**Journal of Computing and Electronic Information Management**
ISSN: 2413-1660 | Vol. 21, No. 1, 2026

# Autoscaling Stateful Microservices Under Variable Load and Traffic Uncertainty

**Minjae Rhee [1,*] and Jiesi Yang [2]**

[1] University of Illinois Urbana-Champaign, United States
[2] University of Utah, United States

*** Corresponding author:** rheeminjae45@gmail.com

**Abstract:** Modern cloud-native applications increasingly adopt microservice architecture (MSA) to achieve modularity, independent deployment, and scalability. While stateless microservices have been extensively studied in the context of autoscaling, stateful microservices (SMS) present unique challenges due to their dependency on persistent state, session continuity, and data locality. Autoscaling SMS under variable load and traffic uncertainty requires sophisticated mechanisms that transcend conventional reactive approaches. This paper provides a comprehensive review of existing methodologies for autoscaling SMS, encompassing reactive, proactive, and hybrid scaling strategies. We examine the role of machine learning (ML) and deep learning (DL) in traffic forecasting and workload prediction, the management of distributed state during scaling events, and the integration of service mesh technologies to mitigate traffic uncertainty. Key challenges including cold-start latency, state migration overhead, and quality of service (QoS) degradation during scaling transitions are discussed in depth. We further review benchmark frameworks and evaluation methodologies used to assess autoscaling systems. The paper concludes by identifying critical open research problems and future directions in this rapidly evolving domain.

**Keywords:** Autoscaling; stateful microservices; Cloud-native; Traffic uncertainty; Workload prediction; Kubernetes; Service mesh; Resource management.

## 1. Introduction

The widespread adoption of microservice architecture (MSA) has fundamentally transformed how large-scale distributed applications are designed, deployed, and managed. Unlike monolithic systems, MSA decomposes complex applications into small, independently deployable services that communicate over well-defined application programming interfaces (APIs). This architectural paradigm enables teams to develop, test, and scale individual components without disrupting the broader system [1]. However, the inherent dynamism of microservice-based deployments introduces significant resource management challenges, particularly when services must respond to variable workloads and unpredictable traffic patterns.

Autoscaling—the automated adjustment of computational resources in response to observed or predicted demand—has emerged as a critical mechanism for maintaining performance guarantees while controlling infrastructure costs in cloud environments [2]. Container orchestration platforms such as Kubernetes have made horizontal pod autoscaler (HPA) and vertical pod autoscaler (VPA) capabilities available as first-class features, enabling operators to define scaling policies based on resource utilization metrics [3]. Despite these advances, conventional autoscaling mechanisms were primarily designed with stateless services in mind, where any replica can serve any request without coordination with other replicas.

Stateful microservices (SMS) constitute a substantial subset of real-world microservice deployments, encompassing databases, caching layers, message queues, session stores, and stream processing operators. Unlike their stateless counterparts, SMS maintain persistent data or session state that must be preserved, replicated, or migrated during scaling operations [4]. The act of adding or removing replicas of a stateful service requires careful orchestration of state transfer to ensure data consistency and minimize service disruption. This characteristic makes autoscaling SMS fundamentally more complex than autoscaling stateless services and prevents direct application of existing techniques.

Traffic uncertainty further compounds the difficulty of autoscaling SMS. Real-world workloads exhibit diurnal patterns, sudden spike events, and long-tail distributions that are inherently difficult to predict [5]. Reactive autoscaling approaches, which respond to observed violations of performance thresholds, suffer from scaling lag—the delay between the onset of overload and the availability of new service replicas. For SMS, this lag is amplified by the additional time required for state initialization and synchronization. Proactive autoscaling methods that leverage ML forecasting models have shown promise in anticipating demand surges, but their effectiveness depends heavily on the accuracy of workload prediction models and their adaptability to concept drift in traffic patterns [6].

The intersection of stateful service management and autoscaling under uncertainty represents an active research frontier with significant practical implications. Cloud providers, enterprise operators, and platform engineers face pressing needs for principled approaches to this problem. This paper surveys the current state of research, synthesizing contributions from container orchestration, distributed systems, workload prediction, and resource management. The review is organized to first examine foundational concepts in stateful microservice architecture and state management, then progress through autoscaling policies and decision mechanisms, traffic prediction methodologies, and evaluation frameworks. Each section draws upon recent literature to characterize the problem space and highlight the most promising technical directions.

The contributions of this survey are threefold. First, it

provides a structured taxonomy of autoscaling approaches applied to SMS, distinguishing reactive, proactive, and hybrid paradigms. Second, it synthesizes findings across the ML-based workload prediction literature as applied to containerized environments. Third, it identifies unresolved challenges and articulates open research questions that warrant further investigation by the distributed systems and cloud computing communities.

## 2. Literature Review

The literature on autoscaling in cloud and microservice environments has grown substantially since the mid-2010s, driven by the rapid adoption of container-based deployment models. Early work on autoscaling focused primarily on virtual machine provisioning in infrastructure-as-a-service (IaaS) platforms, employing threshold-based rules and proportional-integral-derivative (PID) controllers to govern scaling decisions [7]. These rule-based systems, while simple to deploy, struggled with oscillation, under-provisioning, and slow response to non-stationary workloads. The transition to container orchestration, exemplified by the Kubernetes ecosystem, introduced finer-grained resource management and opened the door to more sophisticated autoscaling strategies [8].

Proactive autoscaling using time-series forecasting emerged as a promising alternative to purely reactive mechanisms. Researchers demonstrated that workloads in production cloud environments exhibit predictable periodic components that can be exploited by autoregressive models such as ARIMA [9]. Subsequent work extended these ideas using recurrent neural networks, specifically long short-term memory (LSTM) networks, which proved effective at capturing temporal dependencies in request arrival patterns [10]. The integration of such forecasting models with autoscaling controllers introduced new design questions about how to translate predicted load into concrete replica count decisions, a challenge that remains partially unresolved.

