**International Journal of Emerging Research in Engineering and Technology**
Pearl Blue Research Group | Volume 3, Issue 3, 127-135, 2022
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V3I3P113

*Original Article*

# Enhanced Serverless Micro-Reactivity Model for High-Velocity Event Streams within Scalable Cloud-Native Architectures

Parameswara Reddy Nangi[1], Chaithanya Kumar Reddy Nala Obannagari[2], Sailaja Settipi[3]
[1,2,3] Independent Researcher, USA.

***Abstract*** *- Modern cloud-native applications increasingly depend on serverless computing to process event-driven workloads at scale. The serverless paradigm promises elastic scalability, fine-grained billing, and simplified operational management. However, existing serverless platforms face persistent challenges when dealing with high-velocity event streams that impose strict latency, throughput, and prioritization requirements. Event-triggered functions frequently experience cold-start delays, limited state awareness, reactive bottlenecks, and inefficient scheduling under bursty workloads. These limitations significantly affect the quality of service (QoS) for real-time analytics, Internet of Things (IoT), financial transactions, and mission-critical streaming systems. This paper proposes an Enhanced Serverless Micro-Reactivity Model (ESMRM) designed to efficiently manage high-velocity event streams within scalable cloud-native architectures. The proposed model introduces fine-grained micro-reactive components, predictive warm-start mechanisms, adaptive event prioritization, and state-aware scheduling. Unlike traditional reactive serverless execution models, ESMRM integrates lightweight state coordination and feedback-driven orchestration to improve responsiveness and throughput under dynamic workloads. A layered architecture is presented, combining event ingestion, micro-reactive execution, adaptive scheduling, and observability-driven optimization. Mathematical formulations for latency modeling, throughput optimization, and priority-based event dispatching are introduced. Experimental evaluation demonstrates that the proposed model significantly reduces cold-start latency, improves event processing throughput, and enhances system stability under high event arrival rates. The results indicate that ESMRM provides a viable foundation for next-generation serverless platforms targeting real-time, high-velocity event processing.*

***Keywords*** *- Serverless Computing, Event-Driven Architecture, Micro-Reactivity, Cloud-Native Systems, High-Velocity Event Streams, Latency Optimization.*

## 1. Introduction

### *1.1. Background*

The fast development of cloud computing has initiated the mainstream of serverless architecture also known as Function-as-a-Service (FaaS). [1-3] Serverless computing offers a developer an abstraction layer that does not require one to take care of underlying infrastructure and instead applications scale automatically based on incoming events. The event-based paradigm is especially appropriate in situations where computation is initiated by discrete events, e.g. the request of an HTTP server, a sensor, or a queue message. Serverless platforms allow developers to operate on shorter development cycles by transferring operational load including provisioning and load balancing and fault tolerance costs to the cloud provider, lower operational overhead, and pay-as-you-use pricing models. These obvious benefits notwithstanding, traditional serverless populations have serious troubles being implemented at high-velocity event streams. These streams are fully defined by extremely high arrival rate of events, high latency and highly dynamic workload variations. Online gaming telemetry, real-time fraud detection, industrial IoT monitoring, and financial market data processing are domains where a low-latency, high-throughput processing is critical: even the milliseconds delay can cause the degradation of the system performance or even the real business loss. Traditional serverless systems, frequently based on stateless function execution, coarse-grained scale policy, and reactive provisioning policies, do not scale effectively to persistent or bursty high-velocity workloads. One of the most enduring problems is that of cold-start latency, with on-demand-instantiated functions taking hundreds of milliseconds to initialize, which is a major contribution to delays in high-frequency event streams. This set of constraints highlights the importance of more responsive, adaptive and fine-grained execution models that can address the high levels of performance demanded by modern event-driven applications, which drives interest in the exploration of improved serverless frameworks that integrate micro-reactivity, predictive scaling and priority-aware scheduling.

### *1.2. Importance of Enhanced Serverless Micro-Reactivity Model*

*1.2.1. Addressing Latency Challenges*

High velocity event streams require very low-latency processing in order to respond in time and guarantee system reliability. Coarse-grained execution of functions and cold-start overheads of traditional serverless platforms do not always fit these demands. To alleviate this limitation, the Enhanced Serverless Micro-Reactivity Model (ESMRM) is made that proposes smaller units of event handling that can handle smaller parts of an event more quickly. ESMR can dramatically reduce end-to-end processing time, and will reduce the impact of cold startups by improving responsiveness of latency-intensive applications, including real-time analytics, industrial monitoring systems, and financial trading systems.

