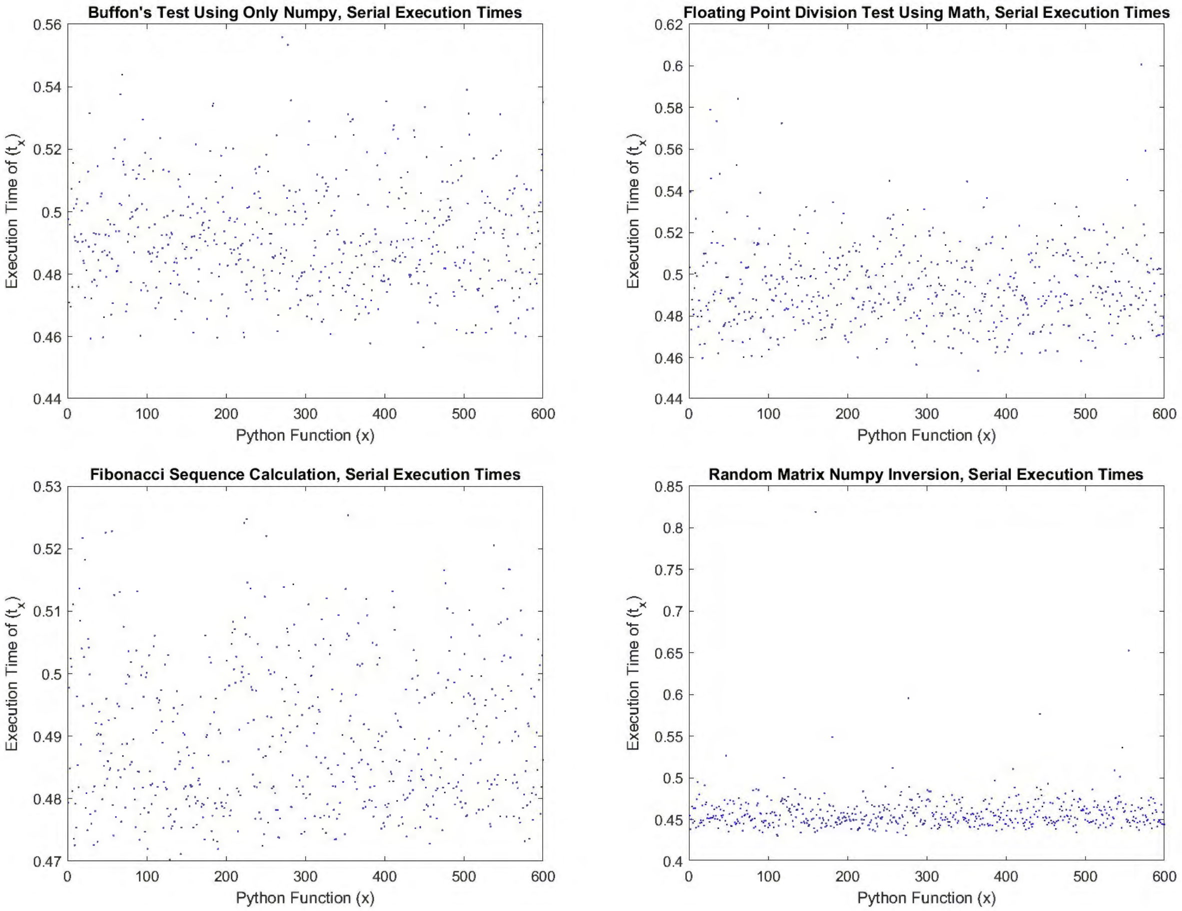
These were considered adequate baselines due to the selected factors. They were also selected to measure the variability of the algorithm and library selection for combinations of functions in later tests. Tests were conducted in both serial executions on a stand-alone environment and in parallel on a Kubernetes Minikube cluster.

3.7.1) Individual Execution Times: Each function adopted for this experiment contained the necessary libraries to receive input and output. An additional module for time allowed for the determination of execution times in seconds from the epoch, 12 midnight on 1 January 1970. A start time was found at the entry point into the code and stored, prior to saving the run-time parameter to an input variable through the system argument. The characteristic code of the algorithm executed and a second time was taken. The difference between the finish and start time was calculated.

Exemplary results of these tests are shown in Fig. 3.6. The code for these tests is in Appendix G.4. For the process of this test, 600 iterations of the function execution were conducted through a Linux shell script.

3.7.2) Parallel Execution Times: A parallel execution of the same test functions was conducted on the selected test platform using the default settings. Four namespaces were established to match the final experiment configuration, each evenly dividing the default resources.

600 individual trial functions matched the number and type of the serial executions in algorithm and input parameters. Ten iterations of 60 functions executed 15 functions in parallel per namespace. The test’s graphic results are in Fig. 3.7.



Buffon's Test Using Only Numpy, Serial Execution Times

Floating Point Division Test Using Math, Serial Execution Times

Fibonacci Sequence Calculation, Serial Execution Times

Random Matrix Numpy Inversion, Serial Execution Times

Fig. 3.6 Execution times of 600 function iterations. Data points are in order of data collection. Top Left: Buffon's Needle Test. Top Right: Floating Point Division. Bottom Left: A Fibonacci Sequence. Bottom Left: Matrix Inversion. Detail of routine methods is in each graph title. Input parameters were selected to center around 0.5 seconds execution time.

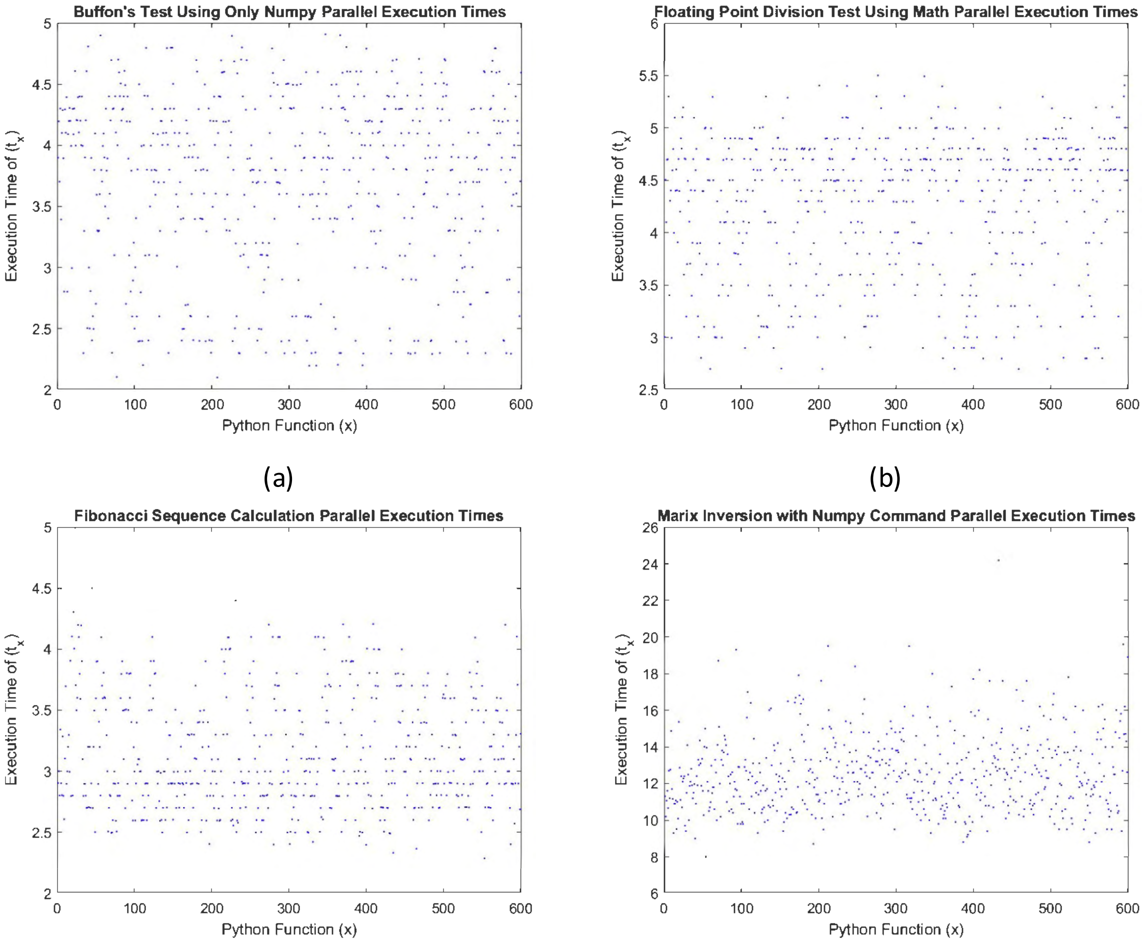
3.7.3) Serial/Parallel Timing Comparison: Characteristic execution times were evaluated to compare the series executions in Section 3.7.1 with parallel execution times in Section 3.7.2. A box-plot comparison indicated that only the routine for a random matrix inversion using Numpy exhibited variation in behavior significant to other algorithms between serial and parallel execution times (see Fig. 3.8).

3.8 Platform Tests

A comparison of run-times for serial functions against run-times of parallel execution on a Kubernetes platform was conducted to determine and compare distributions. This was conducted with the same functions and parameters in previous. Procedures for determination of the effects on the function and Kubernetes pods were evaluated as well as the interrelation of functions and namespaces.

3.8.1) Function and Pod Interaction: Cumulative distribution functions (CDF) for the probability that a randomly selected function executes below a given time is charted for the same timing data in the previous section. Fig. 3.9 graphs CDF the serial executions of the function alone. Fig. 3.10 charts the cumulative distribution for the entire length of the parallel pod execution.

Both serial and parallel function execution distributions show a hysteresis shape, suggesting a normal (Gaussian) distribution. An average, μ, and standard deviation, σ, were determined.



(a)

(b)

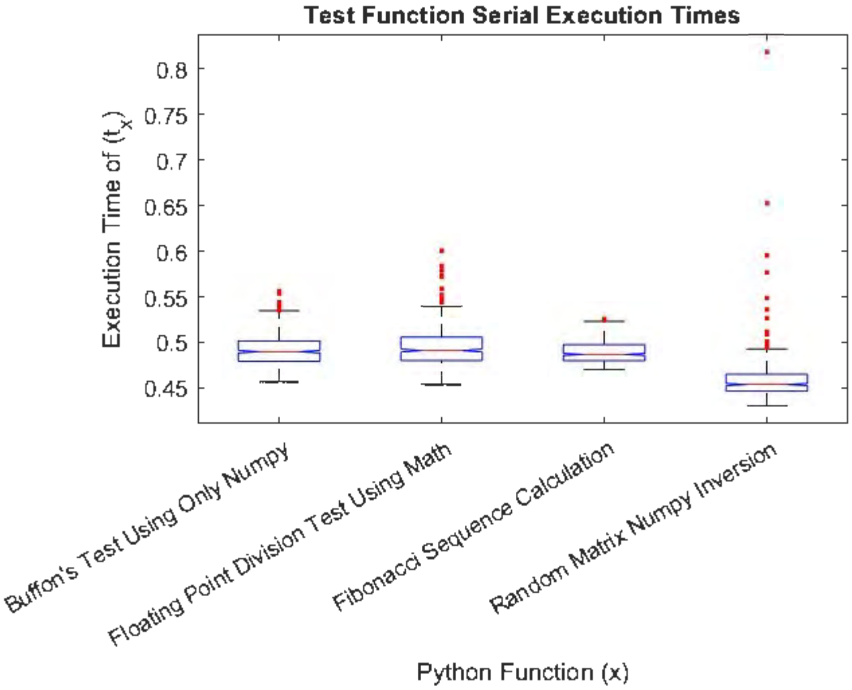
(c)

(d)

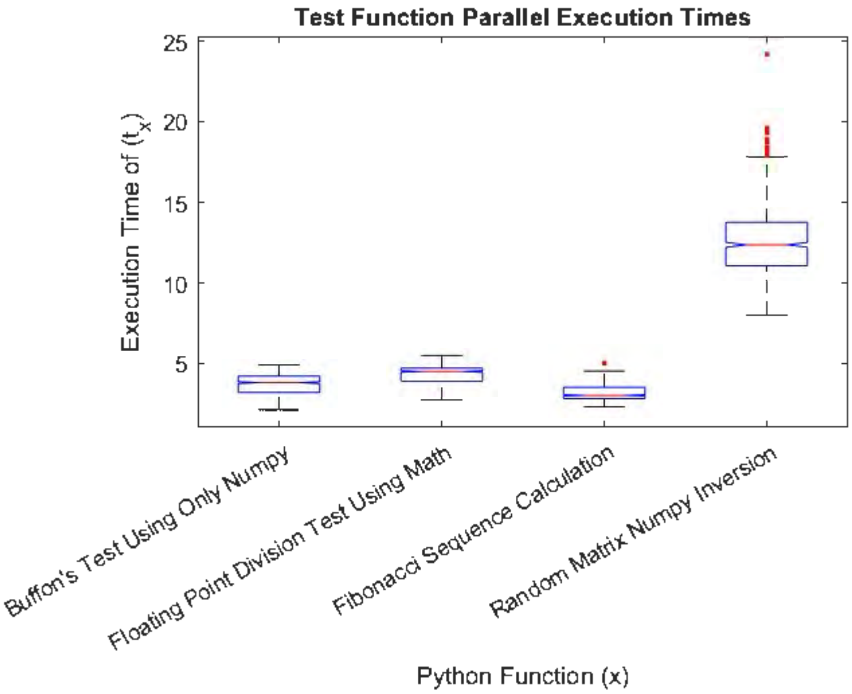
Fig. 3.7 Execution times of 600 function parallel executions. Parallel functions were divided evenly into four name-spaces in 10 tests. Input parameters were the same as in the previous. Representative functions were: (a) Buffon's Needle Test. (b) Floating point division. (c) Fibonacci sequence. (d) Numpy matrix inversion charted for a comparison with the Gaussian distribution

Skewness was also found for each graph. In all cases, a parallel between an idealized normal distribution and the recorded distribution was parallel with the idealized distribution, leading to the recorded CDF. The closely matched serial distributions tend to indicate the presence of noise in the Kubernetes pod formation.

Skewness was less pronounced in the parallel, pod based execution. Both serial and parallel distributions had skewness values close to zero, indicating near Gaussian timing. The lower order of skewness in the parallel Kubernetes environment suggests a greater impact of the central limit theorem and a higher number of independent factors affecting pod execution time.

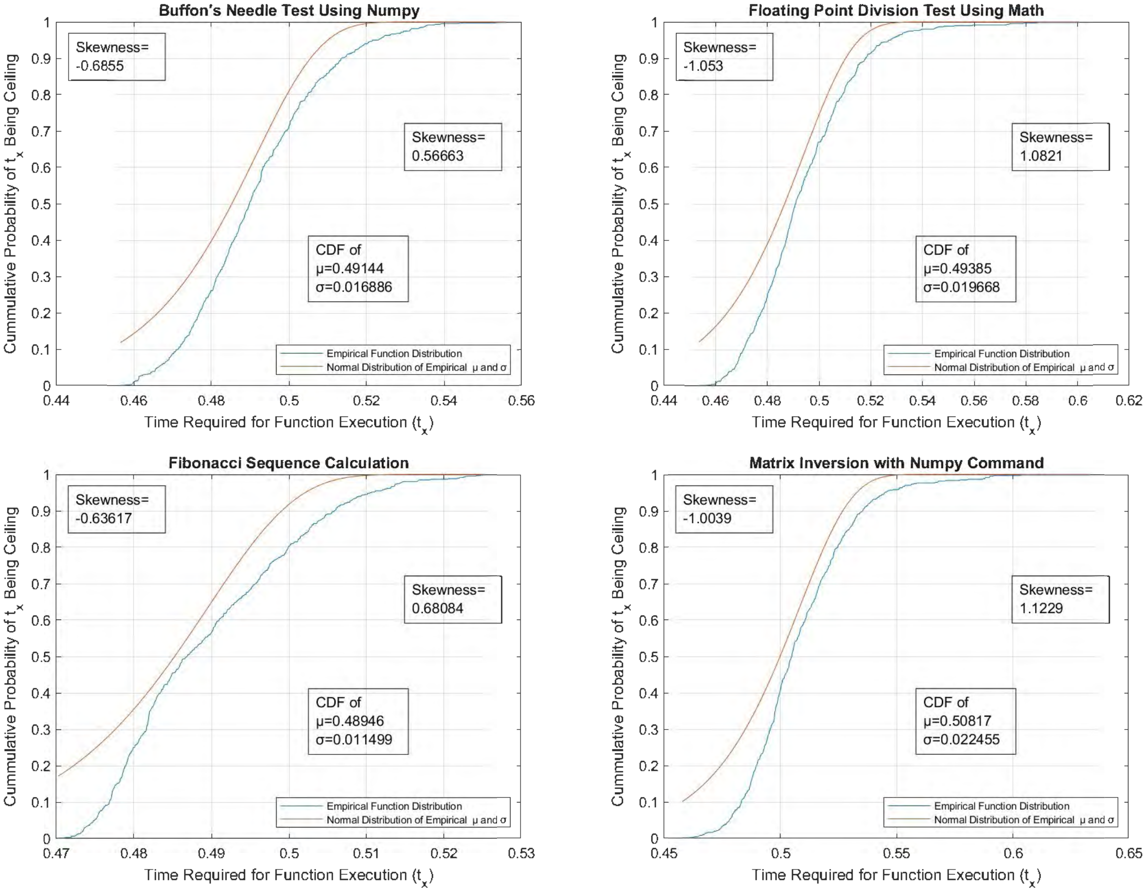


(a) Serial Tests



(b) Parallel Tests

Fig. 3.8 Function box plot comparison. Function data from Figs. 3.6 and 3.7 of the four test Python codes. (a) Serial executions (b) Parallel execution times.



Buffon's Needle Test Using Numpy

Floating Point Division Test Using Math

Fibonacci Sequence Calculation

Matrix Inversion with Numpy Command

Fig. 3.9: CDF's of 600 serial function executions.

3.8.2) Function and Name-Space Interaction: In Section 3.8.1, the function selection was uniform (A homogeneous set.) across time in the serial test and space in the parallelism test. That is, parallelism was introduced across Kubernetes name-spaces. A Kubernetes cluster, the type of platform being used for this test, operates from a hierarchy with multiple name-spaces, each holding multiple pods. The impact of the boundaries between these name-spaces requires closer examination, particularly on how it impacts the running time of non-homogeneous containerized functions.

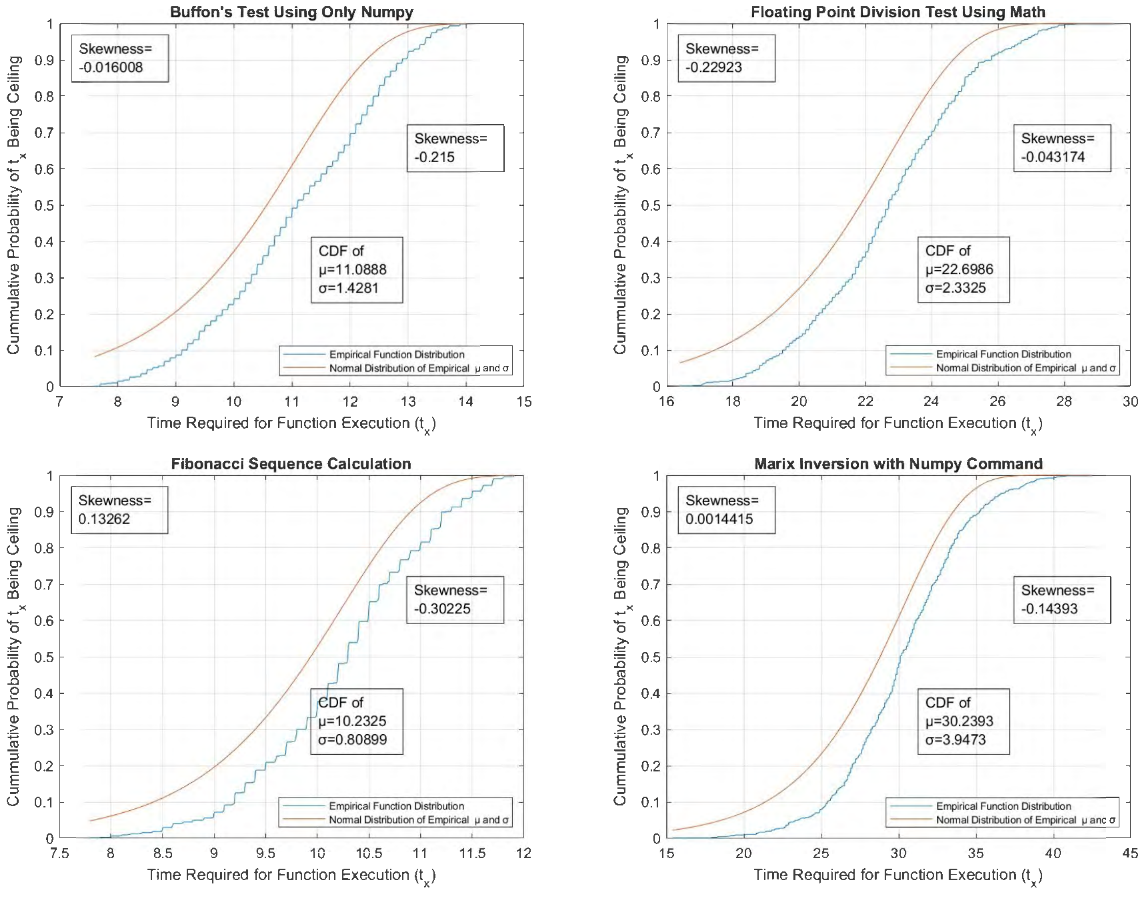


Fig. 3.10. CDF's of parallel executions. Test were from 600 executions of the test functions taken in 10 sets in ascending order.

Name-spaces in the Kubernetes environment act as a boundary that separates the pod-hosting containers of different users. The tests in this section are to determine whether functions running in two namespaces in parallel exhibit variation with alternate functions running in parallel in two separate namespaces. This test is conducted with eight functions running in parallel in each namespace.

The cluster is configured with the default resource allocation. The first test compares the results of Buffon's needle test against a Fibonacci Sequence. The second tests the Buffon’s needle test against Numpy matrix inversion. Results are in Figs. 3.11a and 3.11b.

These tests demonstrated varying start time dispersion of execution within the given namespaces. By arranging the functions by execution start time, either an increasing linear, positive exponential or negative exponential pattern emerges. The execution times of the 3rd and 4th namespace functions averaged less for the later starting functions than those at the beginning. [6] This suggests an availability of resources, even though all pods in each name-space are provided a quota to maximize usage.

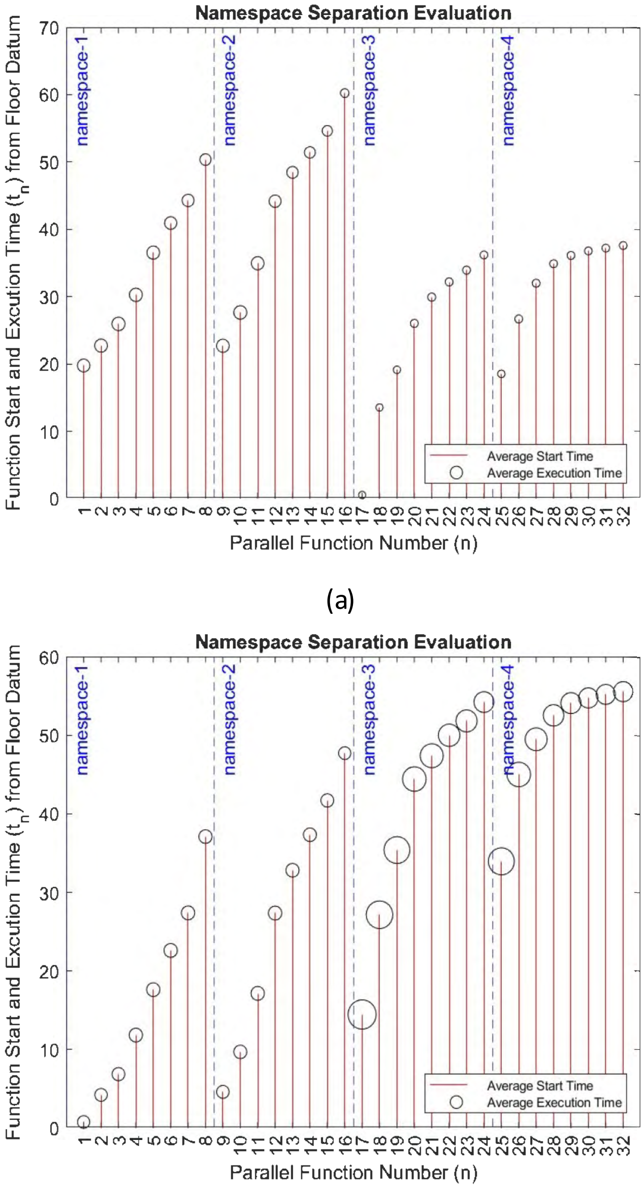
[7] Different name-spaces will be hosted on the same server, which is a necessary condition for this experiment.

3.8.3) Regression Development: A final filtering and extraction of name-spacing effects was conducted on the same data set. This attempts to mitigate the noisy signal of extracting times and produces a comparable value that can be developed across profiles of functions. A fourth-order regression of the ordered execution start times was taken for the extracted data. A MATLAB routine was executed to find the coefficients (a-e) for the minimization of:

[6] The size of the circles are a proportional indicator of the length of function execution. The circles do not indicate the absolute execution time. This is done for the sake of clarity.

[7] In Kubernetes, the pod is the lowest level of run-time deployment. Any pod may have multiple containers, each with a separate operational environment and access identifier. They do not operate independently of the pod. In this experiment, there is a one-to-one correlation between containers and pods. However, they are not interchangeable in terminology or access to performance data.

$$S(a,b,c,d,e) = \sum_{i=1}^{n} (a{x_\mathrm{i}}^4 + b{x_\mathrm{i}}^3 + c{x_\mathrm{i}}^2 + dx_\mathrm{i} + e - y_\mathrm{i})^2 \quad (4.1)$$



(a)

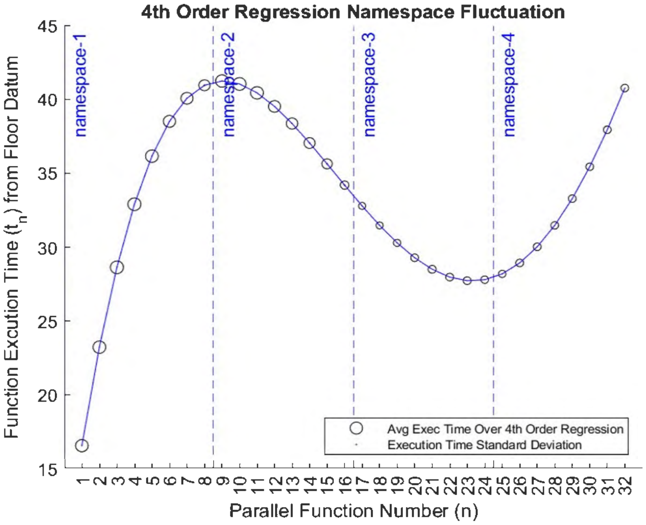
(b)

Fig. 3.11 Name-space separated executions. (a) An execution timing trace for Buffon's needle method in two Kubernetes name-spaces applied with two Fibonacci Sequence in two name-spaces. (b) An execution timing trace for Buffon's needle method in two

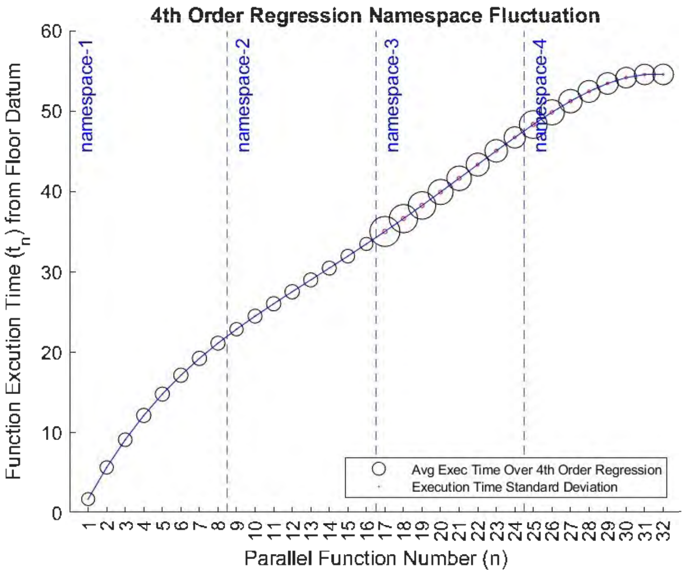
Kubernetes name-spaces and Numpy matrix inversion in two name-spaces. The Kubernetes job for all functions was applied in a single command. The sample size is 100 repetitions of the command. The vertical lines represent the average time to begin execution from a floor (minimum) datum taken from all repetitions. The size of the circle represents the average run-time for the connected function.

The coefficients were stored for the test case iterations to form a profile execution start time data set. Execution start time coefficients were averaged, and a standard deviation was found. These data points were stored for comparison against the profile executions of parallel functions in the four name-spaces. The curve of the regression marked with the mean and standard deviation provided a distinguishing characteristic. See Figs. 3.12a and 3.12b.

A parallel process was conducted for the container start time. Both sets of data were aligned with the data set for function execution times. No set was taken for container stop time, as these were largely uniform.



(a)



(b)

Fig. 3.12 Regression Analysis. Average of 100 function execution times and standard deviation to capture the effects of functions separated across name-spaces, the execution start times were filtered by fourth order regression, one for each namespace captured by order of name-space 14. (a) Regression analysis I. Regression analysis of Buffon's needle test and Fibonacci sequence split two × two between four Kubernetes name-spaces. These execution match those in Fig. 3.11(a). (b) Regression analysis II. Regression analysis of Buffon's needle test and Numpy inversion split two × two between four Kubernetes name-spaces. These execution match those in Fig. 3.11(b). The original order by scheduling was maintained and the normalized timing from the epoch adhered to. The size of the markers represent the length of function execution but are not to scale.

CHAPTER 4

RESULTS

4.1 Simulation Results

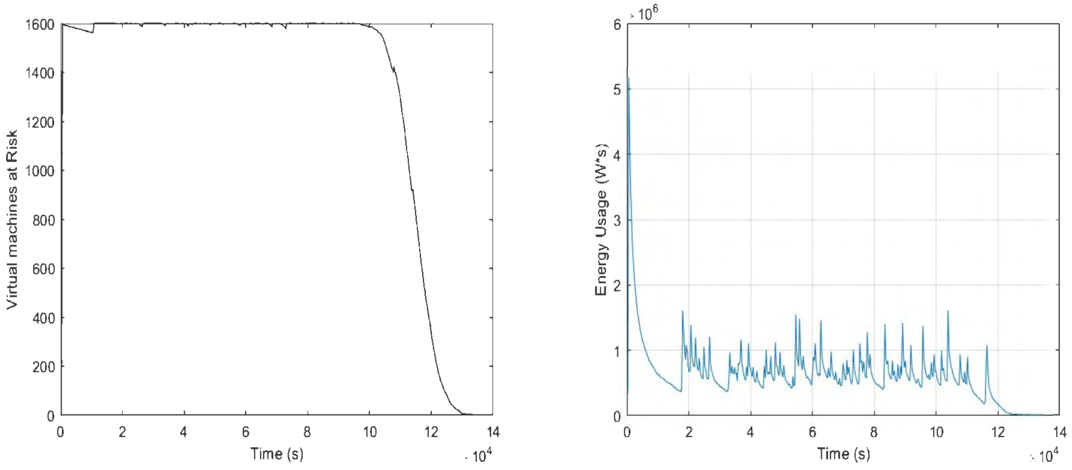
The resource management methods introduced in Paragraph 3.1.2 are simulated with the results shown in Table 4.1. These are outputs at the end of the run. For a fair comparison, all methods are running with the same maximum number of cloudlets available (1600) to load the VMs. Different strategies lead to their own migration rules, which directly influence energy consumption. Additionally, if VMs are assigned to different hosts or after cloudlet completion, a certain number of hosts will be shut down. So, energy consumption, number of migrations, and number of host shutdowns are chosen to illustrate the performance of different methods.

TABLE 4.1
SIMULATION OUTPUT COMPARISON. QUANTITATIVE RESULTS FROM VARIATIONS IN SIMULATION.

	IQR-MU	Zero-Trust	Predation	Altered
Cloudlets	1600	1600	1600	1600
Energy Consumed (MWh)	0.079	1.718	1.725	1.878
Migrations (Million)	0.036	2.048	.988	1.341
Cloudlets Complete	1600	2	247	340
Num. Host Shutdowns	3240	252643	52033	12712

4.1.1) IQR-MU Simulation Results: IQR-MU is chosen from several resource management strategies provided by CloudSim. It serves as a baseline model whose results were extracted from the output methods native to the simulation, and it adopts methods for specific measures of risk. The minimized energy consumption, number of migrations, and number of host shutdowns are at the cost of a maximized co-resident vulnerability. These results showed that the optimization method sporadically lowered the number of co-resident virtual machines (1600). This was a result of removing virtual machines from under-utilized hosts, which at some points in time left a few of them alone on a host. The vast majority were shared among a few hosts (see Fig. 4.1 a).

Redistribution of VMs during the same period resulted in the fluctuation of power as the migration process used by the optimization algorithm re-allocated virtual machines. See Fig. 4.1 b. A steady and fully populated set of hosts was able to complete the workloads in a short time period, as will be shown in the following sections. Though an equal workload, the risk of co-resident attack was almost consistent throughout until the VMs began to shut down for job completion.



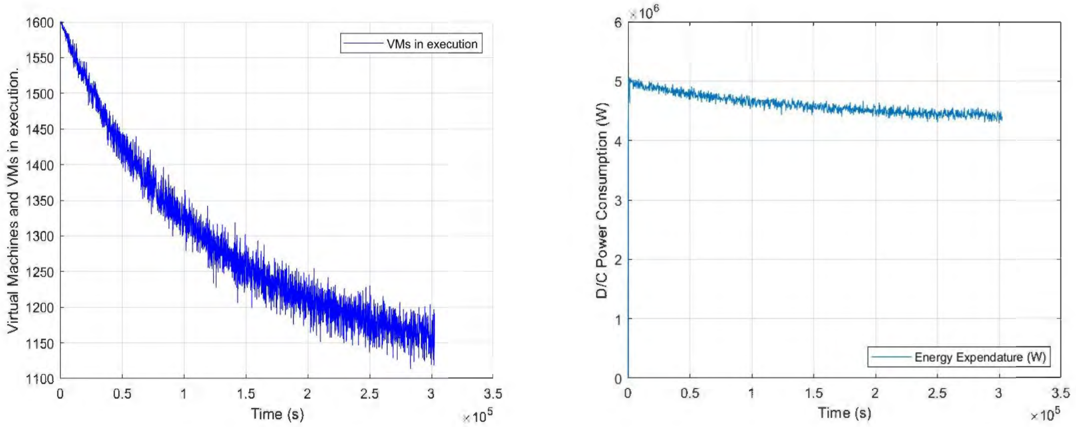
(a)

(b)

Fig. 4.1. IQR-MU results. (a) A the number of virtual machines which where at risk for a coresident attack as a result of being on shared hosts. (b) Power usage during the same time period. Fluctuation resulted from redistribution of VMs in optimization algorithm.

4.1.2) The Zero Trust Model Simulation Results: The zero-trust VM selection algorithm found the longest-running co-resident VM for migration, while the native methods statistical placement policy migrated it to a different host. It has the largest number of migrations and the largest number of hosts involved, but it achieved the elimination of co-resident attacks.

A constant number of VMs remained on the data-center throughout the run-time, with only those in execution drawing power. Cloudlet completion initially allowed the selection of a new workload from a queue of 2000 cloudlets. See Fig. 4.2a. An accompanying trend in power consumption level, as shown in Fig. 4.2b, exhibited a peaked period when VMs picked up queued workload. As the workload was completed and VMs dropped off activity, the number of VMs migrating began a downward trend (see Fig. 4.3).