The rise of service mesh technologies, particularly Istio and Linkerd, provided new observability and traffic management primitives that enriched the data available to autoscaling systems. Service mesh sidecars collect fine-grained latency, error rate, and throughput telemetry at the per-request level, enabling autoscaling controllers to respond to QoS signals beyond simple CPU utilization. Recent work on cross-modal modeling further shows that jointly leveraging heterogeneous system signals—such as log sequences and performance metrics—through attention-based fusion mechanisms can improve the characterization of system dynamics, highlighting the value of multi-modal observability for more responsive and reliable autoscaling decisions in complex microservice environments [11]. Studies showed that latency-aware autoscaling policies outperform utilization-based policies for services with heterogeneous request types and variable processing costs [12].

The problem of autoscaling stateful services specifically has received comparatively less attention in the literature than its stateless counterpart. Early contributions addressed stateful autoscaling in the context of relational and NoSQL databases, focusing on read replicas and sharding strategies for horizontal scaling [13]. More recent work has examined stateful autoscaling in stream processing systems such as Apache Flink and Apache Kafka Streams, where operator state must be redistributed across workers during scale-out events [14]. The key insight from this body of work is that state migration constitutes a first-class concern in any autoscaling framework for stateful workloads, with migration latency directly influencing effective scaling speed and the degree of service disruption experienced by clients.

Reinforcement learning (RL) has attracted considerable interest as a framework for autoscaling decision-making because it can learn policies that trade off between competing objectives without requiring explicit formulations of the optimization problem [15]. Deep reinforcement learning (DRL) variants have been applied to Kubernetes autoscaling, where agents observe cluster state and select scaling actions to maximize long-term reward signals derived from service level objective (SLO) attainment and resource efficiency [16]. However, DRL approaches are typically data-hungry and require extensive simulation or offline training before safe deployment in production environments.

The impact of cold-start latency on autoscaling effectiveness has been examined in the context of serverless computing and containerized microservices [17]. Cold-start delays arise from the time required to provision a new container, pull the service image, and initialize the runtime environment. For SMS, an additional warm-up phase is necessary to load or reconstruct service state before the new replica can begin serving requests. Studies have measured cold-start overheads ranging from hundreds of milliseconds to several seconds depending on image size, state volume, and storage backend characteristics [18]. These overheads motivate research into pre-warming strategies, checkpoint-based state recovery, and incremental state transfer protocols that reduce the time between scaling trigger and replica readiness.

Traffic uncertainty in microservice environments arises from multiple sources, including user behavior variability, upstream service dependencies, and external event-driven demand spikes [19]. Anomalous traffic patterns—such as flash crowds, denial of service (DoS) events, and cascading failures—pose particular challenges for autoscaling systems that rely on historical patterns for prediction. Robust autoscaling designs must therefore incorporate mechanisms for detecting regime changes in traffic and switching between forecasting models or scaling policies accordingly [20]. Several studies have proposed ensemble forecasting approaches that combine multiple predictors to improve robustness against unpredictable workload shifts [21].

Resource heterogeneity in modern cloud infrastructure further complicates autoscaling for SMS. Cloud providers offer diverse instance types with varying CPU, memory, storage, and network characteristics, and cost-optimal resource selection requires autoscaling systems to consider not only the number of replicas but also the type of resources allocated to each [22]. Vertical scaling—adjusting the resource limits of individual pods—interacts with horizontal scaling in complex ways, particularly for stateful services where memory-resident state constrains the feasibility of downscaling [23]. The co-design of horizontal and vertical autoscaling policies for SMS remains an understudied area with significant practical relevance to production deployments.

Benchmark frameworks have played an important role in advancing autoscaling research by providing reproducible evaluation environments that capture the complexity of real microservice workloads. DeathStarBench and similar end-to-end microservice benchmarks have been widely adopted for evaluating autoscaling systems due to their realistic service

dependency graphs and workload generators [24]. More recent benchmark contributions have extended these environments to include stateful components such as distributed caches and message brokers, enabling more faithful assessment of stateful autoscaling strategies [25]. The literature also reflects growing interest in multi-objective autoscaling optimization, where competing concerns such as energy efficiency, cost minimization, latency compliance, and fault tolerance must be simultaneously addressed [26]. Pareto-optimal scaling policies derived from evolutionary algorithms and multi-objective RL formulations offer operators configurable trade-off frontiers, particularly relevant for SMS where the cost of scaling operations varies substantially across different system states.

## 3. Stateful Microservice Architecture and State Management

Understanding the architectural properties of stateful microservices is prerequisite to addressing the autoscaling problem. A stateful microservice (SMS) differs from a stateless service in that it maintains durable or session-bound state that influences the processing of future requests. This state may take the form of in-memory caches, persistent database records, active session tokens, queued messages, or stream processing checkpoints [27]. The lifecycle management of this state—including initialization, replication, migration, and garbage collection—introduces dependencies that do not exist in stateless deployments and that fundamentally constrain the autonomy with which individual replicas can be added or removed without coordination.

Modern container orchestration platforms manage stateful workloads through dedicated primitives. Kubernetes provides the StatefulSet workload type, which assigns stable network identities and persistent volume claims (PVCs) to each pod in the set [28]. Unlike Deployment objects, which treat replicas as interchangeable, StatefulSets ensure that pods are created, scaled, and deleted in a predictable ordered sequence. This ordering guarantees that state initialization and termination procedures can be executed safely, but it also introduces sequential bottlenecks that increase the latency of scale-out and scale-in operations. The tension between safe ordered scaling and rapid elastic response is a central challenge for autoscaling frameworks targeting stateful workloads.

State partitioning is a fundamental design pattern for achieving horizontal scalability in SMS. By dividing the services state space into non-overlapping partitions—commonly using consistent hashing or range-based schemes—each replica becomes responsible for a subset of the global state, and scaling operations require only the migration or reassignment of affected partitions [29]. Distributed key-value stores such as Apache Cassandra and Redis Cluster implement partition-aware scaling protocols that allow the number of storage nodes to be adjusted with bounded data movement. Applying similar techniques to application-layer stateful services requires developers to expose partition keys and implement state transfer callbacks, imposing non-trivial engineering overhead that limits the applicability of these patterns across service types.