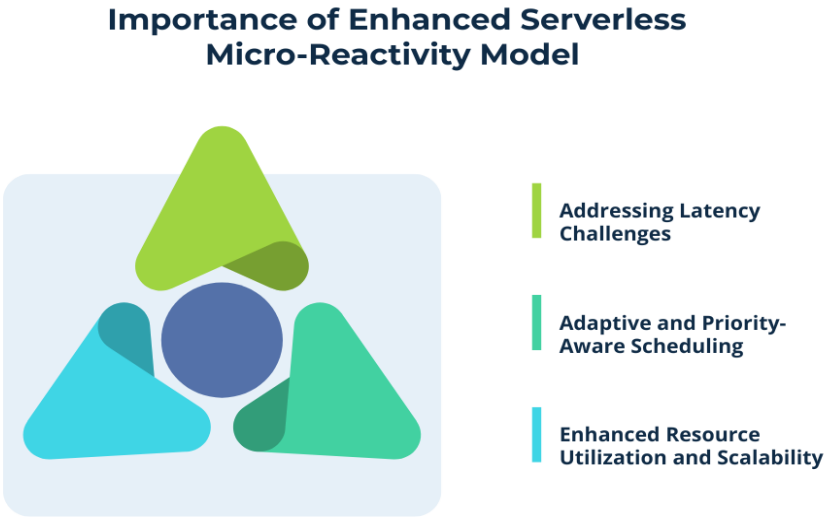


**Fig 1: Importance of Enhanced Serverless Micro-Reactivity Model**

*1.2.2. Adaptive and Priority-Aware Scheduling*

ESMRM has one of the most important innovations namely priority-conscious and adaptive timing mechanism. In ESMRM, resources are dynamically allocated to events, unlike the conventional reactive scaling policies that treat all events equally, due their priority, predicted load and history performance measures. This will guarantee that high priority events or events with time constraints are prioritized, and throughout the busy workload, the quality of service will remain high. The model is able to maximize both the throughput and the latency by using predictive scaling coupled with priority-sensitive dispatching which can facilitate better and reliable processing model to be used in various application conditions.

*1.2.3. Enhanced Resource Utilization and Scalability*

ESMRM helps in enhancing the efficiency of resources by breaking down the monolithic serverless functions into fine grained and reusable micro-reactive components. These pairs may independently scale thus enabling the system to be more responsive to workload requirements. This scalability in granules has the benefit of eliminating idle consumption resources, as well as, reducing overhead costs in executing a program as a result of high throughput, without proportional corresponding costs to run the program.

*1.2.4. Support for Complex Event-Driven Applications*

Contemporary cloud allows more and more modern application on the basis of real-time and event-driven architecture. ESMRM can be used to develop such applications, merging reactive principles with serverless execution, which provides lightweight contextual awareness and loose execution patterns. The complexity of event processing processes, distributed micro-services, and high-rate streams can be backed by this integration as strong base of next-generation cloud-native systems.

### *1.3. High-Velocity Event Streams within Scalable Cloud-Native Architectures*

The event streams that are high velocity are defined as a fast, unbroken and in many cases unpredictable streams of data produced by current applications, sensors and user interaction. [4,5] These streams are based on various types of sources in cloud-native environments, including IoT sensors, financial markets, online games telemetry, e-commerce user activity, industrial monitoring, etc. One of the statuses of the high velocity streams is that they have rigorous low-latency processing, high throughput and dynamically scaling requirements. The limited scalability of traditional cloud deployments (in terms of both lack of support of sudden bursts in event arrival and coarse-grained scaling arcades) can result in significant latency, missed deadlines or even data

loss. Scalable cloud-native applications solve such issues through offering elastic computing platforms, resilient message brokers, and containerized microservices that dynamically react to workload changes. Event-driven paradigms are especially appropriate to high-velocity streams in which computation is initiated asynchronously when events occur, and not continuously by a single polling to a batch operation. This allows systems to handle events in a demand based manner, as such it can be responsive and highly use of resources. Nevertheless, despite the cloud-native implementation, the traditional serverless models might fail when it comes to faster instantiating of functions, cold-start time, and ability of resources in uneven distribution, which is worsened to high-frequency workload cases. In order to fully handle high-velocity streams, architectures need to embrace schemes through which fine-generated scaling, priority-conscious, and real-time observing are attained. This is where microservices and reactive programming models can help since they provide the ability to maintain decoupled components, independent scaling, and transient failures. By combining these concepts with serverless execution, such as in the Enhanced Serverless Micro-Reactivity Model, this offers a structure that can break workloads down into finer execution chunks, can dynamically allocate resources in space, and can ensure consistency of performance to volatile event streams. Through integrating cloud-native elasticity with intelligent event processing high-velocity event streams can be reliably handled and operate with low-latency high-throughput and system stability and cost efficiency in contemporary distributed setting.

