IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

ANALYZE THE COST IMPLICATIONS OF ADOPTING SERVERLESS COMPUTING MODELS

1Mahesh Kadam, 2Ganesh Wayal

1MTech Computer, 2HOD Computer

1TSSM's PADMABHOOSHAN VASANTDADA PATIL INSTITUTE OF TECHNOLOGY (PVPIT),

2TSSM's PADMABHOOSHAN VASANTDADA PATIL INSTITUTE OF TECHNOLOGY (PVPIT)

ABSTRACT

With the rise of serverless computing, new and compelling frameworks for serverless applications have emerged, which are helping to drive the trend toward container and microservice applications. This points to the fact that serverless computing is becoming more vital to business gatherings, conferences, blogging, and development. Key generation time (minimum, average, and maximum), encoded key size, and signature time are some of the security parameters that undergo experimental investigation. Serverless network metrics like as latency, average throughput, average response time, error rate, load distribution, and cost-efficiency are also used for testing. But there was hardly any excitement among academics. Simulation findings on serverless network metrics reveal that, in comparison to identity-based cryptosystems and attribute-based encryption, filter-based security utilizing fuzzy logic delivers better average throughput while reducing average response time, error rate, and load distribution. Additionally, when compared to attribute-based encryption and identity-based cryptosystems, testing on security metrics reveals that filter-based security employing fuzzy achieves lower encoded key size, signature time, and key generation time with reduced minimum, average, and maximum time. The research also demonstrates that fuzzy filter-based security is cost-effective. An important part of making decisions in cloud computing management is analyzing the many kinds of elements that affect the costs of cloud computing services.

KEYWORDS: Serverless Computing Models, Cost, fuzzy logic, security, cost-effective

INTRODUCTION

Initially, in the cloud's infancy, a wide variety of resources at various tiers (IaaS, PaaS, and SaaS) could be made available as a service on demand. Without a question, the advent of cloud services has encouraged the transfer of legacy applications to the cloud, allowing them to reach a wider audience. The difficulty of managing distributed systems, including instance selection, auto scaling, fault tolerance, monitoring, logging, and so on, comes with infinite computing resources. At the same time, micro-services are becoming more popular due to advancements in virtualization and the rise of new container technologies like Docker. A granular cloud service with easy-to-understand controls appears to be in high demand.

With the introduction of serverless computing, a new paradigm has emerged in cloud computing that frees developers from the burden of maintaining and supplying the underlying infrastructure and allows them to concentrate entirely on their applications. Developers may build cloud-based application functions that automatically activate in reaction to events using the Function as-a-Service (FAAS) implementation serverless paradigm. Serverless computing allows businesses to pay solely for the resources actually used by their applications, as opposed to the conventional cloud approach that requires resources to be reserved in advance regardless of consumption.

Gartner Group found that more than three quarters of businesses are using serverless computing or intend to within the next two years. Additionally, the serverless industry is projected to see tremendous growth, going from $3 billion in 2017 to an estimated $22 billion by 2025. Understanding the cost implications and finding relevant workloads are critical for successful adoption. Transitioning to a serverless computing architecture involves various issues, such as legacy system integration, cold start, and state management.

In recent years, serverless computing has gained popularity as a viable option for cloud application hosting. Minimal setup and configuration are required for serverless computing solutions, which provide autonomous fine-grained scaling of computational resources, high availability (24/7), fault tolerance, and charging only for real compute time. Serverless solutions make use of transient infrastructure like MicroVMs or application containers to accomplish these goals. By allowing cloud providers to more quickly combine customer workloads to utilize available capacity and deallocate unneeded servers to conserve energy, the serverless architectural paradigm change eventually promises greater server utilization. With the serverless model's emphasis on on-demand resource provisioning and pricing that reflects only real computation time, re-architecting apps for it offers lower hosting costs.

LITERATURE REVIEW

Andi, Hari (2021) This article provides a concise overview of the serverless cloud computing paradigm, including its idea, advantages, and applications in the IT industry. The old paradigm, which included three service-based models (IaaS, PaaS, and SaaS), held that developers should allocate resources, manage servers, and own servers. Infrastructure as a service (IaaS) allows the cloud provider to store and access data, software as a service (SaaS) allows users to have access to various applications, and platform as a service (PaaS) allows developers to have access to specific services for organizing and running their projects. With serverless cloud computing, the supplier of cloud services handles all aspects of server ownership, management, and maintenance, relieving the developer of this burden. Therefore, this strategy is cost-effective and significantly shortens the time it takes for a system to reach the market. Amazon Lambda, Microsoft Azure, and Google Cloud Platform are the three main types of serverless architecture. Among its drawbacks, which are detailed later in this article, is the fact that it is useless when a procedure takes too long to complete.