State replication strategies govern the durability and availability of service state in the face of replica failures and scaling events. Primary-backup replication maintains one authoritative copy of each state partition on a primary replica and one or more read-only copies on backup replicas, enabling rapid failover but requiring synchronous or asynchronous replication of writes [30]. Consensus-based replication protocols such as Raft provide stronger consistency guarantees but introduce coordination overhead that limits throughput under high write concurrency. For autoscaling purposes, the choice of replication strategy directly influences the cost of adding and removing replicas, as new replicas must receive a full or incremental state snapshot before participating in request processing and this transfer time bounds the minimum achievable scaling latency.

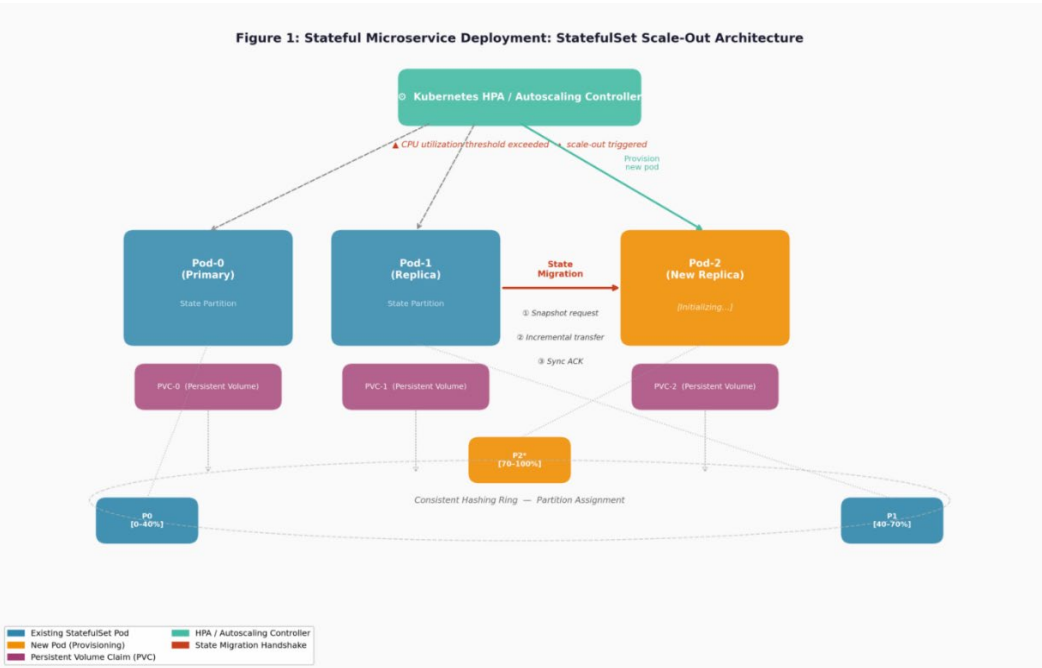


**Figure 1.** Stateful microservice deployment: statefulset scale-out architecture

The placement of stateful replicas across the cluster has important implications for both performance and fault tolerance. Anti-affinity rules that distribute replicas across availability zones protect against correlated failures but may

increase the latency of inter-replica communication required for state synchronization [31]. Topology-aware scheduling that co-locates communicating replicas on the same physical node or rack reduces synchronization latency but concentrates failure risk. Autoscaling systems for SMS must therefore incorporate placement constraints as a first-class decision variable, accounting for the interaction between replica placement, communication topology, and the cost of state transfer under the specific consistency model of each service. As illustrated in Figure 1, the complete lifecycle of a stateful replica—from provisioning through state transfer to active service—involves multiple coordinated handshakes between existing and newly instantiated pods that must complete before the new replica contributes to request handling capacity.

The emergence of disaggregated storage architectures, in which compute and storage are provisioned and scaled independently, offers new possibilities for stateful microservice management. In a disaggregated setting, the service state resides in a shared storage layer such as a distributed file system or object store, while compute replicas remain stateless with respect to persistent data [32]. This architectural choice reduces the coupling between autoscaling decisions and state migration, enabling compute-layer replicas to be added and removed without triggering data movement. However, it introduces a dependency on the latency and throughput characteristics of the shared storage layer, which may become a bottleneck under high concurrency and represents a potential single point of performance degradation for services with stringent tail latency requirements.

# 4. Autoscaling Policies and Decision Mechanisms

Autoscaling policies define the rules or learned behaviors that govern when and how the number of service replicas is adjusted in response to observed or predicted workload conditions. The design space for autoscaling policies is large, spanning reactive threshold-based rules, model predictive controllers, and learned decision agents [33]. Each class of policy embodies different assumptions about workload predictability, system observability, and the acceptable computational cost of scaling decisions, and the optimal choice depends on the statistical properties of the target workload and the operational constraints of the deployment environment.

Reactive autoscaling policies respond to current measurements of resource utilization or service performance that exceed predefined thresholds. The Kubernetes HPA, for example, adjusts the replica count based on observed CPU or memory utilization relative to target values, using a proportional control law to select the desired replica count [34]. While simple to configure and reason about, reactive policies suffer from inherent lag: by the time high utilization is observed and new replicas become operational, service latency may have already degraded substantially. For SMS, this lag is exacerbated by state initialization overhead, making reactive-only approaches particularly vulnerable to transient overload events that last only as long as or shorter than the combined detection and provisioning delay. Table 1 summarizes the principal autoscaling policy classes discussed in this section, comparing their trigger mechanisms, response characteristics, and suitability for stateful workloads.