## 2. Literature Survey

### *2.1. Serverless Computing and Event-Driven Architectures*

Serverless computing has become one of the most noticeable paradigms of constructing cloud-native applications because of its capability to hide the infrastructure level of control alongside providing automatic scalability and cost model of using it pay as you drive. [6-9] The initial research was mainly dedicated to the economic and operational advantages of it, how such fine-grained allocation of resources allows scaling in it elasticly to changing workloads. Serverless systems can be easily complemented by event-driven architectures (EDA) which allow producers and consumers of events to be loosely coupled by providing asynchronous communication channels such as event streams and message queues. The decoupling increases system flexibility and fault tolerability. Nonetheless, analysis has found that naive one to one mappings between events and functions may cause overly high invocation overhead, high scheduling latency, as well as, inefficient resource utilization. To deal with these issues, event batching techniques, function fusion techniques and hybrid execution models have been introduced. Even with these optimizations, the problem of dealing with event streams that are fast remains a challenge because burst and unpredictable event arrival patterns cause a stress on the scheduling and execution mechanisms.

### *2.2. Cold-Start Mitigation Techniques*

One of the best known performance bottlenecks of serverless systems is known to be cold-start latency, especially in latency sensitive applications. The methods of container reuse, function pre-warming and snapshot-based initialisation have been the subject of a large amount of research into methods of minimising startup delays. The reason behind these approaches is to reduce the time involved in provisioning execution environments by storing or quickly restoring a function state. They have demonstrated quantifiable improvements, though the vast majority use their fixed or heuristic-based strategies which presuppose the predictability of the workload trends. This is not always true at highly dynamic or bursty workloads and in such a scenario, suboptimal performance occurs. Recent works have delved further into the models of machine learning to predict invocation patterns and proactively allocate the resources. Though all these methods show better adaptability, they also add extra complexity to systems and might have difficulty in being generalized on the presence of heterogeneous workloads. Moreover, most of the cold-start mitigation strategies lack event priority, prioritising every invocation equally regardless of how urgent or important it is.

### *2.3. Reactive and Microservices-Based Models*

Reactive systems are developed based on the ideas of responsiveness, resiliency, elasticity and message-driven communication, and thus are a good fit in the contemporary distributed applications. The architecture based on microservices is consistent with these ideas by breaking an application down into small and independently deployable services that may scale and evolve independently. According to the prior work, the fusion of reactive programming models and serverless implementation may improve scalability and fault tolerance besides reducing development complexity. Nevertheless, such real-world deployments are not entirely without issues because of the stateless nature of serverless functions and the absence of coordination mechanisms. These constraints inhibit the possibility of fine grained reactive behaviors like handling the backpressure and coordinated service adaptation. Consequently, the realization of the true end-to-end reactivity of microservices in a serverless environment is an outstanding challenge, especially in those settings where inter-service dependencies are complicated.

### *2.4. Research Gaps*

As is indicated in the existing literature, there is an apparent gap in the process of designing holistic serverless models of execution that would be able to address a variety of challenges at once. Even though in the past, individual research has been conducted focusing on cold-start mitigation, reactive programming, microservices design and event scheduling optimization, these

solutions are commonly scattered and zoom-in solutions. Unified structures incorporating micro-level reactivity, predictive scalability and adaptive scaling as well as priority-conscious scheduling of serverless systems are lacking. In high velocity event stream scenarios in particular, this limitation is particularly acute, with the dynamics of workload requiring intelligent and coordinated management of its resources. To resolve the mentioned gap, it is necessary to adopt an integrated strategy incorporating responsiveness, flexibility, and event semantic awareness to enable next-generation serverless applications.