(a) (b)

Fig. 4.2. Zero trust results. (a) The number of VMs and running cloudlets. Constant VM number greater than cloudlet number indicates inactive VMs on the data-center. (b) Power usage was consistent with the time period workload and host shutdown with inactive VMs

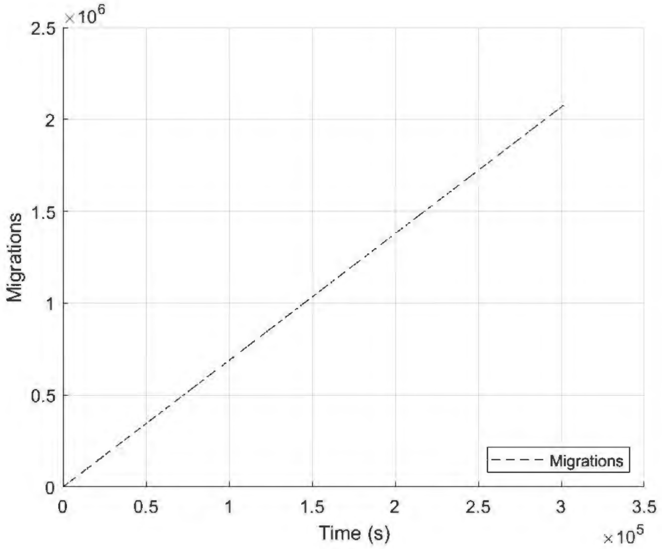
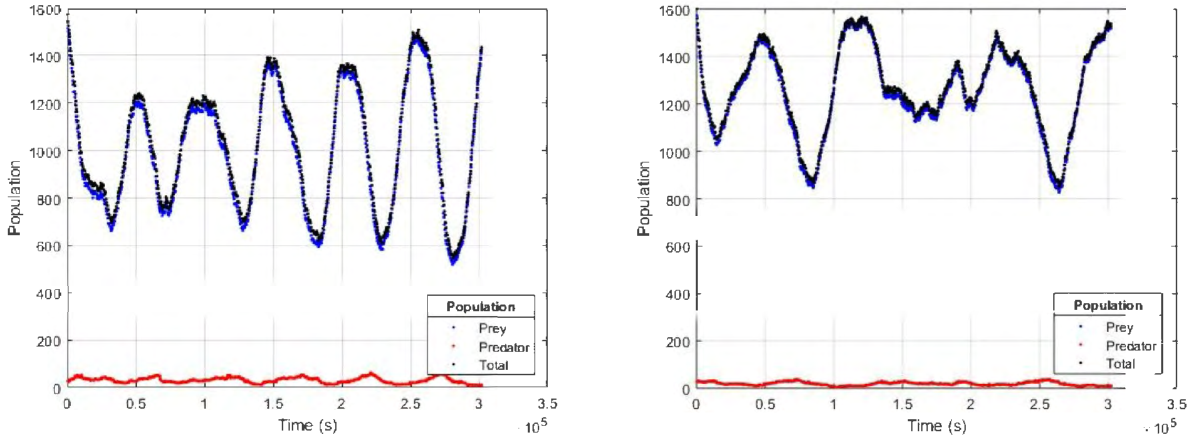


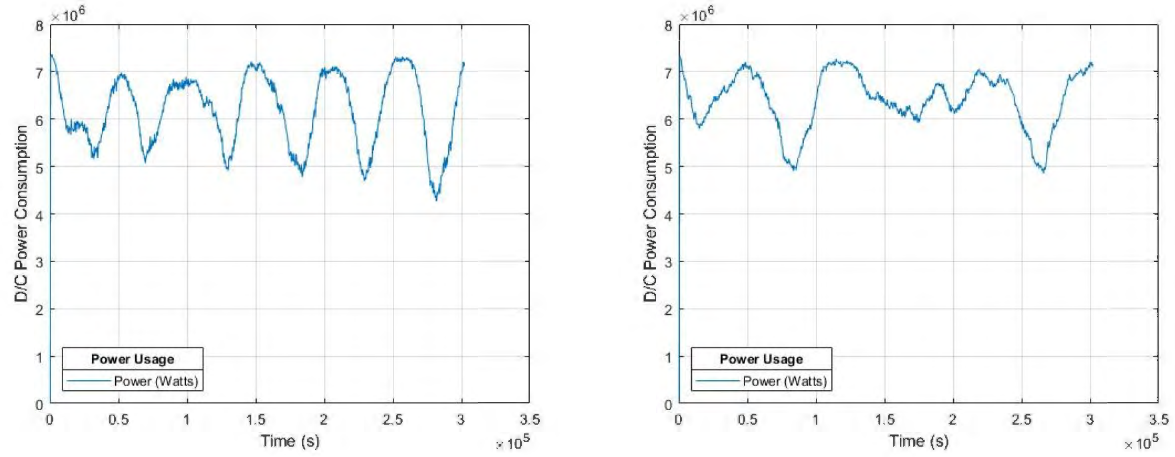
Fig. 4.3. Cloudlet completion migration trends. A downward trend in migrations began when workload assigned to VMs completed a cloudlet

4.1.3) Predation Model Simulation Results: Fig. 4.4a compares the active VMs in the data-center under the predation model. Hacker/victim VMs translate the landscape to the cloud, mirroring foxes/predators and rabbits/prey. In Fig. 4.5a, a high degree of correlation for the time interval exists with the population. An expanded plateau at the peaks was a result of the dispersion of attack locations prior to a precipitous decline in population. The risk of predation, deaths occurring within a time period per number of active VMs, presents as an inverse of the population and power consumption. See Fig. 4.6a. Reproduction rate variations and model outputs are displayed with comparable L-V fluctuation in Appendix 2 and 3.



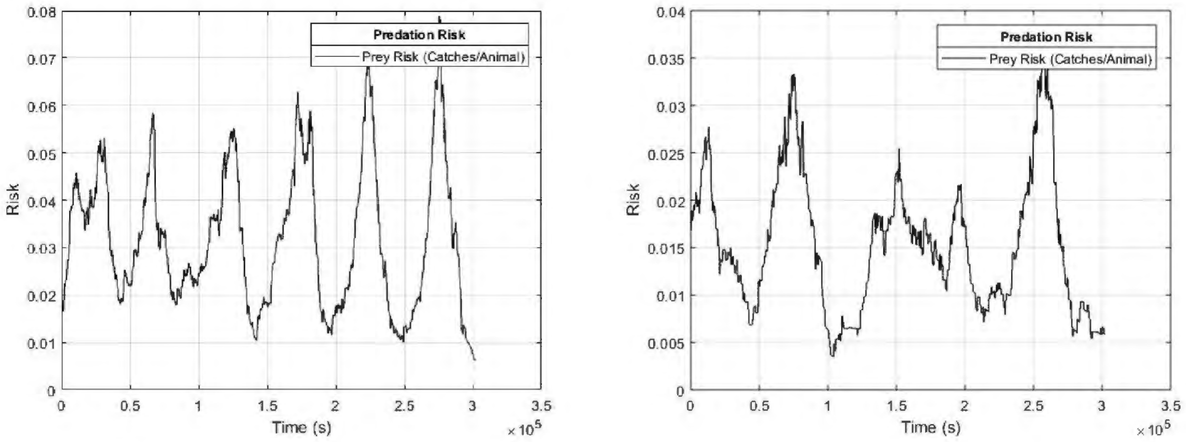
(a) (b)

Fig. 4.4. Population fluctuation of predator and prey within the landscape and data-center. (a) Population of predator and prey exhibit an inverse relationship as predicted by Fig. 3.4 without other controls. (b) Population of predator and prey game space with migration of prey is triggered by locality to VM termination.



(a) (b)

Fig. 4.5. Population/power correlation. Within a data-center of 576 hosts, the fluctuations in power generally correlated with the sum of predator and prey population. (a) Correlating in population varied periodically with a diminished difference between peaks and valleys. (b) Power usage in an altered predations model exhibited a similar trait. The characteristic is explained by host power requiring a 'turn-on' usage.



(a) (b)

Fig. 4.6. Predation risk. The risk of predation within associated time periods within the model. Both (a) and (b) illustrate the risk of becoming a prey within the game space/data-center.

4.1.4) Altered Predation Simulation Results: In the altered predation model, the population variation was more erratic (see Fig. 4.4b). This occurred with long periods of minimal predator population with less peaked raises than the 'laissez faire' model. This is attributable to a widely dispersed group of predators giving the victims nowhere to migrate safely. Fig. 4.5b illustrates power, and Fig. 4.6b illustrates risk. They also form correlated patterns.

4.2 FaaS Data Collection Process

The configuration of the experimental platform reverted to a four-core, 4096-megabyte memory Kubernetes cluster. The verbosity settings of the API server and scheduler were set to provide equivalent data to a FaaS customer maintaining a trusted profile, in accordance with the security discussion in Chapter 3. Platform data were extracted in the format of Fig. 4.7.

A selection of 5 Python functions was taken from the list of candidates based on their imported modules, the potential to stress specific resources in the host, and modularity. They were provided with input parameters tested to execute in the range of 0.5 seconds on a stand-alone platform (see Table 4.2).

TABLE 4.2
TEST FUNCTIONS

#	Image	Description
1	buffon-test_random	A Buffon’s needle test using the Random module
2	lu-decomp	A matrix inversion function using the lowerupper (LU) decomposition method
3	lu-decomp_numpy	A matrix inversion function using the lowerupper (LU) decomposition commands in the Numpy module
4	random-walk	A random walk Monte-Carlo routine
5	random-walk_random	A random walk Monte-Carlo routine using the Random Python module

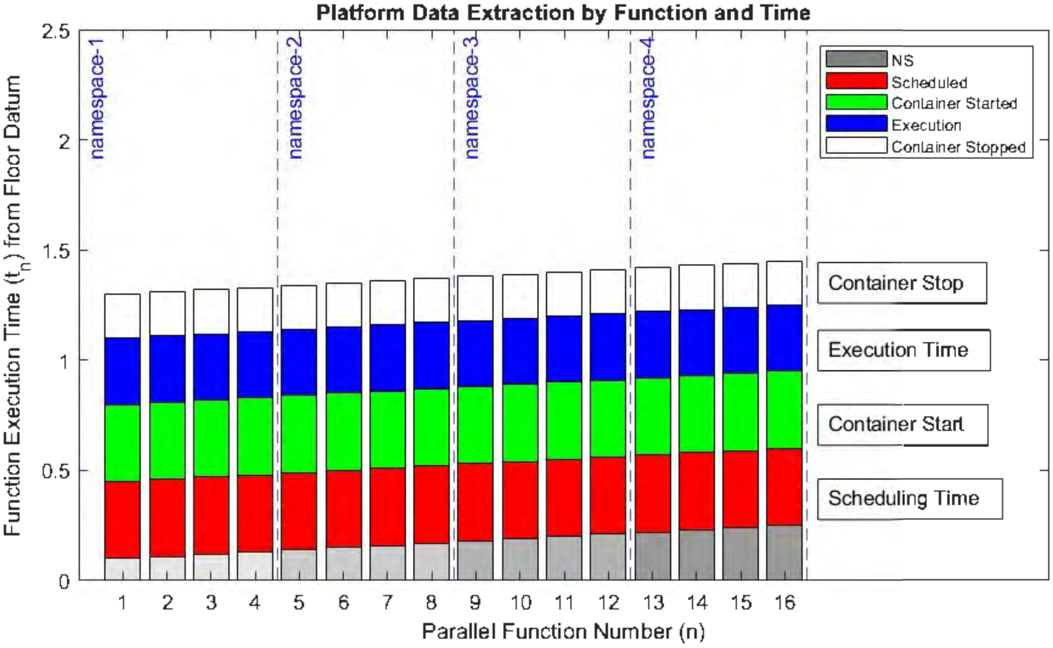


Fig. 4.7: Data Extraction. A representation of data extracted from the Kubernetes platform executing FaaS Functions. Four namespaces separate individual user execution environments. Each environment runs functions in parallel, four in this representation. The first event to occur is the command to begin the process of managing the Kubernetes platform execute user functions, represented in grey, directing that the four parallel environment simultaneously. The scheduling process, represented in red, directs the pods of Kubernetes to start. Container starting processes, in green, prepare the execution run-time. Blue is the individual function execution. White represents the last timed process, the container shutdown. The pod remains on until removed by external commands.

A scripted execution routine cycled through function profiles within a four name-space cluster. Each name-space had an even distribution of resource quotas rationing computation resources. The cluster quota was divided among sixteen parallel running instances of each function in each name-space.

4.2.1) Profile of Name-spaced Functions: The profile of four of the five functions over four name-spaces provided for 256 variations to test on the Kubernetes cluster. Each function was given a number (1-4 in accordance with Table 4.2.) which identified it in the job file which informed the Kubernetes API. For example, a job file executing profile 1122 would execute an LU decomposition routine in Name-space 1 and 2, and LU decomposition using the Numpy module in Name-space 3 and 4.

This selection of functions was chosen to evaluate the impact on the co-resident environment when the same arithmetic operations are being calculated using alternative routines and mixes of Python libraries. While the Buffon’s needle test only shared a randomizing module with random walks, they were also designed to be compared with random-walk that does not share these libraries. The profiles were applied on the cluster in sequence from lowest order to highest order.

Each profile was executed with sufficient repetitions to generate a data set to establish profile characteristics and a test sample set. The interaction of the container and pod run-time with the characteristics of the arithmetic operations, as well as with each other, were extracted as measures of:

- Mean execution time for the functions in the run-time
- Standard Deviation for the execution time for the functions
- Natural log exponent of curve fit of execution times
- Mean of fourth-order regression coefficients of container start times for all iterations of the profile
- Standard Deviation of the fourth order regression coefficients of the above container start times.
- Mean of fourth-order regression coefficients of function execution start times for all iterations of the profile

- Standard Deviation of the fourth-order regression coefficients of the above function execution start times.

These were extracted and saved as a comparison library. The process of developing the comparison library is described in Tables 4.3 through 4.5. This library was held as a representative of the behavior of the Kubernetes cluster executing co-resident functions.

TABLE 4.3
NAME-SPACE DATA PROCESSING AND STORAGE. EACH PROFILE OF FUNCTIONS IS EXECUTED ITERATIVELY TO FORM A DATA SET. FOR EXAMPLE, IF EACH NAME-SPACE IS EXECUTING FUNCTION ‘1’, IT WOULD BE DESIGNATED PROFILE 1111. ON THE OTHER HAND, IF ONE OF EACH OF FOUR FUNCTIONS ‘1’, ‘2’, ‘3’, AND ‘4’ WERE EXECUTING IN NAMESPACE ORDER, ITS PROFILE NUMBER WOULD BE 1234. THIS PROVIDES 4^4 = 256 PROFILES OF FUNCTIONS. FOR EACH OF THE PROFILES, THE AVERAGE, μ, AND STANDARD DEVIATION, Σ, FOR FUNCTION EXECUTION TIME IS RECORDED. THE ORDERED EXECUTION TIMES ARE CURVE FITTED WITH THE EXPONENT OF THE NATURAL LOG, A, WITHIN THE EACH NAMESPACE IS ALSO RECORDED.

	Averaged Function Profile Characteristic Data Set								
Python Code	Average, μ, of namespace (NS) Function Execution Time			STD, σ, of namespace (NS) Function Execution Time			Exponent, α, of natural log curve fit for namespace (NS) Function Execution Time		
Profile	μ of 1st NS Exec Time		μ of 4th NS Exec Time	σ of 1st NS Exec Time		σ of 4th NS Exec Time	α of 1st NS Exec Time		α of 4th NS Exec Time
1111	$\mu_{1111,\ T1}$		$\mu_{1111,\ T4}$	$\sigma_{1111,\ T5}$		$\sigma_{1111,\ T8}$	$\alpha_{1111,\ T9}$		$\alpha_{1111,\ T12}$
1112	$\mu_{1112,\ T1}$		$\mu_{1112,\ T4}$	$\sigma_{1112,\ T5}$		$\sigma_{1112,\ T8}$	$\alpha_{1112,\ T9}$		$\alpha_{1112,\ T12}$
↑↑ 256 Profiles ↓↓									
4444	$\mu_{4444,\ T1}$		$\mu_{4444,\ T4}$	$\sigma_{4444,\ T5}$		$\sigma_{4444,\ T8}$	$\alpha_{4444,\ T9}$		$\alpha_{4444,\ T12}$

The mean for fourth-order regression (see Fig. 3.12.) of container start times, as well as their standard deviations, are found for the many runs. These are aligned with the function execution time mean and standard deviations in Table 4.3 and placed into a tabular form below. A final data set was taken for the function execution start times as in Table 4.5.

4.2.2) Test Set Data Extraction: A second set of data was extracted in the same format at a ratio of one to two. These were processed in the same format and arranged for threat evaluation. These serve as a test set representing information obtained by users attempting to infer co-resident data leakage.

TABLE 4.4
CONTAINER START TIMES DATA PROCESSING AND STORAGE. CONTAINER START TIMES SERVED AS THE SECOND EVALUATION PARAMETER. THESE WERE COLLECTED FOR EACH CONTAINER, ON A ONE TO ONE BASIS, FROM EACH POD. THESE TIMES WHERE EXTRACTED USING THE KUBERNETES «INSPECT» COMMAND AND MAINTAINED THE ORDER OF THE SCHEDULING WITHIN EACH NAMESPACE. THESE TIMES WERE FILTERED THROUGH A FOURTH ORDER REGRESSION PROCESS IN ORDER TO CAPTURE THE EFFECTS OF SEPARATION BY THE FOUR NAMESPACES. THE COEFFICIENTS FOR EACH ORDER OF THE REGRESSION, C4-C0, ARE RECORDED. AN AVERAGE WAS TAKEN ACROSS THE ITERATIONS ALONG WITH THE STANDARD DEVIATION FOR THE AVERAGES. THESE WERE ORDERED IN THE MANNER BELOW.

Averaged /Standard Deviation Function Regression Values						
Python Code	4th Order Regression Means μ of Container Start Times			4th Order Regression STD's σ of Container Start Times		
Profile	μ of 4th Order Coefficients		μ of 0th Order Coefficients	σ of 4th Order Coefficients		σ of 0th Order Coefficients
1111	$\mu_{1111,\ C4}$		$\mu_{1111,\ C0}$	$\sigma_{1111,\ C4}$		$\sigma_{1111,\ C0}$
1112	$\mu_{1112,\ C4}$		$\mu_{1112,\ C0}$	$\sigma_{1112,\ C4}$		$\sigma_{1112,\ C0}$
↑↑ 256 Profiles ↓↓						
4444	$\mu_{4444,\ C4}$		$\mu_{4444,\ C0}$	$\sigma_{4444,\ C4}$		$\sigma_{4444,\ C0}$

TABLE 4.5

EXECUTION START TIMES PROCESSING AND STORAGE. EXECUTION START TIMES SERVED AS THE THIRD EVALUATION PARAMETER. THESE WERE COLLECTED FOR EACH FUNCTION. THESE TIMES WHERE EXTRACTED USING THE KUBERNETES «LOGS» COMMAND AND MAINTAINED THE ORDER OF THE SCHEDULING WITHIN EACH NAMESPACE. THESE TIMES WERE FILTERED THROUGH A FOURTH ORDER REGRESSION PROCESS IN ORDER TO CAPTURE THE EFFECTS OF SEPARATION BY THE FOUR NAMESPACES. THE COEFFICIENTS FOR EACH ORDER OF THE REGRESSION, K4-K0, ARE RECORDED. AN AVERAGE WAS TAKEN ACROSS THE ITERATIONS ALONG WITH THE STANDARD DEVIATION FOR THE AVERAGES. THESE WERE ORDERED IN THE MANNER BELOW.

Averaged /Standard Deviation Function Regression Values						
Python Code	4th Order Regression Means μ of Container Start Times			4th Order Regression STD's σ of Container Start Times		
Profile	μ of 4th Order Coefficients		μ of 0th Order Term	σ of 4th Order Coefficients		σ of 0th Order Term
1111	$\mu_{1111,\ K4}$		$\mu_{1111,\ K0}$	$\sigma_{1111,\ K4}$		$\sigma_{1111,\ K0}$
1112	$\mu_{1112,\ K4}$		$\mu_{1112,\ K0}$	$\sigma_{1112,\ K4}$		$\sigma_{1112,\ K0}$
↑↑ 256 Profiles ↓↓						
4444	$\mu_{4444,\ K4}$		$\mu_{4444,\ K0}$	$\sigma_{4444,\ K4}$		$\sigma_{4444,\ K0}$

TABLE 4.6
TEST CASE FUNCTION EXECUTION TIMES

Averaged Function Profile Characteristic Data Set									
Python Code	Average, μ, of namespace (ns) Function Execution Time			STD, σ, of namespace (ns) Function Execution Time			Exponent, α, of natural log curve fit for namespace (ns) Function Execution Time		
Profile	μ of 1st ns Exec Time		μ of 4th ns Exec Time	σ of 1st ns Exec Time		σ of 4th ns Exec Time	α of 1st ns Exec Time		α of 4th ns Exec Time
????	$\mu_{????, ns1}$		$\mu_{????, ns4}$	$\sigma_{????, ns1}$		$\sigma_{????, ns4}$	$\alpha_{????, ns1}$		$\alpha_{????, ns4}$
↑↑ 18/256 Profiles ↓↓									
????	$\mu_{????, ns1}$		$\mu_{????, ns4}$	$\sigma_{????, ns1}$		$\sigma_{????, ns4}$	$\alpha_{????, ns1}$		$\alpha_{????, ns4}$

TABLE 4.7
TEST CASE CONTAINER START TIMES

Averaged /Standard Deviation Function Regression Values						
Python Code	4th Order Regression Means μ of Container Start Times			4th Order Regression STD's σ of Container Start Times		
Profile	μ of 4th Order Coefficients		μ of 0th Order Coefficients	σ of 4th Order Coefficients		σ of 0th Order Coefficients
????	$\mu_{1111, c4}$		$\mu_{????, c0}$	$\sigma_{????, c4}$		$\sigma_{????, c0}$
↑↑ 18/256 Profiles ↓↓						
????	$\mu_{????, c4}$		$\mu_{????, c0}$	$\sigma_{????, c4}$		$\sigma_{????, c0}$

4.2.3) Experiment Comparison Process: The evaluation of a co-resident threat treats timing data of a multi-function parallel FaaS application as an ensemble signal. The component timing data of the ensemble is considered sequentially by determining at which interval a group of test case profiles has a similarity score calculated to be the characteristic of the candidates that the attacker wishes to discover about the target user.

A collection of these functions matching the activity of the target user will allow the attacker to infer that the target is using. The evaluation framework follows a Bayesian model to match the normal distribution and adopts a normalizing feature to compare the characteristic parameters that they are able to extract from the co-resident platform. A product aggregate of these factors establishes a density of profiles from which to infer similarity.

TABLE 4.8
TEST CASE EXECUTION START TIMES

Averaged /Standard Deviation Function Regression Values						
Python Code	4th Order Regression Means μ of Container Start Times			4th Order Regression STD's σ of Container Start Times		
Profile	μ of 4th Order Coefficients		μ of 0th Order Term	σ of 4th Order Coefficients		σ of 0th Order Term
????	μ????, κ4		μ????, κ0	σ????, κ4		σ????, κ0
↑↑ 18/256 Profiles ↓↓						
????	μ????, κ4		μ????, κ0	σ????, κ4		σ????, κ0

Because there is a wide range of possible profiles that the target can adopt, and the event of a matching profile matches only one of the many orchestrated functions sequentially executing in the co-resident environment, a method was necessary to test results in mass. A product aggregate of these factors establishes a density of profiles from which to infer data leakage. A high-density plot of matching profiles will stand out and be considered accurate if low in the range of possible differences.

From the N^n possible profiles from the N number of equations on n name-spaces a random sample set was taken to test for profile identification leakage. The number of test profiles was calculated to keep the probability of a repeated profile below 50% by Equation 4.1. For a test set of 256, or 4^4, profiles, this was 18 test cases, each selected at random from the test set.

$$.5 = (1-1/N)\times(1-2/N)\times \ldots \times(1-\nu/N) \qquad (4.1)$$

A normalized scalar similarity score for the test case was determined for each entry in the profile library by equation 4.2. These were ordered and aligned with the profile number. An absolute value and logarithmic (base 10) value were taken for each similarity score (see Fig. 4.8). A similarity score close to zero provided an indication that the test profile revealed information about the co-resident workload executing on the platform. An accuracy score was calculated through the indexed separation from the test profile to the profile that had a similarity score closest to zero. See the caption of Fig. 4.8.

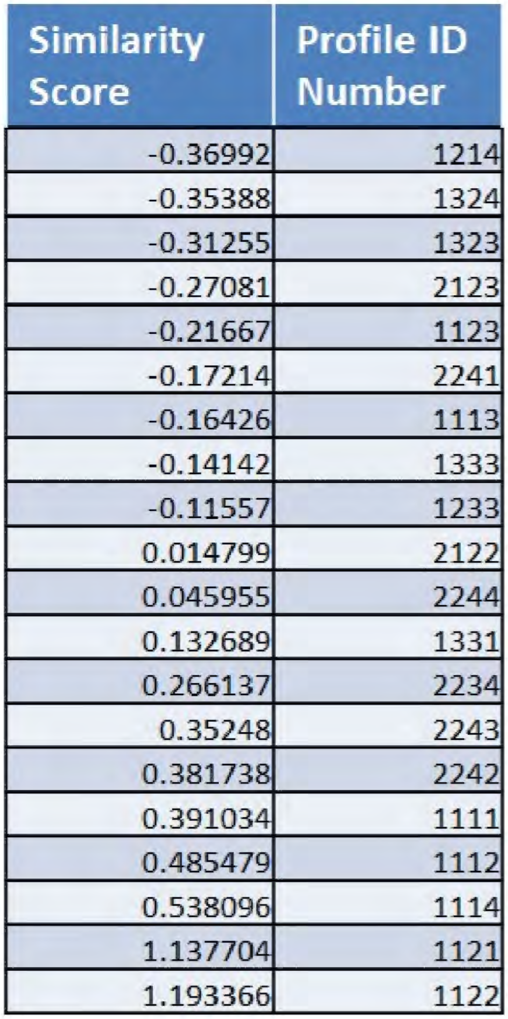
$$\Delta_i = \prod_{n=1}^{12} \frac{\tau_{test}}{T_{i,n}} \times \prod_{n=1}^{10} \frac{c_{test}}{C_{i,n}} \times \prod_{n=1}^{10} \frac{k_{test}}{K_{i,n}} \qquad (4.2)$$

A measurement of how far from 0 the sample value was counted from the results of the normalization process. This value was stored as a ratio and rendered as above or below. The process was repeated for thirty random profile selections.

TABLE 4.9
NOTATION FOR TERMS IN SIMILARITY ALGORITHM. TERMS ARE EXTRACTED FROM TABLES 4.3, 4.4 , AND 4.5 IN HIERARCHICAL ORDER. SUBSCRIPTS FOLLOW A ROW, COLUMN FORMAT WHERE PRESENT.

Term Name	Symbol	Description.
Similarity score	Δ_1	Term expressing the calculated closeness of similarity of the tested profile to the candidate profile i.
Test Case Execution Value	$\tau_{test,n}$	Term from tested function profile index from n = 1-12
Library Candidate Execution Time Value	$T_{i, n}$	Term from candidate function, profile i = 1-256, index from n = 1-12
Test Case Container Start Value	$c_{test,n}$	Term from tested function profile index from n = 1-8
Library Candidate Value	$C_{i, n}$	Term from candidate function, profile i = 1-256, index from n = 1-8
Test Case Execution Start Value	κ_j	Term from tested function profile index from j = 1-8
Library Candidate Value	$K_{i, n}$	Term from candidate function, profile i = 1-256, index from n = 1-8

4.2.4) Comparison Results: Of the 18 samples tested, 4 had a similarity index that brought them outside of 2 orders of magnitude separate from the normal. That is, the result of the normalization and scalar process provided the sample test profile with a score of between 0.01 and 100. The numerical distance of ordered calculation results was, on average, 10 percent of the range of candidate profiles. That is, 90% of the possible profiles could be eliminated from consideration.



Similarity Score	Profile ID Number
-0.36992	1214
-0.35388	1324
-0.31255	1323
-0.27081	2123
-0.21667	1123
-0.17214	2241
-0.16426	1113
-0.14142	1333
-0.11557	1233
0.014799	2122
0.045955	2244
0.132689	1331
0.266137	2234
0.35248	2243
0.381738	2242
0.391034	1111
0.485479	1112
0.538096	1114
1.137704	1121
1.193366	1122

Fig. 4.8: Similarity Scores. Similarity Scores. An ordered list of the similarly scores represented as logarithms (base 10) of the absolute values of the Equations 4.2. The list is about the ideal normalized result, 10 on each side, of zero. The randomly selected profile for this test was 1333, with an evaluated similarity score of -0.14142. This profiles separation from the ideal 0 was two places; 1 for profile 1233 and 2 to achieve zero. A reflective value was also found for the inverse of the logarithmic value, which added three places of separation from the ideal result: 1 for profile 2122, 2 for profile 2244, and three for profile 1331. This provided a error score of 2 + 3 = 5, or 5/256 = .0195. This is about a 2 percent degree of error.

A second test was conducted for the remainder of the five functions using the random Python library. Of the 625 profiles developed, only 339 had a complete set of data to evaluate. A minor extraction error occurred in data-set faulted the evaluation routine. The process to remedy the change would only be applicable to one of the final test sets. Of the twenty-five samples tested, 4 had a similarity index that brought them outside of 2 orders of magnitude separate from the normal. That is, the result of the normalization and scalar process provided the sample test profile with a score of between 0.01 and 100. The numerical distance of ordered calculation results was, on average, 20% of the range of candidate profiles. That is, 80% of the possible profiles could be eliminated from consideration.

4.3 Parallelism Expansion

In an attempt to align container creation with the timing of resident code execution, the namespace parallelism doubled. This was accomplished through finer-grained parsing and division of resources. Two attempts were made to increase the parallelism of the platform. The first was to increase the number of functions running in a name-space four-fold from 16 to 24. This leads to lag beyond the ability to reasonably obtain results.

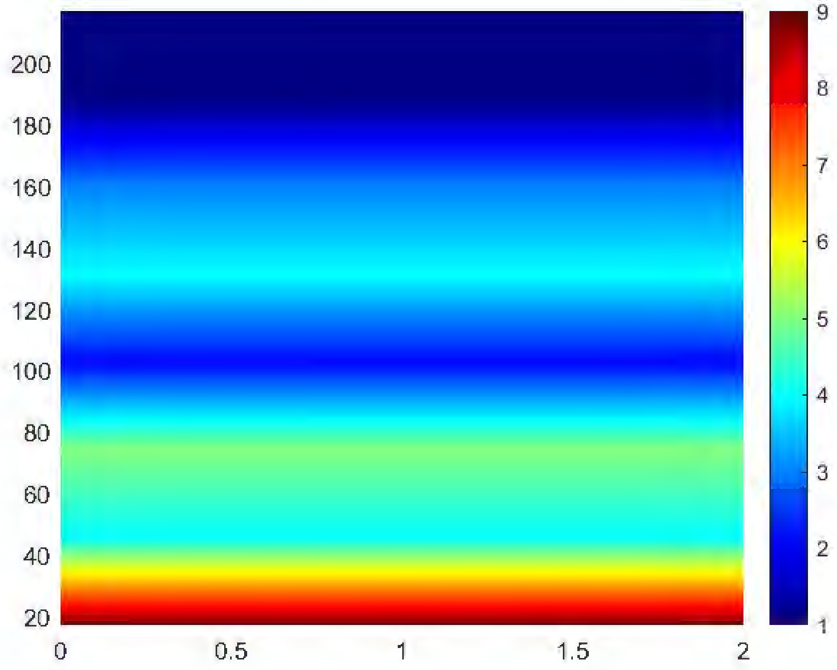


Fig. 4.9: Test case I correlation density. A graphical depiction of offset from success in finding the true fingerprinted function profile. In 18 tests, the algorithm to predict the exact function profile was executed. The characteristic parameters of the test function were compared to the fingerprints of all 256 function profiles. Each of these was given a similarity score, and ordered. The predicted identity of the fingerprinted profile was compared given the score. These were arranged from lowest to highest. Densities of these offsets are depicted in this Fig.. Of the 18 tests 9 were in the 10 percent range and 3 more within the 20 % range. These were represented as red at the bottom of the Fig.

The second test was conducted with parallelism of 32 functions per name-space. The result of the high degree of parallelism was an uneven dispersal of pod scheduling, container start, and function execution (see Fig. 4.11). Execution time notably increased, and container start time increased dramatically. A trend arose that demonstrated latency in container scheduling for the last name-space in the configuring YAML file, mitigating the advantage of having that information provided in co-resident tests. Finally, the API server began to lose the ability to extract information about halfway through the test, and a complete comparison could not be undertaken.

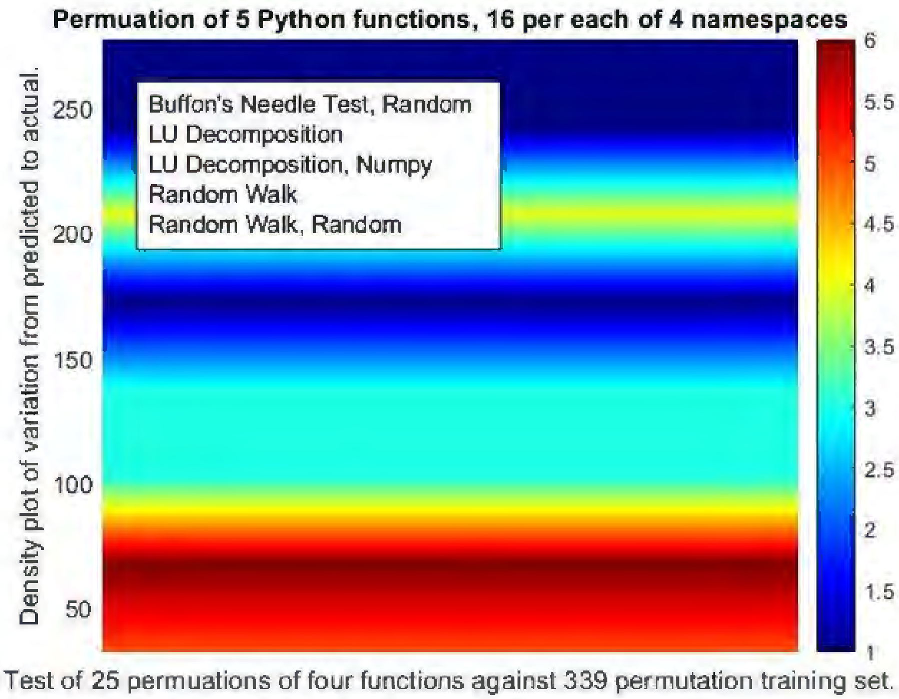


Fig. 4.10: Test case II correlation density. A graphical depiction of offset from success in finding the true fingerprinted function profile. In 25 tests, the algorithm to predict the exact function profile was executed. The characteristic parameters of the test function were compared to the fingerprints of all 339 function profiles. Each of these was given a similarity score, from 1-339. The predicted identity of the fingerprinted profile was compared given the score. These were arranged from lowest to highest, with an exact match having a similarity score of 1. This is represented at the bottom of the Fig.. The offset from one for predictions were found with a simple difference and arranged from lowest to highest. The densities of these offsets are depicted in this Fig..

4.4 Test Cases of Co-Resident Parallel Function Data Leakage

As a balance, the Minikube configuration was increased by an additional CPU and a quarter again of the memory. The parallelism in the name-space was dropped from 32 to 16. Finally, candidate functions were increased from four to sixteen as described in Table 3.6, with each having at least one alternative algorithm or library structure except the Fibonacci sequence and the twin primes finder. These were split out among three different test machines with the same architecture and resources. These resulted in a distinctly different pattern of run-time segments (see Fig. 4.12).