**Table 1.** Comparison of autoscaling policy classes for stateful microservices

**Table 1: Comparison of Autoscaling Policy Classes for Stateful Microservices**

| Policy Class | Trigger Mechanism | Response Latency | Compute Overhead | Stateful Suitability | Representative Implementations | Key Limitations |
|---|---|---|---|---|---|---|
| **Reactive (Threshold-based)** | CPU / memory utilization exceeds threshold | High (detection + provisioning lag) | Very Low | Poor (migration lag amplified) | Kubernetes HPA AWS Auto Scaling Azure VMSS | Oscillation; blind to traffic spikes; ignores state init overhead |
| **Proactive (Forecast-driven)** | Predicted future demand from time-series model | Low (scale ahead of demand) | Medium (model inference) | Moderate (pre-warms state early) | Prophet + HPA SARIMA controller LSTM-scale | Forecast error on non-stationary loads; concept drift |
| **RL / DRL-based** | Learned reward from SLO + cost signals | Variable (policy-dependent) | High (training + inference) | Moderate (needs stateful reward shaping) | DeepScaling ADRL-K8s PPO-autoscaler | Data-hungry; unsafe exploration; poor interpretability |
| **Hybrid (Reactive + Proactive)** | Proactive baseline + reactive correction layer | Low baseline; fast correction | Medium | Good (decoupled layer design) | FIRM (OSDI 2020) Hybrid-MAPE MPC + threshold | Complexity; tuning two layers; forecast and rule conflicts |

Poor suitability / High latency
Moderate suitability
Good suitability / Low latency

Proactive autoscaling policies attempt to anticipate demand changes by forecasting future workload and triggering scale-out actions sufficiently in advance to ensure that new replicas are ready before demand peaks arrive. Time-series models including ARIMA, Holt-Winters exponential smoothing, and ML-based approaches have been employed for short-horizon workload prediction in cloud environments [35]. The accuracy of these forecasts directly determines the effectiveness of proactive scaling: optimistic forecasts lead to over-provisioning and unnecessary resource expenditure, while pessimistic forecasts result in under-provisioning and latency degradation. Incorporating confidence intervals and probabilistic uncertainty estimates into scaling decisions allows controllers to balance these risks in a principled way by provisioning according to a target percentile of the forecast distribution rather than the point estimate alone [36].

Hybrid autoscaling architectures combine reactive and proactive components to capture the strengths of both approaches. A common design uses a proactive model to set the baseline replica count based on predicted demand while a reactive layer monitors real-time performance signals and supplements with additional replicas when the forecast underestimates actual load [37]. This layered approach reduces the occurrence of both over-provisioning due to excessively conservative forecasts and under-provisioning due to forecast inaccuracy. For SMS, hybrid architectures can be further extended to decouple the proactive scaling of stateless request-handling components from the reactive scaling of stateful storage components, acknowledging that the two layers operate on different time scales and with

different overhead profiles.

RL has been proposed as a principled framework for autoscaling policy optimization in settings where the workload dynamics are complex and the cost structure is difficult to specify analytically. In the RL formulation, the autoscaling agent observes a state representation encoding cluster metrics, service performance indicators, and recent workload history, and selects scaling actions from a discrete or continuous action space [38]. The agent receives scalar reward signals encoding the operators preferences over performance, cost, and stability. Policy gradient algorithms and actor-critic methods have been applied to train autoscaling agents in simulated cloud environments, showing improvements over hand-crafted policies on benchmark workloads [39].

DRL extends RL with neural network function approximators that can represent complex, non-linear relationships between observed state and optimal action. DRL-based autoscaling agents have demonstrated the ability to learn nuanced policies that account for the non-stationarity of workload patterns and the delayed consequences of scaling decisions [40]. However, deploying DRL agents in production settings raises important concerns about safety, interpretability, and catastrophic forgetting when the environment distribution shifts. Several studies have proposed constrained DRL formulations that bound the probability of SLO violations during exploration, reducing the risk of unacceptable performance degradation [41].

Model predictive control (MPC) offers a middle ground between purely reactive approaches and fully learned policies. MPC optimizes a sequence of scaling actions over a finite prediction horizon by solving a constrained optimization problem at each decision epoch, using a learned or parameterized model of the system dynamics [42]. For SMS, MPC formulations can explicitly model the state migration cost and scaling lag, enabling the controller to sequence scale-out actions so that state initialization completes before the anticipated demand peak. The stability of autoscaling controllers is a critical practical concern—oscillatory behavior that causes the system to repeatedly scale up and down wastes resources and increases migration frequency for SMS [43]. Cooldown periods that suppress scaling actions following a recent scaling event are commonly used to prevent oscillation, though their optimal duration remains workload-dependent and requires empirical calibration.

# 5. 5. Traffic Prediction and Load Forecasting

Accurate traffic prediction is central to the effectiveness of proactive autoscaling for microservice environments. The challenge of load forecasting for microservices is multidimensional: individual services may receive aggregated traffic from dozens of upstream callers, exhibit diurnal and weekly periodicities overlaid with stochastic variability, and experience abrupt level shifts due to external events such as marketing campaigns or viral content [44]. Forecasting models must therefore capture both the regular periodic structure of workloads and the irregular components that arise from non-stationary processes while remaining computationally lightweight enough for online deployment alongside the services they govern.

Classical statistical time-series models provide interpretable baselines for workload forecasting. Seasonal autoregressive integrated moving average (SARIMA) models have been applied to cloud workload prediction with reasonable accuracy for workloads exhibiting stable seasonal patterns [45]. Prophet, a decomposable time-series forecasting model developed at Meta, has been adopted for cloud capacity planning due to its robustness to missing data and its ability to model multiple seasonalities [46]. These classical approaches are computationally efficient and can be retrained frequently to adapt to gradual workload shifts, but they struggle to capture complex non-linear dependencies between workload components and external covariates, motivating the adoption of DL-based alternatives.

DL architectures, particularly LSTM networks and their variants, have demonstrated strong performance on multivariate workload forecasting tasks in cloud environments [47]. LSTM networks learn to encode temporal context in hidden states that carry information across time steps, enabling the model to capture long-range dependencies that short-window autoregressive models miss. Attention-based transformer architectures have emerged as a competitive alternative to LSTM for time-series forecasting, offering parallelizable training and the ability to selectively attend to distant historical patterns [48]. Empirical evaluations on cloud workload traces have shown that transformer-based forecasters can match or exceed LSTM performance on short to medium prediction horizons while being more amenable to transfer learning across heterogeneous services.

Incorporating auxiliary features into workload forecasting models has been shown to improve prediction accuracy, particularly for services whose demand correlates with external signals. Calendar features such as hour-of-day, day-of-week, and holiday indicators capture the structured temporal patterns of human activity [49]. Application-level signals such as user session counts, API call rates, and upstream service throughput provide finer-grained leading indicators of imminent load changes. Multivariate forecasting models that jointly process these heterogeneous signals alongside raw request rate time series have achieved meaningful reductions in prediction error compared to univariate baselines on large-scale microservice deployments [50].