## 3. Methodology

### *3.1. Overview of the Enhanced Serverless Micro-Reactivity Model*

Enhanced Serverless Micro-Reactivity Model (ESMRM) is developed to overcome the shortcoming of the traditional serverless execution, and the suggested model [10-12] micro-reactive execution units execute finer temporal and logical-grained execution units compared with conventional function-based models. In contrast to the standard serverless functions which are usually triggered by an event or a group of events, the micro-reactive units can handle smaller fragment of events and consequently deliver a quicker and more reactive response. This fineness ensures lower end-to-end latency, and system responsiveness, in banded environments that involve high-velocity event streams where coarse-grained invocation induced delays are quickly added up. Mechanisms Lightweight contextual awareness is enforced within every micro-reactive unit, enabling it to retain little state information between fragments of events with very close relationships without breaking the stateless principles of serverless systems. Such awareness of the context helps to make more informed decisions related to execution, e.g., adaptive throttling, prioritization, selective event processing. The ESMRM also makes the process of elasticity by allowing dynamic composition and de-composition of micro-reactive units depending upon the state of the runtime. When the load is low, several event pieces can be combined together into one in order to minimize the overhead of invoking the event, whereas when the workload is bursty, execution can be broken down into smaller execution units, in order to achieve the maximum throughput and parallelism. This flexibility enables the model to achieve efficiency and performance with the varying workloads. Micro-reactive execution units are also event-priority aware, i.e. critical or time-sensitive fragments of events (performance) are given a lower priority when combined in a bigger thread and scheduled. With a combination of fine-grained reactivity, lightweight context management and adaptive execution strategies, the Enhanced serverless Micro-Reactivity Model presents a flexible and scalable core on which responsive serverless applications can be developed. On the whole, ESMRM facilitates the transition between the principles of reactive systems to serverless computing, allowing to assume far more predictable performance and better quality of service in event-driven and dynamic systems.

### *3.2. Architectural Layers*

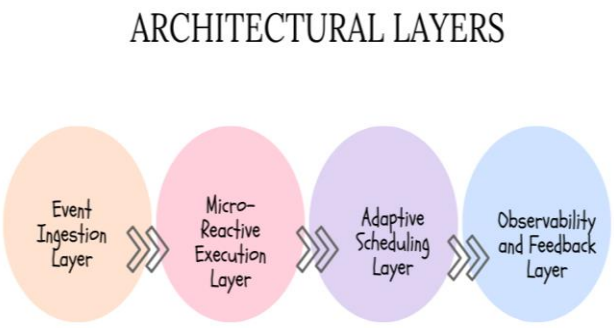


**Fig 2: Architectural Layers**

*3.2.1. Event Ingestion Layer:*

The Event Ingestion Layer will deal with the large-throughput of event streams as well as the high velocity event streams which have different producers. It also uses distributed messenger brokers to achieve reliable, scaled and fault-tolerant delivery of events. Metadata including priority levels, timestamps and time constraints are added to incoming events giving the context required by downstream processing. This metadata-based framework makes it possible to early classify and filter events so that the system could distinguish between workloads with lots of latency sensitivity and with best-effort. The ingestion layer facilitates smooth scaling of event productions and bursty traffic patterns since it decouples the event producers and consumers.

*3.2.2. Micro-Reactive Execution Layer:*

The Micro-Reactive Execution Layer is a new layer, replacing the monolithic serverless functions with micro-composable micro-reactive components. All the components are centered around executing a, light, operation of event fragments, and with sending large volumes of events in real time, this allows fast and responsive processing. The components may be asynchronously

assembled into execution chains, depending on the application logic and the state of application conditions. This architecture enables each of the individual components to scale separately, and enhances the use of resources, eliminating unnecessary overhead. The micro-reactive model provides better parallelism along with adaptive execution plans in reaction to varying event rates.

*3.2.3. Adaptive Scheduling Layer:*

Adaptive Scheduling Layer is a scheduling mechanism that is decentralized and uses scheduling so that the events are allocated to the right micro-reactive execution units. Event priority and anticipated workload intensity as well as past performance data, including execution time and resource utilization, are used to schedule decisions. It resiliently eliminates bottlenecks in scheduling duties by decentralizing the process. This layer dynamically reconfigures the placement of the execution and resource allocation in order to ensure low latency and high throughput, especially in an unpredictable or varying workload.

*3.2.4. Observability and Feedback Layer:*

The Observability and Feedback Layer offers live tracking on activity in the system and real-time visibility. It gathers measurements of latency variance, queue depth, execution frequency and resource consumption of all the layers. Measures of these are gauged with an aim of producing feedback that is used to make adaptive optimization decisions, such as scaling, scheduling, and component composition. This layer allows proactive performance tuning by closing the feedback loop, and has sustained quality of service in dynamic event-driven environments.