Mahajan (2019) Clients might expect lower costs and more flexibility with serverless computing, while cloud providers can expect more profits and better usage of their resources. We examine the possible financial advantages of serverless computing for both cloud service providers and end users in this article. We compare serverless computing (SC) with standard cloud computing (VM) using realistic cost models, queueing theory performance models, and a game theory formulation. The suggested model has the cloud provider setting pricing for SC and VM in order to maximize profit, while consumers divide their workload between SC and VM to save cost while keeping a certain performance restriction. We investigate the effect of provider expenses, service capacity, customer workload, and comparable pricing. The primary outcome of our work is the discovery and description of three provider-and customer-beneficial operating regimes that make use of SC alone, VM alone, or a combination of the two. Cloud providers and their clients may benefit from this paper's many insights on the pros and cons of a hybrid system that uses serverless computing, virtual machine renting, and related pricing methods.

Thatikonda, Vamsi (2023) Serverless computing, or just "serverless," is changing the game when it comes to application development, deployment, and runtime. By removing the need for developers to worry about the underlying infrastructure, serverless computing frees them up to concentrate on code while cloud providers handle server management. Instead of being charged for pre-allocated capacity, customers are now charged for real resource use, thanks to dynamic resource allocation, which is the outcome of this paradigm change. The article describes serverless computing and its main parts, which are FAAS and BaaS, which stand for "function as a service" and "backend as a service," respectively. We highlight the advantages of serverless, which include its cost-effectiveness, built-in scalability, quick development, and decreased operating requirements. But it does have its limits. Discussed are issues like "cold starts," possible vendor lock-in, limited flexibility, and particular security holes. Web apps, data processing, Internet of Things (IoT) backends, chatbots, and temporary jobs are all examples of practical serverless applications. Last but not least, organizations should carefully weigh the benefits and drawbacks of serverless computing, especially as the market changes. Organizations need to assess their preparedness for the serverless revolution since it is the wave of the future.

Hassan (2021) Thanks to its significant impact on a variety of issues—including but not limited to cost reduction, latency reduction, improved scalability, and elimination of server-side management—serverless computing has emerged as a promising new area of study in the last decade. Unfortunately, academics and developers still lack a comprehensive study on the topic of serverless computing and its relevance in many settings. Therefore, it is crucial to provide published study evidence in this field. In order to compile relevant data for this systematic review, 275 studies that looked into serverless computing were culled from reputable literature sources. Afterwards, a number of research concerns about the most recent developments in serverless computing, including its principles, platforms, use, etc., were addressed by analyzing the collected data. We also go over some of the current issues with serverless computing and how future studies can make it easier to use.

Adzic (2017) In late 2014, Amazon Web Services introduced its "Lambda" platform. Since then, all the main cloud infrastructure providers have introduced new services that work in the same way. Instead of deploying and managing large services or virtual machines, users can now deploy individual functions and pay only for the time their code is actually running. The suppliers indicate that these technologies, which are marketed as "serverless," may drastically alter the design, development, and operation of client/server applications. The article delves into two case studies of early adopters from the industrial sector to demonstrate how moving an application to the Lambda deployment architecture cut hosting costs by 66% to 95%. It also explores how this trend could impact common software architecture design practices if it continues to gain traction.

PROPOSED ATTRIBUTE BASED ENCRYPTION MODEL

Using an attributed-based encryption on serverless computing architecture, this section proposes an access control system. First, the data is encrypted with the help of user characteristics. Then, it is divided into cipher text. At last, a decryption algorithm deciphers it. The encrypted text is then disseminated over the network, and the serverless system stores the wrapped letters. In the serverless computing concept, data is encrypted and transferred in a searchable way. Data cannot be accessed prior to its scheduled expiry period and will be deleted at that point. Therefore, with the serverless architecture, no one other than authorized users may access the material. Service provider, owner, user, attacker, third party, and server are the six organizations that the suggested system use to securely access the data. Users are given the option to save and retrieve their data by the service provider.