Transfer learning and meta-learning techniques address the challenge of forecasting accuracy for services with short or sparse request histories. Pre-training forecast models on high-traffic services and fine-tuning on low-traffic target services reduces the data requirements for achieving reliable predictions [51]. Few-shot forecasting approaches enable proactive autoscaling to be deployed rapidly for newly launched services that lack extensive historical data. These techniques are particularly relevant for microservice environments characterized by rapid service proliferation and frequent deployment of new service versions where historical data is sparse or non-existent for weeks following initial launch.

Uncertainty quantification in workload forecasting is essential for robust autoscaling policy design. Point forecasts that underrepresent prediction uncertainty can lead to autoscaling policies that are overconfident in the adequacy of a given replica count, resulting in performance degradation when actual demand exceeds the forecast [52]. Conformal prediction intervals, Bayesian neural networks, and Monte Carlo dropout methods have been applied to produce calibrated uncertainty estimates for workload forecasts.

Autoscaling controllers that consume these uncertainty estimates can implement risk-aware scaling policies that provision additional capacity when uncertainty is high, trading off resource efficiency for performance guarantee robustness in a principled statistical framework.

Anomaly detection and regime change identification complement workload forecasting by alerting autoscaling systems to the onset of abnormal traffic conditions that may invalidate historical-pattern-based forecasts. Statistical process control methods, isolation forests, and autoencoder-based anomaly detectors have been integrated into autoscaling pipelines to detect significant deviations from expected traffic behavior and initiate contingency scaling responses, while recent advances in log-based anomaly detection further demonstrate that modeling temporal dependencies and logical relationships across distributed system events can significantly improve detection accuracy for complex, multi-component anomalies [53]. The multi-tier dependency structure of microservice applications introduces additional complexity for workload forecasting at the individual service level. A change in the traffic pattern of a front-end service propagates through the call graph, creating correlated demand shifts at downstream services with varying delays and amplification factors [54]. Graph neural networks (GNNs) that model the microservice dependency graph have been proposed for joint forecasting of workloads across all services in a deployment, enabling coordinated proactive scaling that accounts for cross-service demand correlations extracted dynamically from distributed tracing systems. Figure 2 contrasts three representative traffic scenarios alongside their corresponding LSTM forecast outputs, demonstrating how prediction interval width expands during regime transitions and how autoscaling trigger thresholds must be calibrated relative to forecast uncertainty to maintain SLO compliance.

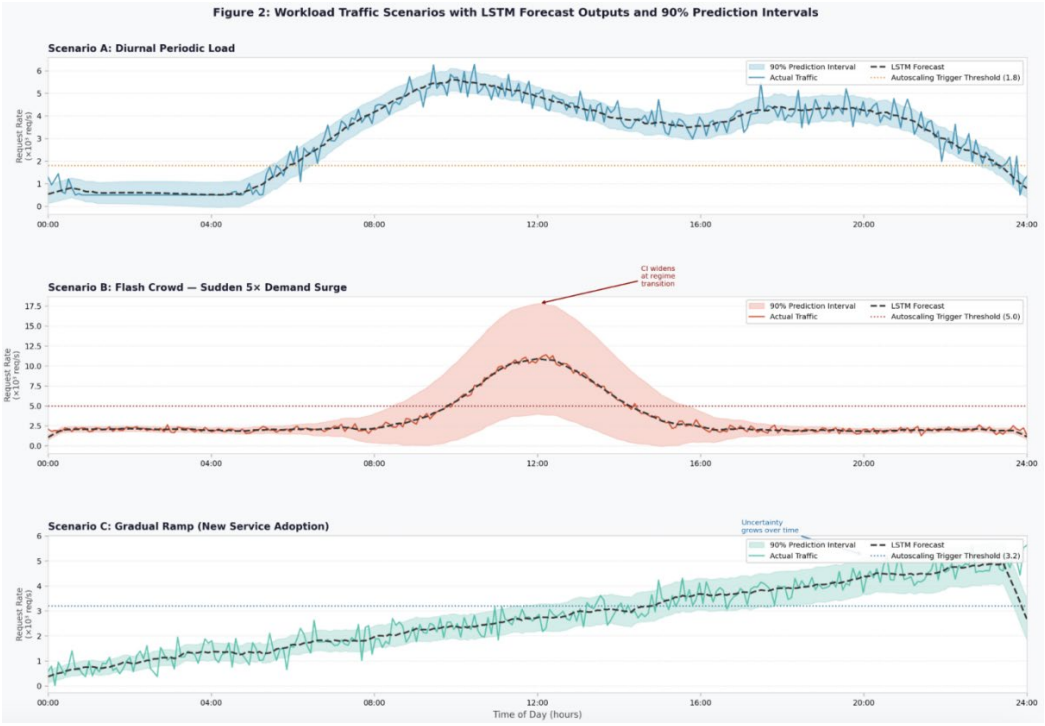


**Figure 2.** Workload traffic scenarios with LSTM forecast outputs and 90% prediction intervals

# 6. Challenges, Evaluation Frameworks, and Open Problems

The deployment of autoscaling systems for stateful microservices in production environments surfaces a set of engineering and research challenges that the preceding contributions have only partially addressed. Chief among these is the cold-start problem for SMS: when a new replica is instantiated, it must load or reconstruct sufficient state to serve requests before it can contribute to the services capacity [55]. For services with large state footprints or slow storage backends, the initialization period can span several seconds to minutes, effectively negating the benefit of a reactive scaling trigger. Incremental state transfer, warm replica pools, and checkpoint-based state recovery are promising mitigations, but their practical application requires close co-design of the autoscaling controller and the services state management layer, a co-design that existing platform abstractions do not yet facilitate cleanly.

