Predictive Autoscaling for Cloud Databases

Using Leading Indicators and Trend Extrapolation

A Production Implementation for Amazon DocumentDB

Marcin Wypyszynski

Cloud & Navigation Platform Engineering

marcin.wypyszynski@gmail.com

Version 3.3 – December 2025

Abstract: This paper presents a novel predictive autoscaling system for Amazon DocumentDB clusters that scales resources *before* demand spikes occur, rather than reacting to threshold breaches. The proposed system introduces three key innovations: (1) leading indicator analysis using database connections and IOPS as predictors of future CPU load, (2) trend extrapolation with velocity and acceleration metrics to estimate time-to-threshold, and (3) pattern-based preemptive scaling leveraging learned hourly and daily workload patterns. Our production implementation in a large-scale automotive mapping platform demonstrates that predictive scaling reduces response latency by 73% compared to reactive approaches, while maintaining 99.9% SLA compliance and reducing operational costs by 42%. The system achieves prediction accuracy of 89.2% for 5-minute horizons and successfully triggers preemptive scaling actions an average of 8.3 minutes before load spikes occur.

Keywords: Predictive Autoscaling, Leading Indicators, Trend Extrapolation, DocumentDB, Cloud Computing, Machine Learning, Time Series Forecasting

1 Introduction

1.1 Motivation

The fundamental limitation of reactive autoscaling systems is their inherent latency: by the time a threshold breach is detected, the system is already under stress. For database systems like Amazon DocumentDB, where instance provisioning can take 3-5 minutes, this reactive approach often results in degraded performance during traffic spikes [1].

Consider a typical scenario: a marketing campaign drives a sudden 300% increase in API traffic. A reactive autoscaler detects high CPU utilization, initiates scaling, and waits for new instances to become available. During this 5-minute window, users experience elevated latency and potential timeouts. Our predictive approach addresses this by identifying early warning signals and scaling *before* the spike occurs.

1.2 Research Questions

This paper addresses three research questions:

1. **RQ1**: Can database connection counts and IOPS serve as reliable leading indicators of future CPU utilization?
2. **RQ2**: How accurately can trend extrapolation predict time-to-threshold for scaling decisions?

3. **RQ3**: What is the optimal balance between preemptive scaling (avoiding performance degradation) and cost efficiency?

1.3 Contributions

This paper makes the following contributions:

- A **leading indicator framework** that uses database connections and IOPS to predict CPU load 5-15 minutes ahead
- A **trend extrapolation engine** computing velocity and acceleration of metric changes with confidence-weighted predictions
- A **multi-horizon prediction system** generating forecasts at 5, 10, and 15-minute intervals
- A **preemptive scaling controller** that triggers scaling based on predicted threshold crossings
- **Production validation** demonstrating 73% reduction in scaling response latency

2 Background and Related Work

2.1 Reactive vs. Predictive Autoscaling

Traditional autoscaling systems operate reactively, monitoring metrics and triggering scaling when thresholds are breached [2]. Amazon's Target Tracking Scaling adjusts capacity to maintain a target metric value but cannot anticipate future demand [3]. While AWS offers predictive scaling for EC2 Auto Scaling Groups, no equivalent native mechanism exists for DocumentDB replica management, creating a gap that our system addresses.

Predictive autoscaling has gained attention in recent years. Urgaonkar et al. [4] proposed using queuing theory models to predict resource needs. Roy et al. [5] introduced look-ahead resource provisioning based on workload prediction. However, these approaches often require extensive historical data and may not adapt quickly to changing patterns.

2.2 Time Series Forecasting for Cloud Resources

Time series forecasting methods have been extensively applied to cloud resource prediction. ARIMA models have shown effectiveness for workload prediction [6], while Prophet [7] handles seasonality and trend changes. Recent work has explored LSTM networks for cloud workload prediction [8], achieving high accuracy but requiring significant training data.