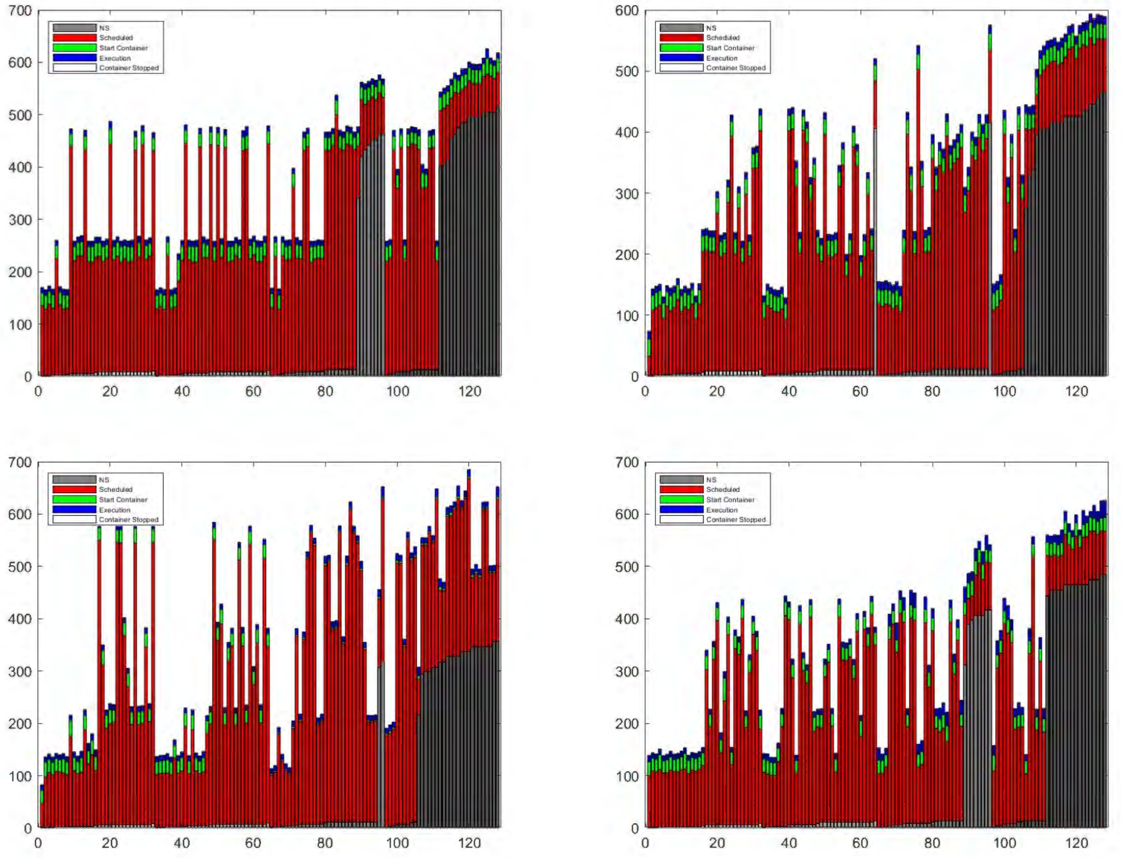
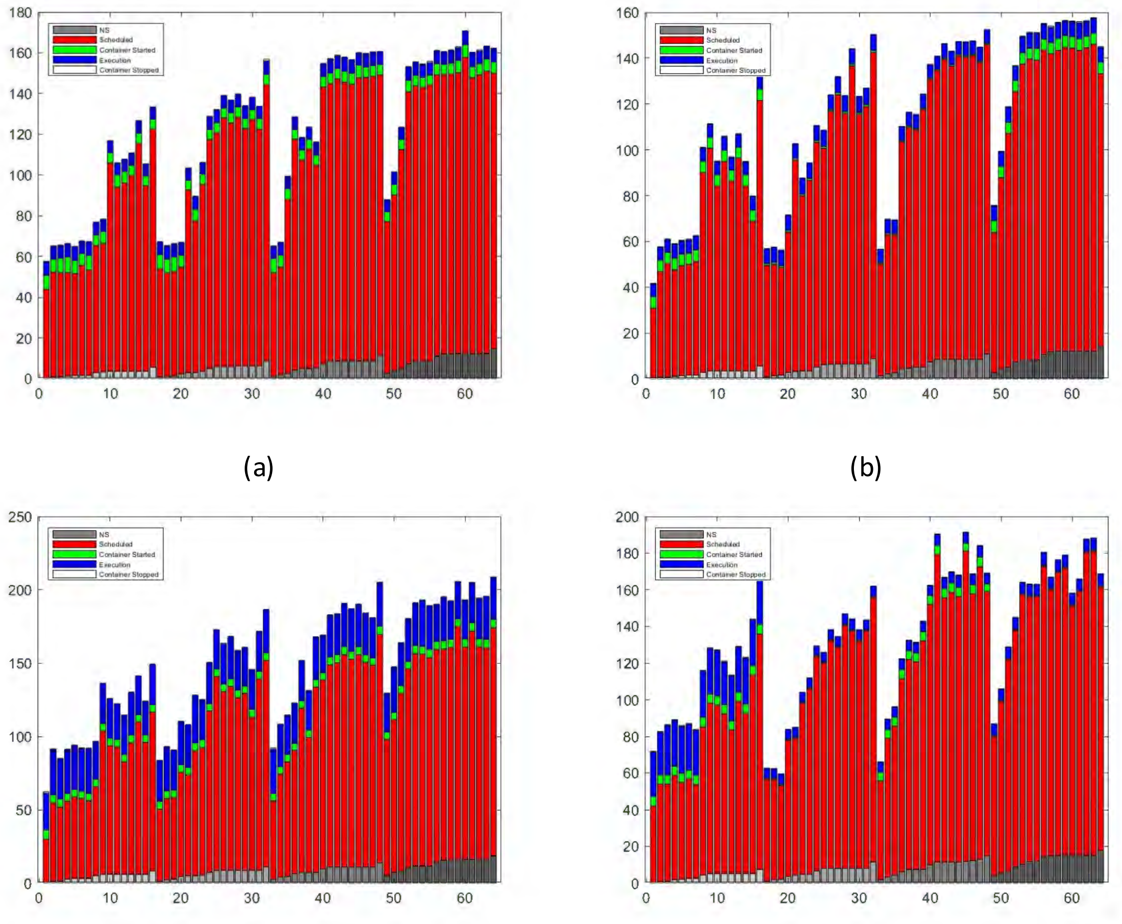


Fig. 4.11: High parallelism execution. An even split of name-space utilization in high parallelism (32 individual executions in four name-spaces) on the Kubernetes cluster. The first two name-spaces were assigned Monte-Carlo methods (Buffon's needle test). The second two namespaces, on the right of each Fig., had (a-d) Buffon's needle test, floating point division, Fibonacci sequence, Numpy matrix inversion.

Just as with the test with 32 parallel functions, the Kubernetes system began to suffer data loss. While scheduling information would fail to emerge in the 32 parallel system, these tests lost execution time data. However, the distribution of run-times and execution times was revealing of the capacity for co-residence. A test of whether this could be controlled was ingrained in the final development for co-resident data leakage in the platform. The results are in the final chapter.



(a) (b)

(c) (d)

Fig. 4.12: Test set data collection. 16 functions running in parallel from a Kubernetes job file. (a) is a uniform function, buffon-test-numpy, across all four name spaces, and (b) is a profile of 1. buffon-test-numpy, 2. buffon-test-piest, 3. division-test-math, 4. division-test-numpy. (c) is a uniform function, comp-fact, across all name-spaces and (d) is a profile of 1. comp-fact, 2. fibonacci, 3. lu-decomp-numpy, 4. random-walk.

CHAPTER 5

DISCUSSIONS AND CONCLUSIONS CHAPTER 5

5.1 Challenges

Although a game-theory model of security in the FaaS environment is dependent on a near infinite range of variables, the options available to any given user of the platform are not. They are limited by the ability to transition from the traditional IaaS, PaaS, and SaaS cloud offerings. They are also limited by their business model's applicability to FaaS platform.

5.1.1) Applying Game Theory Elements: To apply the game theory model, a strategic relationship must exist. That is, a player must be able to influence the other player's options and vice-versa. Implemented in a cloud security environment, this is expressed as a resource contention through variations of execution timing. By observing the effects of this contention on their environment, a player learns something about their opponent.

Considering these two factors, a test was performed to assess whether a potential scientifically or financially based user may encounter an adversary seeking to gain knowledge of their activities. This was conducted with representative workloads on a platform characteristic of FaaS environments. Obtaining and properly interpreting the results of the test was the chief obstacle to this process. The goal of these tests was to determine whether a strategic relationship sufficient to apply game theory exists.

5.1.2) Analysis: Over an equal length of time, the native, zero-trust, and two game theory models were compared on representative runs. Detailed numeric results are displayed in Table 4.1. These show a range of measurable security characteristics that influence system performance.

The IQR-MU and Zero-Trust are two extreme methods: IQR-MU's singular purpose of optimization without security consideration at the boundary of strategy. Squeezing as many VMs as possible in the least number of hosts consumes the least energy and has the least number of host shutdowns. Adjacently, the zero-trust method is an upper bound of rationality being purely deterministic. This leads to the highest number of migrations and utilizes the largest number of hosts. The predation method's zero-sum characteristic borders the payoff characteristic at the extreme of utility.

While the rules governing interaction can be adjusted within this range, those of the individual players in Table 5.1 which specifies VM parameters, do not change. This is exemplified in Appendices C.1 and C.2. In contrast, the altered predation model modified the characteristic VM interaction between hackers and victims as a means to influence the security resulting from migration.

TABLE 5.1

PREDATION SIMULATION SYSTEM PARAMETERS. INPUT PARAMETERS FOR THE SPECIES MODELED IN SIMULATION

Parameters	Value Chosen
Game Space No. of Hosts	48×48
24×24 D/C	576
Max Population (Total VMs) Predator (Malicious VMs)	1600
Initial Creation Probability	0.0189
Breeding Probability Prey (Benign VMs)	0.022245
Initial Creation Probability	0.6725
Breeding Probability	0.022245
Breeding Age	5 time steps
Max Age	100 time steps
Total Run Time (sec)	
24×60×60×3.5	302400

In the altered predation model, an increase in the number of complete cloudlets from the baseline predation model demonstrated an improvement in performance. This, however, came at the cost of migration and consumption of more energy due to reduced host shutdown. However, the trend in risk due to predation over the entire period (see Fig. 4.6b.) was about half of that in the baseline predation model (see Fig. 4.6a). This demonstrates the hypothesis that the trade-offs necessary for scoring in game theory can be obtained from simulation.

A workload delay analysis shows that a narrow range of cloudlet workloads incur a smaller cost in delay. The number of cloudlets taking a higher delay shows a very uneven distribution of the risks associated with a pure predation method (see

Fig. 5.1). While the altered game evens this out over the higher delay, the shorter delays are distributed over a much smaller selection of cloudlets.

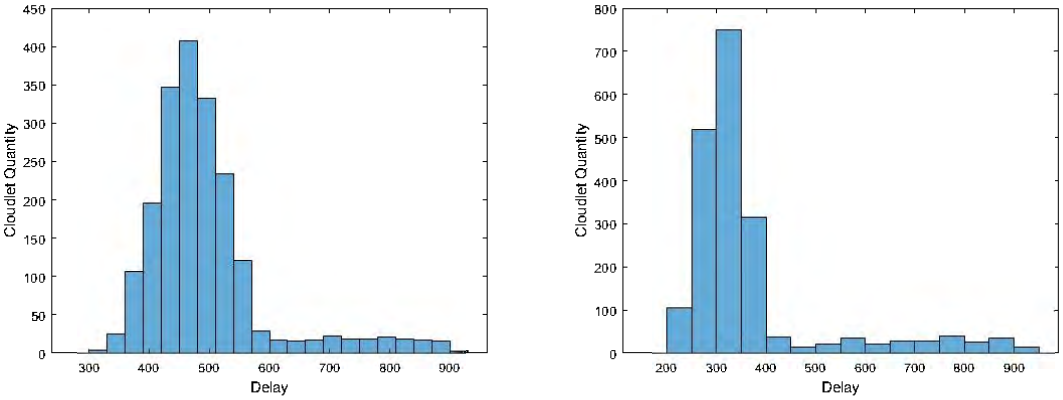


Fig. 5.1: Cloudlet completion delays. Left: Histogram of the number of cloudlets experiencing the number of periods with a delay. Right: A histogram of the altered predation model delays. Both are long tailed indicating an uneven cost in delay. However the narrower range of delay in the altered theory indicates the selective nature of the attack.

TABLE 5.2
SIMULATION OUTPUT COMPARISON. A DATA-CENTER WIDE AGGREGATED PERFORMANCE FOR MODEL VARIATIONS.

	IQR-MU	Zero-Trust	Predation	Altered
Cloudlets	1600	1600	1600	1600
Energy Consumed (MWh)	0.079	1.718	1.725	1.878
Migrations (Million)	0.036	2.048	.988	1.341
Cloudlets Complete	1600	2	247	340
Num. Host Shutdowns	3240	252643	52033	12712

5.2 Data Sets

Three test sets followed the experimental procedure of Chapter 3.1.2. These were executed on the same platform from a selection of functions in Table 3.6. The tested data files were profiled for four test functions, each with a different set from the candidate functions. Data collection and processing procedures were conducted without alteration.

5.2.1) Test Case I: The first test case was performed on a subset of the candidate functions using the input parameters specified. These were:

- Buffon's Needle Test, Numpy
- Buffon's Needle Test, Taylor Series Pi Est.
- Floating Point Division, Math
- Floating Point Division, Numpy

The results of the test showed a multi-modal density plot for successful co-resident data leakage (see Fig. 5.2). These results indicate that between 3 and 4 profiles were in the upper 10 percent of correct determinations (eliminating 172 of the 191 possible function profiles that the target user could be using). Another grouping of 5 to 6 were found in the 35 % range (eliminating only about 120 possible profiles). Finally, the greatest grouping of about profiles appeared at an error rate of 70%, eliminating only 60 possible profiles from the possible 190.

These results indicated a successful information leakage was much less likely than in the validating model. Although no identifying features could be discerned from the input data, possible alignment of similar routine types could suggest an obscuring factor. Similar circumstances could come into play with the multiple uses of Numpy modules.

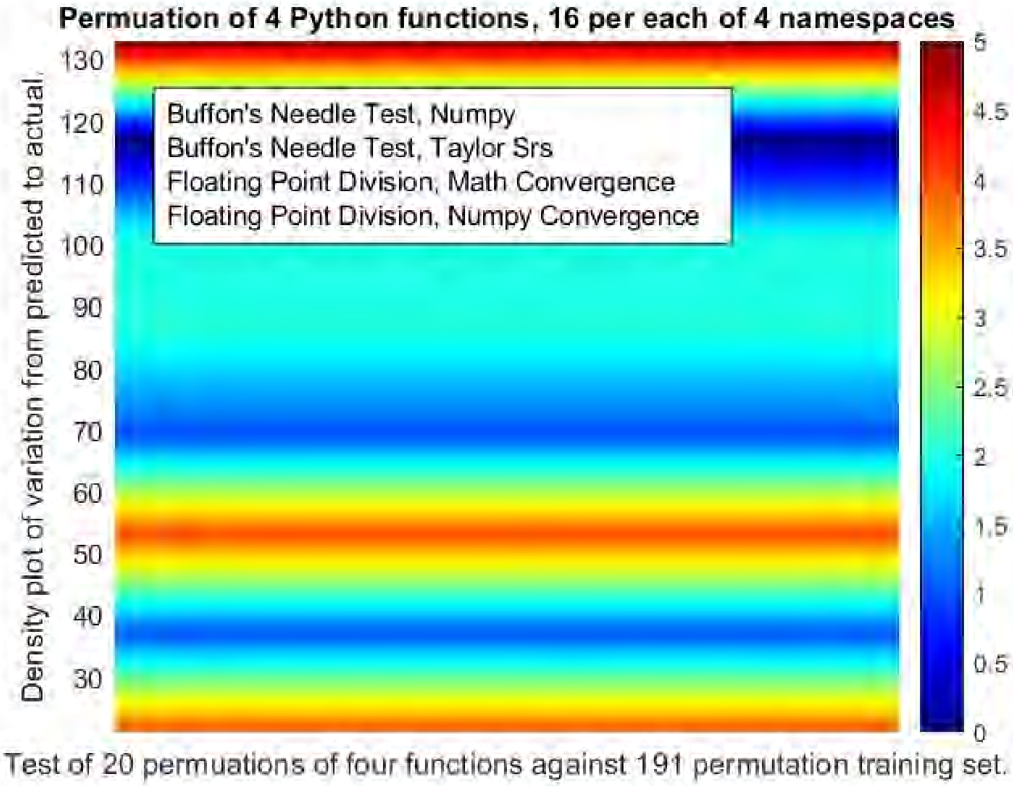


Fig. 5.2: Test Case 1. A density plot of difference between the correct determination of co-resident information. A higher density, displayed in red, closer to the bottom indicates a successful co-resident data leakage. A high density closer to the top indicates that the process of identifying co-resident functions in parallel execution failed. Density of results is indicated by the color of scale on the right.

5.2.2) Test Case II: The second test case was performed on a subset of the candidate functions, also taken from the candidate function list. These functions used none of the same libraries or sub-routines. They are:

- Buffon’s Needle Test, Random
- LU Decomposition
- LU Decomposition, Numpy

- Random Walk

This test case showed a higher degree of agreement in the test case match with the function library (see Fig. 5.3). Here, between 14 and 16 of the 20 test cases were in the range of 20 candidate functions from the possible 256. This eliminated approximately 90% of the candidate profiles from the pool of victim processes.

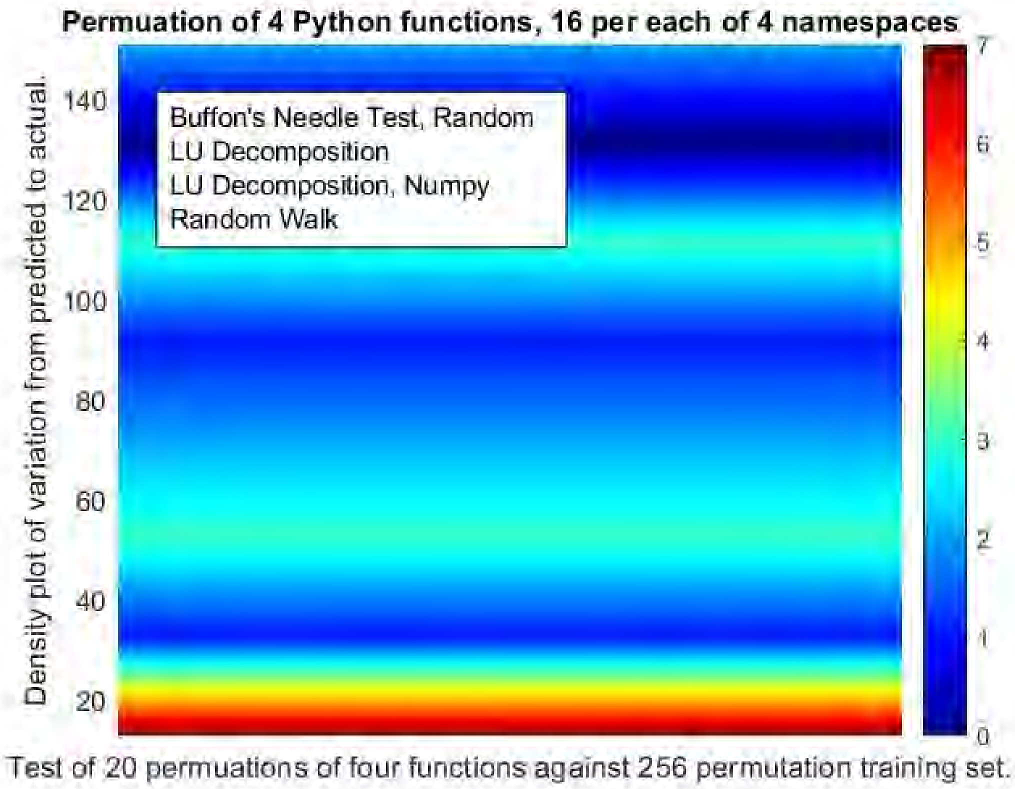


Fig. 5.3: Test Case 2.

5.2.3) Test Case III: The third test case was performed on a subset of the candidate functions that used none of the same libraries or sub-routines. This is done in comparison with the previous test set on two of the four functions:

- Computation of a Factorial
- Fibonacci Sequence

- LU Decomposition, Numpy
- Random Walk

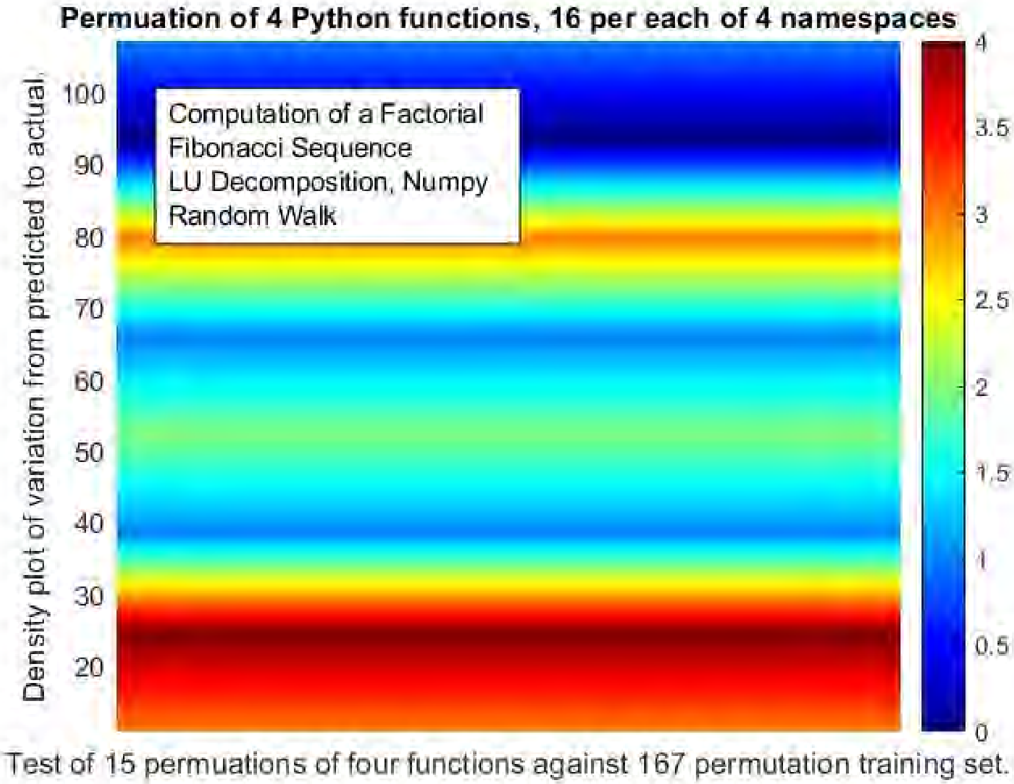


Fig. 5.4: Test Case 3.

5.3 Findings

Within a very narrow range of conditions, a co-resident strategic relationship could be established in an adversarial aspect. Provided the ability to obtain measures of how each user influenced the behavior of an adversary FaaS execution, a sifting process exposes execution timing to leak information. The relationship would be mutually risky for both parties and not definitive as to the exact computational processes being undertaken.

The degree of parallelism had a profound effect on the ability to conduct such a game. The configuration of the host environment, choice of input parameters, and allocation of resources could readily interfere with the ability to conduct such an attack. The degree of parallelism under these conditions greatly skewed the ability to retrieve and interpret this data.

Alternatively, the data also shows that the diffusion of function execution within a namespace, under the conditions above, reduces the risk of co-residence. Assuming that a co-resident, simultaneous execution of code on a host, is a prerequisite for an attack, these conditions may provide an adequate basis to prevent the threat. The factors that cause such dispersion, including library choice, function input parameters, and parallelism, are all under the control of the user.

Within limits, this can form the basis of a mitigation to co-resident threats. While the platform may suffer from slowdown, risk DoS, or limited data availability, it is the scalability that makes FaaS a viable model in the first place. This is, therefore, a consideration of the provider to incorporate into their load balancing and scaling.

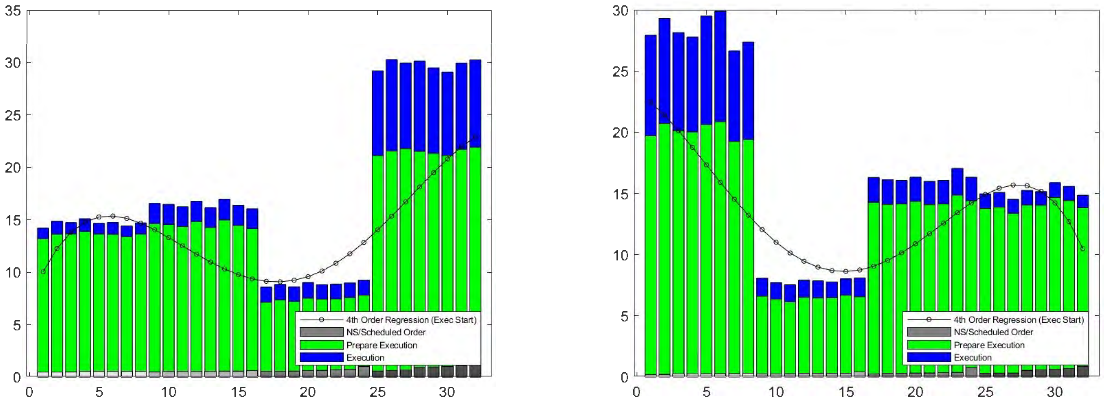
5.4 Module Effects on Resource Utilization

The functions were provided parameters and executed in parallel, such as to maximize the resource consumption of the computer platform. In this manner, the approach of the experiment was a bin-packing problem. The scheduling times, container start times, function execution start and finish times, and finally, container stop were recorded in order to characterize correlated resource contention effects on the distribution of these times.

Upon preliminary tests of the experiment, it was apparent that some data points exposed prominence from an aspect of execution timing that could skew the analysis to overcome the sought-after relationship. Some functions had execution times that revealed which profile of code was executed in parallel without considering the resource contention being tested (see Fig. 5.5). A reversal of the order of the profile exposed the same effect.

Notable in this finding is the apparent effect of including certain Python modules. The Fibonacci sequence in both examples of Fig. 5.5 completes every parallel execution before the first and second functions are started. Each of the first two includes an additional module, Random for the Buffon's needle test and Math for the floating point division. The matrix inversion routine includes the Numpy module.

The effect of these variations is that no two sets of parallel functions are executing at the same time on the platform. The Buffon's needle test and FPD have the closest correlation, but in both forward and backward order their actual co-resident execution time is less than 0.5 seconds per function. This suggests that information leakage is too problematic, assuming that a method to overcome the container isolation can be developed. Therefore, it is possible to mitigate some co-resident data leakage attacks by the judicious distribution of functions based on their composite libraries.



(a) (b)

Fig. 5.5: Module selection effects. A parallel execution of mirror profiles of candidate functions: from left to right in (a) Buffon's needle test, floating point division, Fibonacci sequence, Numpy matrix inversion. In (b) Numpy matrix inversion, Fibonacci sequence, floating point division, Buffon's needle test.

5.5 Further Study

The single host environment in which this experiment took place forced co-residence on the pods and containers that were tested. While this may favor the analysis of information leakage over Kubernetes in a cluster, other factors intervene. The Pods on a Kubernetes cluster are not limited to a single container. In fact, the presence of a sidecar container is common for the separation of functionality in most applications.

In addition, the requirement to include functionality other than the algorithmic logic is part of most applications. This can take the form of databases, webhooks, and parallel logic. In a FaaS environment, this may be necessary to make up for the lack of state and synchronization. An orchestrated application will have this as an inherent component.

So, though the ability to gain co-residence in a FaaS Kubernetes environment may be diminished in a multi-host cluster, it may be increased as well due to both scaling and workflow. Testing these effects will be an important test of whether co-resident attacks can reveal more about the activities of other cloud users and whether there is a strategic model for FaaS computer security.

REFERENCES

[1] R. Patnayakuni and N. Patnayakuni, “Securing serverless computing,” in *WISP 2018 Proceedings*, pp. 1–5, 2018.

[2] J. M. O. Candel, A. Elouali, F. J. M. Gimeno, and H. Mora, “Cloud vs serverless computing: A security point of view,” in *Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2022)*, pp. 1098–1109, 2023.

[3] A. Smith, *The Wealth of Nations*. AuthorHouse, 2009.

[4] S. Ghazanfar and A. Islahi, “Economic thought of an Arab scholastic: Abu hamid al-Ghazali (a.h. 450-505/a.d. 1058-1111),” *History of Political Economy - HIST POLIT ECON*, vol. 22 , pp. 381–403, 06 1990.

[5] P. A. Samuelson, *Economics. International Student Ed.* Louis etc-Tokyo McGraw-HillKogakusha XXIII, 1970.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds: A Berkeley view of cloud computing,” tech. rep., University of California at Berkeley, 2 2009.

[7] B. Schiller, *Essentials of Economics*. 1221 Ave. of the Americas, New York, New York, 10020: McGraw-Hill, 2009.

[8] U. Cantner, J. A. Cunningham, E. E. Lehmann, and M. Menter, “Entrepreneurial ecosystems: a dynamic lifecycle model,” *Small Business Economics*, vol. 57, no. 1, pp. 407–423 , 2021.

[9] G. Irazoqui, *Cross-core Microarchitectural Side Channel Attacks and Countermeasures*. PhD thesis, Worcester Polytechnic Institute, April 2017.

[10] W. J. Lloyd, S. Pallickara, O. David, M. Arabi, T. Wible, J. Ditty, and K. Rojas, “Demystifying the clouds: Harnessing resource utilization models for cost effective infrastructure alternatives,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 667–680, 2017.

[11] M. Kayaalp, N. B. Abu-Ghazaleh, D. V. Ponomarev, and A. Jaleel, “A high-resolution sidechannel attack on last-level cache,” *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[12] S. van Schaik, C. Giuffrida, H. Bos, and K. Razavi, “Malicious management unit: Why stopping cache attacks in software is harder than you think,” in *USENIX Security Symposium*, 2018.

[13] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan, “Security of os-level virtualization technologies,” in *Secure IT Systems* (K. Bernsmed and S. Fischer-Hübner, eds.), (Cham), pp. 77–93, Springer International Publishing, 2014.

[14] M. Pearce, S. Zeadally, and R. Hunt, “Virtualization: Issues, security threats, and solutions,” *ACM Comput. Surv.*, vol. 45, mar 2013.

[15] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of virtual machine live migration in clouds: A performance evaluation,” in *International Conference on Cloud Computing*, vol. 5931, 09 2011.

[16] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, “DRAMA: Exploiting DRAM addressing for Cross-CPU attacks,” in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 565–581, USENIX Association, 8 2016.

[17] A. T. Litchfield and A. Shahzad, “Virtualization technology: Cross-vm cache side channel attacks make it vulnerable,” *CoRR*, vol. abs/1606.01356, 2016.

[18] J. P. Shen and M. H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*. 4180 IL Route 83, Suite 101, Long Grove, IL 60047-9580: Waveland Press, Inc., 2002.

[19] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Burlington, MA: Morgan Kaufmann, 01 2008.

[20] D. Tam, R. Azimi, L. Baldini Soares, and M. Stumm, “Managing shared l2 caches on multicore systems in software,” in *Proc. Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA)*, (New York, NY, USA), pp. 26–33, ACM, 01 2007.

[21] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, , and D. Ponomarev, “Nonmonopolizable caches: Low-complexity mitigation of cache side channel attacks,” *ACM Trans. Architec. Code Optim.*, vol. 8, no. 4, p. 21, 2012.

[22] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, “Side-channel vulnerability factor: A metric for measuring information leakage,” in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 106–117, 2012.

[23] J. Szefer, “Survey of microarchitectural side and covert channels, attacks, and defenses,” *Journal of Hardware and Systems Security*, vol. 3, 09 2019.

[24] K. H. Kim, A. Beloglazov, and R. Buyya, “Power-aware provisioning of virtual machines for real-time cloud,” *Concurrency and Computation: Practice and Experience*, vol. 23 , pp. 1491–1505, 09 2011.

[25] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” *Operating Systems Review*, p. 164–177 , 2003.

[26] P. Sempolinski and D. Thain, “A comparison and critique of eucalyptus, opennebula and nimbus,” in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 417–426, 2010.

[27] Y. Han, J. Chan, T. Alpcan, and C. Leckie, “Virtual machine allocation policies against co-resident attacks in cloud computing,” *2014 IEEE International Conference on Communications (ICC)*, pp. 786–792, 2014.

[28] Z. Wu, Z. Xu, and H. Wang, “Whispers in the hyper-space: High-speed covert channel attacks in the cloud,” in *21st USENIX Security Symposium (USENIX Security 12)*, (Bellevue, WA), pp. 159–173, USENIX Association, 8 2012.

[29] R. Rosenberger and R. M. Greenberg, “Computer virus myths,” *ACM Sigsac Review*, vol. 7 , pp. 21–24, 1990.

[30] M. Ahmed and M. Hossain, “Cloud computing and security issues in the cloud,” *International Journal of Network Security & Its Applications*, vol. 6, pp. 25–36, 01 2014.

[31] M. Botacin, F. Ceschin, R. Sun, D. Oliveira, and A. R. A. Grégio, “Challenges and pitfalls in malware research,” *Comput. Secur.*, vol. 106, p. 102287, 2021.

[32] T. Garfinkel and M. Rosenblum, “When virtual is harder than real: Security challenges in virtual machine based computing environments,” in *In 10th Workshop on Hot Topics in Operating Systems*, 2005.

[33] Z. Wang and R. B. Lee, “Covert and side channels due to processor architecture,” *2006 22nd Annual Computer Security Applications Conference (ACSAC’06)*, pp. 473–482, 2006.

[34] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-vm side channels and their use to extract private keys,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS ’12, (New York, NY, USA), p. 305–316, Association for Computing Machinery, 2012.

[35] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, “Last-level cache side-channel attacks are practical,” *2015 IEEE Symposium on Security and Privacy*, pp. 605–622, 2015.

[36] D. Gullasch, E. Bangerter, and S. Krenn, “Cache games – bringing access-based cache attacks on aes to practice,” in *2011 IEEE Symposium on Security and Privacy*, pp. 490–505 , 2011.

p. D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, “Branchscope: A new side-channel attack on directional branch predictor,” *SIGPLAN Not.*, vol. 53, 693–707, mar 2018.

[37] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-tenant side-channel attacks in paas clouds,” *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[38] P. Chen, L. Li, and Z. Yang, “Cross-VM and Cross-Processor covert channels exploiting processor idle power management,” in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 733–750, USENIX Association, Aug. 2021.

[39] R. Vanathi and S. Chokkalingam, “Side channel attacks in iaas and its defense mechanisms,” *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 8, 02 2019.

[40] F. Miao, L. Wang, and Z. Wu, “A vm placement based approach to proactively mitigate co-resident attacks in cloud,” in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00285–00291, 2018.

[41] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler, “On detecting coresident cloud instances using network flow watermarking techniques,” *International Journal of Information Security*, vol. 13, no. 2, pp. 171–189, 2014.

[42] S. Homsi, *Cloud Workload Allocation Approaches for Quality of Service Guarantee and Cybersecurity Risk Management*. PhD thesis, Florida International University, 2019.

[43] Y. Han, *Defending against co-resident attacks in cloud computing*. PhD thesis, University of Melbourne, September 2015.

[44] M. Gawali and S. Shinde, “Task scheduling and resource allocation in cloud computing using a heuristic approach,” *Journal of Cloud Computing*, vol. 7, 02 2018.

[45] W. Attaouiy and E. Sabir, “Multi-criteria virtual machine placement in cloud computing environments: A literature review,” *ArXiv*, vol. abs/1802.05113, 2018.

[46] X. Sui, D. Liu, L. Li, H. Wang, and H. Yang, “Virtual machine scheduling strategy based on machine learning algorithms for load balancing,” *EURASIP Journal on Wireless Communications and Networking*, vol. 9, no. 160, 2019.

[47] S. Homsi, S. Liu, G. A. Chaparro-Baquero, O. Bai, S. Ren, and G. Quan, “Workload consolidation for cloud data centers with guaranteed quality of service using request reneging,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 2103 – 2116, July 2017.

[48] X. Wang, L. Wang, F. Miao, and J. Yang, “Svmdf: A secure virtual machine deployment framework to mitigate co-resident threat in cloud,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, 2019.

[49] H. Xu, Y. Liu, W. Wei, and W. Zhang, “Incentive-aware virtual machine scheduling in cloud computing,” *The Journal of Supercomputing*, vol. 74, pp. 1–23, 07 2018.

[50] M. Schwarzl, E. Kraft, M. Lipp, and D. Gruss, “Remote memory-deduplication attacks,” 2021.

[51] H. B. Lee, T. M. Jois, C. W. Fletcher, and C. A. Gunter, “Dove: A data-oblivious virtual environment,” *ArXiv*, vol. abs/2102.05195, 2021.

[52] J. V. Bulck, F. Piessens, and R. Strackx, “Sgx-step: A practical attack framework for precise enclave execution control,” *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, 2017.

[53] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, “Homealone: Co-residency detection in the cloud via side-channel analysis,” in *Security and Privacy (SP), 2011 IEEE Symposium on*, pp. 313–328, IEEE, 2011.

[54] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapôtre, G. Gogniat, and P. Benoit, “Whisper: A tool for run-time detection of side-channel attacks,” *IEEE Access*, vol. 8 , pp. 83871–83900, 2020.

[55] M.-M. Bazm, M. Lacoste, M. Südholt, and J.-M. Menaud, “Side-channels beyond the cloud edge: New isolation threats and solutions,” in *2017 1st Cyber Security in Networking Conference (CSNet)*, pp. 1–8, 2017.

[56] F. Abazari, M. Analoui, and H. Takabi, “Multi-objective response to co-resident attacks in cloud environment,” *International Journal of Information Communication Technology Research*, vol. 9, pp. 25–36, 09 2017.

[57] M. Khaleel, S. Alqithami, M. Zhu, D. Che, and W.-C. Hou, “A cooperative game theorybased approach for energy-aware job scheduling in cloud,” *International Journal of Computers and Their Applications*, vol. 20, pp. 221–235, 12 2013.

[58] P. S. Pillai and S. Rao, “Resource allocation in cloud computing using the uncertainty principle of game theory,” *IEEE Systems Journal*, vol. 10, no. 2, pp. 637–648, 2016.

p. Z. Xu, H. Wang, and Z. Wu, "A measurement study on co-residence threat inside the cloud," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, (USA), 929–944, USENIX Association, 2015.

[59] T. Zhang, S. Chen, F. Liu, and R. Lee, "Side channel vulnerability metrics: the promise and the pitfalls," in *ACM International Conference Proceeding Series*, 06 2013.

[60] Y. Han, T. Alpcan, J. Chan, C. Leckie, and B. Rubenstein, "A game theoretical approach to defend against co-resident attacks in cloud computing: Preventing co-residence using semisupervised learning," *IEEE Transactions on Information Forensics and Security*, vol. 11 , no. 3, p. 15, 2015.