The data is encrypted before being stored in the serverless system. Authorized users may access the original data by decrypting it using the decryption keys and the appropriate characteristics. The entities that attempt to access the server without authorization are known as attackers. Before and after the release and expiry times, respectively, they try to access the data.

By allowing a third party to manage the properties, the serverless architecture safeguards the data from malevolent users. You may create the system's parameters—public parameters, secret, and decryption key—through a third party. The reference time is recorded by the time server without interactions. It keeps track of the exact moment when the time-sensitive key changes are released. The network nodes are used for key management and storage. In their last-ditch effort, the criminals try to steal the protected key shares.

SECURITY REQUIREMENTS

In this section, we detail the security criteria that will be applied in this investigation.

i. **Data Confidentiality:** Even a malevolent user cannot obtain sensitive information. In order for an authorized user to have access to the data, access permissions must be met, which includes having sufficient credentials.
ii. **Collision Resistance:** Due to collision resistance, many users are unable to decode encrypted material, even when they work together and combine their unique encryption keys.
iii. **Attack Resistance:** The suggested system is resistant against several cyber and brute-force assaults. In a brute-force assault, the bad guy tries every conceivable key until he finds one that works, and in a cyber-attack, he utilizes many identities to crack the code.
iv. **Non-accessibility of sensitive data before release time:** The sensitive data is available before the appropriate release time within the permission period, but users are not permitted to access it.
v. **Delete data after expiration time:** After the expiry period has passed, the sensitive data will be automatically deleted.

SECURITY ASSUMPTION - BILINEAR MAP

Let G_0 and G_1 it is thought of as prime-order multiplicative cyclic bilinear groupings p. Envision g as a generator of G_0. As a kind of map, a bilinear map $e : G_0 \times G_0 \rightarrow G_1$ possessing the qualities listed below:

- **Bilinearity:** For all $g \in G_0$ and $a,b \in Z_p$, we have $e(g^a, g^b) = e(g,g)^{ab}$.
- **Non-degeneracy:** $e(g,g) \neq 1$.
- **Computability:** There is an efficient algorithm to compute $e(u,v)$ for $u,v \in G_0$.

Consider G_1, G_2 and GT constitutes a bilinear group, where g and h are the generators of G_1 and G_2.

Define $g_i = g^{\alpha i}$ for an unknown $\alpha \in Z_p^*$ and set $y = (g_1, g_2, \ldots, g_n)$ When given g, h, and y as input, algorithm B has a better chance of solving the BDHE issue ε. .

$$[\Pr[B(e(g_{n+1},h))=1]-\Pr[B(Z)=1]] \geq \varepsilon$$

FIS IMPLEMENTATION

In Figure 1, we can see that the output variable linked to the fuzzy logic system is expected to fall somewhere between very low, low, medium, high, and very high.

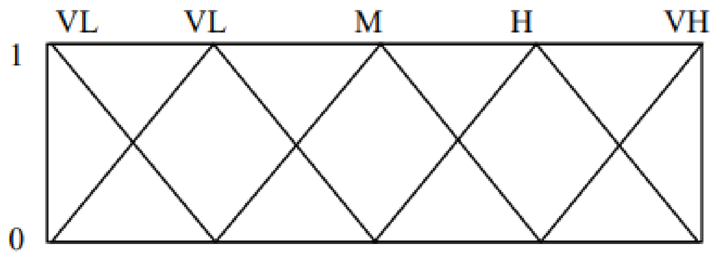

Figure 1. Cost value of Fuzzy membership functions

To get the desired result from each input, the FIS is programmed using If-Then rules. As a result of optimizing the fuzzy system's inputs VM reliability, maximum residual energy, and maximum connection costs may be reduced. The following rule forms the basis of the proposed system:

The output is deemed to be very low if R_l is medium, Rn is low, and E_r is low. In defuzzification, the input variables (R_n, R_l, E_r) determine the trust level of each message forwarded to Virtual Machines (VMs) using the value acquired from FIS.

RESULT ANALYSIS

In order to estimate the computational cost of the suggested technique, the Pairing-Based (PBC) library is used. A desktop computer with an Intel Core i7 3.4 GHz CPU, 500 GB of storage, and 8 GB of RAM is used for the research. We do the calculations for several iterations and then present the average results. File sizes ranging from 20 kilobytes to 240 kilobytes are used in the studies. Data encryption, ciphertext extraction, decryption key overhead, ciphertext share creation, and ciphertext share distribution constitute the anticipated computational overhead. As seen in Figure 2, the computational cost varies with file size.