Consistency requirements during scale-in operations present a complementary challenge. When a replica is decommissioned, in-flight requests must be completed, active sessions must be migrated, and state partitions must be transferred to surviving replicas without violating correctness invariants [56]. Implementing graceful termination protocols that satisfy these requirements under tight time bounds is non-trivial, particularly for services with complex transactional semantics or long-lived client connections. The interaction between autoscaling-driven scale-in events and the services concurrency control mechanisms has received limited systematic study, representing a significant gap in the current literature with direct consequences for the correctness of production deployments.

Network topology and data gravity effects constrain the autoscaling decisions available to orchestration systems operating at cloud scale. When stateful replicas must maintain strong consistency through consensus protocols, the placement of new replicas in remote availability zones may violate latency constraints imposed by the consensus algorithms quorum requirements [57]. Geo-distributed

autoscaling for SMS must therefore jointly optimize replica count, placement, and consistency protocol parameters to meet both performance and correctness objectives across geographically distributed deployments, a multi-dimensional optimization problem that has received limited formal treatment.

The evaluation of autoscaling systems presents its own set of methodological challenges. Laboratory benchmarks based on synthetic workload generators may fail to capture the statistical properties of production traffic, leading to inflated performance estimates that do not generalize to real deployments [58]. Several research groups have addressed this concern by releasing curated traces from production cloud workloads that can drive more realistic benchmark evaluations, though the representativeness of publicly released traces for specific application domains remains a concern [59]. Standardized evaluation protocols that specify common metrics, workload scenarios, and baseline comparators are needed to enable fair comparison across the growing body of autoscaling proposals that currently report results on incompatible experimental setups.

Benchmarking stateful autoscaling specifically requires evaluation environments that include representative stateful services with configurable state sizes, migration protocols, and consistency requirements. The adoption of widely used microservice benchmarks for this purpose is limited by the relatively small state footprints of standard benchmark applications [60]. Developing benchmark suites that stress the state management aspects of autoscaling—including large-volume state migration, concurrent read-write workloads during migration, and partition reassignment under heterogeneous request distributions—represents an important infrastructure contribution for the research community that would enable more rigorous empirical comparison across proposed approaches.

Open problems in the autoscaling of SMS extend beyond the technical issues discussed above to include economic and governance dimensions. Multi-tenant cloud environments require autoscaling systems to operate within budget constraints and to allocate finite cluster resources fairly across competing workloads [61]. Incentive-compatible autoscaling policies that prevent individual services from gaming allocation mechanisms remain an open area of inquiry with implications for both cloud providers and enterprise operators managing shared Kubernetes clusters at scale.

The integration of autoscaling with continuous delivery pipelines introduces additional complexity for stateful services. Canary deployments and blue-green deployment strategies that are straightforward for stateless services require careful orchestration for SMS to avoid data corruption or inconsistency when multiple service versions access the same state store concurrently. Autoscaling systems must be version-aware and capable of coordinating scaling actions across old and new service versions during rollout, ensuring that the migration of both compute replicas and state partitions proceeds in a controlled manner that respects consistency invariants throughout the deployment transition window.

Finally, the sustainability implications of autoscaling decisions are attracting increasing research attention as cloud providers commit to carbon neutrality goals. Carbon-aware autoscaling that modulates resource provisioning in response to real-time carbon intensity signals from the electrical grid has been proposed as a mechanism for reducing the environmental footprint of cloud workloads without sacrificing performance. For SMS, carbon-aware autoscaling must account for the energy cost of state migration operations as well as steady-state compute consumption, making the optimization problem more complex than for stateless workloads. Developing principled frameworks that jointly optimize for performance, cost, and carbon emissions in the context of stateful microservice autoscaling represents a compelling direction for future research at the intersection of systems engineering and environmental sustainability.

# 7. Conclusion

This paper has reviewed the current state of research on autoscaling stateful microservices under variable load and traffic uncertainty, tracing the evolution of the field from early threshold-based reactive controllers to sophisticated ML-driven and DRL-based policy frameworks. The review has highlighted that while substantial progress has been made in autoscaling for stateless cloud services, the stateful regime introduces fundamentally new challenges related to state migration overhead, consistency preservation during scaling transitions, and cold-start latency that require dedicated solutions rather than direct adaptations of existing approaches.

The synthesis of the literature reveals several recurring themes. First, the coupling between compute-layer autoscaling and state management operations is the defining characteristic of SMS autoscaling, and frameworks that treat these as separable concerns tend to underperform those that address them jointly. Second, proactive and hybrid autoscaling approaches consistently outperform purely reactive methods for workloads with predictable structure, but their advantage erodes under highly non-stationary traffic conditions, motivating robust and adaptive forecasting architectures that can detect regime changes and adjust prediction strategies accordingly. Third, DRL-based autoscaling agents show significant promise but face practical barriers around safety during exploration, interpretability, and adaptation to distributional shift that must be addressed before widespread production deployment can be realized without risk to operational correctness.

Important open challenges remain at the intersection of distributed systems consistency, cloud economics, and ML forecasting. The development of benchmark environments that faithfully capture the state management complexities of real-world SMS is a prerequisite for rigorous empirical comparison of autoscaling approaches. Standardized evaluation metrics that account for not only tail latency and resource efficiency but also the cost of state migration and the frequency of consistency violations are needed to enable the field to make measurable and reproducible progress across research groups. The integration of autoscaling with multi-version deployment strategies, geo-distributed placement optimization, and carbon-aware resource scheduling represents a rich agenda for future investigation that will require collaboration between the systems, ML, and sustainability research communities.

As microservice deployments continue to grow in scale and complexity, and as the proportion of stateful components in cloud-native applications increases, the ability to autoscale SMS reliably and efficiently will become an increasingly critical operational capability. The research community is well-positioned to address this challenge by leveraging advances in ML, distributed systems theory, and cloud platform engineering while developing tighter integrations

between autoscaling controllers and the state management substrates of the services they govern.

# References


[1] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77-97.

[2] Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745-9760.

[3] Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., ... & Wilkes, J. (2020, April). Autopilot: workload autoscaling at google. In proceedings of the fifteenth european conference on computer systems (pp. 1-16).

[4] Harve, B. M., Bidkar, D. M., Krishnappa, M. S., Pandy, G., Jayaram, V., Veerapaneni, P. K., & Mehta, G. (2024, December). The cloud-native revolution: Microservices in a cloud-driven world. In 2024 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA) (pp. 1043-1048). IEEE.