[61] J. Pawlick, E. Colbert, and Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *ACM Computing Surveys (CSUR)*, vol. 52, pp. 1 – 28, 2019.

[62] M. Mehedi Hasan and M. A. Rahman, "A signaling game approach to mitigate co-resident attacks in an iaas cloud environment," *J. Inf. Secur. Appl.*, vol. 50, feb 2020.

[63] R. U. Ayres, "On the life cycle metaphor: where ecology and economics diverge," *Ecological Economics*, vol. 48, no. 4, pp. 425–438, 2004.

[64] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[65] M. Begon, C. R. Townsend, and J. L. Harper, *Ecology: From Individuals to Ecosystems*. Hoboken, NJ: Blackwell, 4 ed., 2005.

[66] H. K., E. T.H.G., O. Jones, S.-G. R, and B. Y.M., "Animal life history is shaped by the pace of life and the distribution of age-specific mortality and reproduction.," *Nat Ecol Evol.*, vol. 3, no. 8, pp. 1217–1224, 2019.

[67] J. Wellington, "Lectures of ecology, biology 4468." https://uh.edu/ biolcz/class/eco4468/ lect13.htm.

[68] J. C. Krebs, "Ecology: The experimental analysis of distribution and abundance," 2014.

[69] E. M, H. M.R., T. A.J., and S. C.A., "Evidence of partial migration in a large coastal predator: Opportunistic foraging and reproduction as key drivers?," *PLOS ONE*, vol. 11, February 2016.

[70] L. Näsén, *Synchronizing Migration with Birth: An Exploration Of Migratory Tactics In Female Moose*. PhD thesis, Swedish university of agricultural sciences, 2015.

[71] N. Serra, “Utility functions and lotka-volterra model: A possible connection in predatorprey game,” *Journal of Game Theory*, vol. 3, pp. 31–34, 2014.

[72] M. Garvie, “Finite-difference schemes for reaction–diffusion equations modeling predator–prey interactions in matlab,” *Bulletin of mathematical biology*, vol. 69, pp. 931–56, 05 2007.

[73] G. Morabito, C. Sicari, A. Ruggeri, A. Celesti, and L. Carnevale, “Secure-by-design serverless workflows on the edge-cloud continuum through the osmotic computing paradigm,” *Internet Things*, vol. 22, p. 100737, 02 2023.

[74] S. Sarkar, R. Wankar, S. N. Srirama, and N. K. Suryadevara, “Serverless management of sensing systems for fog computing framework,” *IEEE Sensors Journal*, vol. 20, pp. 1564 – 1572, 2020.

[75] A. Hall and U. Ramachandran, “An execution model for serverless functions at the edge,” in *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI ’19, (New York, NY, USA), p. 225–236, Association for Computing Machinery, 2019.

[76] L. Baresi and D. F. Mendonça, “Towards a serverless platform for edge computing,” *2019 IEEE International Conference on Fog Computing (ICFC)*, pp. 1–10, 2019.

[77] F. Alvarez, D. Breitgand, D. Griffin, P. Andriani, S. Rizou, N. Zioulis, F. Moscatelli, J. Serrano, M. Keltsch, P. Trakadas, T. K. Phan, A. Weit, U. Acar, O. Prieto, F. Iadanza, G. Carrozzo, H. Koumaras, D. Zarpalas, and D. Jimenez, “An edge-to-cloud virtualized multimedia service platform for 5g networks,” *IEEE Transactions on Broadcasting*, vol. 65, no. 2 , pp. 369–380, 2019.

[78] J. Spillner, C. Mateos, and D. A. Monge, “Faaster, better, cheaper: The prospect of serverless scientific computing and hpc,” in *CARLA*, 2017.

[79] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Isahagian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, “Serverless computing: Current trends and open problems,” in *Research Advances in Cloud Computing*, (Singapore), pp. 1–20, Springer Singapore, 12 2017.

[80] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in *2015 10th Computing Colombian Conference (10CCC)*, pp. 583–590 , 10 2015.

[81] J. Wei and M. Gao, “Workload prediction of serverless computing,” in *Proceedings of the 2021 5th International Conference on Deep Learning Technologies*, ICDLT ’21, (New York, NY, USA), p. 93–99, Association for Computing Machinery, 2021.

[82] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” *Concurrency and Computation: Practice and Experience*, vol. 24, 2012.

[83] V. Gupta, S. R. Phade, T. A. Courtade, and K. Ramchandran, “Utility-based resource allocation and pricing for serverless computing,” *CoRR*, vol. abs/2008.07793, 2020.

[84] J. Kuhlenkamp, S. Werner, and S. Tai, “The ifs and buts of less is more: A serverless computing reality check,” in *2020 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 154–161, 04 2020.

[85] M. Filho, E. Pimentel, W. Pereira, P. H. M. Maia, and M. I. Cortés, “Self-adaptive microservice-based systems – landscape and research opportunities,” 3 2021.

[86] J. Manner, M. Endreb, S. Bohm, and G. Wirtz, “Optimizing cloud function configuration via local simulations,” in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, (Los Alamitos, CA, USA), pp. 168–178, IEEE Computer Society, 9 2021.

[87] M. Shahrad, R. Fonseca, I. n. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, (USA), pp. 205–218, USENIX Association, jul 2020.

[88] G. Adzic and R. Chatley, “Serverless computing: Economic and architectural impact,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, (New York, NY, USA), p. 884–889, Association for Computing Machinery, 2017.

[89] B. Jambunathan and K. Yoganathan, “Architecture decision on using microservices or serverless functions with containers,” *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pp. 1–7, 2018.

[90] J. M. Hellerstein, J. M. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, “Serverless computing: One step forward, two steps back,” *ArXiv*, vol. abs/1812.03651, 2019.

[91] P. Maissen, P. Felber, P. Kropf, and V. Schiavoni, “Faasdom: A benchmark suite for serverless computing,” in *Proceedings of the 14th ACM International Conference on*

Distributed and Event-Based Systems, DEBS ’20, (New York, NY, USA), p. 73–84, Association for Computing Machinery, 2020.

[92] V. Sazawal and N. Sudan, “Modeling software evolution with game theory,” in *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*, ICSP ’09, (Berlin, Heidelberg), p. 354–365, Springer-Verlag, 2009.

[93] Z. Wang, “Can “micro vm” become the next generation computing platform?: Performance comparison between light weight virtual machine, container, and traditional virtual machine,” in *2021 IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE)*, pp. 29–34, 8 2021.

[94] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, “Firecracker: Lightweight virtualization for serverless applications,” in *NSDI*, 2020.

[95] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, “My vm is lighter (and safer) than your container,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP ’17, (New York, NY, USA), p. 218–233 , Association for Computing Machinery, 2017.

[96] J. Thalheim, P. Bhatotia, P. Fonseca, and B. Kasikci, “Cntr: Lightweight os containers,” in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC ’18, (USA), p. 199–212, USENIX Association, 2018.

[97] S. Qin, H. Wu, Y. Wu, B. Yan, Y. Xu, and W. Zhang, “Nuka: A generic engine with millisecond initialization for serverless computing,” in *2020 IEEE International Conference on Joint Cloud Computing (JCC)*, (Los Alamitos, CA, USA), pp. 78–85, IEEE Computer Society, 8 2020.

[98] M. Kerrisk, “cgroups(7) — linux manual page,” 2021.

[99] N. Yang, W. Shen, J. Li, Y. Yang, K. Lu, J. Xiao, T. Zhou, C. Qin, W. Yu, J. Ma, and K. Ren, “Demons in the shared kernel: Abstract resource attacks against os-level virtualization,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’21, (New York, NY, USA), p. 764–778, Association for Computing Machinery, 2021.

[100] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues, challenges, and the road ahead,” *IEEE Access*, vol. 7, pp. 52976–52996, 2019.

[101] L. Rice, *Container Security*. Sebastopol, CA: O’Reilly, 2020. ISBN 978-1-492-05670-6.

[102] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, (Savannah, GA), pp. 689 – 703, 2016.

[103] A. Wong, E. Chekole, M. Ochoa, and J. Zhou, "Threat modeling and security analysis of containers: A survey," *CoRR*, vol. abs/2111.11475, 2021.

[104] K. Alpernas, C. Flanagan, S. Fouladi, L. Ryzhyk, S. Sagiv, T. Schmitz, and K. Winstein,

"Secure serverless computing using dynamic information flow control," *Proceedings of the ACM on Programming Languages*, vol. 2, pp. 1 – 26, 2018.

[105] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, "Valve: Securing function workflows on serverless computing platforms," in *Proceedings of The Web Conference 2020*, WWW '20, (New York, NY, USA), p. 939–950, Association for Computing Machinery, 2020.

[106] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), p. 418–429 , Association for Computing Machinery, 2018.

[107] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang, "Houdini's escape: Breaking the resource rein of linux control groups," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, (New York, NY, USA), p. 1073–1086 , Association for Computing Machinery, 2019.

[108] J. Sun, C. Wu, and J. Ye, "Blockchain-based automated container cloud security enhancement system," in *2020 IEEE International Conference on Smart Cloud (SmartCloud)*, (Los Alamitos, CA, USA), pp. 1–6, IEEE Computer Society, 10 2020.

[109] J. Edge, "The bogus cve problem," 9 2023.

[110] X. Li, X. Leng, and Y. Chen, "Securing serverless computing: Challenges, solutions, and opportunities," *IEEE Network*, pp. 1–8, 2022.

[111] I. Mavridis and H. Karatza, "Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud," *Concurrency and Computation: Practice and Experience*, vol. 35, 05 2021.

[112] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, “Experimenting with docker: Linux container and base os attack surfaces,” in *2016 International Conference on Information Society (i-Society)*, pp. 17–21, 2016.

[113] W. Viktorsson, C. Klein, and J. Tordsson, “Security-performance trade-offs of kubernetes container runtimes,” in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, (Los Alamitos, CA, USA), pp. 1–4, IEEE Computer Society, 10 2020.

[114] A. Sabbioni, C. Mazzocca, M. Colajanni, R. Montanari, and A. Corradi, “A fully decentralized architecture for access control verification in serverless environments,” in *2022 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, June 2022.

[115] E. Marin, D. Perino, and R. D. Pietro, “Serverless computing: a security perspective,” *Journal of Cloud Computing*, vol. 11, pp. 1–12, 2022.

[116] D. Sgandurra and E. Lupu, “Evolution of attacks, threat models, and solutions for virtualized systems,” *ACM Computing Surveys*, vol. 48, pp. 1–38, 02 2016.

[117] J.-A. Kabbe, “Security analysis of docker containers in a production environment,” 2017.

[118] Y. Yang, W. Shen, B. Ruan, W. Liu, and K. Ren, “Security challenges in the container cloud,” in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, (Los Alamitos, CA, USA), pp. 137–145 , IEEE Computer Society, 12 2021.

[119] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, “Linux security module framework,” in *Conference: 2002 Ottawa Linux Symposium*, 07 2002.

[120] V. S. R. Pusuluri, “Taxonomy of security and privacy issues in serverless computing,” 6 2022.

[121] L. Rao, “Picloud launches serverless computing platform to the public.”

[122] P. G. López, M. Sánchez-Artigas, G. Paris, D. Barcelona-Pons, A. Ollobarren, and D. Pinto, “Comparison of faas orchestration systems,” *CoRR*, vol. abs/1807.11248, pp. 148–153, 12 2018.

[123] A. Eivy and J. Weinman, “Be wary of the economics of "serverless" cloud computing,” *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.

[124] H. Lee, K. Satyam, and G. C. Fox, “Evaluation of production serverless computing environments,” *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 442–450, 2018.

[125] J. Hu, A. Bruno, B. Ritchken, B. Jackson, M. Espinosa, A. Shah, and C. Delimitrou, “Hivemind: A scalable and serverless coordination control platform for UAV swarms,” *CoRR*, vol. abs/2002.01419, 2020.

[126] A. Bocci, S. Forti, G. L. Ferrari, and A. Brogi, “Secure faas orchestration in the fog: how far are we?,” *Computing*, vol. 103, pp. 1025–1056, 2021.

[127] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, “Benchmarking, analysis, and optimization of serverless function snapshots,” in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’21)*, ACM, 2021.

[128] K. K. Bailey, “Challenges in research experimentation with open-source platforms for serverless computing,” 05 2022.

[129] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, “Serverless applications: Why, when, and how?,” *IEEE Software*, vol. 38 , pp. 32–39, 1 2021.

[130] V. Yussupov, U. Breitenbücher, F. Leymann, and C. Müller, “Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC’19, (New York, NY, USA), p. 273–283, Association for Computing Machinery, 2019.

[131] T. Zois, “The evolution of serverless services,” tech. rep., Vrije Universiteit and University of Amsterdam, 02 2021.

[132] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, “The serverless trilemma: Function composition for serverless computing,” in *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2017, (New York, NY,

USA), p. 89–103, Association for Computing Machinery, 2017.

[133] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, “Serverless execution of scientific workflows,” in *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings*, (Berlin, Heidelberg), p. 706–721, Springer-Verlag, 2017.

[134] M. Malawski, “Towards serverless execution of scientific workflows – hyperflow case study,” in *WORKS 2016 Workshop, Workflows in Support of Large-Scale Science, November 2016, Salt Lake City, Utah*, 11 2016.

[135] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, “Sand: Towards high-performance serverless computing,” in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC ’18, (USA), p. 923–935 , USENIX Association, 2018.

[136] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, “A mixed-method empirical study of function-as-a-service software development in industrial practice,” *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.

[137] L. Feng, P. Kudva, D. D. Silva, and J. Hu, “Exploring serverless computing for neural network training,” *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 334–341, 2018.

[138] T. Zhang, D. Xie, F. Li, and R. Stutsman, “Narrowing the gap between serverless and its state with storage functions,” in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’19, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2019.

[139] D. Kelly, F. G. Glavin, and E. Barrett, “Denial of wallet – defining a looming threat to serverless computing,” 2021.

[140] J. Wen, Z. Chen, X. Jin, and X. Liu, “Rise of the planet of serverless computing: A systematic review,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, 01

2023.

[141] W. O’Meara and R. G. Lennon, “Serverless computing security: Protecting application logic,” in *2020 31st Irish Signals and Systems Conference (ISSC)*, pp. 1–5, 2020.

[142] Y. Kim, J. Koo, and U.-M. Kim, “Vulnerabilities and secure coding for serverless applications on cloud computing,” in *Human-Computer Interaction. User Experience and Behavior: Thematic Area, HCI 2022, Held as Part of the 24th HCI International Conference, HCII 2022, Virtual Event, June 26 – July 1, 2022, Proceedings, Part III*, (Berlin, Heidelberg), p. 145–163, Springer-Verlag, 2022.

[143] N. Mateus-Coelho and M. Cruz-Cunha, “Serverless service architectures and security minimals,” in *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6, 6 2022.

[144] F. Romero, G. I. Chaudhry, I. n. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca, C. Kozyrakis, and R. Bianchini, “Faa$t: A transparent auto-scaling cache for serverless applications,” in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’21, (New York, NY, USA), p. 122–137, Association for Computing Machinery, 2021.

[145] Y. Han, J. Chan, T. Alpcan, and C. Leckie, “Using virtual machine allocation policies to defend against co-resident attacks in cloud computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 95–108, 2015.

[146] M. Orzechowski, B. Balis,´ R. G. Słota, and J. Kitowski, “Reproducibility of computational experiments on kubernetes-managed container clouds with hyperflow,” *Computational Science – ICCS 2020*, vol. 12137, pp. 220 – 233, 2020.

[147] A. Mampage, S. Karunasekera, and R. Buyya, “A holistic view on resource management in serverless computing environments: Taxonomy and future directions,” *ACM Comput. Surv.*, vol. 54, 9 2022.

[148] D. Ustiugov, T. Amariucai, and B. Grot, “Analyzing tail latency in serverless clouds with stellar,” in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 51–62, 11 2021.

[149] N. C. Wanninger, J. J. Bowden, K. Shetty, A. Garg, and K. C. Hale, “Isolating functions at the hardware limit with virtines,” in *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys ’22, (New York, NY, USA), p. 644–662, Association for Computing Machinery, 2022.

[150] M. C. Loring, M. Marron, and D. Leijen, “Semantics of asynchronous javascript,” in *Proceedings of the 13th ACM SIGPLAN International Symposium on on Dynamic Languages*, DLS 2017, (New York, NY, USA), p. 51–62, Association for Computing Machinery, 2017.

[151] K. Alpernas, A. Panda, and M. Sagiv, “This is not the end: Rethinking serverless function termination,” 2022.

[152] D. Lehmann, J. Kinder, and M. Pradel, “Everything old is new again: Binary security of webassembly,” in *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC’20, (USA), USENIX Association, 2020.

[153] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, “A review of serverless use cases and their characteristics,” 2020.

[154] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, “Experimental analysis of the application of serverless computing to iot platforms,” *Sensors (Basel, Switzerland)*, vol. 21 , 2021.

[155] R. Chard, T. J. Skluzacek, Z. Li, Y. Babuji, A. Woodard, B. Blaiszik, S. Tuecke, I. Foster, and K. Chard, “Serverless supercomputing: High performance function as a service for science,” 2019.

[156] M. Malawski, K. Figiela, A. Gajek, and A. Zima, “Benchmarking heterogeneous cloud functions,” in *Euro-Par Workshops*, pp. 415–426, 02 2018.

[157] S. Werner, J. Kuhlenkamp, M. Klems, J. Müller, and S. Tai, “Serverless big data processing using matrix multiplication as example,” *2018 IEEE International Conference on Big Data (Big Data)*, pp. 358–365, 2018.

[158] E. Paraskevoulakou, “Machine learning pipelines in serverless environments,” 2 2020. [161] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, “Cold start influencing factors in function as a service,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 181–188, 12 2018.

[162] D. Barcelona-Pons, P. Sutra, M. Sánchez-Artigas, G. París, and P. García-López, “Stateful serverless computing with crucial,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, 3 2022.

[163] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, “Sprocket: A serverless video processing framework,” in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’18, (New York, NY, USA), p. 263–274, Association for Computing Machinery, 2018.

[164] L. Samuelson, *Evolutionary Games and Equilibrium Selection*. Boston, Massachusetts: The MIT Press, 1997.

[165] W. Spaniel, *Game Theory 101, The complete textbook*. CreateSpace Independent Publishing, 2013.

[166] B. Seibold, M. R. Flynn, A. R. Kasimov, and R. R. Rosales, “Constructing set-valued fundamental diagrams from jamiton solutions in second order traffic models,” *Networks and Heterogeneous Media*, vol. 8, no. 3, p. 745–772, 2013. https://math.mit.edu/traffic/.

[167] S. Mitchell-Malm, “what really happened in tuscan gp’s huge restart crash,” 9 2020.

[168] I. A. Villalobos, A. S. Poznyak, and A. Malo, “Urban traffic control problem a game theory approach,” *2008 47th IEEE Conference on Decision and Control*, pp. 2168–2172, 2008.

[169] R. Dholakia and Kshetri, “Factors impacting the adoption of the internet among smes,” *Small Business Economics*, vol. 23, pp. 311–322, 02 2004.

[170] C. Low, Y. Chen, and M. Wu, “Understanding the determinants of cloud computing adoption,” *Industrial Management and Data Systems*, vol. 111, 08 2011.

[171] M. Böhm, S. Leimeister, C. Riedl, and H. Krcmar, “Cloud computing and computing evolution,” *Cloud Computing Technologies, Business Models, Opportunities and Challenges*, 01 2011.

[172] A. Sokri, “Optimal resource allocation in cyber-security: A game theoretic approach,” *Procedia Computer Science*, vol. 134, pp. 283–288, 01 2018.

[173] G. Kramer, M. Gastpar, and P. Gupta, “Cooperative strategies and capacity theorems for relay networks,” *IEEE Transactions on Information Theory*, vol. 51, no. 9, pp. 3037–3063 , 2005.

[174] H. S. Bedi and S. Shiva, “Securing cloud infrastructure against co-resident dos attacks using game theoretic defense mechanisms,” in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI ’12, (New York, NY, USA), p. 463–469, Association for Computing Machinery, 2012.

[175] M. Ni, A. K. Srivastava, R. Bo, and J. Yan, “Design of a game theory based defense system for power system cyber security,” *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 1049–1054 , 2017.

[176] R. M. Axelrod and W. D. Hamilton, *The evolution of cooperation*. New York, USA: Basic Books, 1984.

APPENDIX A

GAME THEORY FUNDAMENTALS

Game theory is a branch of applied math that analyzes interactive relationships between at least two parties, each with established preferences over outcomes [164]. It has found application in politics, gambling, biology, and economics. Game theory's link with economics is well established defining balance points, called equilibrium, strategic relationships, where an agent's actions have effects on others, and limited resources. Holding to many principles for various strategic circumstances, the 'games' in game theory do not necessarily exemplify sports or board games. They are not capable of informing a player what they should or should not want. Nor are they capable of providing options or moves which were not previously existent in the game space. It is also quite possible that if each player follows the tenets of the game theory, they will both reach inefficient outcomes [165].

The theory does, however, provide insight into the evaluation of agent objectives, utility quantification, and strategy profiles in a number of single-move and multi-move adversarial and cooperative interactions. In addition to adopting a specific vocabulary, it can deliver non-intuitive results. It has found application in several aspects of the cloud.

A.1. Traditional Game Theory

Game theory is an economic model applied to (1) strategy selection of (2) two or more players gaining some (3) utility in the form of quantifiable payoffs (see Table A.1). These three features imply competing interests for each rational entity seeking the highest utility, given the common knowledge of the strategies of the other player. They do not, however, imply complete knowledge.

Within game theory, players must avoid a dominated strategy. This is a strategy that leads to worse outcomes for that player than other available strategies, regardless of the strategy selected by another player/agent. In the traditional model, strategy is selected based on the concept of common knowledge: recursive shared knowledge *ad infinitum*.

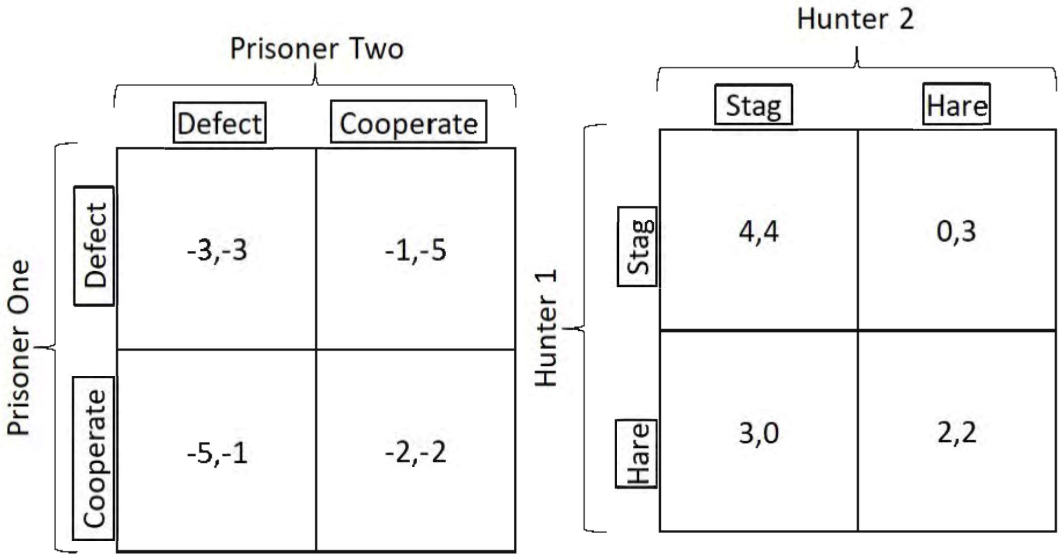
TABLE A.1
GAME THEORY COMPONENTS

Component	Element	Explanation
Players	Rationality	A player operates in his or her best interests.
Strategies	Inter-dependence	What one player does effects the outcomes of another player.
Pay-offs	Utility	Established by the utility function which assigns a numeric value to actions.

A.1.1) Game Theory in the Normal Form: As a central feature, game theory researchers seek a game's Nash equilibrium. This condition provides that no player, in this case, a cloud user, has an incentive to alter their behavior, given what the other user is doing. This concept can be applied in both a single-move game and in extended form games (covered in Section A.1.3).

The concept of a Nash equilibrium can be demonstrated by a pure strategy selected from the scoring in a normal-form game (see Fig. A.1a). This game is called the prisoners dilemma. The upper-left block (scored as -3,-3) is a Nash equilibrium: The vertical player is unable to unilaterally improve his score by moving down, while the horizontal player is unable to unilaterally improve her score by moving right. However, if both were able to cooperate and change their strategy simultaneously, they could achieve the more efficient score -2,-2) at the upper left, which is not a Nash Equilibrium. So, a Nash equilibrium can lead to inefficient outcomes.

Alternatively, a game may have multiple Nash equilibrium. In the stag hunt game (see Fig. A.1b.), the players can play Stag-Stag or Hare-Hare to achieve a Nash equilibrium. Less obvious is the third equilibrium which results from playing a probabilistic distribution of strategies (see Section A.11).



		Prisoner Two	
		Defect	Cooperate
Prisoner One	Defect	-3,-3	-1,-5
	Cooperate	-5,-1	-2,-2

(a)

		Hunter 2	
		Stag	Hare
Hunter 1	Stag	4,4	0,3
	Hare	3,0	2,2

(b)

Fig. A.1. Nash equilibrium examples. Players' are represented on the vertical and horizontal axes. By convention, the strategy profile and scores for player are positioned in the coordinate box as an ordered pair. The left number of the ordered pair is assigned to the vertical player and the second number is assigned to the vertical player. These numbers represent the utility for the coordinate strategy profile. a. A game with a single pure strategy Nash equilibrium (The prisoner's dilemma). Preferences over outcomes dictates that negative scoring is better for an agent closer to zero. Neither player is unilaterally able to improve from Defect-Defect. b. A multiple Nash equilibrium game. The stag hunt is an example of a game with more than one Nash equilibrium. The strategies of Stag-Stag as well as the Hare-Hare are both Nash equilibrium. A third

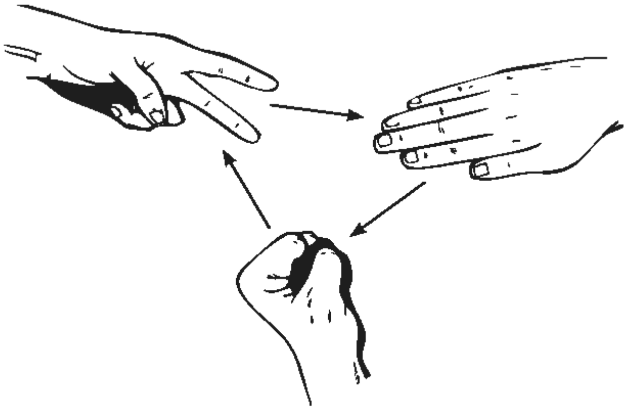
equilibrium in mixed strategies (see Appendix A.3) preserves the lemma that an odd number of Nash equilibrium are required.

A.1.2) Limits of Game Theory: The ability to decide on a pure or mixed strategy is subject to gaming in cloud security based on the actual value of any pay-off. While the scoring is dynamic and provided on the basis of cloud customer behavior, profile, or trends, these are subject to manipulation, obscuring, or malformation, often based on a central tenet of the game theory the concept of common knowledge.

Common knowledge, which forms the idea of each player being able to place themselves in each other's shoes, is reciprocated *ad infinitum*. It presents a cyclic basis for inference about the best strategies of each player and provides an incentive to alter beliefs. It is a subtle and permeating concept in extensive form games. These are covered in Sections A.1.3 and A.4. The lack of reliability of such scoring and strategic details in numeric scoring is suggested in related literature [60].

Lacking such measures of utility is an obstacle to adapting the classical game theory methods. While the number of possible strategies of players may be immense, without scoring, the iterated elimination of dominated strategies (IEDS) used to reduce the normal form is not possible. In non-cooperative games, it would be irrational to identify opponents.

A.1.3) Extended Game Theory: The game theory models presented above represent the normal form of simultaneous move games. In this model, players execute their moves at the same time, much like a game of paper-rock-scissors (see Fig. A.2). Neither player has a strategy that dominates nor is dominated, as the options are circular in precedence (see Fig. A.2(a). Furthermore, the game is symmetrical in payoffs across the draws on the diagonal (see Fig. A.2(b).



(a)

1 \ 2	P	R	S
P	1,1	2,0	0,2
R	0,2	1,1	2,0
S	2,0	0,2	1,1

(b)

Fig. A.2 Paper-Rock-Scissors. (a) The paper-rock-scissors strategies available for players one and two. These strategies are executed simultaneously, generally on a count of three. (b) The pay-offs of paper-rock-scissors in the normal form. These provide for a victory, loss, or draw for players one and two. Note, neither player has a dominate strategy.

Alternatively, a game can be expressed in the extended form. Rather than a tabular format, the strategies and payoffs are presented on a directed acyclic graph (DAG) (see Fig. A.3). In the case of the paper-rock-scissors game, it is still a simultaneous move game. The vertical dashed line relays this characteristic. Games with sequential moves are covered in Section A.4, Sub-Game Perfect Equilibrium and Creditable Threats.

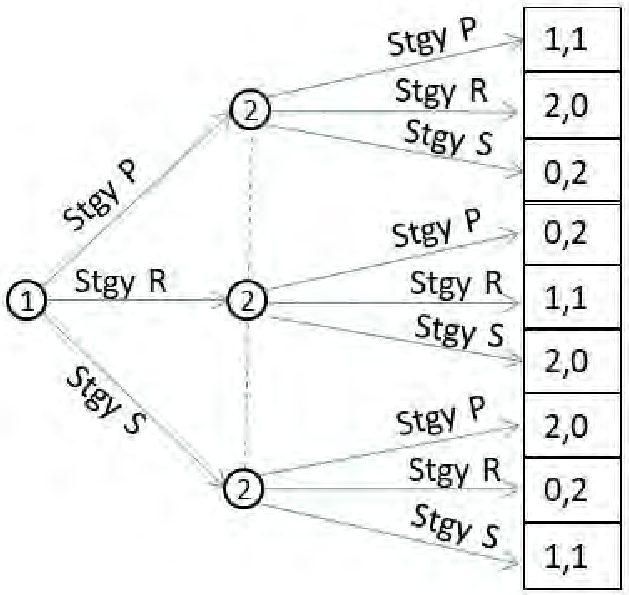


Fig. A.3: Paper-Rock-Scissors extended form. The paper-rock-scissors expressed in the extended form occurs on a directed acyclic graph. Note that the dashed line indicates a lack of knowledge by player two as to which node she is at. Nodes are numbered by the player making a move, strategies are denoted by the edges of the graph. Note, the normal form game may be developed from an extended form game, but not vice-versa.

A.2. Best Responses and the Nash Equilibrium

The Nash equilibrium can be described in several ways. One is the normal form format that fits the description that no player is able to unilaterally deviate from their current strategy and do better. Another is to say that a Nash equilibrium is a law that everyone would follow, even in the absence of an effective police force. This is in contradiction to that description that would make the prisoner’s dilemma in Fig. A.1a (see also Appendix F.2.2) an effective model. The final definitions were that an Nash equilibrium was a mutual best response. This section will develop the process, and define the alternative for when the test fails.

A.2.1) Game Description: In this game two craftsmen must make a simultaneous decision upon what tools they are going to buy. They are in an economy with a limited amount of wood, steel, and brass. Also, the two products that they can make are either a hinge or a latch. The constraints upon the use and availability of materials and capability to manufacture are:

Production of a hinge requires the ability to remove metal. The production of a latch requires the ability to shape metal. Neither the drill nor the welder require the use of wood to construct them. But they both require brass to construct. The hammer and the file both require wood to construct, but the hammer requires more of both, leaving the more malleable brass as the dominant material. Due to the available of materials and manufacturing processes, there are limited choices in what can be manufactured and constraints on profits. They are found in the pay-off matrix A.4.

A.2.2) Evaluation for Dominated Strategies: An examination of the pay-off matrix reveals that there are no dominant strategies. Neither the craftsmen number 1 nor craftsman number two have a strategy where they are able to do better, regardless of the strategy the opposing craftsman plays. There are further, no pure strategy Nash equilibrium.

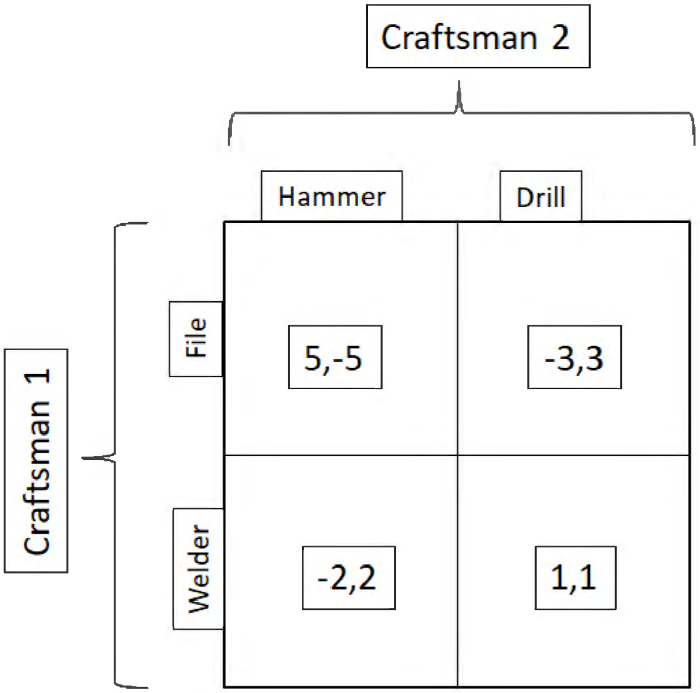
A.2.2.1) Craftsman 1:

If Craftsman 2 plays Hammer (Purchases a Hammer), the scores for Craftsman 1 are as on Table A.2.

TABLE A.2

PAY-OFFS, CRAFTSMAN 1 WITH THE PURCHASE OF A HAMMER BY CRAFTSMAN 2.

	Hammer
File	5,-
Welder	-2,-



	Hammer	Drill
File	5,-5	-3,3
Welder	-2,2	1,1

Fig. A.4: Mixed strategy Nash equilibrium. A game requiring a mixed strategy Nash equilibrium.

Likewise, if Craftsman 2 plays Drill (Purchases a drill), the scores for Craftsman 1 are as in Table A.3.

TABLE A.3
PAY-OFFS, CRAFTSMAN 1 WITH THE PURCHASE OF A DRILL BY CRAFTSMAN 2

	Drill
File	-3,-
Welder	1,-

Notice that the scores of the Craftsman 2 are not considered in this analysis. They are left with a blank (-), as the test for dominance only considers the player that is testing if they have a dominant strategy. The implications for Craftsman 1 are that if Craftsman 2 plays hammer, File beats Welder. But if Craftsman 2 plays Drill, Welder beats File. It is not necessary that the absolute pay-off beat the pay-off for the other strategy for both of the opposing player's strategy. -2 beats -3 for Craftsman 2 playing Hammer and Drill respectively. So, for Craftsman 1, the pay-off of strategy depends on the strategy that Craftsman 2 plays. Neither of the strategies dominates the other.

A.2.2.2) Craftsman 2: Conducting the same analysis, if Craftsman 1 plays File (Purchases a File), the scores for Craftsman 2 are as in Table A.4:

TABLE A.4
PAY-OFFS, CRAFTSMAN 2 WITH THE PURCHASE OF A FILE BY CRAFTSMAN 1

	Hammer	Drill
File	-,-5	- , 3

Similarly, if Craftsman 1 plays Welder (Purchases a welder), the scores for Craftsman 2 are as in Table A.5:

TABLE A.5
PAY-OFFS, CRAFTSMAN 2 WITH THE PURCHASE OF A FILE BY CRAFTSMAN 1.

	Hammer	Drill
Welder	-,2	- , 1

The implications for Craftsman 2 are congruent with those of Craftsman 1. If Craftsman 1 plays File, Drill beats Hammer. But if Craftsman 1 plays Welder, Hammer beats Drill. So, for Craftsman 2, the pay-off of strategy depends on the strategy that Craftsman 1 plays. Neither of the strategies dominates the other.

A.2.3) Evaluation for Pure Strategy Nash Equilibrium: Evaluation for a Nash equilibrium will be determined by the method of bests responses. A mutual best response will be an NE, which will meet the condition that no player can unilaterally improve their condition, given what the other player is doing. For each player, the best response will be marked.

A.2.3.1 Craftsman 1: If Craftsman 2 plays Hammer (Purchases a Hammer), the best response for Craftsman 1 is marked as the highest score. See Table A.6.

TABLE A.6
BEST RESPONSE, CRAFTSMAN 1 WITH THE PURCHASE OF A HAMMER BY CRAFTSMAN 2

	Hammer
File	(5),-
Welder	-2,-

Likewise, if Craftsman 2 plays Drill (Purchases a drill), the best response for Craftsman 1 are as in Table A.7.