Table 1: Computational overhead for different file size

File Size (kB)	Top K	SEED	Attribute Based Encryption
1	120	99	90
2000	155	134	125
4000	190	169	160
6000	225	199	190
8000	260	229	220
10000	295	259	250

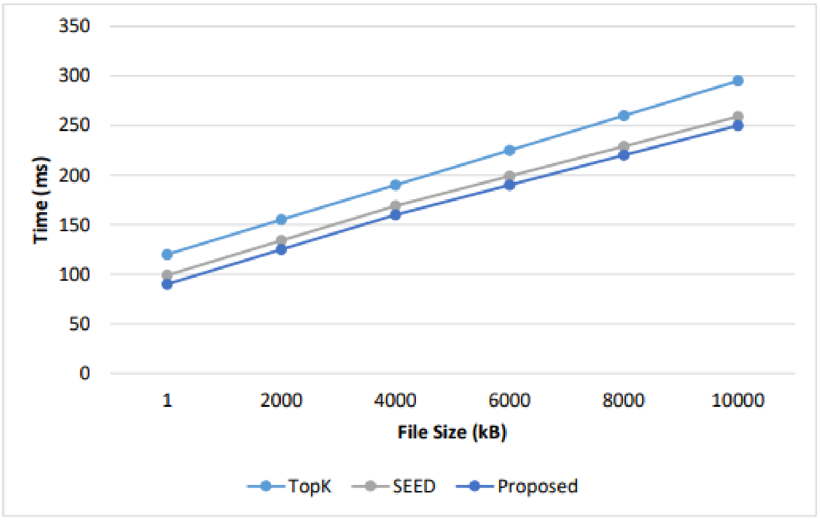

Figure 2: Computational overhead for different file size

In each of these cases, the suggested solution outperforms the current ones in terms of overhead. The current approaches' increased overhead is caused by the fact that encryption text must be associated before its components can be extracted. Conversely, computational overhead is reduced when encrypted text is not associated with it while it is being extracted. As both execution time and file size increase, the overhead in all the approaches, including the suggested one, seems to be linear.

Table 2: Encryption time with different attribute size

File Size (kB)	Top K	SEED	Attribute Based Encryption
1	122	50	42
10	277	200	187
20	389	270	244
30	494	400	365
40	603	500	463

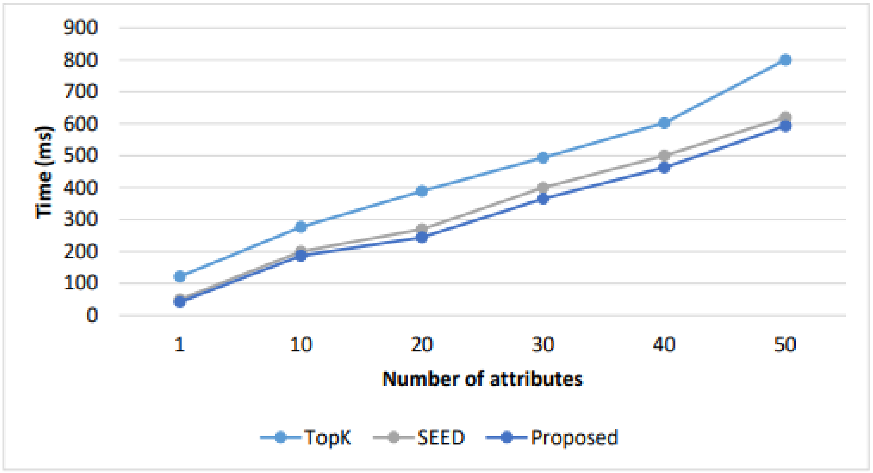

Figure 3: Encryption time with different attribute size

Look at Figure 3 and Table 2 for the encryption time results, then look at Figure 4 and Table 3 for the decryption time findings. For different types of users. The assessment takes place in milliseconds and compares the suggested approach to the current ones. The results are linear with rising user characteristics, as seen by the varied user attributes between 1 and 50. According to the simulation results, the suggested approach has a shorter encryption time compared to the current methods. The suggested approach also has a shorter decryption time than the current ones.