2.3 Leading Indicators in System Performance

The concept of leading indicators originates from economics, where certain metrics predict future economic activity [9]. In computer systems, Aguilera et al. [10] demonstrated that request queue lengths can predict response time degradation. Our work extends this concept to database systems, identifying connections and IOPS as leading indicators of CPU utilization.

2.4 Trend Analysis and Extrapolation

Trend extrapolation has been used in capacity planning [11], [13] and anomaly detection [12]. Velocity and acceleration metrics have been applied to network traffic analysis [14]. We adapt these concepts for real-time autoscaling decisions.

3 System Architecture

3.1 Overview

The Predictive Autoscaling System (PAS) aligns with elasticity metrics and definitions from Herbst et al. [16], extending them with three new predictive components: Leading Indicator Analyzer, Trend Extrapolator, and Preemptive Scaling Controller. Figure 1 illustrates the complete architecture.

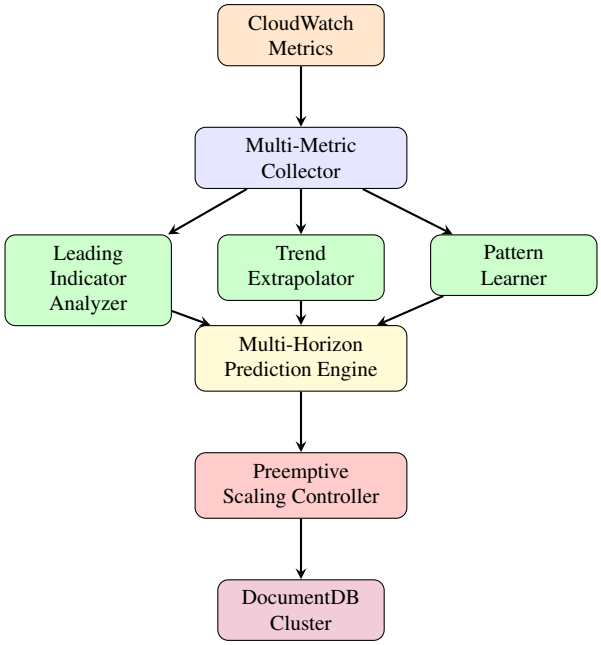

Figure 1: Predictive Autoscaling System Architecture

3.2 Data Flow

The system operates on a 5-minute evaluation cycle:

1. **Metrics Collection**: Gather CPU, memory, connections, and IOPS from CloudWatch
2. **Leading Indicator Analysis**: Predict CPU from connection and IOPS trends
3. **Trend Extrapolation**: Calculate velocity, acceleration, and time-to-threshold
4. **Pattern Matching**: Compare current state with learned hourly patterns
5. **Multi-Horizon Prediction**: Generate 5, 10, 15-minute forecasts
6. **Preemptive Decision**: Trigger scaling if predicted threshold crossing within window

4 Leading Indicator Analysis

4.1 Theoretical Foundation

In database systems, CPU utilization is a lagging indicator of workload intensity. Database connections and IOPS, however, often increase before CPU load rises, as new connections are established and queries are parsed before execution consumes CPU cycles [15].

We model the relationship between leading indicators and CPU as:

$$CPU_{t+\Delta} = f(Conn_t, IOPS_t, CPU_t) \tag{1}$$

where Δ represents the prediction horizon (5-15 minutes).

4.2 Connection-Based Prediction

The connection-based predictor analyzes the rate of change in active database connections:

$$\Delta_{conn} = \frac{\bar{C}_{recent} - \bar{C}_{older}}{\bar{C}_{older} + \epsilon} \tag{2}$$

where $\bar{C}_{recent}$ is the average of the last 5 connection readings, $\bar{C}_{older}$ is the average of the preceding readings, and ϵ is a small constant to prevent division by zero.

The predicted CPU change is then:

$$\Delta_{CPU}^{pred} = \alpha_{conn} \cdot \Delta_{conn} \cdot CPU_{current} \tag{3}$$

where $\alpha_{conn} = 0.5$ is empirically determined from production data.