TABLE A.7
BEST RESPONSE, CRAFTSMAN 1 WITH THE PURCHASE OF A DRILL BY CRAFTSMAN 2.

	Drill
File	-3,-
Welder	(1),-

A.2.3.2) Craftsman 2: Looking for the best responses of Craftsman 2, if Craftsman 1 plays File (Purchases a File) , the best response for Craftsman 2 is marked as in Table A.8.

TABLE A.8
BEST RESPONSE, CRAFTSMAN 2 WITH THE PURCHASE OF A FILE BY CRAFTSMAN 1.

	Hammer	Drill
File	-,-5	-,(3)

Similarly, if Craftsman 1 plays Welder (Purchases a welder), the best response for Craftsman 2 is marked as well

TABLE A.9
BEST RESPONSE, CRAFTSMAN 2 WITH THE PURCHASE OF A FILE BY CRAFTSMAN 1

	Hammer	Drill
Welder	-,(2)	- , 1

A.2.3.3) Mutual Best Responses: Testing for mutual best responses is checked by combing tables:

TABLE A.10
BEST RESPONSES FOR BOTH PLAYERS

	Hammer	Drill
Welder	(5),-5	-3,(3)
File	-2,(2)	(1),1

Neither player shares any bests responses within a strategy profile. Given what any given player is doing, the other player will always seek to find a different strategy. This lack of a pure strategy NE will be resolved through a probabilistic solution in the next section.

A.3. Mixed Strategy Calculation

The concept of finding a solution to a game without a pure strategy is found from the reverse circumstances. If one can imagine a game where there was complete and mutual knowledge and in order for one player to win, the other must necessarily lose and equal amount (a zero-sum game), it would soon become apparent that an impasse would form.

A.3.1) Concept of Mixed Strategy: Consider a game where players would split a deck of cards, each with an equal number of red cards and black cards. They will simultaneously place a card on the table. If the cards are both the same color, red or black, Player 1 will take a dollar from Player 2. If they are opposite colors, Player 2 will take a dollar from Player 1. The game matrix is shown below in A.11. But if both players were, for instance able to see a mirror that showed the opponent's card, they each would want to change their strategy.

TABLE A.11
A ZERO-SUM GAME. A ZERO-SUM GAME PROVIDES MODEL CHARACTERISTICS TO DEMONSTRATE THAT A MIXED STRATEGY CAN IMPOSE CONDITIONS THAT MAKE OPPOSING PLAYERS INDIFFERENT TO THE UTILITY OF A GIVEN STRATEGY.

	Black	Red
Black	$1, $-1	$-1, $1
Red	$-1, $1	$1, $-1

If, instead, Player One was to shuffle his cards to the point that he was unable to make any determination about the card from drawing from the middle of the deck, Player Two would be indifferent to knowledge of Player One's strategy. Regardless of the color of the card that Player Two put down, she could not have an expected utility greater than found by a random selection of a black or red card.

$$EU_2 = \$1\times .5+(-\$1\times .5) = 0 \qquad \text{(A.1)}$$

By the same token for player one, their expected utility would be the same if player two underwent the same randomization.

$$EU_1 = \$1\times .5+(-\$1\times .5) = 0 \qquad \text{(A.2)}$$

For both players, they can expect to both receive a utility of zero. This would make them indifferent to the strategy which the other player was going to play, as the other player could not possibly know what their move would be. The antithesis of the dominated strategy, both would adopt the same means of measuring their expected utility.

A.3.2) Application of the Mixed Strategy: Consider the game played between the two craftsmen in Fig. A.4. If Craftsman Two were to find a strategy which would make Craftsman One indifferent to his strategy selection, it would be picked with a probability which provided an equal expected utility for Craftsman One's strategies. For the up strategy

$$EU_{1up} = 5\times p_{left} +(-3\times q_{left}) \qquad \text{(A.3)}$$

For playing down:

$$EU_{1down} = (-2\times p_{left})+1\times q_{left} \qquad \text{(A.4)}$$

In these equations q_{left} is the same as p_{right}. However:

$$p_{left} + q_{left} = 1 \qquad \text{(A.5)}$$

or:

$$q_{left} = 1 - p_{left} \qquad \text{(A.6)}$$

Establishing the condition of indifference in the EU of Craftsman 1:

$$EU_{1up} = EU_{1down} \quad \text{(A.7)}$$

For playing down:

$$5\times p_{left} + (-3\times q_{left}) = (-2\times p_{left}) + 1\times q_{left} \qquad \text{(A.8)}$$

Substituting equation A.6:

$$5\times p_{left} + (-3\times(1 - p_{left})) = (-2\times p_{left}) + 1\times(1 - p_{left}) \qquad \text{(A.9)}$$

Collecting terms on each side:

$$-3 + 8\times p_{left} = -3\times p_{left} + 1 \qquad \text{(A.10)}$$

Reducing terms:

$$11\times p_{left} = 4 \qquad \text{(A.11)}$$

Or:

$$p_{left} = 4/11 \qquad \text{(A.12)}$$

Also:

$$p_{right} = 7/11 \qquad \text{(A.13)}$$

A similar set of calculations applies to Craftsman One:

For playing left:

$$EU_{2left} = -5\times p_{down} + 3\times q_{down} \quad \text{(A.14)}$$

For playing right:

$$EU_{2right} = 2\times p_{down} + 1\times q_{down} \quad \text{(A.15)}$$

Also:

$$q_{down} = 1 - p_{down} \quad \text{(A.16)}$$

Establishing the condition of indifference in the EU of Craftsman 2:

$$EU_{2left} = EU_{2right} \quad \text{(A.17)}$$

Which leads to:

$$-5\times p_{down} + 3\times q_{down} = 2\times p_{down} + 1\times q_{down} \qquad \text{(A.18)}$$

Substituting equation A.16:

$$-5\times p_{down} + 3\times(1- p_{down} = 2\times p_{down}) + 1\times(1- p_{down}) \quad \text{(A.19)}$$

Collecting terms on each side:

$$3-8\times p_{down} = p_{down} + 1 \text{ (A.20)}$$

Reducing terms:

$$9\times p_{down} = 2 \qquad \text{(A.21)}$$

Or:

$$p_{down} = 2/9 \qquad \text{(A.22)}$$

Also:

$$p_{up} = 7/9 \qquad \text{(A.23)}$$

Therefore, the Craftsman 1 would play strategies of up with probability 7/9 and down with probability 2/9. Craftsman 2 would play strategies of left with probability 4/11 and right with probability 7/11. The mixed strategy method can be applied in a variety of strategic situations; from pitching and hitting in a baseball game, to political negotiations. In this example, the industrial relationship was selected as it relates to an economic environment. It serves as a basis for how game theory can apply in more complex relationships.

A.4. Sub-Game Perfect Equilibrium and Creditable Promises

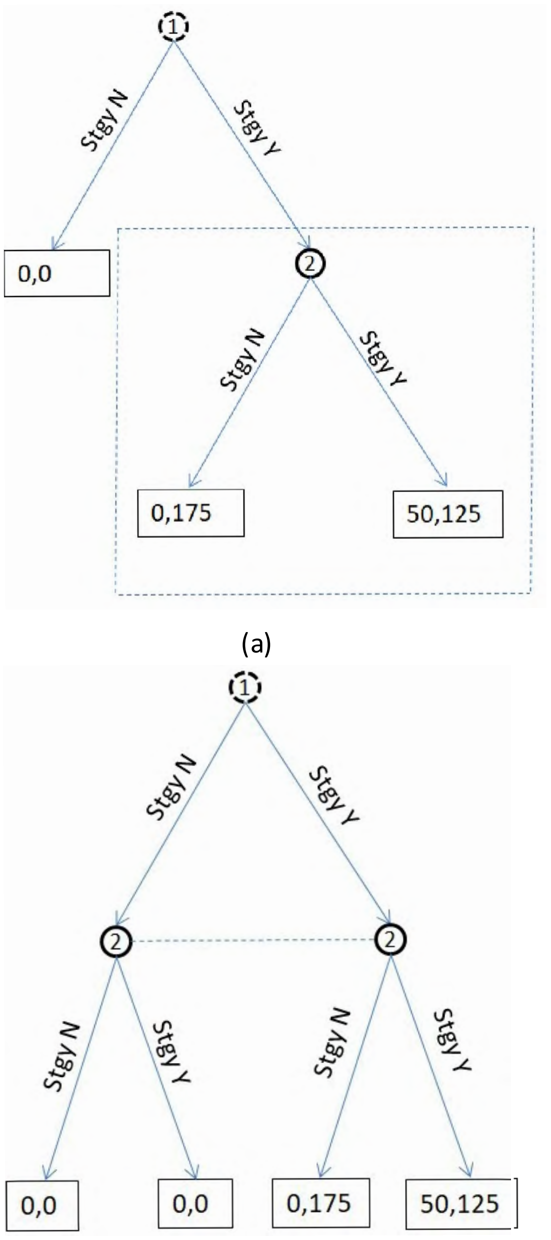
Problematic factors arise in using game-theory in cloud computing. Each cloud user is incentivized to operate in their own best interest. While they may be able to increase their own utility by promising to providing information that helps all of the cloud user, they may not do so when it comes time to actually provide it. They will not have incentive to do if they are able to increase their own utility.

Consider the case where the Craftsman One wants to build benches, which require both metal working tools and wood working tools. The wood costs $100 and the metal costs $125, but Craftsman 2 only has $75. She promises to evenly split the $400 earnest money once the materials are purchased. The welder and file is necessary to build the frame of bench and seat mounts before the wood seats can be attached. Should the Craftsman One put forward the remaining $150?

Once Craftsman Two receives the earnest money, she is at a sub-game, marked by the dashed lines. She has a choice to either follow through with her promise to split or to only provide Craftsman One the $150 in return. This will result in the extended game in Fig. A.5.

The resolution of the question of a creditable promise in the subgame of Fig. A.5a is to represent the simultaneous game in Fig. A.5b in the normal form. Doing so demonstrates that Craftsman Two has no incentive to follow through on their promise. This is a commitment problem, and in the absence of an effective police force, Craftsman Two may have incentive to recompense Craftsman One even less.

A similar set of circumstances exist in the cloud game-theory methods of security. A cloud user may promise to provide accurate information, and they may even benefit from doing so. But is a subgame, and the scores are indicated in the Fig. b. To represent this in a normal form game, the scores for both craftsmen are represented the same way that the paper-rock-scissors game represents a simultaneous move game. In this case, Craftsman Two does not know where she is at when Craftsman One is at the initial node.



(a)

(b)

Fig. A.5. Extended game with sub-game. (a) At the initial node Craftsman One decides whether to enter into the business deal with Craftsman Two. If he takes the No Strategy (Stgy N) the game ends at 0,0. But if he goes with the Yes Strategy (Stgy Y), Craftsman Two will then decide whether to renege on their promise or to honor their promise. Thus once any other player commits to providing security information, they lose incentive to do so. (b) Player two is then unable to determine where they are in the game. The outcome is that no normal form game translation is possible.

A.5. Grid Uncontrolled Intersection Game

An idealized model of traffic flow in an uncontrolled street modeled as a agents acting in their own best interest seeking a resource demonstrates patterns of population density. Without any direct aggression, the attempt to maneuver into a more advantageous position tends to diminish the utility to the aggregate. This is an example of the 'tragedy of the common.' The model is an extension of the uncontrolled intersection in Fig. 1.1 in Fig. A.6.

At a given intersection each direction of traffic flow, up and to the left, are considered without flow in the opposite direction. Acceleration is not taken into account and each vehicle with an arrow is moving at a constant speed outside of the intersection. At the threshold and inside of the intersection, each driver is operating in their own best interest.

In the top panel a single intersection is illustrated for patterns that develop. Cars queue into the intersection at a constant rate. All cars in the queue to cross the intersection are speed regulated by maintaining an arbitrary constant distance. The arrows (in blue) mark the constant buffer speed all cars are operating to prevent a collision outside of the intersection. Preferences (utilities) are determined by maintaining a blue arrow (+) or a red arrow (-). Players, pay-offs, and the strategic elements necessary for a game environment are present.

Panel 1. The first car at the threshold moving upward (1) passes through the intersection. As this car exits the intersection, the cars following move into position at the threshold, and another car enters the queue. As the car moving from bottom to top enters the queue, it must revert to follow the rule of maintaining constant distance. In that time, the car at the threshold moving from right to left (*1*) must clear the intersection.

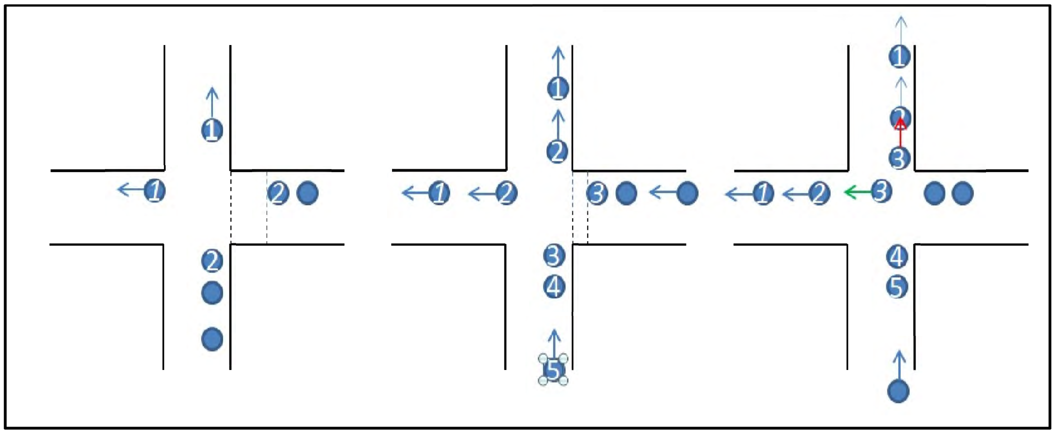
Panel 2. While waiting in the queue, any driver loses their blue arrow. As they reach the threshold, they seek to leave the intersection within a shorter interval. Because the cars in queue are limited by the distance between them, and not by position they will shorten time at the threshold by reducing its width (*2*), and increase acceleration through the intersection. The interval between the last car in the queue and the next car entering increases, between 3 and 4.

Panel 3. At the limit of this idealized pattern of increase, the last car traveling from bottom to top (3) must exit the intersection prior to the next car moving from right to left (*3*) can enter the intersection, they are bound by the arrow not being able to intersect the car in front of them. Two consequences occur from this pattern.

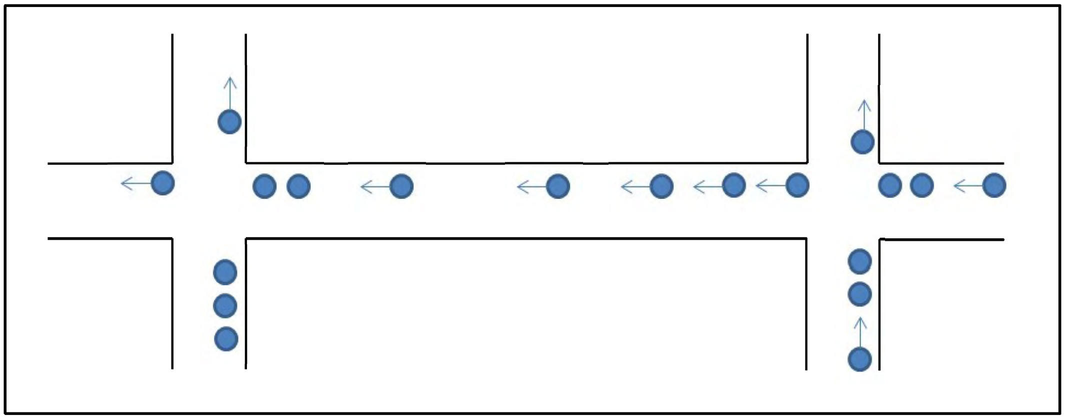
- 1. Cars bunch up at the exit of the intersection, regardless of direction of travel.
- 2. The queue empties, and the pattern begins again.

Within a grid of uncontrolled intersections, the grouping of vehicles at the exit of an intersection establishes a phased wave pattern that is inverted from the constant traffic pattern. The phenomena is not limited to intersections. Most notably, a circular roadway with cars operating with no competitive aspect will form wave traffic patterns due to driver effort to maintain distance [166]. In competitive environments, the phenomena is controlled by deliberate grouping [167].

While several aspects of traffic management can be modeled as game theory [168] the consideration here is aligned to the uncontrolled transient sharing of resources. Game-theory predicts semi-periodic or periodic fluctuation in population density. The model aligns well with the biphased populations in the Lotka-Voltarra predator-prey model in Chapter 2.



(a)



(b)

Fig. A.6. Uncontrolled intersection population density. Dynamics and results from idealized intersection without control mechanisms for right of way. (a) Individual intersections resolve to increased rate of traffic resulting from drivers acting in their own self interests. (b) The larger grid resolves to wave patterns of traffic density.

APPENDIX B

OWASP Top 10 2021

- 1 Broken Access Control. e.g. Accessing API with missing access controls for POST, PUT and DELETE.
- 2 Cryptographic Failures. e.g. This concerns protocols such as HTTP, SMTP, FTP also using TLS upgrades like STARTTLS.
- 3 Injection. e.g. Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.
- 4 Insecure Design. e.g. lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.
- 5 Security Misconfiguration. e.g. Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- 6 Vulnerable and Outdated Components. e.g. If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- 7 Identification and Authentication Failures. e.g. not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.
- 8 Software and Data Integrity Failures. e.g. objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.
- 9 Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

– Auditable events, such as logins, failed logins, and high-value transactions, are not logged.

– Warnings and errors generate no, inadequate, or unclear log messages.

– Logs of applications and APIs are not monitored for suspicious activity.

– Logs are only stored locally.

– Appropriate alerting thresholds and response escalation processes are not in place or effective.

– Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.

– The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

You are vulnerable to information leakage by making logging and alerting events visible to a user or an attacker (see A01:2021-Broken Access Control).

- 10 SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

source https://owasp.org/Top10/

APPENDIX C

ALTERED GAME-THEORY MODELS

The extended time population distribution of altered game theory models demonstrated a cyclical, out of phase population of both the predator and prey species. These exhibited a fluctuation pattern that was less wide than the predator prey models without any kind of discernible intelligence on the part of the prey. They also demonstrated a perceivable rougher fluctuation.

An adaptation of the victim to the attacker in this way precedes a discussion of mutation in the cloud. The loss of contact with an opposing population leaves parties to their own characteristic integration. Reproduction, competition, and life-cycle phases (see Fig. 2.7.1) provides incentives to mutate in the evolutionary game-theory model (see Appendix F).

C.1. Game Theory Predation Model Metric Variation 1

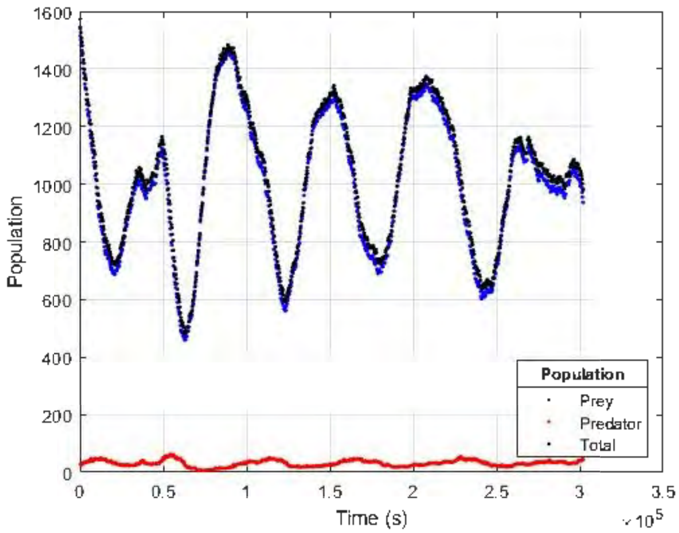


Fig. C.1. Game Theory Predation Model Metric Variation 1. Population fluctuation for predator and prey with reproduction probability at 0.02249. Representation in the cloud

resulted in 239 complete workloads, energy consumption of 1737.54 kWh, 996087 VM migrations, 51608 host shutdowns and 2807.02 seconds mean time before shutdown.

C.2. Game Theory Predation Model Metric Variation 2

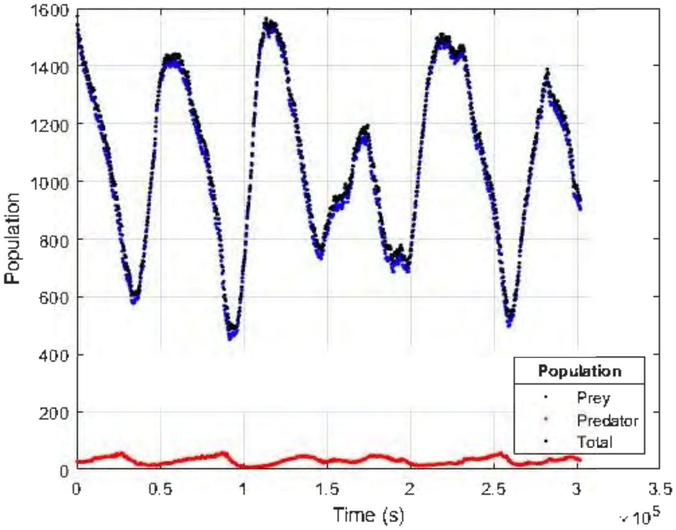


Fig. C.2. Game Theory Predation Model Metric Variation 2. Population fluctuation for predator and prey with reproduction probability at 0.022. Representation in the cloud resulted in 235 complete workloads, energy consumption of 1771.44 kWh, 1049386 VM migrations, 47228 host shutdowns and 3110.02 seconds mean time before a shutdown.

APPENDIX D

CONTAINER SECURITY ARCHITECTURE

D.1 Name-Space Descriptions

- Mount for file system isolation; CLONE_NEWNS; isolates mount points. Mount namespaces provide isolation of the list of mounts seen by the processes in each name-space instance. Thus, the processes in each of the mount name-space instances will see distinct single-directory hierarchies.
- UTS for hostname and domain name isolation; CLONE_NEWUTS; isolates host name and NIS domain name. UTS name-spaces provide isolation of two system identifiers: the hostname and the NIS domain name. These identifiers are set using sethostname(2) and setdomainname(2), and can be retrieved using uname(2), gethostname(2), and getdomainname(2). Changes made to these identifiers are visible to all other processes in the same UTS namespace, but are not visible to processes in other UTS name-spaces.
- IPC for IPC and message queue isolation; CLONE_NEWIPC; Each IPC name-space has its own set of System V IPC identifiers and its own POSIX message queue filesystem. Objects created in an IPC name-space are visible to all other processes that are members of that name-space, but are not visible to processes in other IPC name-spaces.
- PID for process ID isolation; CLONE_NEWPID; isolates process IDs. PID name-spaces isolate the process ID number space, meaning that processes in different PID name-spaces can have the same PID. PID name-spaces allow containers to provide functionality such as suspending/resuming the set of processes in the container and migrating the container to a new host while the processes inside the container maintain the same PIDs. PIDs in a new PID name-space start at 1, somewhat like a standalone system, and calls to fork(2), vfork(2), or clone(2) will produce processes with PIDs that are unique within the name-space.

Network for network resource isolation; CLONE_NEWNET; isolates Network devices, stacks, ports, etc. Network name-spaces provide isolation of the system resources associated with networking. A physical network device can live in exactly one network name-space. When a network name-space is freed, that is, when the last process in the name-space terminates, its physical network devices are moved back to the initial network name-space (not to the parent of the process).

- User for UID/GID isolation; CLONE_NEWUSER, isolates user and group IDs. User namespaces isolate security-related identifiers and attributes, in particular, user IDs and group IDs (see credentials(7)), the root directory, keys (see keyrings(7)), and capabilities (see capabilities(7)). A process's user and group IDs can be different inside and outside a user name-space. In particular, a process can have a normal unprivileged user ID outside a user name-space while at the same time having a user ID of 0 inside the name-space.
- Cgroup for control group isolation; CLONE_NEWCGROUP, When a process creates a new cgroup name-space using clone(2) or unshare(2) with the CLONE_NEWCGROUP flag, its current cgroups directories become the cgroup root directories of the new name-space.
- Time for clock time isolation; CLONE_NEWTIME; isolates hostname and NIS domain name.

D.2 Control Groups Descriptions

- blkio — this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, or USB). Two policies are available. The first is a proportional-weight time-based division of disk implemented with CFQ. This is in effect for leaf nodes using CFQ. The second is a throttling policy which specifies upper I/O rate limits on a device. CONFIG_BLK_CGROUP
- cpu — This subsystem uses the scheduler to provide cgroup tasks access to the CPU. Cgroups can be guaranteed a minimum number of "CPU shares" when a system is busy.

This does not limit a cgroup’s CPU usage if the CPUs are not busy.

CONFIG_CGROUP_SCHED

- cpuacct — This subsystem provides accounting for CPU usage by groups of processes. It generates automatic reports on CPU resources used by tasks in a cgroup.

CONFIG_CGROUP_CPUACCT

- cpuset — this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup. This cgroup can be used to bind the processes in a cgroup to a specified set of CPUs and NUMA nodes. CONFIG_CPUSETS
- device — This subsystem allows or denies access to devices by tasks in a cgroup. This supports controlling which processes may create (mknod) devices as well as open them for reading or writing. The policies may be specified as allow-lists and deny-lists. Hierarchy is enforced, so new rules must not violate existing rules for the target or ancestor cgroups. CONFIG_CGROUP_DEVICE
- freezer — this subsystem suspends or resumes tasks in a cgroup. The freezer cgroup can suspend and restore (resume) all processes in a cgroup. Freezing a cgroup /A also causes its children, for example, processes in /A/B, to be frozen.

CONFIG_CGROUP_FREEZER

- hugetlb — This supports limiting the use of huge pages by cgroups.

CONFIG_CGROUP_HUGETLB

- memory — This subsystem sets limits on memory use by tasks in a cgroup and generates automatic reports on memory resources used by those tasks. The memory controller supports reporting and limiting of process memory, kernel memory, and swap used by cgroups.

CONFIG_MEMCG

- net_cls — this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task. This places a classid, specified for the cgroup, on network packets created by a cgroup. These classids can then be used in firewall rules, as well as used to shape traffic using tc(8).

This applies only to packets leaving the cgroup, not to traffic arriving at the cgroup.

CON-FIG_CGROUP_NET_CLASSID

net_prio — this subsystem provides a way to dynamically set the priority of network traffic per network interface. This allows priorities to be specified, per network interface, for cgroups. CONFIG_CGROUP_NET_PRIO

- ns — the namespace subsystem. As described above.
- perf_event — this subsystem identifies cgroup membership of tasks and can be used for performance analysis. This controller allows perf monitoring of the set of processes grouped in a cgroup. CONFIG_CGROUP_PERF
- pid This controller permits limiting the number of process that may be created in a cgroup

(and its descendants). CONFIG_CGROUP_PIDS

- rdma The RDMA controller permits limiting the use of RDMA/IB-specific resources per cgroup. CONFIG_CGROUP_RDMA

APPENDIX E
Top Tens

E.1 THE OWASP Serverless TOP 10, 2017

- 1 Injection. If a function is triggered via email or a database, there is nowhere to put a Firewall or any other control that will validate the event.
- 2 Broken Authentication. Attackers will try to look for a forgotten resource, like a public cloud storage, or open APIs.
- 3 Sensitive Data Exposure Most of the methods used in traditional architectures, such as stealing keys, performing man-in-the-middle (MitM) attacks and stealing readable data at rest or in transit, still apply to serverless applications. However, the data sources might be different.
- 4 XML External Entities (XXE) In serverless, executing remote requests (OOB) might not be possible if the function is running inside the internal virtual private network (VPC). Scanning will be less likely to take effect in the few seconds the function has and DoS attacks are less of a concern, because the function is running in a designated container which will affect only the current execution.
- 5 Broken Access Control Attackers will target over-privileged functions in order to gain unauthorized access to resources in the account rather than having control over the environment.
- 6 Security Misconfiguration Unused pages are replaced with unlinked triggers, unprotected files and directories are changed to public resources, like public buckets. Attackers will try to identify misconfigured functions with long timeout or low concurrency limit in order to cause a Denial of Service (DoS).

7 Cross-Site Scripting (XSS) The source of traditional XSS attacks are usually databases or reflective inputs. While in serverless they could also originate from different sources like emails, cloud storage, logs, IoT and others.

- 8 Insecure Deserialization Dynamic languages like Python and NodeJS, together with the common use of JSON, a serialized data type, could make deserialization attacks a little more common in the serverless world.
- 9 Using Components with Known Vulnerabilities To be able to execute the desired tasks, functions make use of many dependencies and 3rd-party libraries. Vulnerability introduced by the supply chain is one of most common risks these days and attackers will target code that makes use of vulnerable libraries as an entry point to the application.
- 10 Insufficient Logging and Monitoring. The fact that serverless auditing is now even more difficult than in traditional applications, where we use our own logging system, and not the one provided by the infrastructure, just makes it easier for the attackers.

E.2 The Serverless Architectures Security Top

It is clear that there are trade-offs to developing security in the serverless structure. A shortlived execution cycle limits the ability to launch persistent attacks. But at the same time, the fine granularity and inter-connectivity tends to expand the attack surface. [1]. Several problem areas have been identified for examination at a technical level E.1. On the other hand other studies suggest a change in the culture surrounding serverless computing [143].

TABLE E.1
THE SERVERLESS ARCHITECTURES SECURITY TOP 10[1], [2]

Number	Threat
SAS-1	Function Event Data Injection
SAS-2	Broken Authentication
SAS-3	Insecure Serverless Deployment Configuration
SAS-4	Over-Privileged Function Permissions & Roles
SAS-5	Inadequate Function Monitoring and Logging
SAS-6	Insecure 3rd Party Dependencies
SAS-7	Insecure Application Secrets Storage
SAS-8	Denial of Service & Financial Resource Exhaustion
SAS-9	Serverless Function Execution Flow Manipulation
SAS-10	Improper Exception Handling and Verbose Error Messages

APPENDIX F

FUNCTIONS-as-a-SERVICE EVOLUTION

F.1. Evolution of the Serverless and Parallel Threats

The establishment of the cloud was a progressive development. Grid and point-to-point (P2P) networks existed prior and remote servicing was a common business model [169]. Virtualization has existed since mainframe computing as an economic measure [170]. Virtualization in the cloud developed in an evolutionary manner [171] and the security considerations right along with it [118].

The innovation of serverless computing is also an evolutionary change [133]. It's economic development [90] emerged from a unique virtualization environment [113]. Serverless computing altered the stable landscape, introducing a new equilibrium.

F.2. The Evolution Game Theory Model of FaaS

Game Theory has been applied to almost every aspect of cloud computing. Networking, power, memory, cyber-security [63], resource allocation [172], and co-resident threats [62]. Many, if not all models of game theory have also been applied: signaling games [64], cooperative games: [173] , zero-sum [174], and Stackleberg [175].

The implementation of these methods has been shown to face the challenge of inadequate scoring [172]. The stochastic simulation in the first part of this study is one method of addressing this matter (see 2). Game-theory is plainly not limited to the case of security. But again, the utility of any given agent is not easily obtained, their rationality is subject to question, and a precise description of their available strategies is evasive.

F.2.1) Evolutionary Game Theory: An adaption of game theory can operate unconstrained by rationality and expands the applicability of utility to abstract quantities. This method considers repeated iterations of game play, but not in terms of turns, such as extended forms. Strategies are adaptive and, rather than holding fixed positions, the games are played as encounters.

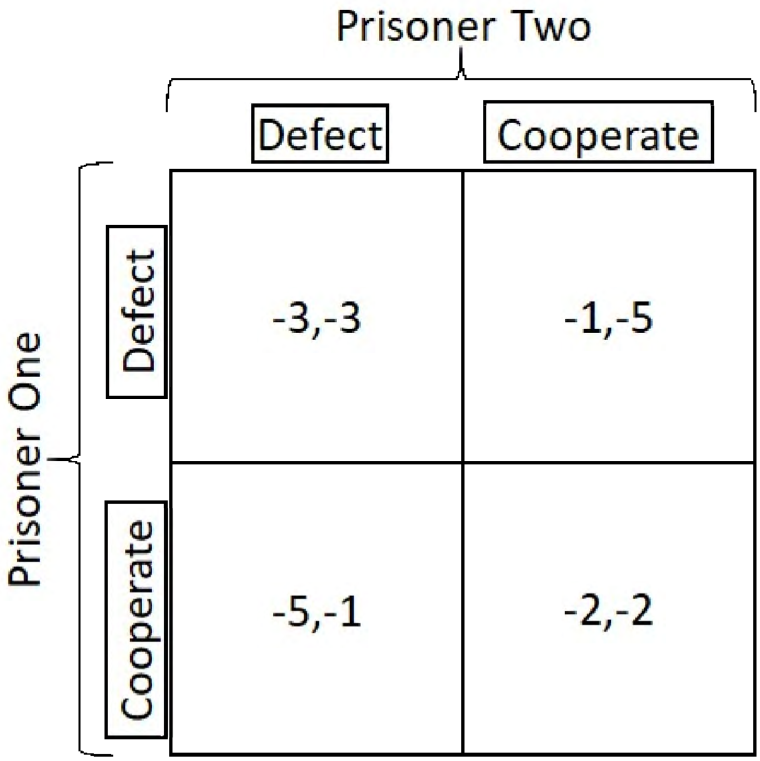
Evolutionary game theory is applicable in biological, economic, and social sciences, as well as technical studies. It defers the common knowledge requirement assumed by traditional game theory and contemplates an alternative set of assumptions. The rational player evaluation of a pay-off matrix is replaced by mutations occurring as environmental adaptations.

Evolutionary game theory promotes a different notion of evolution than survival of the fittest. It instead considers the ability to pass on characteristics to following generations and rewards proliferation. Such a mechanism provides for the more optimal usage of resources, redefines the relationship between players, and trends to elimination of the zero-sum game.

F.2.2) Example One: The Iterated Prisoner's Dilemma: The prisoner's dilemma is one of the most common and well studied games in game theory. It exemplifies the significance and structure of the Nash Equilibrium in the same form as the uncontrolled intersection and stag hunt games. In its iterated from, it demonstrates one of the evolutionary game theory principles that supports the serverless FaaS environment.

The components of the game consider two criminals that the police catch in the act of minor offense. The police suspect them of being in the act of a more serious charge, but are unable to prove it without the confession and implication of one or both. The prisoners are separated and each informed that they will receive a reduced sentence for defecting on their accomplice. But, if the other confesses and defects, they will receive the maximum punishment. Both can confess/defect, but the value of the confession will reduce the benefit of their own confession. The pay-off matrix is in Fig. F.1.

The prisoner's dilemma solution is for each of them to play defect, or defect/defect, which is a Nash Equilibrium. Neither player is able to unilaterally improve their position by changing their strategy. Had both chosen, and more significantly adhered to, cooperate/cooperate, a more efficient outcome would result. However, either party could then change their strategy to defect. Altering the game to repeated interactions presents the opportunity to incorporate mutable strategies.



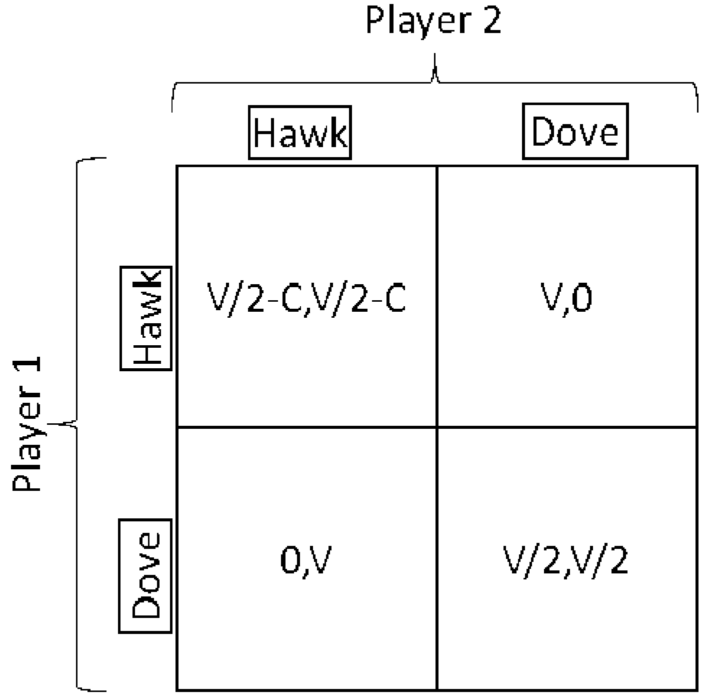
		Prisoner Two	
		Defect	Cooperate
Prisoner One	Defect	-3,-3	-1,-5
	Cooperate	-5,-1	-2,-2

Fig. F.1. The payoff matrix for the prisoner’s dilemma. Two would-be thieves are apprehended by law enforcement and held in separate interrogation rooms. Prosecutors believe that they were in the process of committing a greater crime, but are not able to prove it. The police make an offer to each of the prisoners: confess to the greater crime for a more lenient sentence which will implicate the prisoner’s partner for the maximum sentence. If both prisoners confess, the value of the confession is less, and they both receive medial punishment. The scores in the payoff matrix are arranged into boxes which correspond to the strategies indexed by row and column. Within the box, the left digit applies to the vertical strategies. Separated by a comma, the right digit applies to the horizontal player’s strategies.