Table 3: Decryption time with different attribute size

File Size (kB)	Top K	SEED	Attribute Based Encryption
1	100	44	23
10	250	190	190
20	322	272	250
30	458	358	332
40	588	449	435
50	683	632	603

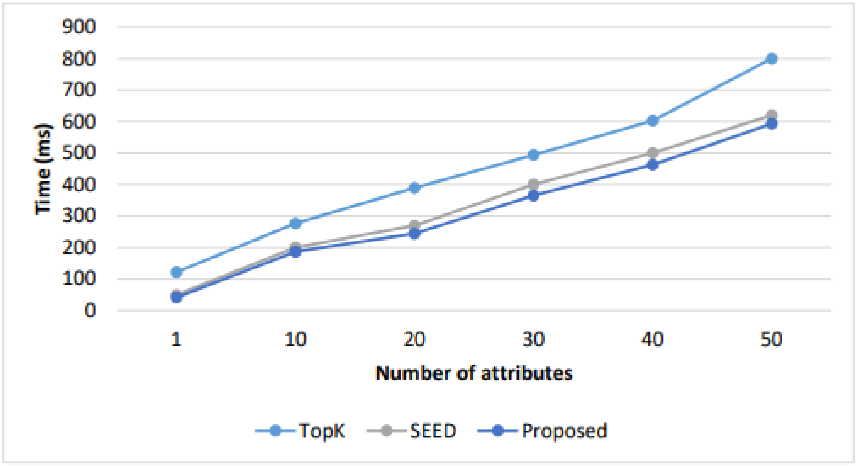

Figure 4: Decryption time with different attribute size

Figure 6 shows the results of the decryption time with regard to different file sizes, whereas Table 4 and Figure 5 show the results of the encryption time with respect to different file sizes. In milliseconds, we compare the suggested approaches to the current ones and evaluate them.

Table 4: Encryption time with different file size

File Size (kB)	Top K	SEED	Attribute Based Encryption
1	1100	700	100
100	1500	1250	740
200	1900	1700	1323
300	2200	2000	1743
400	2423	2300	2239
500	2746	2500	2493

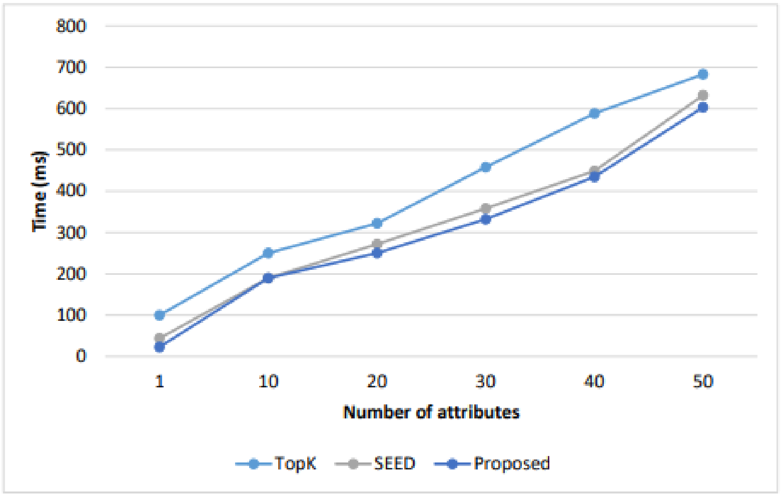

Figure 5: Encryption time with different file size

Table 5: Decryption time with different file size

File Size (kB)	Top K	SEED	Attribute Based Encryption
1	1023	600	250
100	1423	1002	734
200	1966	1432	1233
300	2342	1765	1734
400	2954	2303	2102
500	3124	2643	2543