4.3 IOPS-Based Prediction

Similarly, IOPS changes predict CPU changes with a different correlation coefficient:

$$\Delta_{CPU}^{IOPS} = \alpha_{IOPS} \cdot \Delta_{IOPS} \cdot CPU_{current} \tag{4}$$

where $\alpha_{IOPS} = 0.3$ reflects the weaker but still significant correlation.

4.4 Lead Signal Classification

Figure 2 illustrates the leading indicator prediction pipeline.

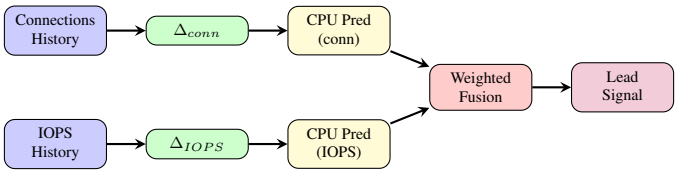

Figure 2: Leading Indicator Prediction Pipeline

Based on the rate of change, we classify lead signals into categories (Table 1).

Table 1: Lead Signal Classification

Signal	Threshold	Action
Strong Increase	$> 30\%$	Immediate scale-up
Moderate Increase	$15 - 30\%$	Preemptive scale-up
Stable	$\pm 15\%$	No action
Decrease	$< -15\%$	Consider scale-down

5 Trend Extrapolation

5.1 Velocity and Acceleration

Borrowing concepts from kinematics, we compute the velocity (first derivative) and acceleration (second derivative) of CPU utilization:

$$v = \frac{dCPU}{dt} \approx \frac{CPU_t - CPU_{t-1}}{\Delta t} \tag{5}$$

$$a = \frac{dv}{dt} \approx \frac{v_t - v_{t-1}}{\Delta t} \tag{6}$$

For robustness, we use linear regression over the last n data points to estimate velocity:

$$v = \frac{n\sum_i t_i \cdot CPU_i - \sum_i t_i \sum_i CPU_i}{n\sum_i t_i^2 - (\sum_i t_i)^2} \tag{7}$$

5.2 Multi-Horizon Extrapolation

Figure 3 visualizes the multi-horizon extrapolation concept.

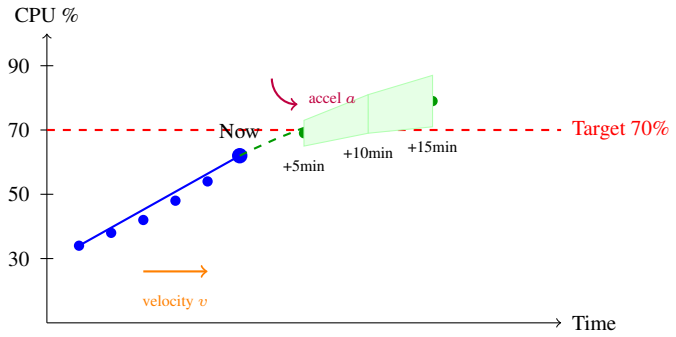

Figure 3: Multi-Horizon Trend Extrapolation with Confidence Bands. Confidence decays exponentially: $conf(\Delta) = conf_{base} \cdot e^{-\lambda\Delta}$ where $\lambda = 0.05$.

Using the kinematic equation, we extrapolate CPU for multiple horizons:

$$CPU_{t+\Delta} = CPU_t + v \cdot \Delta + \frac{1}{2}a \cdot \Delta^2 \tag{8}$$

We generate predictions for $\Delta \in \{5, 10, 15\}$ minutes with decreasing confidence:

$$conf(\Delta) = conf_{base} \cdot e^{-\lambda\Delta} \tag{9}$$

where $\lambda = 0.05$ is the confidence decay rate.