Rather than conducting the prisoner’s dilemma a single round of play, the game can be conducted multiple times, starting with each player similarly situated. For a single instance, the strategies of defect/defect do achieve the only Nash equilibrium in the normal form. This is not, however, the most efficient outcome.

This game was analyzed by pitting different algorithms against each other [176]. Going through several hundred iterations, a set of scores developed for entrants. This revealed a very different result than the pure strategy result of defect. By starting off in a cooperative strategy and playing that strategy until the other player defected, a kind of reprisal system was found to be most efficient of the contest. If, on some subsequent move, a the opposing player cooperated, the strategy was to revert to cooperation.

F.2.3) Example Two: Hawk and Dove: An alternative model in evolutionary game theory two agents assuming the roles of hawk and dove. In this game a contested resource can be obtained based on the aggressive or passive profiles chosen by agents. A hawk profile may encounter another hawk and fight for the resource until one is injured, from which the winner will take the resource. Each is equally likely to win in any given encounter, however, they both assume the cost of injury. Two doves will encounter each other and posture until both split the value of the resource. Finally, when the hawk meets the dove, it simply takes all of the resource while the dove gets nothing. A pay-off matrix for this game is in Fig. F.2.



		Player 2	
		Hawk	Dove
Player 1	Hawk	V/2-C,V/2-C	V,0
	Dove	0,V	V/2,V/2

Fig. F.2. The payoff matrix for the hawk and dove game

Characteristic of the hawk and dove game is that rather than numeric values, V for ’value’ and C for ’cost’ are in the pay-off matrix. The solution to the hawk and dove game is dependent, not on the actual values of the V and C, but rather the events of players meeting. Although a hawk meeting a dove will result in zero pay-off for the dove, relationship where V/2-C is less than zero will still be less than that of zero when a hawk meets a dove. Additionally, the V/2 pay-offs from when a dove meets a dove does not include any cost, C, due to damage. In the environment that is largely populated by hawks, a local region of higher scoring can exist only for doves.

Once again, cyclic changes emerge in this uncontrolled environment, unless there is a definitive means of ensuring the value of V/2 is greater than C. Even though the hawks are stronger, two doves are able to build stable cooperation due to losing nothing in cooperation. Such an environment is only stable as the collective score is able to maintain a balance with the cost at the collective boundaries. A single instance of a mutation to hawk within the boundary also risks loss of the benefit. These conditions are called an evolutionary stable strategy, which are covered in more detail in Appendix F.2.5.

F.2.4) Games Promoting Serverless Models: Certain aspects of the iterated prisoner's dilemma can be aligned with the serverless models. The specter of vendor lock-in creates a strategically binding scenario between the provider and the user. However, the applicability of the extended game maybe diminished in scope due to the Nash equilibrium's alternative definition. An NE is also described as a rule that all agents in a game would follow in the absence of an effective police force. This would contradict the ability of the police to hold any suspect at all. Also this strategy mutates into its descendant, altruistic tit-for-tat, which forgives some instances of defect. Upon analysis, this strategy mutated to unconditional cooperation across a wider range of environments. Again, a cyclic rise and fall of populations would occur as incentive to defect began to form.

The serverless evolution can also be modeled in the hawk and dove game. It is not necessary that the serverless cloud require complete homogeneity e.g. all hawks and doves. Rather, it is sufficient that an even dispersion of agents is present and the population can include both hawks and doves. The fine grained, ability to scale to infinity in the serverless cloud provides the effect of a homogeneous environment necessary for the application of this model of evolutionary game theory. Therefore, two or more cooperating players are able to build a locally stable region of higher scoring than an environment largely populated by hawks.

The applicability of this model is also limited, though for a different reason than the prisoner's dilemma. First, a local region of stability is accelerated by the ratio of internal area to external circumference. So, the internal region will generate higher pay-offs faster than the ability to control those pay-offs by the hawks. Although this may appear to be a good thing, the single instance of a mutation to hawk within the region of doves will result in elimination of doves due to hawks. Once again, a cyclic pattern emerges.

F.2.5) Alternative Game Theory in the Cloud: Evolutionary Stability in FaaS: A lack of common knowledge, disjoint scoring and utility, and absence of rationality is not entirely prohibitive of the adoption of game theory in the cloud. From both an ecological perspective and from a security perspective, cloud computing development can be modeled without the alignment of payoffs on a normal form game. Players need not be rational in the sense that their utility maximization is the only objective.

F.2.5.1) Players: While evolutionary game theory allows for the designation of predator and prey, it adopts a different means of describing the interaction that goes beyond conflict. In the game theory model, a set of players is considered to be identical within an environment, but are subject to mutation. That mutation is then capable of a different strategy in its interactions and use of resources. A successful mutation will have the ability to carry on to future generations.

F.2.5.2) Strategies: In the evolutionary game theory, the development of characteristics is expanded to community members. This aspect permits the growth of altruistic characteristics transferring to the proliferation of the species. The traditional competition gives way to the interactions that occur when agents meet each other.

F.2.5.3) Pay Offs: The ability to measure a score as a reward for the behavior of a player is subservient in the evolutionary game-theory model to the proliferation of the species. A functional relationship is further probabilistic as moves of nature are considered due to the extended form characteristic of the game. Additionally, the division of benefits and cost is more fungible to the collective.

F.2.6) Example: An altered model of the predator-prey game need not only adjust the parameters that characterize the individual and have implications for the collective. A victim may have the ability to issue an alert upon encountering an attacker. While the tendency may be to hide and avoid detection, instead a victim then announces their presence to the attacker. The alert places the victim at greater danger.

Such an altruistic behavior sets the conditions for the proliferation of the species. The benefit to the group is that the prey is able to evaluate the danger to the group upon the meeting of a single individual. This may not require an eminent sacrifice on the part of that individual, but it places them in greater contrast to the surroundings observed by the attacker. The prey group is then able to adapt to additional mutations, such as specialization of look-outs, evasions connected with alerting mechanisms, and signal passing which attenuates the vulnerability of the original alert givers risk.

Of course, the attacker develops their own mutations. These may be related to the ability of the prey to issue an alert directly or indirectly. An alert may be intercepted by more than the original attacker. Alerts may imply that the volume of target victims is enough to amass attackers. Alerts are subject to falsification.

These are all factors that become organic to each of the component agents in an environment. Rather than becoming a rational choice, the behavior becomes characteristic and instinctive in a complex system. The cloud is capable of that degree of complexity. This will appear in the adaption of more intricate service models.

F.3 Evolutionary Game Theory and the Emergence of FaaS

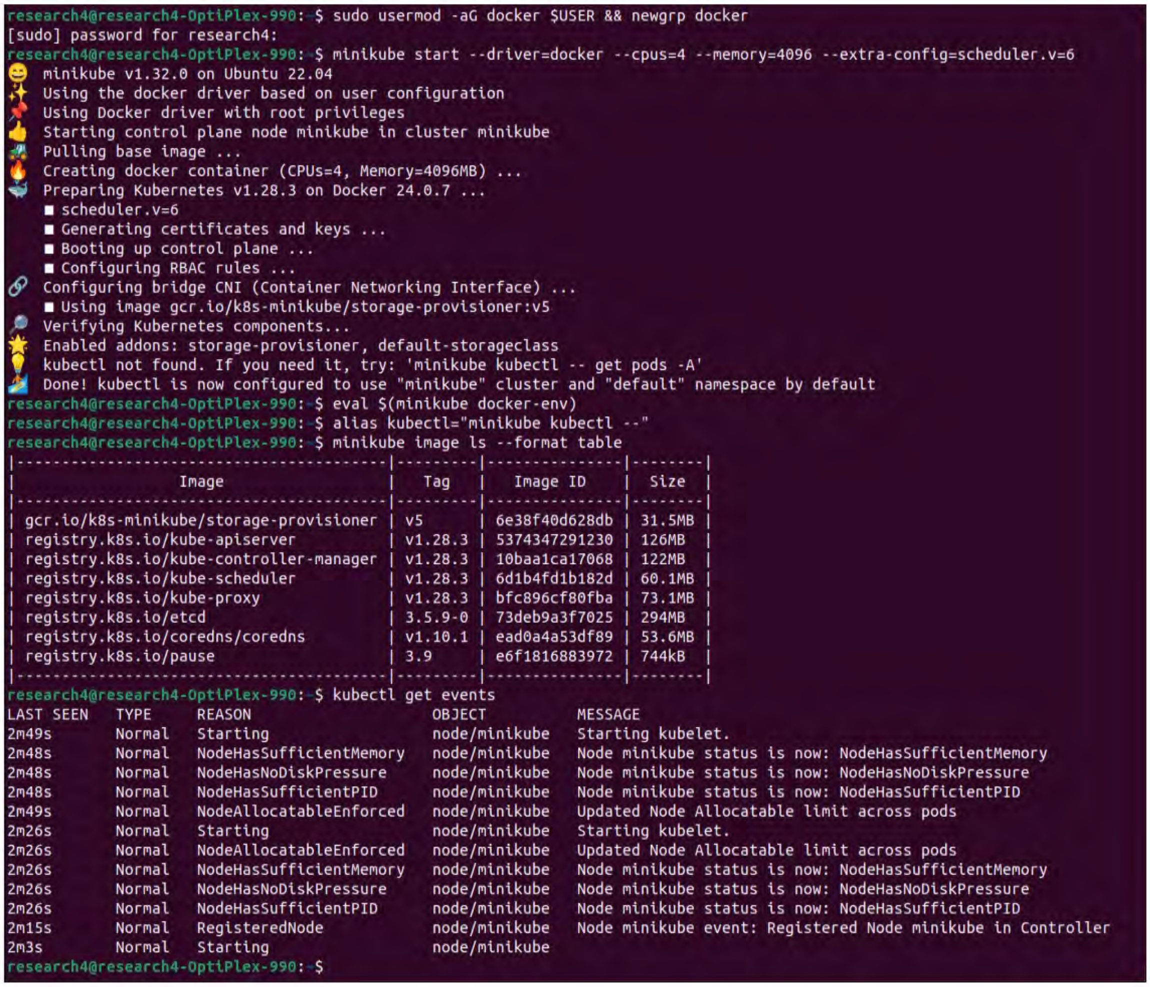
In the evolutionary model, the ability to pass on genes and thrive is tested by the mutation of a player appearing on the game-space and meeting another player. A single mutation is likely to meet one of the players without a mutation. From this encounter, with a hawk or defector, meeting a dove or cooperator respectively, it is likely they will not thrive enough to reproduce. But when two cooperative parties meet, their payoffs are high enough that they are able to reproduce. By this means, an enclave of cooperation can form, even in a field of defectors. This system works even though it may result in greater harm to any one player. For instance, a call out in alarm may cause a greater risk to a prey. But it ensures the survival of its offspring. This is the same result from the altered predation model, which informed them to flee.

A different ecology has emerged in cloud computing through the dynamics in game theory. This is a mutation of the programmatic environment that has established itself in several forms in both commercial and open-source venues. Its economic factors include the same economies of scale that compose the cloud in this game theory model. Yet, the variation in programmatic behavior has allowed for an evolutionary stable set of specialized behaviors. It reflects a series of mutations to the various -as-a-Service models and an altered strategy on the parts of the players of cloud security games.

APPENDIX G

DATA DEVELOPMENT

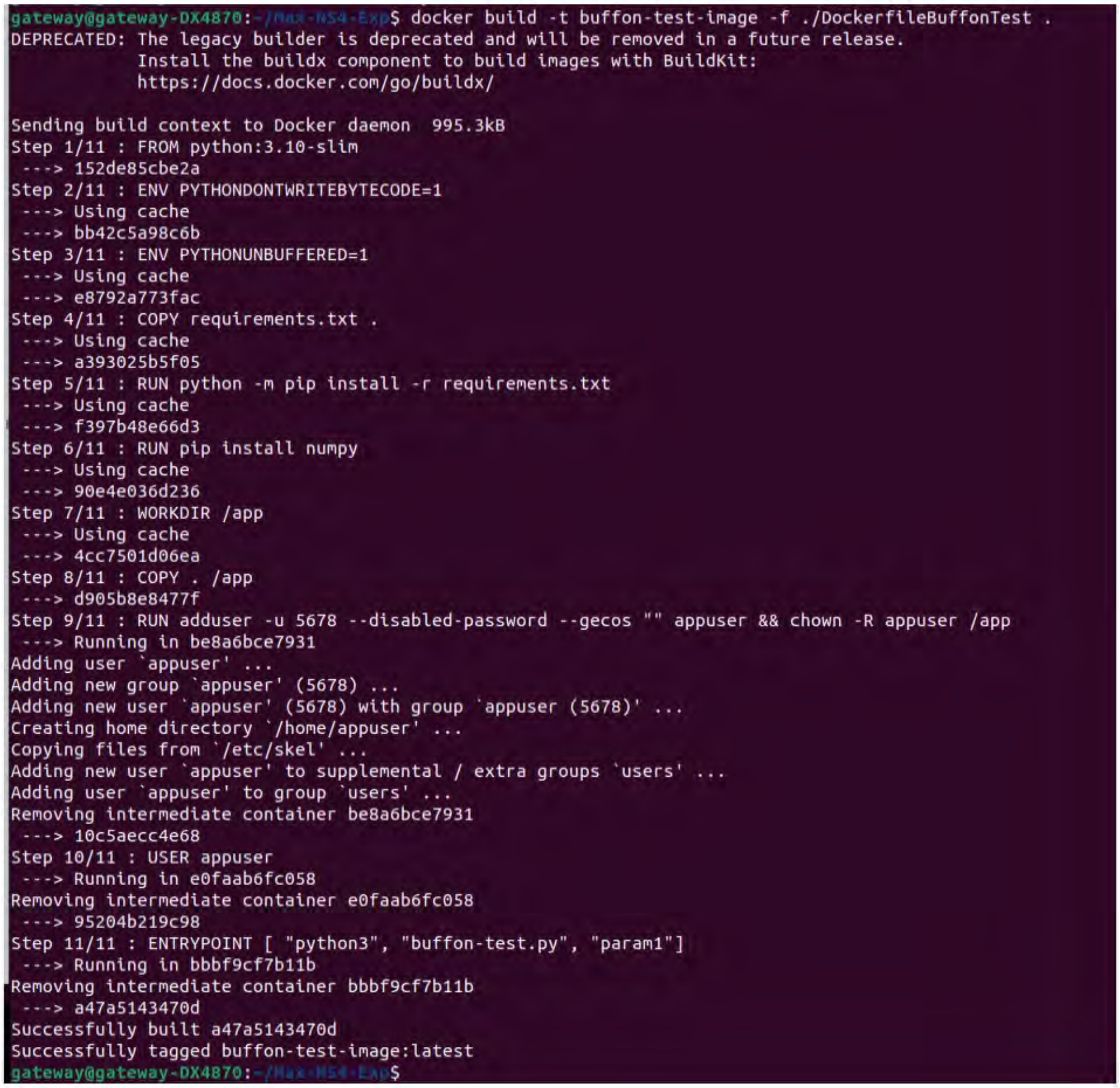
G.1 Kubernetes Configuration



```
research4@research4-OptiPlex-990:~$ sudo usermod -aG docker $USER && newgrp docker
[sudo] password for research4:
research4@research4-OptiPlex-990:~$ minikube start --driver=docker --cpus=4 --memory=4096 --extra-config=scheduler.v=6
😄  minikube v1.32.0 on Ubuntu 22.04
✨  Using the docker driver based on user configuration
📌  Using Docker driver with root privileges
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=4, Memory=4096MB) ...
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
    ■ scheduler.v=6
    ■ Generating certificates and keys ...
    ■ Booting up control plane ...
    ■ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
    ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔎  Verifying Kubernetes components...
🌟  Enabled addons: storage-provisioner, default-storageclass
💡  kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
research4@research4-OptiPlex-990:~$ eval $(minikube docker-env)
research4@research4-OptiPlex-990:~$ alias kubectl="minikube kubectl --"
research4@research4-OptiPlex-990:~$ minikube image ls --format table
|-----------------------------------------|----------|---------------|--------|
|                  Image                  |   Tag    |   Image ID    |  Size  |
|-----------------------------------------|----------|---------------|--------|
| gcr.io/k8s-minikube/storage-provisioner | v5       | 6e38f40d628db | 31.5MB |
| registry.k8s.io/kube-apiserver          | v1.28.3  | 5374347291230 | 126MB  |
| registry.k8s.io/kube-controller-manager | v1.28.3  | 10baa1ca17068 | 122MB  |
| registry.k8s.io/kube-scheduler          | v1.28.3  | 6d1b4fd1b182d | 60.1MB |
| registry.k8s.io/kube-proxy              | v1.28.3  | bfc896cf80fba | 73.1MB |
| registry.k8s.io/etcd                    | 3.5.9-0  | 73deb9a3f7025 | 294MB  |
| registry.k8s.io/coredns/coredns         | v1.10.1  | ead0a4a53df89 | 53.6MB |
| registry.k8s.io/pause                   | 3.9      | e6f1816883972 | 744kB  |
|-----------------------------------------|----------|---------------|--------|
research4@research4-OptiPlex-990:~$ kubectl get events
LAST SEEN   TYPE     REASON                    OBJECT          MESSAGE
2m49s       Normal   Starting                  node/minikube   Starting kubelet.
2m48s       Normal   NodeHasSufficientMemory   node/minikube   Node minikube status is now: NodeHasSufficientMemory
2m48s       Normal   NodeHasNoDiskPressure     node/minikube   Node minikube status is now: NodeHasNoDiskPressure
2m48s       Normal   NodeHasSufficientPID      node/minikube   Node minikube status is now: NodeHasSufficientPID
2m49s       Normal   NodeAllocatableEnforced   node/minikube   Updated Node Allocatable limit across pods
2m26s       Normal   Starting                  node/minikube   Starting kubelet.
2m26s       Normal   NodeAllocatableEnforced   node/minikube   Updated Node Allocatable limit across pods
2m26s       Normal   NodeHasSufficientMemory   node/minikube   Node minikube status is now: NodeHasSufficientMemory
2m26s       Normal   NodeHasNoDiskPressure     node/minikube   Node minikube status is now: NodeHasNoDiskPressure
2m26s       Normal   NodeHasSufficientPID      node/minikube   Node minikube status is now: NodeHasSufficientPID
2m15s       Normal   RegisteredNode            node/minikube   Node minikube event: Registered Node minikube in Controller
2m3s        Normal   Starting                  node/minikube
research4@research4-OptiPlex-990:~$
```

Fig. G.1. Minikube startup and configuration

G.2 Docker Builds



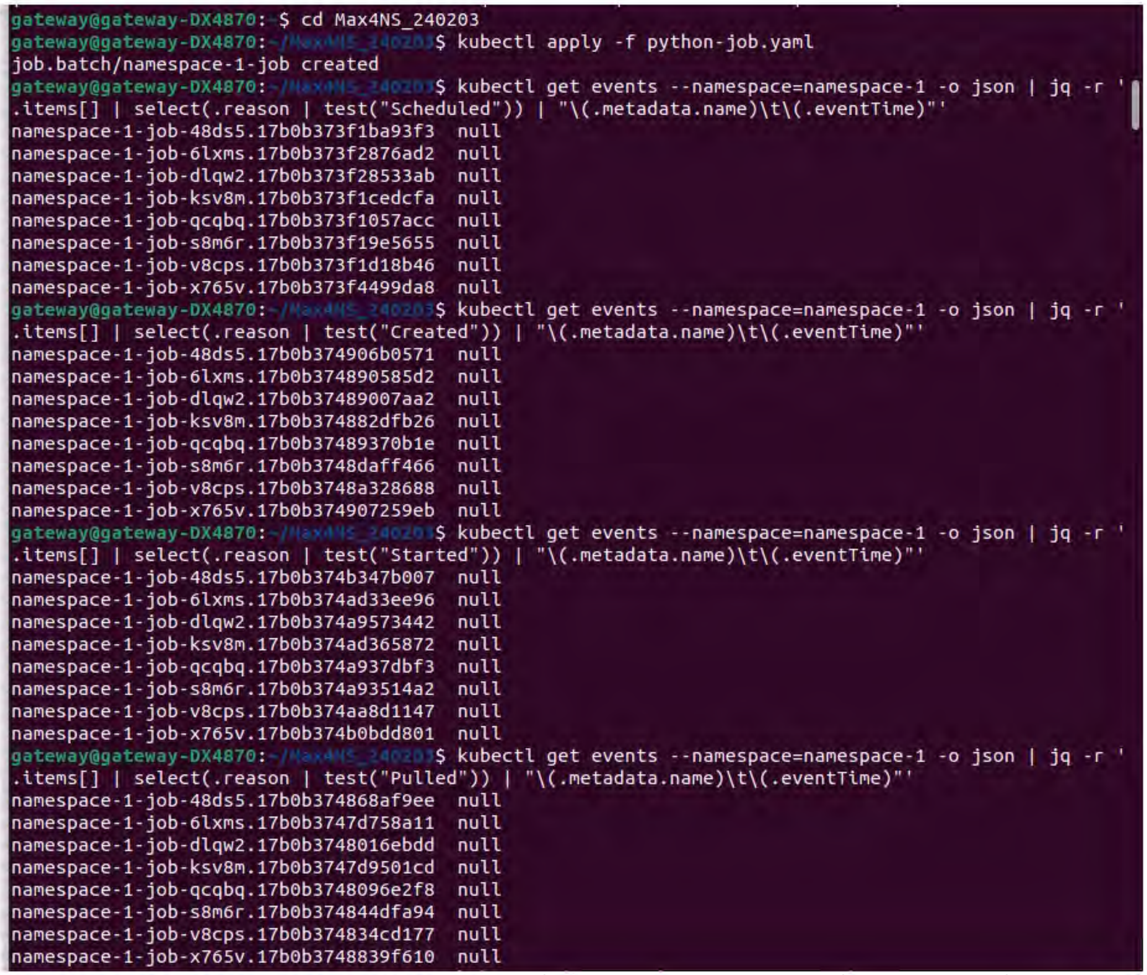
```
gateway@gateway-DX4870:~/Max-NS4-Exp$ docker build -t buffon-test-image -f ./DockerfileBuffonTest .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  995.3kB
Step 1/11 : FROM python:3.10-slim
 ---> 152de85cbe2a
Step 2/11 : ENV PYTHONDONTWRITEBYTECODE=1
 ---> Using cache
 ---> bb42c5a98c6b
Step 3/11 : ENV PYTHONUNBUFFERED=1
 ---> Using cache
 ---> e8792a773fac
Step 4/11 : COPY requirements.txt .
 ---> Using cache
 ---> a393025b5f05
Step 5/11 : RUN python -m pip install -r requirements.txt
 ---> Using cache
 ---> f397b48e66d3
Step 6/11 : RUN pip install numpy
 ---> Using cache
 ---> 90e4e036d236
Step 7/11 : WORKDIR /app
 ---> Using cache
 ---> 4cc7501d06ea
Step 8/11 : COPY . /app
 ---> d905b8e8477f
Step 9/11 : RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
 ---> Running in be8a6bce7931
Adding user `appuser' ...
Adding new group `appuser' (5678) ...
Adding new user `appuser' (5678) with group `appuser (5678)' ...
Creating home directory `/home/appuser' ...
Copying files from `/etc/skel' ...
Adding new user `appuser' to supplemental / extra groups `users' ...
Adding user `appuser' to group `users' ...
Removing intermediate container be8a6bce7931
 ---> 10c5aecc4e68
Step 10/11 : USER appuser
 ---> Running in e0faab6fc058
Removing intermediate container e0faab6fc058
 ---> 95204b219c98
Step 11/11 : ENTRYPOINT [ "python3", "buffon-test.py", "param1"]
 ---> Running in bbbf9cf7b11b
Removing intermediate container bbbf9cf7b11b
 ---> a47a5143470d
Successfully built a47a5143470d
Successfully tagged buffon-test-image:latest
gateway@gateway-DX4870:~/Max-NS4-Exp$
```

Fig. G.2. Docker container image build process. The container image serves as the template to execute function code in the Kubernetes cluster. The images will be present in the local registry. See Fig. G.1.

G.3. Event and Inspection Data

Two process' of data extraction were obtained from the Kubernetes cluster. The event data for pods and inspection commands provided container start and stop time.

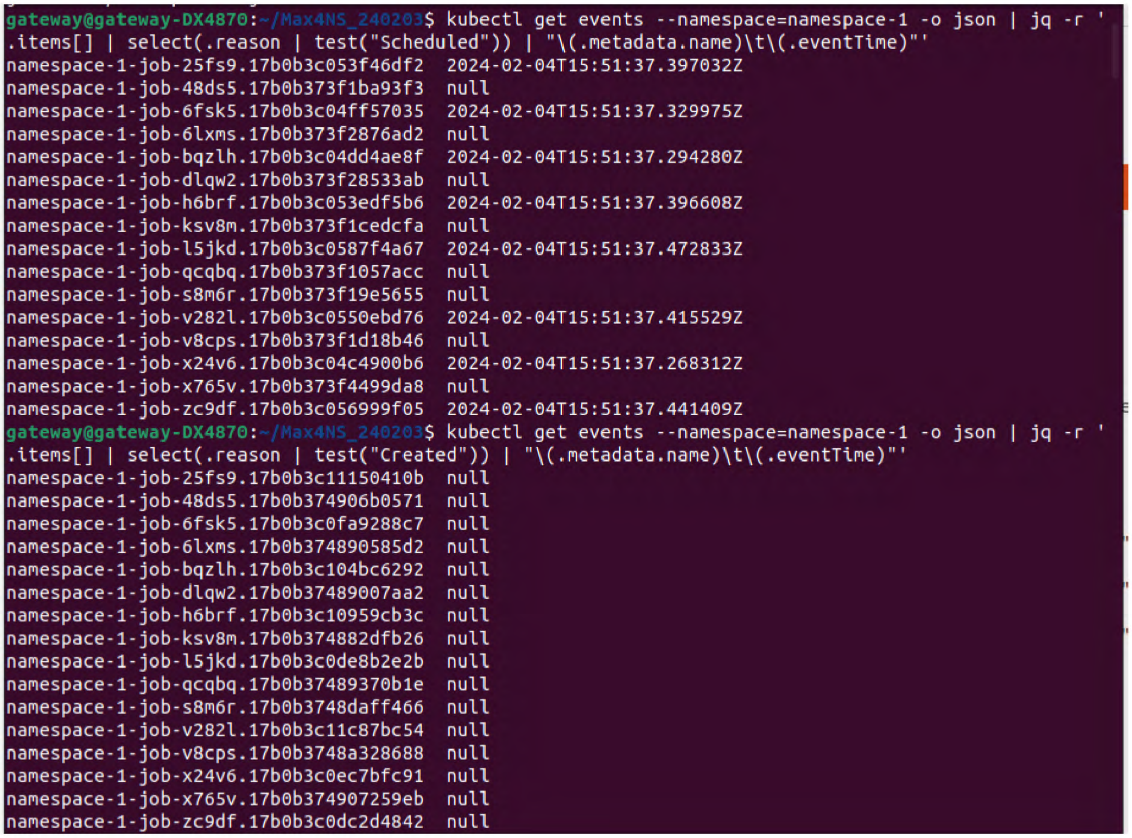
G.3.1) Event Data: Pod events where logged in the scheduler of the Kubernetes cluster. The default setting was inadequate to provide information at the necessary granularity. This was remedied by setting verbosity levels to a higher setting permitting readings to the micro-second.



```
gateway@gateway-DX4870:~$ cd Max4NS_240203
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl apply -f python-job.yaml
job.batch/namespace-1-job created
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Scheduled")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-48ds5.17b0b373f1ba93f3  null
namespace-1-job-6lxms.17b0b373f2876ad2  null
namespace-1-job-dlqw2.17b0b373f28533ab  null
namespace-1-job-ksv8m.17b0b373f1cedcfa  null
namespace-1-job-qcqbq.17b0b373f1057acc  null
namespace-1-job-s8m6r.17b0b373f19e5655  null
namespace-1-job-v8cps.17b0b373f1d18b46  null
namespace-1-job-x765v.17b0b373f4499da8  null
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Created")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-48ds5.17b0b374906b0571  null
namespace-1-job-6lxms.17b0b374890585d2  null
namespace-1-job-dlqw2.17b0b37489007aa2  null
namespace-1-job-ksv8m.17b0b374882dfb26  null
namespace-1-job-qcqbq.17b0b37489370b1e  null
namespace-1-job-s8m6r.17b0b3748daff466  null
namespace-1-job-v8cps.17b0b3748a328688  null
namespace-1-job-x765v.17b0b374907259eb  null
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Started")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-48ds5.17b0b374b347b007  null
namespace-1-job-6lxms.17b0b374ad33ee96  null
namespace-1-job-dlqw2.17b0b374a9573442  null
namespace-1-job-ksv8m.17b0b374ad365872  null
namespace-1-job-qcqbq.17b0b374a937dbf3  null
namespace-1-job-s8m6r.17b0b374a93514a2  null
namespace-1-job-v8cps.17b0b374aa8d1147  null
namespace-1-job-x765v.17b0b374b0bdd801  null
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Pulled")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-48ds5.17b0b374868af9ee  null
namespace-1-job-6lxms.17b0b3747d758a11  null
namespace-1-job-dlqw2.17b0b3748016ebdd  null
namespace-1-job-ksv8m.17b0b3747d9501cd  null
namespace-1-job-qcqbq.17b0b3748096e2f8  null
namespace-1-job-s8m6r.17b0b374844dfa94  null
namespace-1-job-v8cps.17b0b374834cd177  null
namespace-1-job-x765v.17b0b3748839f610  null
```

Fig. G.3. Pod scheduling events at the default setting. At the default setting for the Kubernetes cluster, pod events are logged. However, each data point is absent numeric timing data. This the case for scheduling, image pulling from the registry, for the

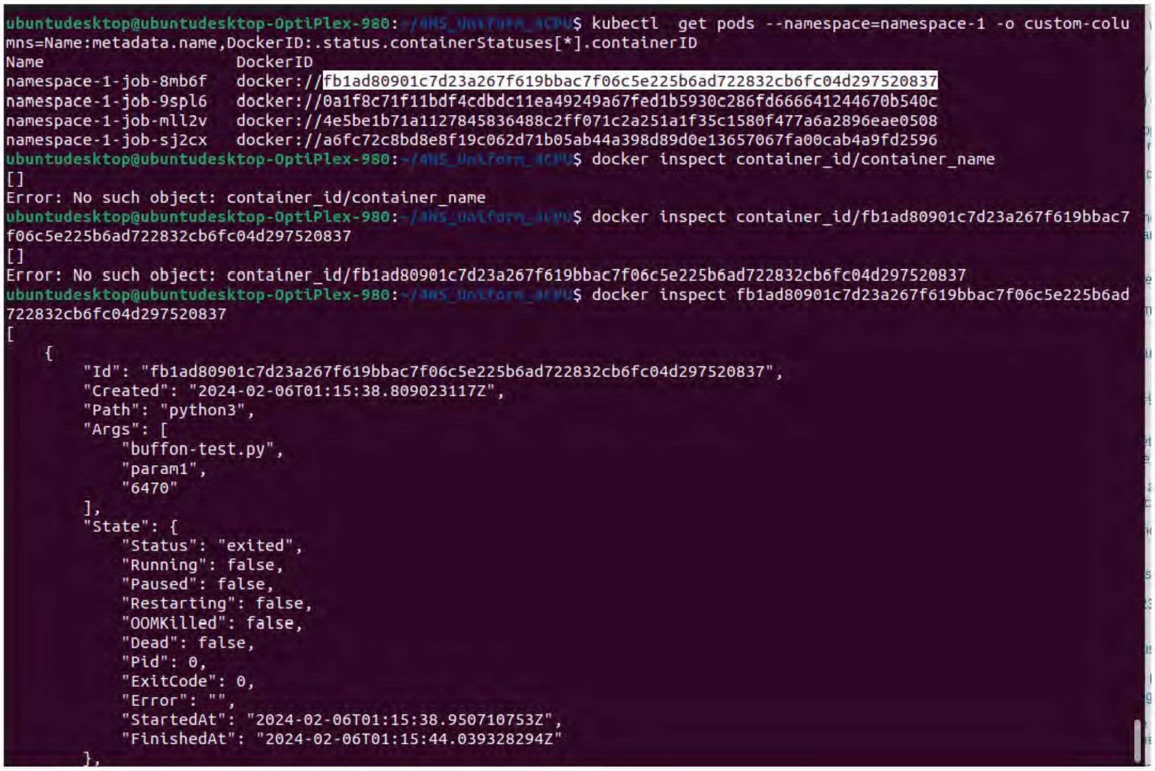
creation, and starting of the pod. Under normal circumstances, the record will hold this data for an hour.



```
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Scheduled")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-25fs9.17b0b3c053f46df2  2024-02-04T15:51:37.397032Z
namespace-1-job-48ds5.17b0b373f1ba93f3  null
namespace-1-job-6fsk5.17b0b3c04ff57035  2024-02-04T15:51:37.329975Z
namespace-1-job-6lxms.17b0b373f2876ad2  null
namespace-1-job-bqzlh.17b0b3c04dd4ae8f  2024-02-04T15:51:37.294280Z
namespace-1-job-dlqw2.17b0b373f28533ab  null
namespace-1-job-h6brf.17b0b3c053edf5b6  2024-02-04T15:51:37.396608Z
namespace-1-job-ksv8m.17b0b373f1cedcfa  null
namespace-1-job-l5jkd.17b0b3c0587f4a67  2024-02-04T15:51:37.472833Z
namespace-1-job-qcqbq.17b0b373f1057acc  null
namespace-1-job-s8m6r.17b0b373f19e5655  null
namespace-1-job-v282l.17b0b3c0550ebd76  2024-02-04T15:51:37.415529Z
namespace-1-job-v8cps.17b0b373f1d18b46  null
namespace-1-job-x24v6.17b0b3c04c4900b6  2024-02-04T15:51:37.268312Z
namespace-1-job-x765v.17b0b373f4499da8  null
namespace-1-job-zc9df.17b0b3c056999f05  2024-02-04T15:51:37.441409Z
gateway@gateway-DX4870:~/Max4NS_240203$ kubectl get events --namespace=namespace-1 -o json | jq -r '
.items[] | select(.reason | test("Created")) | "\(.metadata.name)\t\(.eventTime)"'
namespace-1-job-25fs9.17b0b3c11150410b  null
namespace-1-job-48ds5.17b0b374906b0571  null
namespace-1-job-6fsk5.17b0b3c0fa9288c7  null
namespace-1-job-6lxms.17b0b374890585d2  null
namespace-1-job-bqzlh.17b0b3c104bc6292  null
namespace-1-job-dlqw2.17b0b37489007aa2  null
namespace-1-job-h6brf.17b0b3c10959cb3c  null
namespace-1-job-ksv8m.17b0b374882dfb26  null
namespace-1-job-l5jkd.17b0b3c0de8b2e2b  null
namespace-1-job-qcqbq.17b0b37489370b1e  null
namespace-1-job-s8m6r.17b0b3748daff466  null
namespace-1-job-v282l.17b0b3c11c87bc54  null
namespace-1-job-v8cps.17b0b3748a328688  null
namespace-1-job-x24v6.17b0b3c0ec7bfc91  null
namespace-1-job-x765v.17b0b374907259eb  null
namespace-1-job-zc9df.17b0b3c0dc2d4842  null
```

Fig. G.4. Pod scheduling events at the augmented verbosity setting. By entering the extra configuration scheduler verbosity level, it is possible to extract more precise information about the behavior of the cluster. This applicable to micro-second information about the pod scheduling. However, other metrics (Creation in this case) in the pod creation time remained invisible to the user.