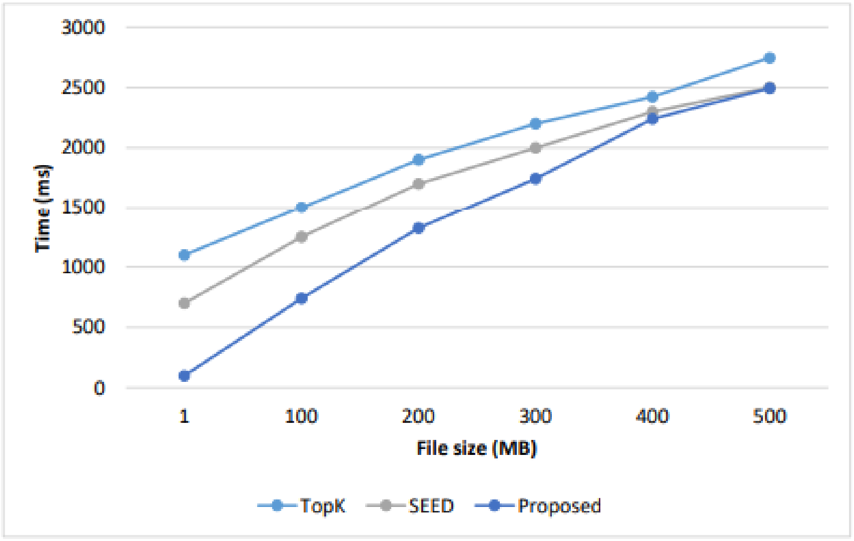

Figure 6: Decryption time with different file size

The findings are linear over the range of 1–500 MB, and then they start to show a steady increase as the file size increases. Due to the use of public key cryptography in the proposed technique, this criterion holds true for both the encryption and decryption times. The results of the simulation demonstrate that, for files of different sizes, the suggested method's encryption time is lower than that of the current approaches. Similarly, for files of varied sizes, the suggested method's decryption time is lower than that of the current approaches.

Cost implication in Function-level

Programming languages, performance, restrictions, composability, and deployment should always take precedence while creating serverless computing. Composability refers to various function compositions, while performance and limitations denote the maximum available memory and CPU resources; both are influenced by the design strategy of functions. Furthermore, the execution efficiency and the environment setup time are both directly impacted by the efficiency of the programming language.

- **Language Selection Strategy:** After the design is complete, implementing it is a breeze. But which language is best for a serverless platform? Jackson and Clynch investigated the two most prominent serverless technologies, AWS Lambada and Azure Functions, in order to find a solution to this issue. Their experiment with a tiny use case compared all the programming languages on these platforms, which are also typical on other platforms, in order to clear up the influence of language runtime on performance and cost. A single POST request to the API was used to invoke the function during cold start (when a function is called for the first time, there is a very high delay to initialize the container and the language runtime environment) and warm start conditions, during which all tests were executed in batches. Table 6 displays the outcome.

Table 6: Relationship Between AWS Performance and Cost

Language Runtime (ms)	**Warm Start**	**Cold Start**		
	Average Execution time	**Average Execution time**	**Average Billed Duration**	**Average Cost Per Million**
C#. Net	6.32	2500.09	2600	5.61775
Golang	19.21	8.97	100	0. 408375
JAVA 8	11.13	391.91	400	1.0335
NodeJS	11.46	23.67	100	0. 408375
Python	6.13	2.67	100	0. 408375

Mathematical Modeling of Cost Types

Strategic Choice, Cloud Computing Service and Cloud Type Selection (str): The time investment (eot) required to make a choice (in monetary terms) determines how much it will cost to implement a strategy and choose the best Cloud Computing Services. The expenditures for potential decision-making information (inf), such as scientific literature or market research, and the expenditures for external consulting services (cons). The sum of all workers' time spent on the task is what determines the entire cost of the spending. A worker's hourly wage multiplied by this number gives the total. $p^{str}_{eot,m}$ due to the time invested $a^{str}_{eot,m}$ as well as the total for all participating workers m: $C^{str}_{eot} = \sum p^{str}_{eot,m} * a^{str}_{eot,m}$. in times i<1, decision-making costs occur. Also, the overall price of all bought materials is the same as the total cost of purchased information materials (inf). Finally, the expenses associated with advising C^{str}_{cons} add up to a grand amount, as shown in Table 7 formula G.3. At long last, the sum of the cost elements of cost type str is: $C^{str} = C^{str}_{eot} + C^{str}_{cons} + C^{str}_{inf}$.

Table 7. General formulas

#	Formula
G.1	$TCO_{CCS} = \sum C^t$ with $t \in T$
G.2	$C^t = \sum C_f^t$ with $t \in T, f \in F$
G.3	$C_f^t = \sum_i^n C_{f,i}^t$ with $i = \{1, \dots, n\}, t \in T$ and $f \in F$
G.4	$C_{f,i}^t = a_{f,i}^t * \mathrm{p}_{\mathrm{f,i}}^{\mathrm{t}}$

Serverless computing platform performance modeling in time

The creation of an innovative model for temporal analytical performance that can forecast many key performance indicators across time. Because of the unpredictable and ever-changing nature of serverless computing platforms, it is crucial to use temporal performance models to keep performance measurements within reasonable limits; otherwise, steady-state performance models may no longer be applicable. A simple M/G/∞ queuing system with a Markovian arrival process, a general service process, and infinite processing capacity can be used to model an ideal platform for developing an accurate analytical performance model for serverless computing platforms. In this model, every request goes through the same service process and there is no latency caused by queuing. Cold starts can take orders of magnitude longer than warm starts, and there are limits on maximum concurrency levels, which can cause request rejection when the system reaches its capacity limit. Sadly, modern serverless computing platforms are still far from perfect. To construct an all-encompassing temporal performance model for serverless computing systems, we update the M/G/∞ queuing system to account for these alterations in this study.