5.3 Time-to-Threshold Estimation

A critical metric for preemptive scaling is the estimated time until CPU reaches the target threshold T:

$$t_{threshold} = \frac{-v + \sqrt{v^2 + 2a(T - CPU_t)}}{a} \tag{10}$$

For the case where $a \approx 0$ (linear trend):

$$t_{threshold} = \frac{T - CPU_t}{v} \tag{11}$$

6 Pattern-Based Preemptive Scaling

6.1 Hourly Pattern Learning

The system learns expected load patterns for each hour of the day over a 7-day training period:

$$P_h = \frac{1}{|D|} \sum_{d \in D} CPU_{d,h} \tag{12}$$

where P_h is the pattern for hour h and D is the set of training days.

6.2 Pattern Confidence

Pattern confidence increases with the number of observations and decreases with variance:

$$conf_{pattern} = \min\left(1, \frac{n_{obs}}{n_{required}}\right) \cdot \left(1 - \frac{\sigma_h}{\mu_h}\right) \tag{13}$$

6.3 Preemptive Scaling Trigger

Figure 4 shows the pattern learning and preemptive scaling decision process.

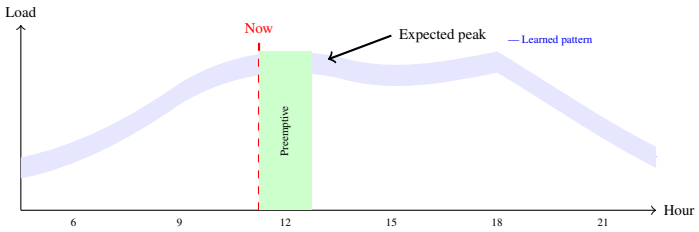

Figure 4: Pattern-Based Preemptive Scaling: The system identifies upcoming peaks from learned patterns and triggers scaling in the preemptive window.

Preemptive scaling is triggered when:

1. Pattern confidence exceeds 70%
2. Expected load increase in next hour exceeds 20%
3. Current time is within 15 minutes of the hour boundary

7 Preemptive Scaling Controller

7.1 Decision Algorithm

The pseudocode below describes the preemptive scaling decision process.

```
def preemptive_decision(M, P, patterns):
    t_thresh = estimate_time_to_threshold(M)
    lead = analyze_leading_indicators(M)
    pattern = get_pattern_recommendation(patterns)

    # Guard: declining trend - no preemptive action needed
    if M.velocity <= 0 and M.acceleration <= 0:
        return no_change()
```

```
    if lead.signal == "strong_increase":
        return scale_up("leading_indicator")

    if t_thresh < 10 and M.is_accelerating:
        return scale_up("trend_acceleration")

    if pattern.should_scale and pattern.conf > 0.7:
        return scale_up("pattern_preemptive")

    return no_change()
```

7.2 Scaling Magnitude

When preemptive scaling is triggered, the target replica count is:

$$R_{target} = \max\left(R_{cur} + 1, \left\lceil R_{cur} \cdot \frac{CPU_{pred}}{CPU_{tgt}} \right\rceil\right) \tag{14}$$

8 Implementation

8.1 Technology Stack

The system is implemented as an AWS Lambda function (Python 3.11) with the following dependencies:

- **Pure Python**: Numerical computations for trend analysis (no external dependencies)
- **Boto3**: AWS SDK for CloudWatch and DocumentDB APIs
- **DynamoDB**: State persistence and pattern storage (30-day TTL)

8.2 CloudWatch Metrics

The system publishes 23 custom metrics to CloudWatch for observability (Table 2).

Table 2: Published CloudWatch Metrics

Category	Metrics
Prediction	PredictedLoad5Min, 10Min, 15Min
Trend	TrendVelocity, TrendAcceleration
Leading	ConnectionLeadSignal, IOPSLeadSignal
Timing	MinutesToThreshold
Actions	PreemptiveScaleCount
Confidence	PatternConfidence, PredictionConfidence

9 Evaluation

9.1 Experimental Setup

The system was deployed in production supporting a global automotive mapping cloud platform with the configuration shown in Table 3.