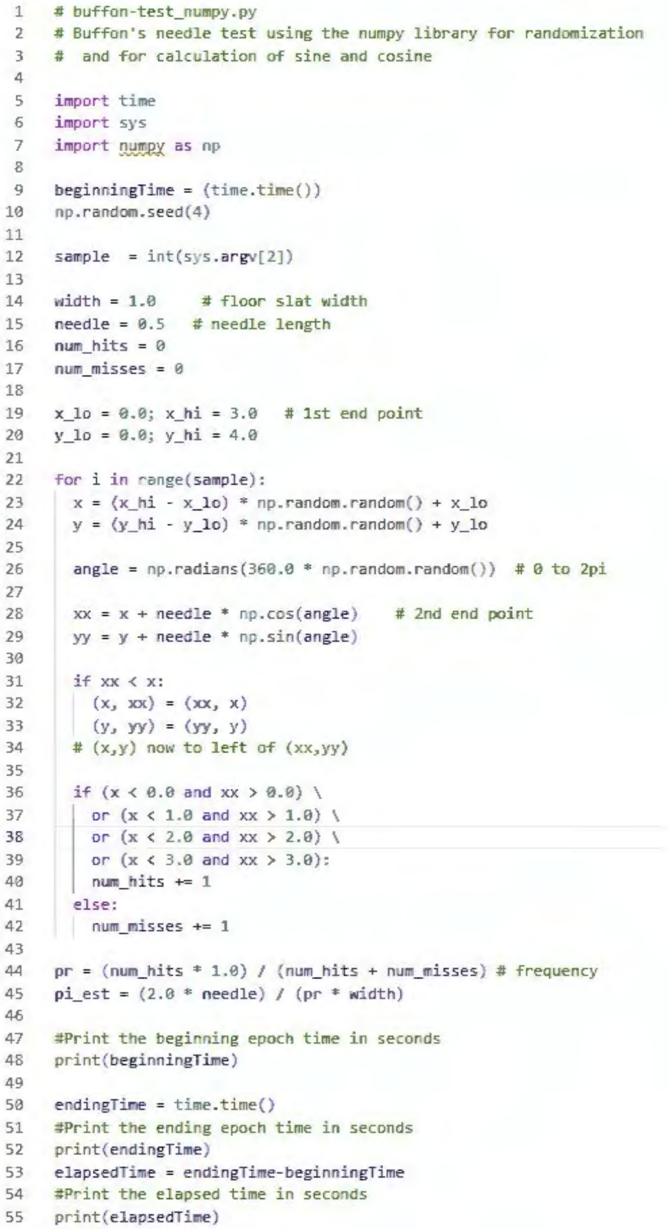
G.3.2 Container Inspection



```
ubuntudesktop@ubuntudesktop-OptiPlex-980:~/4NS_Uniform_4CPU$ kubectl  get pods --namespace=namespace-1 -o custom-colu
mns=Name:metadata.name,DockerID:.status.containerStatuses[*].containerID
Name                DockerID
namespace-1-job-8mb6f   docker://fb1ad80901c7d23a267f619bbac7f06c5e225b6ad722832cb6fc04d297520837
namespace-1-job-9spl6   docker://0a1f8c71f11bdf4cdbdc11ea49249a67fed1b5930c286fd666641244670b540c
namespace-1-job-mll2v   docker://4e5be1b71a1127845836488c2ff071c2a251a1f35c1580f477a6a2896eae0508
namespace-1-job-sj2cx   docker://a6fc72c8bd8e8f19c062d71b05ab44a398d89d0e13657067fa00cab4a9fd2596
ubuntudesktop@ubuntudesktop-OptiPlex-980:~/4NS_Uniform_4CPU$ docker inspect container_id/container_name
[]
Error: No such object: container_id/container_name
ubuntudesktop@ubuntudesktop-OptiPlex-980:~/4NS_Uniform_4CPU$ docker inspect container_id/fb1ad80901c7d23a267f619bbac7
f06c5e225b6ad722832cb6fc04d297520837
[]
Error: No such object: container_id/fb1ad80901c7d23a267f619bbac7f06c5e225b6ad722832cb6fc04d297520837
ubuntudesktop@ubuntudesktop-OptiPlex-980:~/4NS_Uniform_4CPU$ docker inspect fb1ad80901c7d23a267f619bbac7f06c5e225b6ad
722832cb6fc04d297520837
[
    {
        "Id": "fb1ad80901c7d23a267f619bbac7f06c5e225b6ad722832cb6fc04d297520837",
        "Created": "2024-02-06T01:15:38.809023117Z",
        "Path": "python3",
        "Args": [
            "buffon-test.py",
            "param1",
            "6470"
        ],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2024-02-06T01:15:38.950710753Z",
            "FinishedAt": "2024-02-06T01:15:44.039328294Z"
        },
```

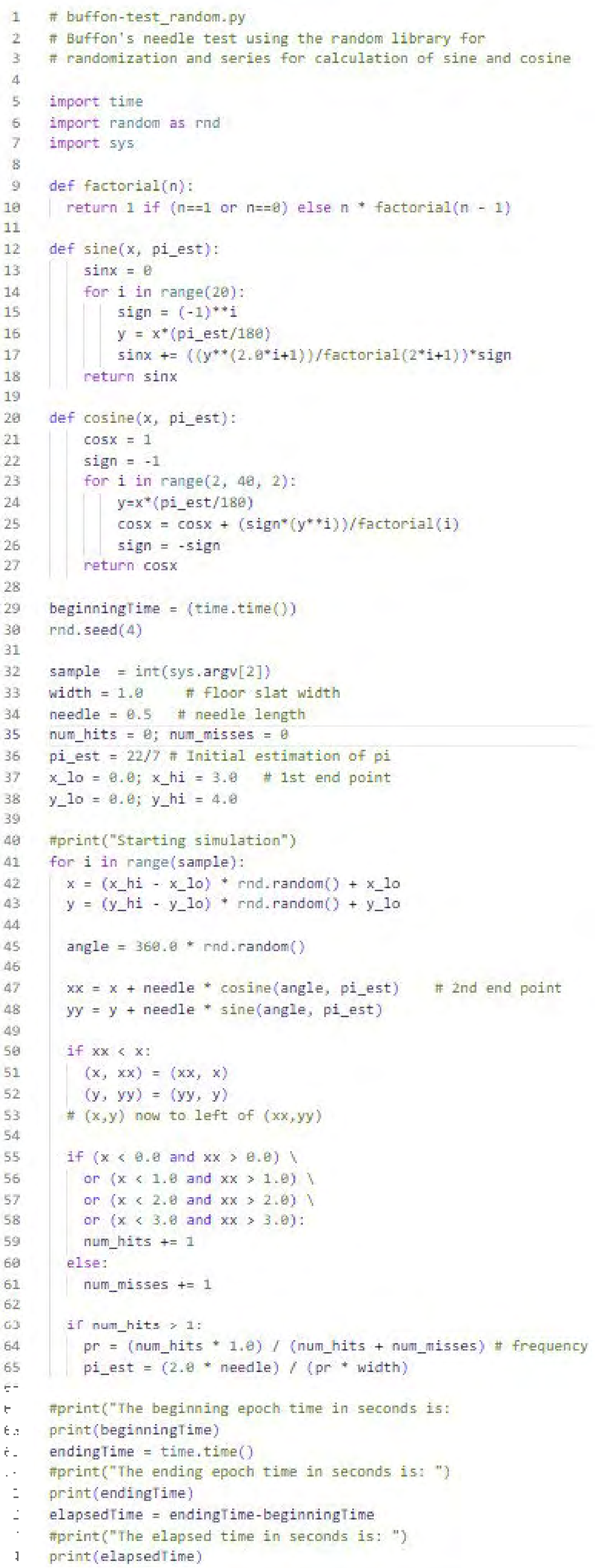
Fig. G.5. Container data inspection. Container data is extracted using the «inspect» command with the container ID. This is obtained through alignment with pod IDs and resident containers. Under normal circumstances, the record will hold this data for an hour.

G.4 Python Codes



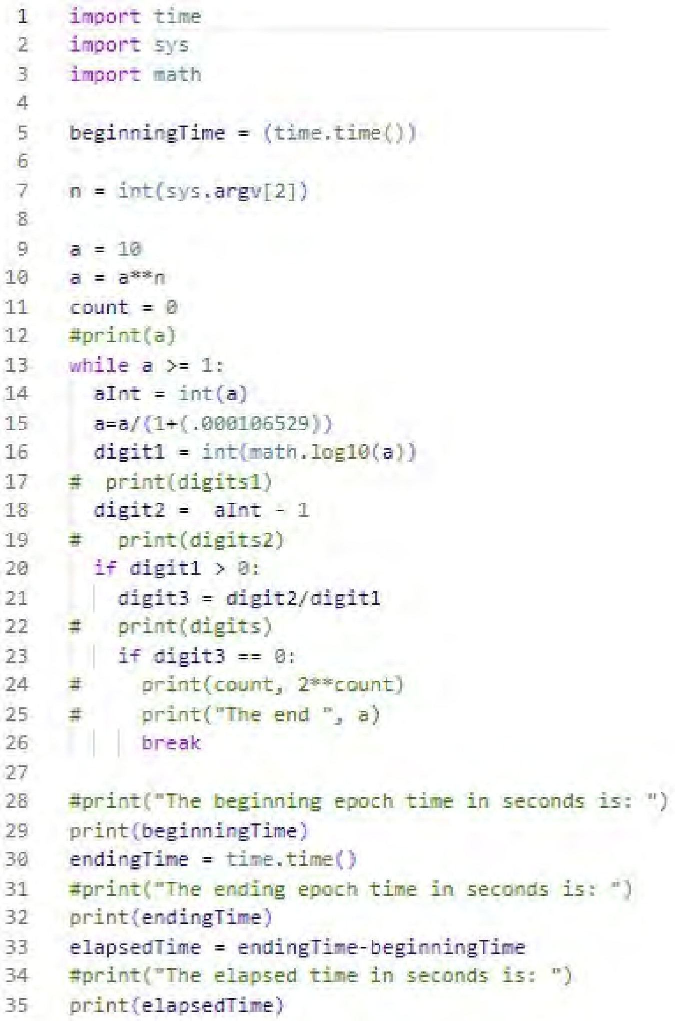
```
# buffon-test_numpy.py
# Buffon's needle test using the numpy library for randomization
#  and for calculation of sine and cosine

import time
import sys
import numpy as np

beginningTime = (time.time())
np.random.seed(4)

sample  = int(sys.argv[2])

width = 1.0     # floor slat width
needle = 0.5   # needle length
num_hits = 0
num_misses = 0

x_lo = 0.0; x_hi = 3.0   # 1st end point
y_lo = 0.0; y_hi = 4.0

for i in range(sample):
  x = (x_hi - x_lo) * np.random.random() + x_lo
  y = (y_hi - y_lo) * np.random.random() + y_lo

  angle = np.radians(360.0 * np.random.random())  # 0 to 2pi

  xx = x + needle * np.cos(angle)    # 2nd end point
  yy = y + needle * np.sin(angle)

  if xx < x:
    (x, xx) = (xx, x)
    (y, yy) = (yy, y)
  # (x,y) now to left of (xx,yy)

  if (x < 0.0 and xx > 0.0) \
    or (x < 1.0 and xx > 1.0) \
    or (x < 2.0 and xx > 2.0) \
    or (x < 3.0 and xx > 3.0):
    num_hits += 1
  else:
    num_misses += 1

pr = (num_hits * 1.0) / (num_hits + num_misses) # frequency
pi_est = (2.0 * needle) / (pr * width)

#Print the beginning epoch time in seconds
print(beginningTime)

endingTime = time.time()
#Print the ending epoch time in seconds
print(endingTime)
elapsedTime = endingTime-beginningTime
#Print the elapsed time in seconds
print(elapsedTime)
```

Fig. G.6. Buffon’s needle test using Numpy module.



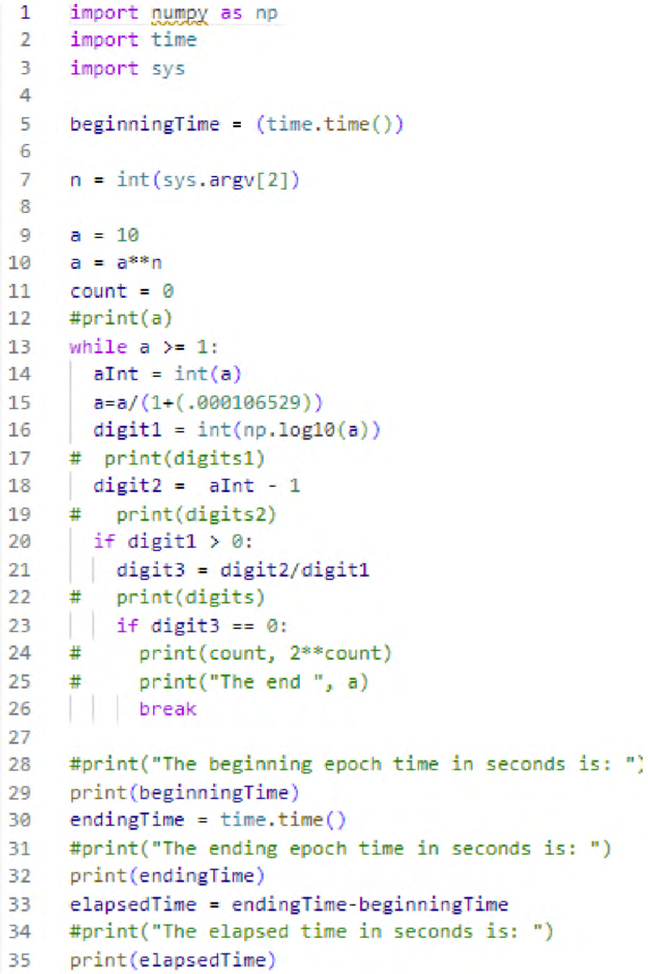
```python
# buffon-test_random.py
# Buffon's needle test using the random library for
# randomization and series for calculation of sine and cosine

import time
import random as rnd
import sys

def factorial(n):
  return 1 if (n==1 or n==0) else n * factorial(n - 1)

def sine(x, pi_est):
    sinx = 0
    for i in range(20):
        sign = (-1)**i
        y = x*(pi_est/180)
        sinx += ((y**(2.0*i+1))/factorial(2*i+1))*sign
    return sinx

def cosine(x, pi_est):
    cosx = 1
    sign = -1
    for i in range(2, 40, 2):
        y=x*(pi_est/180)
        cosx = cosx + (sign*(y**i))/factorial(i)
        sign = -sign
    return cosx

beginningTime = (time.time())
rnd.seed(4)

sample  = int(sys.argv[2])
width = 1.0     # floor slat width
needle = 0.5   # needle length
num_hits = 0; num_misses = 0
pi_est = 22/7 # Initial estimation of pi
x_lo = 0.0; x_hi = 3.0   # 1st end point
y_lo = 0.0; y_hi = 4.0

#print("Starting simulation")
for i in range(sample):
  x = (x_hi - x_lo) * rnd.random() + x_lo
  y = (y_hi - y_lo) * rnd.random() + y_lo

  angle = 360.0 * rnd.random()

  xx = x + needle * cosine(angle, pi_est)    # 2nd end point
  yy = y + needle * sine(angle, pi_est)

  if xx < x:
    (x, xx) = (xx, x)
    (y, yy) = (yy, y)
  # (x,y) now to left of (xx,yy)

  if (x < 0.0 and xx > 0.0) \
    or (x < 1.0 and xx > 1.0) \
    or (x < 2.0 and xx > 2.0) \
    or (x < 3.0 and xx > 3.0):
    num_hits += 1
  else:
    num_misses += 1

  if num_hits > 1:
    pr = (num_hits * 1.0) / (num_hits + num_misses) # frequency
    pi_est = (2.0 * needle) / (pr * width)

#print("The beginning epoch time in seconds is:
print(beginningTime)
endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)
elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.7. Buffon’s needle test using the Random module.



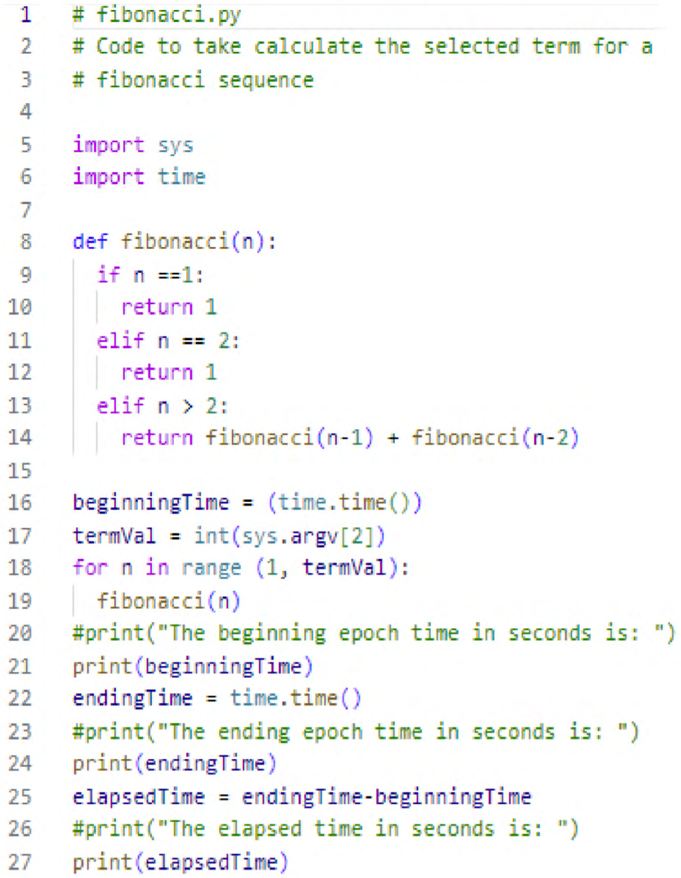
```
import time
import sys
import math

beginningTime = (time.time())

n = int(sys.argv[2])

a = 10
a = a**n
count = 0
#print(a)
while a >= 1:
  aInt = int(a)
  a=a/(1+(.000106529))
  digit1 = int(math.log10(a))
#  print(digits1)
  digit2 =  aInt - 1
#   print(digits2)
  if digit1 > 0:
    digit3 = digit2/digit1
#   print(digits)
    if digit3 == 0:
#     print(count, 2**count)
#     print("The end ", a)
      break

#print("The beginning epoch time in seconds is: ")
print(beginningTime)
endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)
elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.8. A code to conduct floating division on large numbers using the math module.



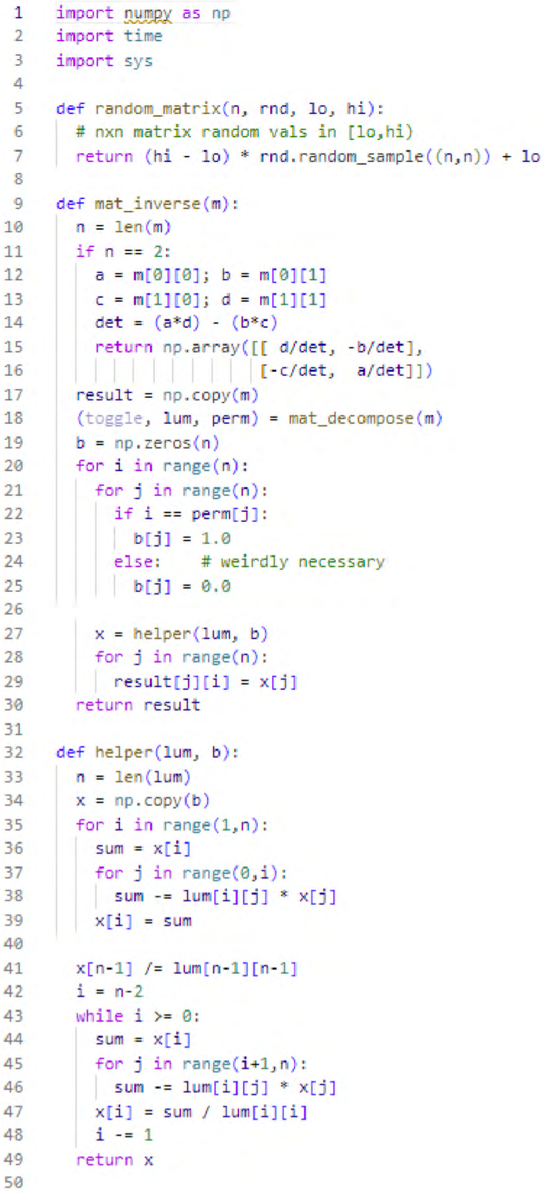
```
import numpy as np
import time
import sys

beginningTime = (time.time())

n = int(sys.argv[2])

a = 10
a = a**n
count = 0
#print(a)
while a >= 1:
  aInt = int(a)
  a=a/(1+(.000106529))
  digit1 = int(np.log10(a))
#  print(digits1)
  digit2 =  aInt - 1
#   print(digits2)
  if digit1 > 0:
    digit3 = digit2/digit1
#   print(digits)
    if digit3 == 0:
#     print(count, 2**count)
#     print("The end ", a)
      break

#print("The beginning epoch time in seconds is: ")
print(beginningTime)
endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)
elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.9. A code to conduct floating division on large numbers using the Numpy module.



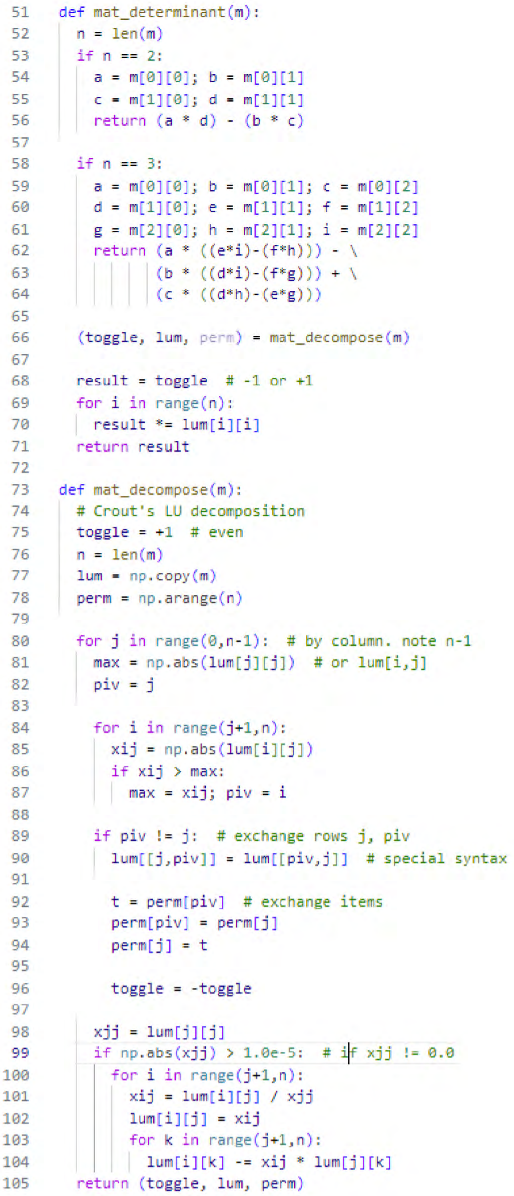
```
# fibonacci.py
# Code to take calculate the selected term for a
# fibonacci sequence

import sys
import time

def fibonacci(n):
  if n ==1:
    return 1
  elif n == 2:
    return 1
  elif n > 2:
    return fibonacci(n-1) + fibonacci(n-2)

beginningTime = (time.time())
termVal = int(sys.argv[2])
for n in range (1, termVal):
  fibonacci(n)
#print("The beginning epoch time in seconds is: ")
print(beginningTime)
endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)
elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.10. A code to generate a Fibonacci sequence to a given number of terms.



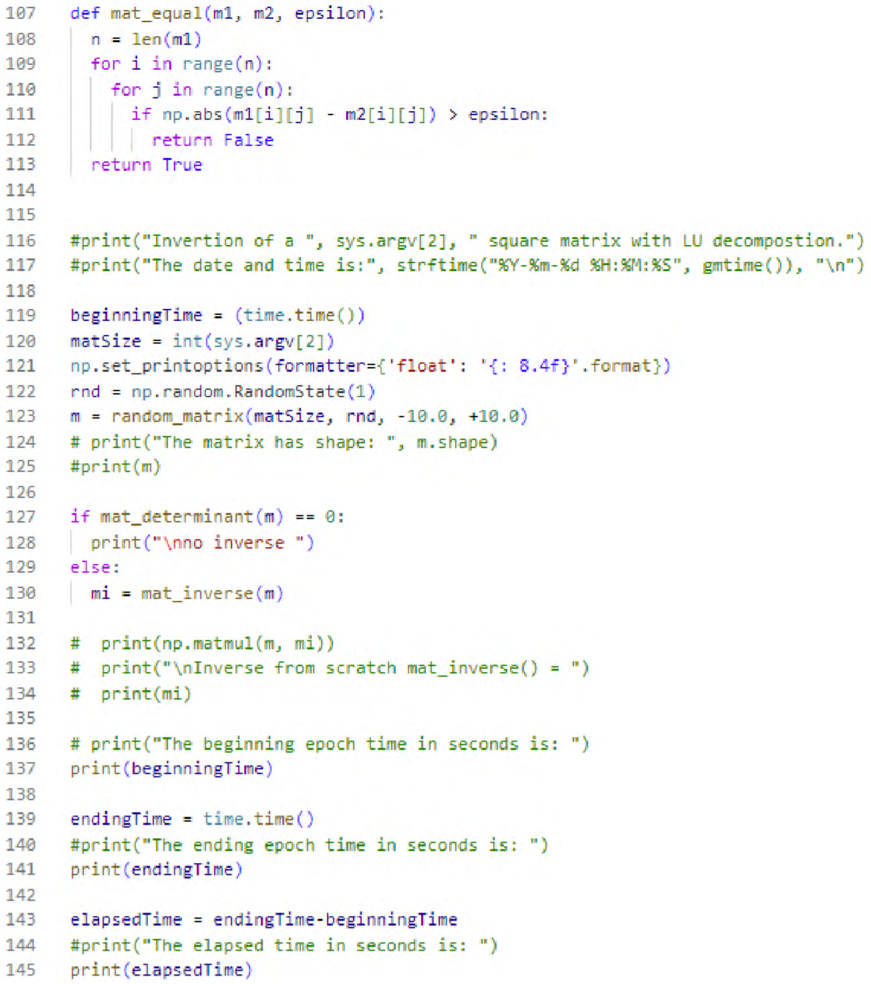
```
import numpy as np
import time
import sys

def random_matrix(n, rnd, lo, hi):
  # nxn matrix random vals in [lo,hi)
  return (hi - lo) * rnd.random_sample((n,n)) + lo

def mat_inverse(m):
  n = len(m)
  if n == 2:
    a = m[0][0]; b = m[0][1]
    c = m[1][0]; d = m[1][1]
    det = (a*d) - (b*c)
    return np.array([[ d/det, -b/det],
                     [-c/det,  a/det]])
  result = np.copy(m)
  (toggle, lum, perm) = mat_decompose(m)
  b = np.zeros(n)
  for i in range(n):
    for j in range(n):
      if i == perm[j]:
        b[j] = 1.0
      else:    # weirdly necessary
        b[j] = 0.0

    x = helper(lum, b)
    for j in range(n):
      result[j][i] = x[j]
  return result

def helper(lum, b):
  n = len(lum)
  x = np.copy(b)
  for i in range(1,n):
    sum = x[i]
    for j in range(0,i):
      sum -= lum[i][j] * x[j]
    x[i] = sum

  x[n-1] /= lum[n-1][n-1]
  i = n-2
  while i >= 0:
    sum = x[i]
    for j in range(i+1,n):
      sum -= lum[i][j] * x[j]
    x[i] = sum / lum[i][i]
    i -= 1
  return x

```

Fig. G.11. A code to invert a matrix using Lower-Upper Decomposition using Numpy, 1 / 3.



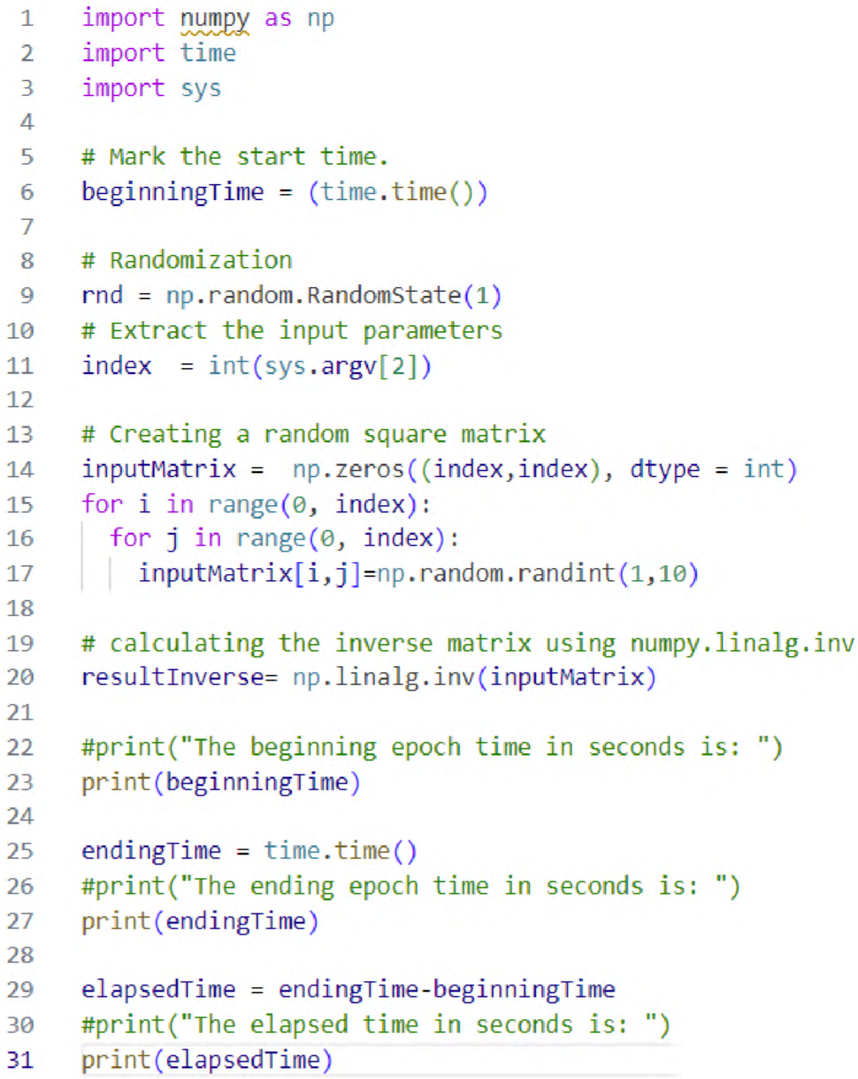
```
def mat_determinant(m):
  n = len(m)
  if n == 2:
    a = m[0][0]; b = m[0][1]
    c = m[1][0]; d = m[1][1]
    return (a * d) - (b * c)

  if n == 3:
    a = m[0][0]; b = m[0][1]; c = m[0][2]
    d = m[1][0]; e = m[1][1]; f = m[1][2]
    g = m[2][0]; h = m[2][1]; i = m[2][2]
    return (a * ((e*i)-(f*h))) - \
           (b * ((d*i)-(f*g))) + \
           (c * ((d*h)-(e*g)))

  (toggle, lum, perm) = mat_decompose(m)

  result = toggle  # -1 or +1
  for i in range(n):
    result *= lum[i][i]
  return result

def mat_decompose(m):
  # Crout's LU decomposition
  toggle = +1  # even
  n = len(m)
  lum = np.copy(m)
  perm = np.arange(n)

  for j in range(0,n-1):  # by column. note n-1
    max = np.abs(lum[j][j])  # or lum[i,j]
    piv = j

    for i in range(j+1,n):
      xij = np.abs(lum[i][j])
      if xij > max:
        max = xij; piv = i

    if piv != j:  # exchange rows j, piv
      lum[[j,piv]] = lum[[piv,j]]  # special syntax

      t = perm[piv]  # exchange items
      perm[piv] = perm[j]
      perm[j] = t

      toggle = -toggle

    xjj = lum[j][j]
    if np.abs(xjj) > 1.0e-5:  # if xjj != 0.0
      for i in range(j+1,n):
        xij = lum[i][j] / xjj
        lum[i][j] = xij
        for k in range(j+1,n):
          lum[i][k] -= xij * lum[j][k]
  return (toggle, lum, perm)
```

Fig. G.12. A code to invert a matrix using Lower-Upper Decomposition using Numpy, 2 / 3.



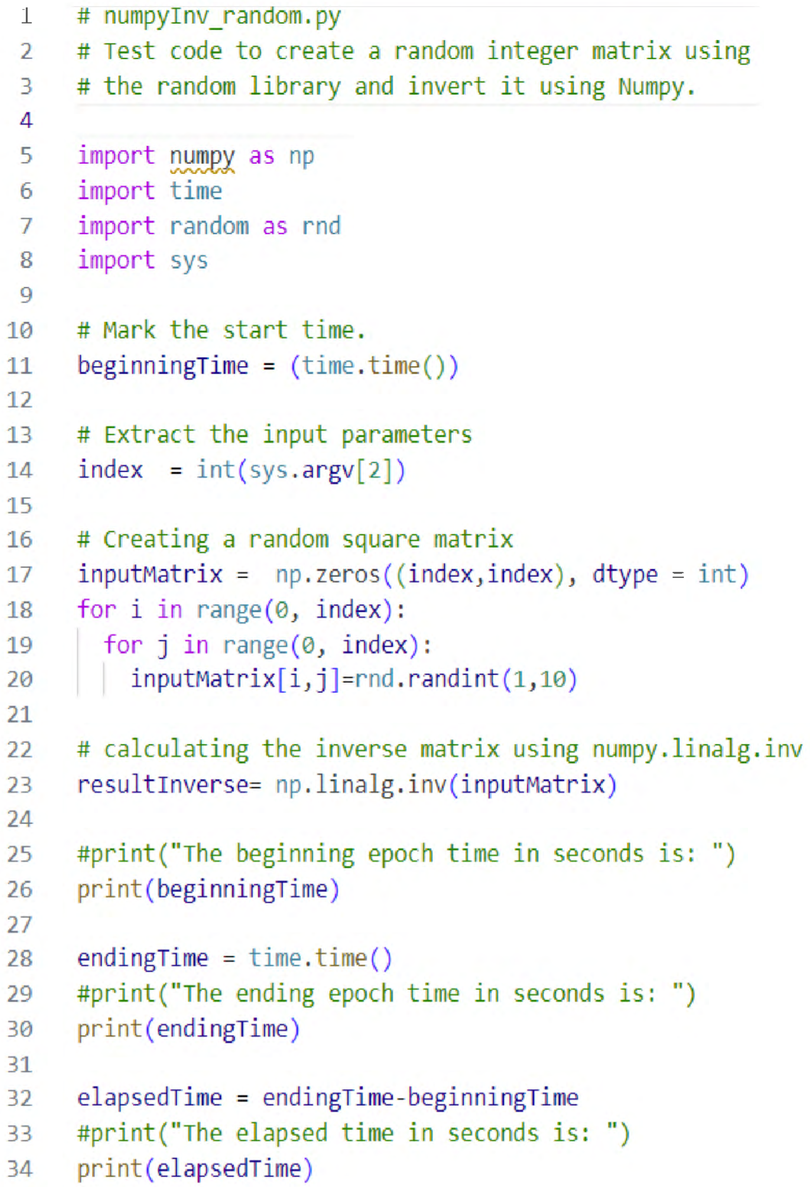
```
def mat_equal(m1, m2, epsilon):
  n = len(m1)
  for i in range(n):
    for j in range(n):
      if np.abs(m1[i][j] - m2[i][j]) > epsilon:
        return False
  return True


#print("Invertion of a ", sys.argv[2], " square matrix with LU decompostion.")
#print("The date and time is:", strftime("%Y-%m-%d %H:%M:%S", gmtime()), "\n")

beginningTime = (time.time())
matSize = int(sys.argv[2])
np.set_printoptions(formatter={'float': '{: 8.4f}'.format})
rnd = np.random.RandomState(1)
m = random_matrix(matSize, rnd, -10.0, +10.0)
# print("The matrix has shape: ", m.shape)
#print(m)

if mat_determinant(m) == 0:
  print("\nno inverse ")
else:
  mi = mat_inverse(m)

#  print(np.matmul(m, mi))
#  print("\nInverse from scratch mat_inverse() = ")
#  print(mi)

# print("The beginning epoch time in seconds is: ")
print(beginningTime)

endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)

elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.13. A code to invert a matrix using Lower-Upper Decomposition using Numpy, 3 / 3.



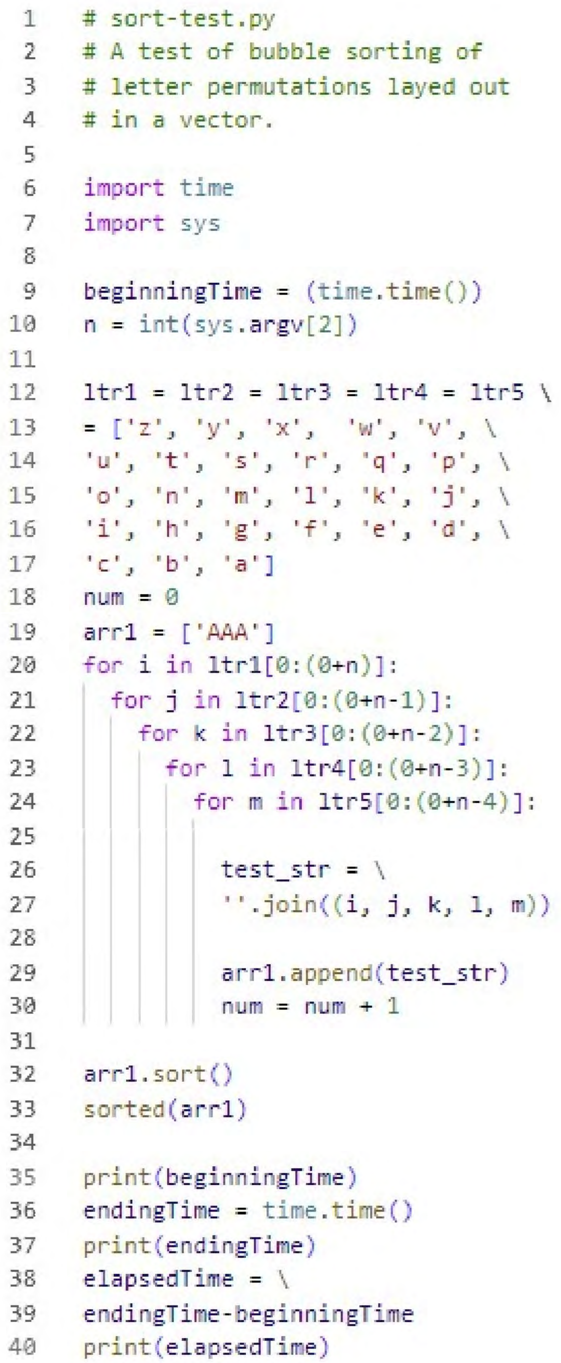
```
import numpy as np
import time
import sys

# Mark the start time.
beginningTime = (time.time())

# Randomization
rnd = np.random.RandomState(1)
# Extract the input parameters
index  = int(sys.argv[2])

# Creating a random square matrix
inputMatrix =  np.zeros((index,index), dtype = int)
for i in range(0, index):
  for j in range(0, index):
    inputMatrix[i,j]=np.random.randint(1,10)

# calculating the inverse matrix using numpy.linalg.inv
resultInverse= np.linalg.inv(inputMatrix)

#print("The beginning epoch time in seconds is: ")
print(beginningTime)

endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)

elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig G.14. Test code for data extraction using the Numpy library for inversion of a matrix.