[5] Camilli, M., & Russo, B. (2022). Modeling Performance of Microservices Systems with Growth Theory: Modeling Performance of Microservices Systems with Growth Theory. Empirical Software Engineering, 27(2), 39.

[6] Hofmann, M. (2019). Developing a streaming-based architecture for demand prediction of taxi trips in the presence of concept drift.

[7] Sabuhi, M., Mahmoudi, N., & Khazaei, H. (2021). Optimizing the performance of containerized cloud software systems using adaptive PID controllers. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 15(3), 1-27.

[8] Balla, D., Simon, C., & Maliosz, M. (2020, April). Adaptive scaling of Kubernetes pods. In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium (pp. 1-5). IEEE.

[9] Yadav, M. P., Pal, N., & Yadav, D. K. (2021, January). Workload prediction over cloud server using time series data. In 2021 11th international conference on cloud computing, data science & engineering (confluence) (pp. 267-272). IEEE.

[10] Karim, M. E., Maswood, M. M. S., Das, S., & Alharbi, A. G. (2021). BHyPreC: a novel Bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine. IEEE Access, 9, 131476-131495.

[11] Xing, S., & Wang, Y. (2025). Cross-Modal Attention Networks for Multi-Modal Anomaly Detection in System Software. IEEE Open Journal of the Computer Society.

[12] Rossi, F., Nardelli, M., & Cardellini, V. (2019, July). Horizontal and vertical scaling of container-based applications using reinforcement learning. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 329-338). IEEE.

[13] Singh, P., Gupta, P., Jyoti, K., & Nayyar, A. (2019). Research on auto-scaling of web applications in cloud: survey, trends and future directions. Scalable Computing: Practice and Experience, 20(2), 399-432.

[14] Russo Russo, G., Cardellini, V., & Lo Presti, F. (2023). Hierarchical auto-scaling policies for data stream processing on heterogeneous resources. ACM transactions on autonomous and adaptive systems, 18(4), 1-44.

[15] Ma, X., Zong, K., & Rezaeipanah, A. (2024). Auto-scaling and computation offloading in edge/cloud computing: a fuzzy Q-learning-based approach. Wireless Networks, 30(2), 637-648.

[16] Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. (2020, May). Adaptive AI-based auto-scaling for Kubernetes. In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID) (pp. 599-608). IEEE.

[17] Varshney, R. P., & Sharma, D. K. (2020, October). Cold start in function as a service: a systematic study, analysis and evaluation. In International Conference on Futuristic Trends in Networks and Computing Technologies (pp. 337-349). Singapore: Springer Singapore.

[18] Mahmoudi, N., & Khazaei, H. (2020). Performance modeling of serverless computing platforms. IEEE Transactions on Cloud Computing, 10(4), 2834-2847.

[19] Soldani, J., Forti, S., Roveroni, L., & Brogi, A. (2025). Explaining Microservices' Cascading Failures From Their Logs. Software: Practice and Experience, 55(5), 809-828.

[20] Lima, M., Neto, M., Silva Filho, T., & Fagundes, R. A. D. A. (2022). Learning under concept drift for regression—a systematic literature review. IEEE Access, 10, 45410-45429.

[21] Bawa, J., Kaur Chahal, K., & Kaur, K. (2025). Improving cloud resource management: an ensemble learning approach for workload prediction: J. Bawa et al. The Journal of Supercomputing, 81(10), 1138.

[22] Fu, K., Zhang, W., Chen, Q., Zeng, D., & Guo, M. (2021). Adaptive resource efficient microservice deployment in cloud-edge continuum. IEEE Transactions on Parallel and Distributed Systems, 33(8), 1825-1840.

[23] Baarzi, A. F., & Kesidis, G. (2021, November). Showar: Right-sizing and efficient scheduling of microservices. In Proceedings of the ACM Symposium on Cloud Computing (pp. 427-441).

[24] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., ... & Delimitrou, C. (2019, April). An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems (pp. 3-18).

[25] Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., & Liu, X. (2022). Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering, 27(1), 25.

[26] Horn, A., Fard, H. M., & Wolf, F. (2022, August). Multi-objective hybrid autoscaling of microservices in kubernetes clusters. In European Conference on Parallel Processing (pp. 233-250). Cham: Springer International Publishing.

[27] Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Challenges and solution directions of microservice architectures: A systematic literature review. Applied sciences, 12(11), 5507.

[28] Aqasizade, H., Ataie, E., & Bastam, M. (2025). Kubernetes in action: Exploring the performance of kubernetes distributions in the cloud. Software: Practice and Experience, 55(10), 1711-1725.

[29] Ziegler, T., Bernstein, P. A., Leis, V., & Binnig, C. (2023). Is scalable OLTP in the cloud a solved problem?. In CIDR.

[30] Wen, L., Xu, M., Gill, S. S., Hilman, M., Srirama, S. N., Ye, K., & Xu, C. (2025). Statuscale: Status-aware and elastic scaling strategy for microservice applications. ACM Transactions on Autonomous and Adaptive Systems, 20(1), 1-25.

[31] Chen, Y., Wu, C., Zhang, F., Lu, C., Huang, Y., & Lu, H. (2025). Topology-aware microservice architecture in edge networks: Deployment optimization and implementation. IEEE Transactions on Mobile Computing.

[32] Ghandeharizadeh, S., Bernstein, P. A., Borthakur, D., Huang, H., Menon, J., & Puri, S. (2022, September). Disaggregated database management systems. In Technology Conference on Performance Evaluation and Benchmarking (pp. 33-48). Cham: Springer Nature Switzerland.

[33] Alharthi, S., Alshamsi, A., Alseiari, A., & Alwarafy, A. (2024). Auto-scaling techniques in cloud computing: Issues and research directions. Sensors, 24(17), 5551.

[34] Nguyen, T. T., Yeom, Y. J., Kim, T., Park, D. H., & Kim, S. (2020). Horizontal pod autoscaling in kubernetes for elastic container orchestration. Sensors, 20(16), 4621.