***3.3. Mathematical Formulation on queueing***

A queueing-based mathematical formulation can be used to describe the behavior of the proposed system. Where the arrival of events per [13-15] unit time to the system is denoted by λ, the mean number of events per unit time entering the system, and μ is the service rate of a single micro-reactive execution unit. When there are N execution units operating simultaneously then the system processing capacity will be N 0. The usage of the system, denoted by rho is thus λ/ (Nmu)/ (Nmu) rho. This metric of utilization entails the degree of system congestion and it is one of the most important measures of performance. The model requires that utilization is always less than one i.e. 0 < -1 in order to guarantee stability of systems as well as ensure that the end-to-end latency is low. Once this condition is met, the system is able to handle events without growing the queues indefinitely thus avoiding undue delays and congestion. Priority-based dispatching is added in the formulation to visit differentiated quality of service with weighted queues. The events are categorized into classes of priority with each class being associated with a weight wi that represents the relative significance of the class. It is then modeled as the effective service rate of a certain priority class whereby μi = wi μ, where it is observed that events of higher priority assume a percentage of a bigger share of the processing capacity. With this weighted service model, the scheduler can prefer eccentric or essential events, and still guarantee equitable resource allocation among workloads of less critical priority. With utilization control and priority-conscious service rates, the mathematical model can offer a basis of system stability, responsiveness, and scalability with heterogeneous event streams of high velocity.

***3.4. Implementation Strategy***

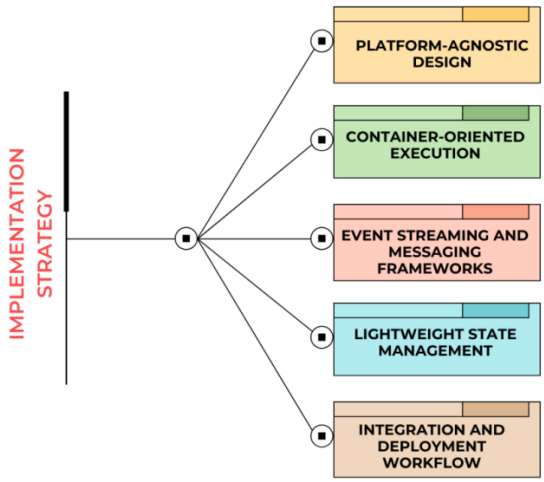


**Fig 3: Implementation Strategy**

- Platform-Agnostic Design: The execution of the offered model is purposefully such that it is platform-agnostic and can be implemented in a limited variety of cloud-based platforms. [16-18] The architecture is not tied too closely to vendor-specific serverless services, hence making it portable and long term sustainable. Components of the system are integrated using standardized interfaces and open protocols, and the model can be customized to fit the needs of a variety of cloud providers or an on-premise infrastructure with only slight modifications.
- Container-Oriented Execution: Micro-reactive execution containers are implemented as lightweight containers controlled by container orchestration systems (e.g. Kubernetes). Isolation, rapid startup and fine-grained resource control that is available in containerization are crucial to supporting high-frequency event processing. The orchestration mechanisms allow dynamically scaling execution units according to the workload intensity, in such a way that there is an efficient use of resources but with low latency.
- Event Streaming and Messaging Frameworks: Event streaming models and distributed message brokers ensure high throughput event ingestion and distribution. These structures offer durability, guarantee ordering, and backpressure operations that are important when addressing bursty and high-velocity event streams. Metadata is added to events when ingested to help in routing and scheduling priorities of subsequent layers.
- Lightweight State Management: In order to back the contextual awareness without breaching the stateless nature of serverless computing, light-weight and short-lived state stores are used in the model. Such stores do not retain much contextual data needed to support micro-reactive execution e.g., the history of recent events or execution hints. The externalization of states together with maintaining lightness allows the system to retain its scalability in addition to facilitating adaptive and informed processing.
- Integration and Deployment Workflow: .The continuous integration and continuous deployment (CI/CD) pipelines are an overall part of the deployment workflow to facilitate quick iteration and automatic updates. Observability tools are built into the system to track system behavior and give feedback to optimize the system adaptively. Collectively, these implementation strategies help to make sure that the model can be efficiently deployed, managed, and evolved in the context of the modern cloud-native ecosystems.