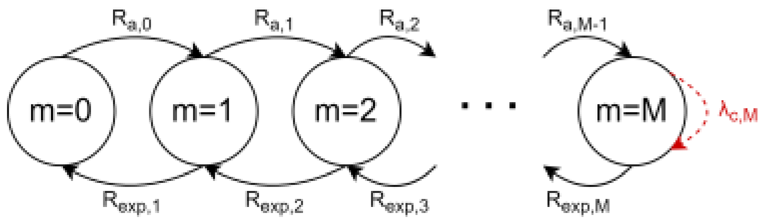

Figure 7. The warm pool's state transition diagram. The red dashed self-loop represents requests that were refused because there wasn't enough capacity.

- **Cold Start Rate**

As previously mentioned, we use M/G/m/m queuing systems to simulate the warm pool in all of its states. Assuming the warm pool rejects the model, this modeling approach creates a new instance, runs it through a cold start, and then adds it to the warm pool. Therefore, in order to get the cold start rate for every state, we must determine the rate of request rejection in the corresponding M/G/m/m queuing system.

- **Arrival Rate of Warm Instances**

In the suggested paradigm, we need to determine the arrival rate for each instance in the warm pool before we can get the instance expiry rate. In order to do this, we will first pretend that we have a warm pool of m instances shown as $\{I_1, I_2, \dots, I_m\}$, with I_1 having the top priority for new arrivals and I_m having the worst priority. Think of $\lambda_{w,m,n}$ as the rate at which the nth instance in the pool arrives.

CONCLUSION

This paper introduces a state-of-the-art serverless method that uses an effective security mechanism to safeguard data in a serverless computing environment. In order to assist with improved cost awareness in serverless computing, the aims to provide guidance on how to establish a serverless platform or design serverless apps. Our goal is to create a temporary analytical performance model that can understand serverless computing systems, their specific features, and forecast future resource needs and important service quality indicators. One of the most talked-about issues with cloud computing is the associated costs. Serverless computing offers promising future prospects because to its cheap cost, ease of use, and good performance, despite its relative youth. Protected data in a serverless environment enhances information and resource exchange while warding off several threats. This technique enhances serverless security and performance against the PBC library by deploying user characteristics. The simulation results demonstrate the effectiveness of the suggested strategy in safeguarding resources or data prior to their access by unauthorized third parties.

REFERENCES

1. Adzic, Gojko & Chatley, Robert. (2017). Serverless computing: economic and architectural impact. 884-889. 10.1145/3106237.3117767.
2. Hassan, Hassan & Barakat, Saman & Sarhan, Qusay. (2021). Survey on serverless computing. Journal of Cloud Computing. 10. 10.1186/s13677-021-00253-7.
3. Thatikonda, Vamsi. (2023). Serverless Computing: Advantages, Limitations and Use Cases. European Journal of Theoretical and Applied Sciences. 1. 341-347. 10.59324/ejtas.2023.1(5).25.
4. Mahajan, Kunal & Figueiredo, Daniel & Misra, Vishal & Rubenstein, D. (2019). Optimal Pricing for Serverless Computing. 1-6. 10.1109/GLOBECOM38437.2019.9013156.
5. Andi, Hari. (2021). Analysis of Serverless Computing Techniques in Cloud Software Framework. Journal of ISMAC. 3. 221-234. 10.36548/jismac.2021.3.004.
6. P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and virtual machines at scale: A comparative study,” in ACM International Middleware Conference, 2016.
7. L. Kleinrock, Theory, Volume 1, Queueing Systems, 1975.
8. M. J. Osborne and A. Rubinstein, A course in game theory, 1994.
9. D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, “A survey on spot pricing in cloud computing,” Journal of Network and Systems Management, 2017.
10. B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” Journal of Network and Systems Management, 2015