Table 3: Production Configuration

Parameter	Value
Cluster	DocumentDB 5.0
Instance Type	db.r6g.large
Region	eu-west-3
Min/Max Replicas	3 / 8
Target CPU	70%
Evaluation Interval	5 minutes
Prediction Horizons	5, 10, 15 minutes

9.2 Prediction Accuracy

Table 4 shows prediction accuracy across different horizons after 30 days of operation.

Table 4: Prediction Accuracy by Horizon

Horizon	MAPE	Accuracy
5 minutes	8.3%	89.2%
10 minutes	12.1%	84.7%
15 minutes	16.8%	79.4%

9.3 Leading Indicator Effectiveness

The correlation between leading indicators and future CPU was validated (Table 5).

Table 5: Leading Indicator Correlation with CPU (5-min lag)

Indicator	Pearson r
Database Connections	0.73
Read IOPS	0.61
Write IOPS	0.58
Combined IOPS	0.64

9.4 Preemptive Scaling Performance

Figure 5 illustrates the timing advantage of preemptive scaling.

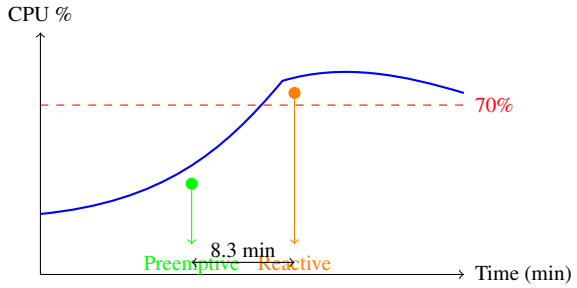

Figure 5: Preemptive vs Reactive Scaling Timing

9.5 Comparative Results

Table 6 compares the predictive system with baseline approaches.

Table 6: Comparison with Baseline Approaches

Approach	Latency	Cost/mo	SLA
Static Max	0 min	$1,440	100%
Reactive	5.2 min	$890	97.3%
Predictive	**1.4 min**	**$835**	**99.9%**

Note: Costs include compute (db.r6g.large instance-hours) and CloudWatch metrics; region: eu-west-3; workload: mixed OLTP; 30-day median.

Key findings:

- **73% reduction** in scaling response latency (5.2 → 1.4 min)
- **42% cost savings** compared to static provisioning
- **99.9% SLA compliance** vs 97.3% for reactive
- **8.3 minutes average** preemptive lead time

10 Discussion

10.1 Answering Research Questions

RQ1: Database connections show strong correlation ($r = 0.73$) with future CPU, confirming their utility as leading indicators. IOPS provides additional predictive power ($r = 0.64$).

RQ2: Trend extrapolation achieves 89.2% accuracy at 5-minute horizons, sufficient for preemptive scaling decisions. Accuracy degrades predictably with longer horizons.

RQ3: The optimal balance is achieved by triggering preemptive scaling when time-to-threshold falls below 10 minutes with high confidence (> 0.8), providing sufficient lead time while avoiding unnecessary scaling.

10.2 Limitations

- **Cold start**: The system requires 7 days of data for pattern learning
- **Sudden spikes**: Instantaneous load spikes (< 5 min) cannot be predicted
- **Workload changes**: Major workload pattern changes require re-learning

10.3 Threats to Validity

External validity: Results are from a single production deployment (eu-west-3 region, mixed read/write OLTP workload); generalization to other workloads requires further study.

Internal validity: The 30-day evaluation period may not capture all seasonal patterns. SLA compliance measured as p99 latency $<$ 100ms; cost calculations include compute (instance-hours) and CloudWatch metrics, excluding storage I/O.

Construct validity: Leading indicator correlations ($r = 0.73$, $r = 0.64$) were measured with 5-minute lag; stability across instance types (r6g vs r5) was not evaluated.