[35] Luo, S., Xu, H., Ye, K., Xu, G., Zhang, L., Yang, G., & Xu, C. (2022, November). The power of prediction: microservice auto scaling via workload learning. In Proceedings of the 13th symposium on cloud computing (pp. 355-369).

[36] Qiu, H., Banerjee, S. S., Jha, S., Kalbarczyk, Z. T., & Iyer, R. K. (2020). {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In 14th USENIX symposium on operating systems design and implementation (OSDI 20) (pp. 805-825).

[37] Gupta, S., Islam, M. T., & Buyya, R. (2025). A Hybrid Reactive-Proactive Auto-scaling Algorithm for SLA-Constrained Edge Computing. arXiv preprint arXiv:2512.14290.

[38] Zou, Y., Qi, N., Deng, Y., Xue, Z., Gong, M., & Zhang, W. (2025, July). Autonomous resource management in microservice systems via reinforcement learning. In 2025 8th International Conference on Computer Information Science and Application Technology (CISAT) (pp. 991-995). IEEE.

[39] Wang, Z., Zhu, S., Li, J., Jiang, W., Ramakrishnan, K. K., Zheng, Y., ... & Liu, A. X. (2022, November). Deepscaling: microservices autoscaling for stable cpu utilization in large scale cloud systems. In Proceedings of the 13th symposium on cloud computing (pp. 16-30).

[40] Zhou, Y. (2025). Adaptive Resource Scheduling for IoT Big Data Stream Processing Based on Deep Reinforcement Learning. Computers and Artificial Intelligence, 2(2), 16-28.

[41] Mayerhofer, R. (2023). Reinforcement-learning-based, application-agnostic, and explainable auto-scaling in the cloud utilizing high-level SLOs (Doctoral dissertation, Technische Universität Wien).

[42] Joshi, N. S., Raghuwanshi, R., Agarwal, Y. M., Annappa, B., & Sachin, D. N. (2024). ARIMA-PID: container auto scaling based on predictive analysis and control theory. Multimedia Tools and Applications, 83(9), 26369-26386.

[43] Luo, S., Xu, H., Lu, C., Ye, K., Xu, G., Zhang, L., ... & Xu, C. (2021, November). Characterizing microservice dependency and performance: Alibaba trace analysis. In Proceedings of the ACM symposium on cloud computing (pp. 412-426).

[44] Ogundipe, A., Okunlola, O., & Alao, O. (2024). Adaptive Load Balancing and Auto Scaling Algorithms for Resource Optimization in Distributed Microservices based Cloud Applications. International Journal of Science Architecture Technology and Environment, 101-111.

[45] Bhat, C. R., Prabha, B., Donald, C., Sah, S., & Patil, H. (2023, December). SARIMA Techniques for Predictive Resource Provisioning in Cloud Environments. In 2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS) (pp. 1-5). IEEE.

[46] Bankole, F. A., & Tewogbade, L. (2024). Optimizing subscription cost structures in technology enterprises using scalable, data-informed forecasting techniques. International Journal of Scientific Engineering Research and Science Education and Technology, 11(6), 359-392.

[47] Nwachukwu, C. Real-Time Workload Prediction and Dynamic Autoscaling: A New Paradigm for Cloud Performance Management.

[48] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021, May). Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 12, pp. 11106-11115).

[49] Tamiru, M. A. (2021). Automatic resource management in geo-distributed multi-cluster environments (Doctoral dissertation, Université de Rennes).

[50] Kumar, B., Verma, A., & Verma, P. (2025). A multivariate transformer-based monitor-analyze-plan-execute (MAPE) autoscaling framework for dynamic resource allocation in cloud environment. Computing, 107(3), 69.

[51] Sun, Y., Keung, J. W., Yu, H. K., & Luo, W. (2026). LogMeta: A Few-Shot Model-Agnostic Meta-Learning Framework for Robust and Adaptive Log Anomaly Detection. Journal of Systems and Software, 112781.

[52] Hang, H., Tang, X., Sun, J., Bao, L., Lo, D., & Wang, H. (2024, May). Robust auto-scaling with probabilistic workload forecasting for cloud databases. In 2024 IEEE 40th International Conference on Data Engineering (ICDE) (pp. 4016-4029). IEEE.

[53] Liu, Y., Ren, S., Wang, X., & Zhou, M. (2024). Temporal logical attention network for log-based anomaly detection in distributed systems. Sensors, 24(24), 7949.

[54] Fu, N., Cheng, G., Teng, Y., Dai, G., Yu, S., & Chen, Z. (2025). Intelligent root cause localization in microservice systems: A survey and new perspectives. ACM Computing Surveys, 57(12), 1-37.

[55] Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gill, S. S., & Uhlig, S. (2024). Cold start latency in serverless computing: A systematic review, taxonomy, and future directions. ACM Computing Surveys, 57(3), 1-36.

[56] Ahsan, S. B. (2020). New consistency orchestrators for emerging distributed systems (Doctoral dissertation, University of Illinois at Urbana-Champaign).

[57] Xu, M., Wen, L., Liao, J., Wu, H., Ye, K., & Xu, C. (2025). Auto-scaling Approaches for Cloud-native Applications: A Survey and Taxonomy. arXiv preprint arXiv:2507.17128.

[58] Laaber, C., Scheuner, J., & Leitner, P. (2019). Software microbenchmarking in the cloud. how bad is it really?. Empirical Software Engineering, 24(4), 2469-2508.

[59] Guo, J., Chang, Z., Wang, S., Ding, H., Feng, Y., Mao, L., & Bao, Y. (2019, June). Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In Proceedings of the international symposium on quality of service (pp. 1-10).

[60] Dashtbani, M., & Tahvildari, L. (2025, November). Key Considerations for Auto-Scaling: Lessons from Benchmark Microservices. In 2025 IEEE International Conference on Collaborative Advances in Software and COmputiNg (CASCON) (pp. 118-123). IEEE.

[61] Pi, A., Zhao, J., Wang, S., & Zhou, X. (2021, December). Memory at your service: Fast memory allocation for latency-critical services. In Proceedings of the 22nd International Middleware Conference (pp. 185-197).