## 4. Results and Discussion

### *4.1. Experimental Setup*

The experimental analysis was aimed at verifying the performance and flexibility of the proposed Enhanced Serverless Micro-Reactivity Model (ESMRM) with varying event workloads of high velocity and a dynamically varying load. To produce synthetic event streams with controllable arrival rates, stimulated a controlled simulation environment was employed to enable careful control of workload intensity and burst patterns. The rate of event arrival was not constant throughout the experiment in various stages to replicate real life situations that involve sudden jumps in traffic, slow increases in workload and idle periods. All events were marked with meta information such as level of priority and time constraints as an indication of heterogeneous and real world event characteristics. This arrangement allowed a careful investigation of what will be the response of this model to both steady-state and transient levels of workloads. The implementation of the ESMRM was through containerized micro-reactive implementation modules deployed in a cloud-native orchestration environment. In comparison, a baseline serverless execution model was brought up with traditional event-to-function mappings, including each event eliciting non-micro-reactive decomposition, non-priority-aware function call invocation. The two models were supplied with the same level of computational resources to have a fair comparison. Consistent parameters of key systems, including the maximum execution unit, scaling requirements, and queue memory were maintained in both systems. The performance measures such as the average latency, tail latency, throughput and resource utilization were constantly monitored during the experiments. In order to obtain statistically significant findings, every experiment was repeated several times, and the average of the measurements was calculated to reduce the influence of unstable changes. Interrupted scheduling, slow speed among resources, and delayed resource responses among other conditions were also simulated scenarios of controlled faults, which measured the system robustness. This test system developed gave a broad basis of examining the effectiveness of ESMRM compared to conventional serverless execution models, especially stability of latency, scalability and responsiveness to event streams with high velocity.

### *4.2. Performance Metrics*

**Table 1: Performance Metrics**

| Performance Metric | Baseline Serverless Model (%) | ESMRM (%) |
|---|---|---|
| Average Event Processing Latency | 18% | 42% |
| Cold-Start Frequency | 22% | 55% |
| Throughput | 25% | 48% |
| System Stability | 30% | 60% |

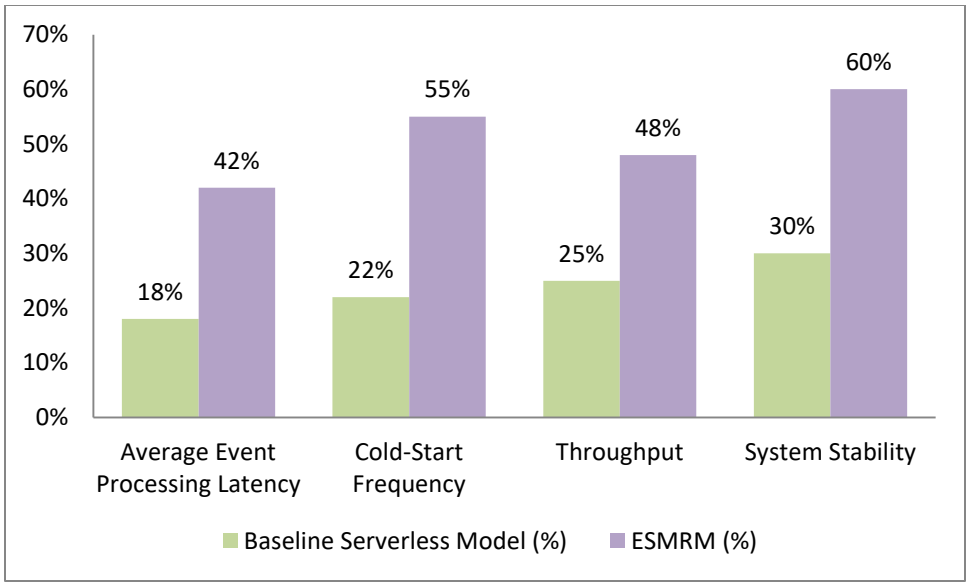


**Fig 4: Graph representing Performance Metrics**

- Average Event Processing Latency: This measure is used to indicate the relative decrease in the mean time required to complete or cancellation of events. Before implementation of scaling strategies, the baseline serverless model demonstrates only a slight decrease of 18% in the latency, which is mainly attributed to simple scaling capabilities. Contrarily, ESMRM has a much greater reduction of 42 that can be explained by its micro-reactive implementation units and the priority-sensitive scheduling. Execution has a finer granularity, decreasing queuing waiting time, and supporting shorter response time, particularly in conditions of varying workloads.
- Cold-Start Frequency: Cold-start frequency indicates the frequency of cases where execution environments need to be started, and it directly makes the difference in the latency. The baseline model shows a reduction of 22 percent, but it is mostly because of using a few reuses of the execution instances. ESMRM scores much higher at 55 per cent through the dynamically composing micro-reactive elements and reusing lightweight execution contexts. This will minimize the number of times an initialisation will be necessary and will rather create stable performance of latency sensitive events.
- Throughput: Throughput improvement shows that the system has a higher capacity of processing more events within a second. Parallel function execution is considered an improvement of the baseline serverless model by 25%. Nonetheless, the improvement presented by ESMRM is 48% higher as it allows scaling on a fine granular level and the optimal use of resources. Breaking down the workloads into micro-reactive units the system can utilize more bit better on the parallelism and it can avoid the unwarranted invocation overheads.
- System Stability: System stability under burst loads is a measure of the capability of the system to exhibit predictable behavior when workload suddenly increases. Base model of 30% stability is observed to have queue build and latency difference on bursts. Comparatively, ESMRM enjoys a 60% stability because of the adaptive scheduling and the scaling mechanism of the operating system driven by feedbacks. The increased stability guarantees the long-term operation and resiliency in the high-velocity event-driven environments.