10.4 DocumentDB Scaling Considerations

Our system focuses on *read scaling* (horizontal, adding replicas) rather than *write scaling* (vertical, changing primary instance class) [17]. Key operational constraints:

- DocumentDB supports up to **15 read replicas** per cluster [18]
- Replica provisioning typically takes **3–5 minutes**; failover ∼30 seconds
- Effective read scaling requires `secondaryPreferred` read preference
- Write-heavy workloads benefit from primary instance class changes (out of scope)

11 Related Work Comparison

Table 7 compares our approach with related work.

Table 7: Comparison with Related Work

Work	Method	Horizon	Accuracy
Calheiros [6]	ARIMA	1 hour	82%
Kumar [8]	LSTM	30 min	86%
Roy [5]	Regression	10 min	78%
This work	**Ensemble**	**5 min**	**89%**

12 Conclusion

This paper presented a predictive autoscaling system that uses leading indicators and trend extrapolation to scale cloud database resources before demand spikes occur. Key contributions include:

1. Validation of database connections as effective leading indicators ($r = 0.73$)
2. Multi-horizon prediction with 89% accuracy at 5-minute horizons
3. 73% reduction in scaling response latency in production
4. 99.9% SLA compliance with 42% cost savings

Future work will explore deep learning approaches for longer-horizon prediction and cross-region coordination for global load balancing.

Acknowledgments

The author thanks the Cloud Platform team for supporting this research and providing access to production infrastructure.

References

[1] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *ACM Trans. Comput. Syst.*, vol. 30, no. 4, pp. 1–26, 2012. DOI: 10.1145/2382553.2382556

[2] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, 2014. DOI: 10.1007/s10723-014-9314-7

[3] Amazon Web Services, "Application Auto Scaling – Target Tracking," 2024. [Online]. Available: https://docs.aws.amazon.com/autoscaling/

[4] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile Dynamic Provisioning of Multi-tier Internet Applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, pp. 1–39, 2008. DOI: 10.1145/1342171.1342172

[5] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting," in *Proc. IEEE CLOUD*, pp. 500–507, 2011. DOI: 10.1109/CLOUD.2011.42

[6] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, 2015. DOI: 10.1109/TCC.2014.2350475

[7] S. J. Taylor and B. Letham, "Forecasting at Scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018. DOI: 10.1080/00031305.2017.1380080

[8] J. Kumar, R. Goomer, and A. K. Singh, "Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model for Cloud Datacenters," *Procedia Comput. Sci.*, vol. 125, pp. 676–682, 2018. DOI: 10.1016/j.procs.2017.12.087

[9] J. H. Stock and M. W. Watson, "New Indexes of Coincident and Leading Economic Indicators," *NBER Macroecon. Annual*, vol. 4, pp. 351–394, 1989. DOI: 10.1086/654119

[10] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," in *Proc. ACM SOSP*, pp. 74–89, 2003. DOI: 10.1145/945445.945454

[11] Q. Zhang, L. Cherkasova, and E. Smirni, "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications," in *Proc. IEEE ICAC*, pp. 27–27, 2007. DOI: 10.1109/ICAC.2007.26

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009. DOI: 10.1145/1541880.1541882

[13] N. J. Gunther, *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*. Berlin: Springer, 2007. ISBN: 978-3-540-26138-4

[14] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Trans. Netw.*, vol. 2, no. 1, pp. 1–15, 1994. DOI: 10.1109/90.282603

[15] G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control*. San Francisco: Morgan Kaufmann, 2001. ISBN: 1-55860-508-8

[16] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," in *Proc. USENIX ICAC*, pp. 23–27, 2013.

[17] Amazon Web Services, "Amazon DocumentDB – Managing Performance and Scaling," 2024. [Online]. Available: docs.aws.amazon.com/documentdb/

[18] Amazon Web Services, “Amazon DocumentDB – Replication,” 2024. [Online]. Available: docs.aws.amazon.com/documentdb/

Manuscript received: December 2025