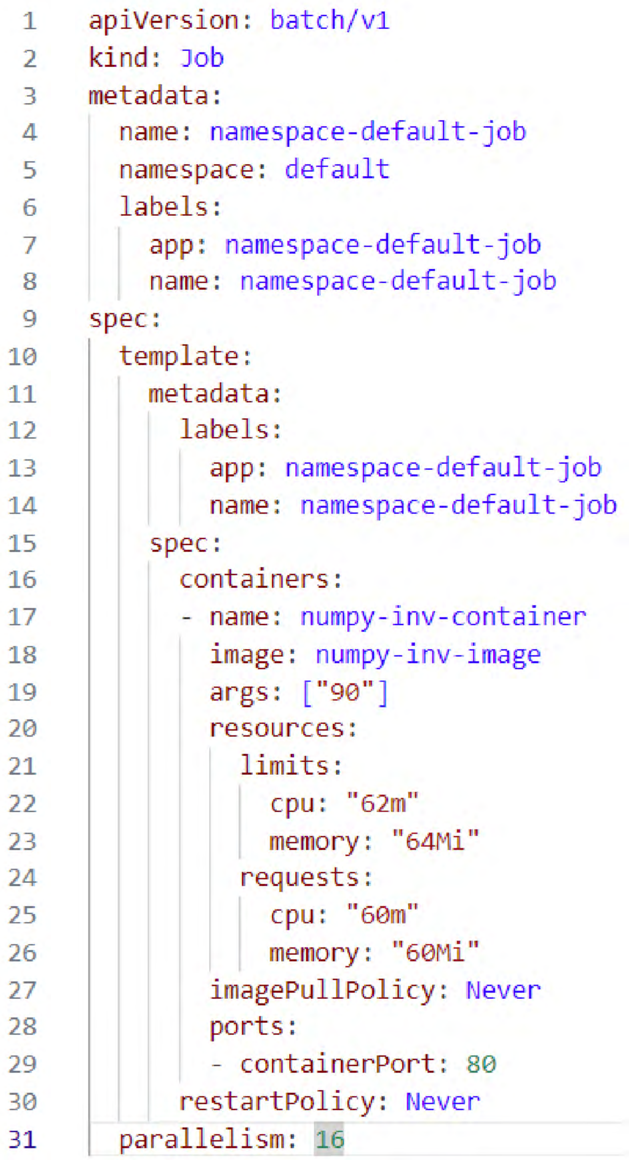
```python
# numpyInv_random.py
# Test code to create a random integer matrix using
# the random library and invert it using Numpy.

import numpy as np
import time
import random as rnd
import sys

# Mark the start time.
beginningTime = (time.time())

# Extract the input parameters
index  = int(sys.argv[2])

# Creating a random square matrix
inputMatrix =  np.zeros((index,index), dtype = int)
for i in range(0, index):
  for j in range(0, index):
    inputMatrix[i,j]=rnd.randint(1,10)

# calculating the inverse matrix using numpy.linalg.inv
resultInverse= np.linalg.inv(inputMatrix)

#print("The beginning epoch time in seconds is: ")
print(beginningTime)

endingTime = time.time()
#print("The ending epoch time in seconds is: ")
print(endingTime)

elapsedTime = endingTime-beginningTime
#print("The elapsed time in seconds is: ")
print(elapsedTime)
```

Fig. G.15. Test code for data extraction using the Numpy library for inversion of a matrix generated with the random library.



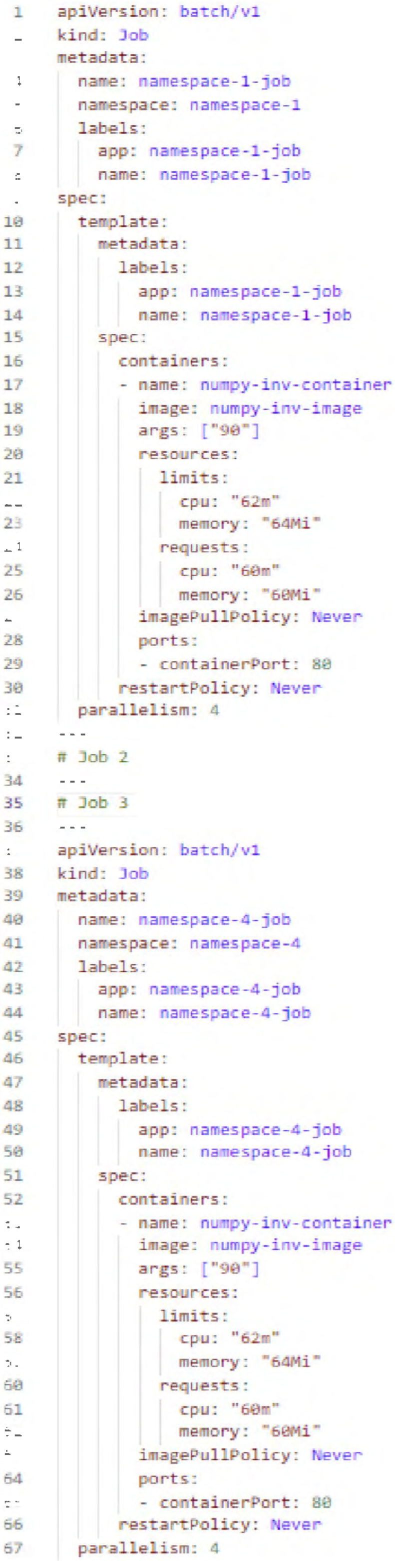
```
# sort-test.py
# A test of bubble sorting of
# letter permutations layed out
# in a vector.

import time
import sys

beginningTime = (time.time())
n = int(sys.argv[2])

ltr1 = ltr2 = ltr3 = ltr4 = ltr5 \
= ['z', 'y', 'x',  'w', 'v', \
'u', 't', 's', 'r', 'q', 'p', \
'o', 'n', 'm', 'l', 'k', 'j', \
'i', 'h', 'g', 'f', 'e', 'd', \
'c', 'b', 'a']
num = 0
arr1 = ['AAA']
for i in ltr1[0:(0+n)]:
  for j in ltr2[0:(0+n-1)]:
    for k in ltr3[0:(0+n-2)]:
      for l in ltr4[0:(0+n-3)]:
        for m in ltr5[0:(0+n-4)]:

          test_str = \
          ''.join((i, j, k, l, m))

          arr1.append(test_str)
          num = num + 1

arr1.sort()
sorted(arr1)

print(beginningTime)
endingTime = time.time()
print(endingTime)
elapsedTime = \
endingTime-beginningTime
print(elapsedTime)
```

Fig. G.16. A bubble sort routine conducted on a vector of letter permutations.



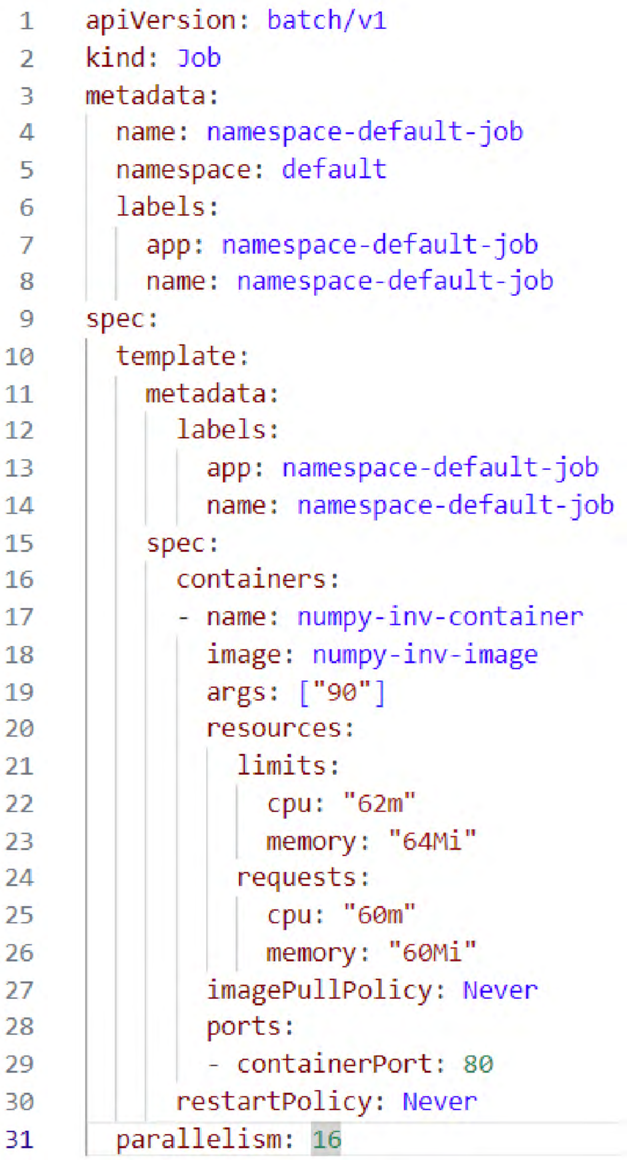
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: namespace-default-job
  namespace: default
  labels:
    app: namespace-default-job
    name: namespace-default-job
spec:
  template:
    metadata:
      labels:
        app: namespace-default-job
        name: namespace-default-job
    spec:
      containers:
      - name: numpy-inv-container
        image: numpy-inv-image
        args: ["90"]
        resources:
          limits:
            cpu: "62m"
            memory: "64Mi"
          requests:
            cpu: "60m"
            memory: "60Mi"
        imagePullPolicy: Never
        ports:
        - containerPort: 80
      restartPolicy: Never
  parallelism: 16
```

Fig. G.17. YAML configuration of 16 parallel executions of NumpyInv.py. Execution in the default Kubernetes name-spaces.



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: namespace-1-job
  namespace: namespace-1
  labels:
    app: namespace-1-job
    name: namespace-1-job
spec:
  template:
    metadata:
      labels:
        app: namespace-1-job
        name: namespace-1-job
    spec:
      containers:
      - name: numpy-inv-container
        image: numpy-inv-image
        args: ["90"]
        resources:
          limits:
            cpu: "62m"
            memory: "64Mi"
          requests:
            cpu: "60m"
            memory: "60Mi"
        imagePullPolicy: Never
        ports:
        - containerPort: 80
      restartPolicy: Never
  parallelism: 4
---
# Job 2
---
# Job 3
---
apiVersion: batch/v1
kind: Job
metadata:
  name: namespace-4-job
  namespace: namespace-4
  labels:
    app: namespace-4-job
    name: namespace-4-job
spec:
  template:
    metadata:
      labels:
        app: namespace-4-job
        name: namespace-4-job
    spec:
      containers:
      - name: numpy-inv-container
        image: numpy-inv-image
        args: ["90"]
        resources:
          limits:
            cpu: "62m"
            memory: "64Mi"
          requests:
            cpu: "60m"
            memory: "60Mi"
        imagePullPolicy: Never
        ports:
        - containerPort: 80
      restartPolicy: Never
  parallelism: 4
```

Fig. G.18. YAML configuration of four parallel executions of NumpyInv.py. Execution in four Kubernetes name-spaces for a total of 16 executions.



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: namespace-default-job
  namespace: default
  labels:
    app: namespace-default-job
    name: namespace-default-job
spec:
  template:
    metadata:
      labels:
        app: namespace-default-job
        name: namespace-default-job
    spec:
      containers:
      - name: numpy-inv-container
        image: numpy-inv-image
        args: ["90"]
        resources:
          limits:
            cpu: "62m"
            memory: "64Mi"
          requests:
            cpu: "60m"
            memory: "60Mi"
        imagePullPolicy: Never
        ports:
        - containerPort: 80
      restartPolicy: Never
  parallelism: 16
```

Fig. G.19. YAML configuration of four parallel executions of NumpyInv.py, each in four Kubernetes name-spaces.

G.5 Python Code Execution Analysis

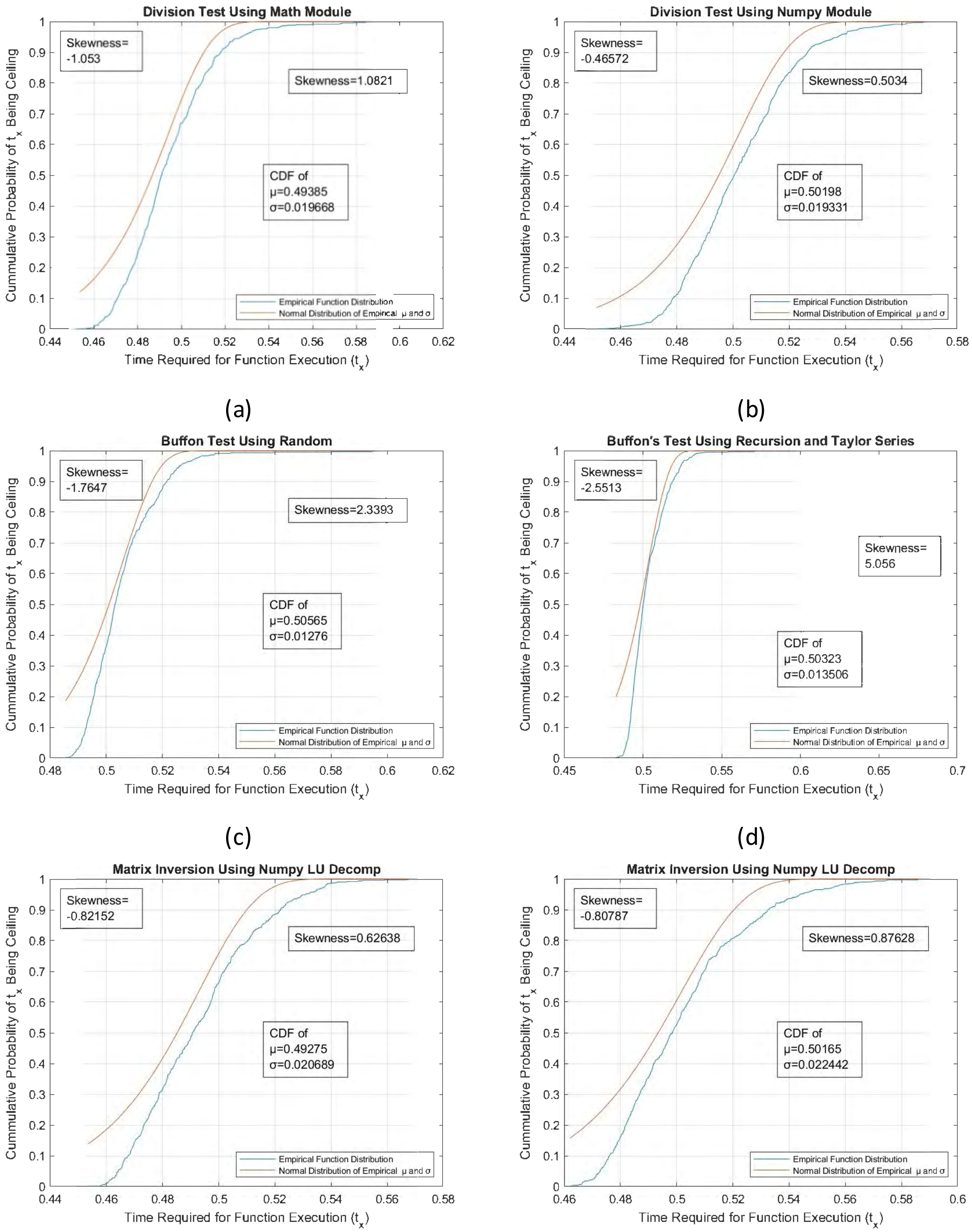
Candidate functions were evaluated for stationarity in stand alone execution and as functions executing in pods on a Minikube cluster. A comparison of distribution characteristics, mean, standard deviation, and skew was evaluated for both the empirical case of data extracted, as well as an ideal distribution for the same mean and standard deviation.

Individual cases were examined for both serial and parallel executions. As a general rule, the ideal case was forward of the empirical case. See Figs. G.20. With a single exception, floating point division, the mean time of execution for parallel execution hovered around 20 times the average serial execution time. In all cases, the distribution was elongated within the parallel analysis.

G.5.1 Serial Execution

In Fig. G.20, Serial execution timing analysis of candidate Python functions indicate that the execution time of any individual function follows closely to a normal distribution. This indicates, by the central limit theorem, that a large number of independent factors influence the execution time of the function or that the execution time is subject to near additive white Gaussian noise.

In most cases, the cumulative distribution of actual and ideal (based on the mean and standard deviation) of the candidate python codes track very closely, with the skew in both cases mostly in the order of -1 and in no case being above the first order. A notable trend is that an increase in skewness occurred as the empirical and ideal graphs converged.



(a)

(b)

(c)

(d)

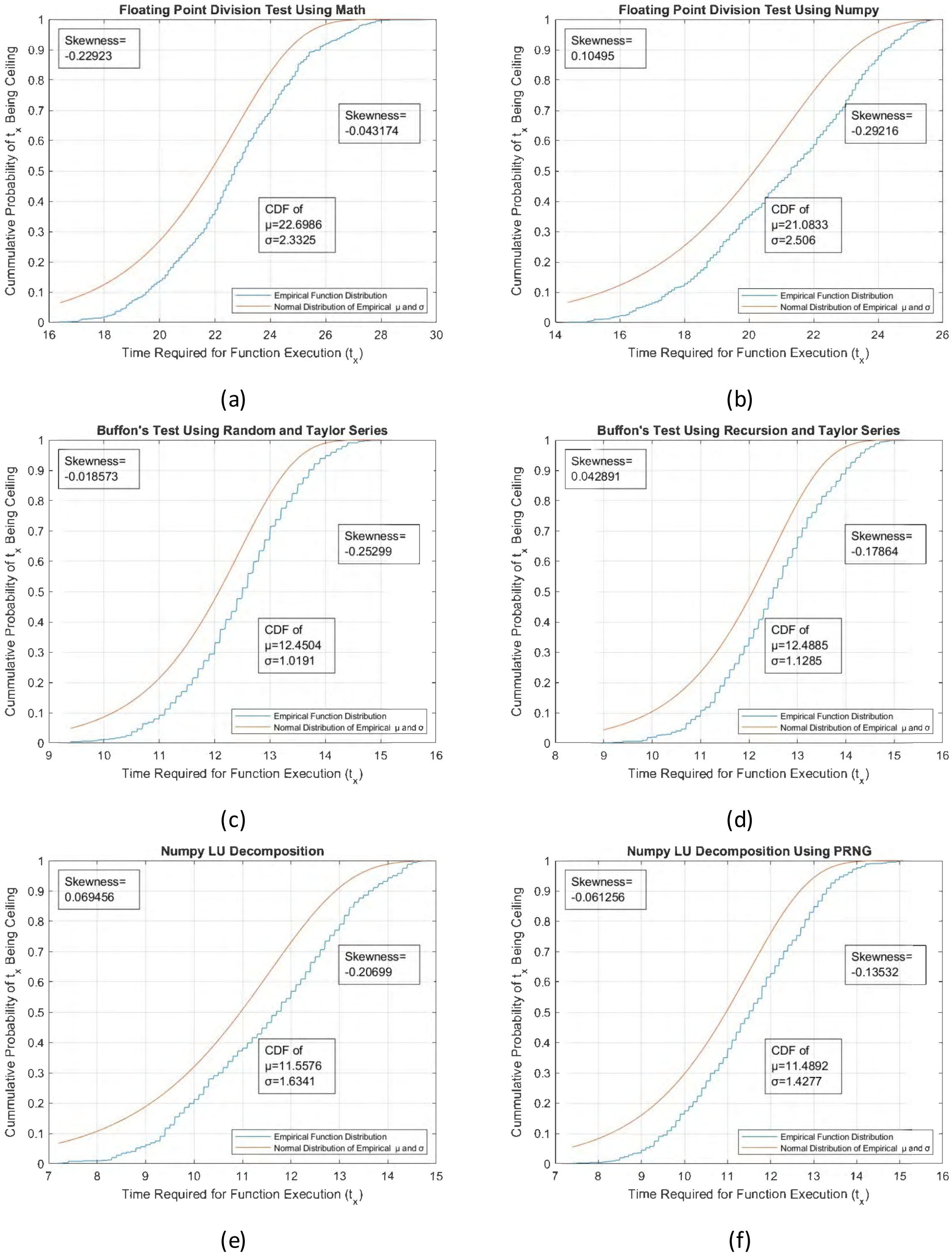
(e)

(f)

Fig. G.20. Comparison of alternate subroutines in Python code, serial execution. (a) and (b) compare the floating point division test using the Math library a routine using the Numpy library for convergence testing. See comparison of Fig. 4.5 for (c) and (d) for Buffon's needle test routines and (e) and (f) for matrix inversion.

G.5.2) Parallel Code Execution) In Fig. G.21, parallel code execution was conducted on a four name-space Kubernetes cluster with sixteen functions executing in separate pods per each name-space. Execution was initiated from a Kubernetes YAML file in the configuration of G.19. The data set was collected over a series of 10 tests.
Timing analysis of candidate Python functions indicate that the execution time of any individual function follows closely to a normal distribution. Like the serial case, the possibility of influence of many independent factors or near additive white Gaussian noise is suggested by the empirical cumulative distribution near the ideal normal distribution. The large number floating point division characteristic remains an outlier.

In all cases, the cumulative distribution of empirical and ideal the candidate python codes track very closely, with the skew in no case being above the -1 order. Each graph is separated at the empirical and ideal by a greater margin than in the serial case. This tends to indicate some degree of effect from the overhead of running on a Kubernetes environment.



(a)

(b)

(c)

(d)

(e)

(f)

Fig. G.21. Comparison of alternate subroutines in Python code, parallel execution. (a) and (b) compare the floating point division test using the Math library a routine using the Numpy library for convergence testing. See comparison of Fig. 4.6 for (c) and (d) for Buffon’s needle test routines and € and (f) for matrix inversion.

The proximity of the tested functions' empirical cumulative distribution and the idealized CDF with the same mean and standard deviation was accompanied by similar measures of skew: -.6885 and .4918 respectively. These results closeness to those from a normal distribution suggested that a large number of independent factors influenced the execution time of the functions. The central limit theorem informs this consideration. However, the appearance of a long tail outlier at the longest execution time as well as wide percentages of dispersion (3.44 percent and 3.30 percent respectively) prompted further evaluation in the parallel context of the chosen orchestration platform.

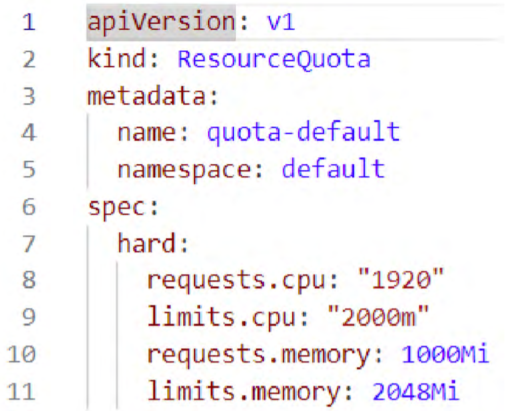
The statistical evaluation of the functions considered their alignment with the curve which most closely matched their cumulative distribution. In both cases above, the knee and inflection points of the curve where not symmetrical. However, a low skewness (3rd order moments of .5666 and .4918 respectively), prompted evaluation of the cumulative distribution and comparison to normal. The results of this analysis are in Fig. G.20.

G.6 Unscheduled Parallel Container Data

The Figs. G.21 (a-f) display the execution time of Python functions execution times executing from the parallelism tag in a Kubernetes YAML file (see Fig. G.17) shading on the base of each column indicates name-space (NS) in the Kubernetes cluster. Figs. a - d all execute on the same name-space, therefor have the uniform black color. The bottom Figs. (e and f) are executed in four separate name-spaces. This is a convention used throughout to depict name-spaces running in parallel.

The function is a matrix inversion using the Numpy library. Figs. in the left column (a, c, e) depict execution on a Dell Inspiron. Figs. on the right (b, d, f) depict execution on a Gateway i3 with 10GiB memory. Both are using he default CPU and memory allocations on a Minikube environment.

Data was ordered from the earliest start time to the last start time. The top Figs (a and b) depict the execution on the default name-space without resource quota specification. Figs. c and d are executed on the default name-space with limits at the default settings (see Fig. G.22).



```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota-default
  namespace: default
spec:
  hard:
    requests.cpu: "1920"
    limits.cpu: "2000m"
    requests.memory: 1000Mi
    limits.memory: 2048Mi
```

Fig. G.22. Resource quota set to the Minikube defaults at the limits. CPU and memory are set to requests at 5 milli CPUs and 3 MegaBytes per pod respectively.

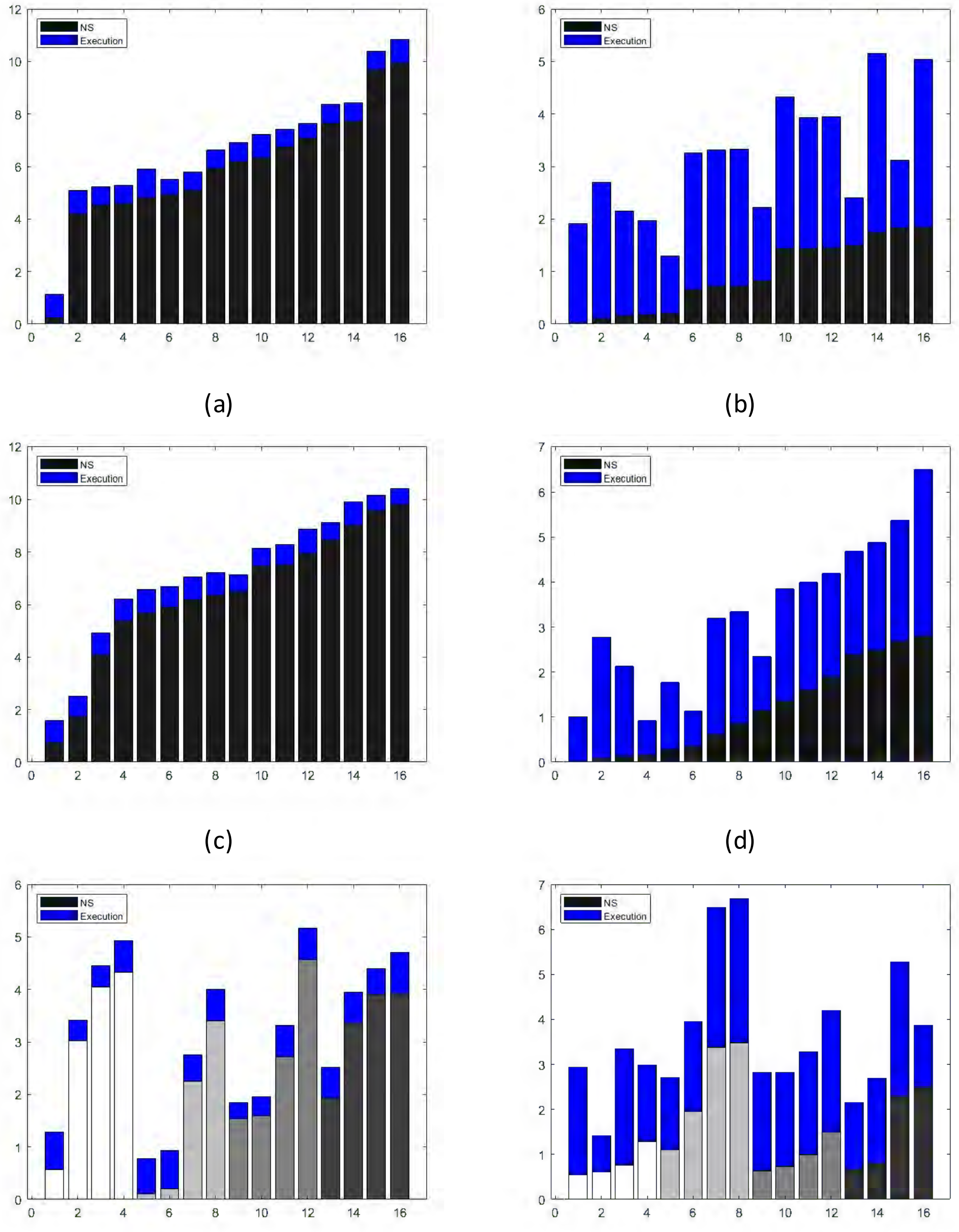
G.7 Platform Comparison

The execution times of functions are displayed in the following figures. Panels (a-f) display the execution time of Python functions on two platforms, a Dell Inspiron laptop running Linux LTS 22.04, and a Gateway i3 also running Linux LTS 22.04. Each function is the same, executing a Python matrix inversion routine activated by the parallelism tag in a Kubernetes YAML file. The name-space shading follows the convention as in Appendix G.6.

Panels (a) and (b) depict the execution on the Python code on a single, default namespace within the Minikube cluster. The start time (0.0 seconds in the graphs) is the floor taken from the earliest execution time start. The remaining pods are placed in order of function execution start time. Panels (c) and (d) depict the same function execution, but in a quota based namespace. The quota matched the default resources assigned to the default quota. Panels (e) and (f) depict execution of the same Python function split out into 4 namespaces. Each divided the default amount of resource equally, and each namespace executed the code in 4 parallel running pods.

The platform of execution influenced the characteristic of the run-time and execution. This was not tested for each function used in this experiment, but this characteristic was notable on other platforms used in further tests. Finally, the phenomena of multiple start times and a reduction in overall execution time bolstered the argument that parallelism characteristics effect the overall performance. This was central to the basis for the final test of the experiment.

G.7.1) Unscheduled Pod Data:



(a)

(b)

(c)

(d)

(e)

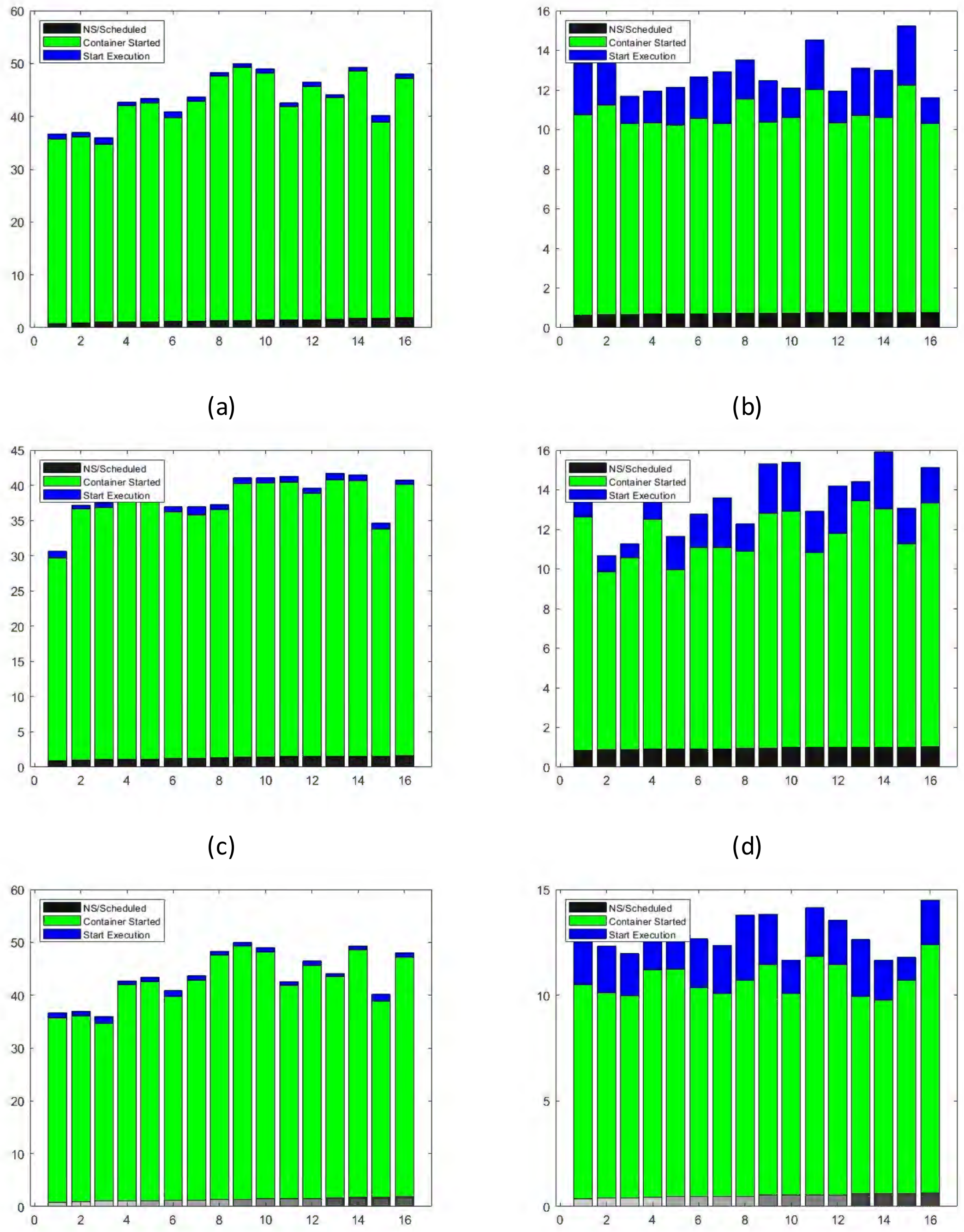
(f)

Fig. G.23. Graphical depiction of a Python function execution times on a Kubernetes cluster executing in parallel. The x-axis is functions numbered x = 1-16. The y-axis is the execution time (t_x) stacked onto the next nearest integer second.

G.7.2 Scheduled Pod Data

The Figs. G.24 (a-f) display the execution time of Python functions executing from the parallelism tag in a Kubernetes YAML file. The name-space shading follows the convention as in Appendix G.6. As in Appendix G.6, the platform of execution influenced the characteristic of the run-time and execution.

Data was ordered from the earliest scheduling time to the last scheduling time. The graph indicates a wide gap in the timing for configuration of the name-spaces and pods. Additional metrics were sought to evaluate the timing data of functions executing in an orchestrated container environment.



(a)

(b)

(c)

(d)

(e)

(f)

Fig. G.24. Graphical depiction of a Python function execution times arranged in order of Kubernetes cluster scheduling by Kubernetes name-space. The x-axis is functions numbered x = 1-16 in the default name-space and 1-4 in separate name-spaces. The y-axis is (t_x) stacked onto the next nearest integer second.

Damon Alsup
dalsup@pvamu.edu

EDUCATION

- B.A. Russians Studies, University of Houston, Houston, Texas, 1993
- B.S. Electrical Engineering, Prairie View A&M University, Prairie View, Texas, 2017
- M.S. Electrical Engineering, Prairie View A&M University, Prairie View, Texas, 2019

WORK EXPERIENCE

- University of Houston
  Computer Consultant
  Advisor to academic data management

- US Army Reserves
  Signal Officer
  Telecommunications management

- Houston Independent School District
  Teacher
  Computer and math instruction

- Blinn College
  Professor of Engineering
  STEM educator

PROFESSIONAL, TECHNICAL, AND SERVICE RELATED

- Humane animal treatment: TNR, rescue, foster and health
- Society of American Military Engineers
- Volunteer EMT

PUBLICATIONS AND PRESENTATIONS

- Game Theory Scoring of Cloud Co-Resident Risk Through Predation Modeling, D. Alsup, S. Cui, M. Putluru, Y. Zhang, Cluster Computing, 2023
- Security in the Serverless Functions as a Service Environment: A Stakeholder Based Approach, D. Alsup, S. Cui, Discover Computing, Pending
- Smart Meter Radiation, M. N. O. Sadiku, D. Alsup, S. M. Musa, International Journal of Advances in Scientific Research and Engineering (IJASRE), 2020