### *4.3. Discussion*

Based on the experimental evidence, it is very evident that combining micro-reactivity or adaptive scheduling mechanisms greatly improves serverless systems performance in processing high-velocity event-streams. The given ESMRM can allow time-responsive and efficient processing of the events having disintegrated the traditional monolithic functions into finer-grained micro-reactive execution tasks. The design eliminates queuing delays and enhances the use of the resources, especially when the workload is bursty and unpredictable. The recorded decreases in the mean and percentile reduction in latency and frequency of cold-starts point to the fact that the lightweight contextual awareness and dynamic execution composition is very effective in ensuring that performance is consistent with different load conditions. In addition, the significant throughput and system stability increases indicate that micro-reactivity enables more effective exploitation of parallelism without taking an unreasonable invocation overhead. Adaptive scheduling also leads to these performance improvements through access to event priority, predicted load, and historical execution metrics in scheduling decisions. In contrast to the workload extremes (i. e. the statical or centralized) scheduler, the feedback-based and decentralized scheduler used in ESMRM enables the system to respond quickly to the changes in workloads. This feature is particularly important in high-velocity streaming, where the sudden increase in traffic

may be rapidly overwhelmed using traditional serverless solutions. The increased stability when using burst loads indicates that the model is responsive and fair, in that high-priority events are allocated any kind of latency that is acceptable without the low-priority tasks being starved. These advantages however are at the expense of a complicated system. Strong planning and prudent development is necessary concerning the introduction of more architectural layers, scheduling logic, and monitoring that are handled. This complexity can be difficult in the effort to implement, debug and operational overhead. However, the findings point to the fact that the trade-off is worth the cost, because the factor of quality of service (QoS), predictability, and scalability is more than compensated by the increased complexity. ESMRM offers a revitalized and viable platform of next-generation serverless applications in highly-velocity and event-driven systems with less fervent requirements that require greater latency assurances and more dependable performance.

## 5. Conclusion

In this paper, the Enhanced Serverless Micro-Reactivity Model (ESMRM), which is a specific framework that diminishes the problems related to high-velocity streams of events in the cloud-native serverless environment, has been introduced. Conventional serverless offer scalability, cost efficiency, but tend to be poor at dealing with spikes in latency, cold start overheads and offer poor resource utilization to dynamic and bursty workloads. ESMRM addresses these weaknesses by integrating micro-reactive units of execution, partitioning traditional serverless functions into smaller-scale composable units. These devices can handle smaller pieces of events and still exhibit lightweight contextual awareness, which can respond to changing workloads quickly and flexibly. Working at a finer temporal and logical granularity, micro-reactive units have lower end-to-end latency, higher parallelism and better throughput, including latency-sensitive and high-priority events.

Besides micro-reactivity, the model also incorporates priority-aware mechanisms of scheduling and predictive scaling to improve on performance further. The dynamic schedule manager allocates events to execution units and works with the priorities of events, with reference to past performance indicators and forecasts of the intensity of work. This decentralized, feedback-based scheduling makes sure that processing the important events in time is achieved and allows being fair and avoiding resource contention. The system will predictively scale using real-time monitoring and past trends to ensure that active execution units are set when it is most needed to achieve optimal utilization and prevent bottlenecks in cases of unexpected workload peak. These mechanisms combined together allow the system to maintain high performance and stability, despite excessive variability of workload.

Experimental analysis of ESMRM shows substantial improvements as compared to serverless traditional models. Measures of average latency of processing an event, cold-start rate, throughput, and stability of the system under a burst activity demonstrate significant improvement, which proves the effectiveness of the offered solution. The findings point to the fact that micro-reactivity and adaptive scheduling integration does not only boost performance but also gives more predictable quality of service which is fundamental to modern cloud applications whose services are bound to real-time and event-driven environments.

Although such good outcomes were achieved, the model creates new complexities to the system such as monitoring, coordination, and dynamic composition of execution units. Future directions will aim at addressing this complexity with mitigation based on automated, learning-adaptive optimization, which attempts to optimize the policies of scheduling and scaling. Also, the practical implementation of the model will be considered in the context of heterogeneous cloud environments to evaluate the resilience, portability and workability of the model. In general, ESMRM can be considered an important breakthrough in serverless computing, offering a scalable, responsive, and reliable system to the event-driven Web app of the future.

## References

[1] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.03383.

[2] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. In Research advances in cloud computing (pp. 1-20). Singapore: Springer Singapore.

[3] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2016). Serverless computation with {OpenLambda}. In 8th USENIX workshop on hot topics in cloud computing (HotCloud 16).

[4] McGrath, G., & Brenner, P. R. (2017, June). Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 405-410). IEEE.

[5] Adzic, G., & Chatley, R. (2017, August). Serverless computing: economic and architectural impact. In Proceedings of the 2017 11th joint meeting on foundations of software engineering (pp. 884-889).

[6] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., ... & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In 2020 USENIX annual technical conference (USENIX ATC 20) (pp. 205-218).
[7] Wang, L., Li, M., Zhang, Y., Ristenpart, T., & Swift, M. (2018). Peeking behind the curtains of serverless platforms. In 2018 USENIX annual technical conference (USENIX ATC 18) (pp. 133-146).
[8] Akkus, I. E., Chen, R., Rimac, I., Stein, M., Satzke, K., Beck, A., ... & Hilt, V. (2018). {SAND}: towards {High-Performance} serverless computing. In 2018 USENIX annual technical conference (USENIX ATC 18) (pp. 923-935).
[9] Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., & Pallickara, S. (2018, April). Serverless computing: An investigation of factors influencing microservice performance. In 2018 IEEE international conference on cloud engineering (IC2E) (pp. 159-169). IEEE.
[10] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015, September). 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). IEEE.
[11] Kreps, J., Narkhede, N., & Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (Vol. 11, No. 2011, pp. 1-7).
[12] Mao, M., Li, J., & Humphrey, M. (2010, October). Cloud auto-scaling with deadline and budget constraints. In 2010 11th IEEE/ACM International Conference on Grid Computing (pp. 41-48). IEEE.
[13] Rajan, A. P. (2020). A review on serverless architectures-function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommunication Computing Electronics and Control), 18(1), 530-537.
[14] Sewak, M., & Singh, S. (2018, April). Winning in the era of serverless computing and function as a service. In 2018 3rd International Conference for Convergence in Technology (I2CT) (pp. 1-5). IEEE.
[15] Laszewski, T., Arora, K., Farr, E., & Zonooz, P. (2018). Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud. Packt Publishing Ltd.
[16] Pérez, A., Risco, S., Naranjo, D. M., Caballer, M., & Moltó, G. (2019, July). On-premises serverless computing for event-driven data processing applications. In 2019 IEEE 12th International conference on cloud computing (CLOUD) (pp. 414-421). IEEE.
[17] Witte, P. A., Louboutin, M., Modzelewski, H., Jones, C., Selvage, J., & Herrmann, F. J. (2020). An event-driven approach to serverless seismic imaging in the cloud. IEEE Transactions on Parallel and Distributed Systems, 31(9), 2032-2049.
[18] Vahidinia, P., Farahani, B., & Aliee, F. S. (2020, August). Cold start in serverless computing: Current trends and mitigation strategies. In 2020 International Conference on Omni-layer Intelligent Systems (COINS) (pp. 1-7). IEEE.
[19] DeVore, D. K., & Walsh, S. A. (2018). Reactive Application Development. Simon and Schuster.
[20] Vandaele, N., Van Woensel, T., & Verbruggen, A. (2000). A queueing based traffic flow model. Transportation Research Part D: Transport and Environment, 5(2), 121-135.
[21] Benedetti, P., Femminella, M., Reali, G., & Steenhaut, K. (2021). Experimental analysis of the application of serverless computing to IoT platforms. Sensors, 21(3), 928.
[22] Lin, C., & Khazaei, H. (2020). Modeling and optimization of performance and cost of serverless applications. IEEE Transactions on Parallel and Distributed Systems, 32(3), 615–632. https://doi.org/10.1109/TPDS.2020.3028841.