Topology and Scale Effects on Performance and Energy in µBench Microservices

Irena Ristova
VU Amsterdam
i2.ristova@student.vu.nl

ABSTRACT

Microservice architectures rely on diverse communication patterns that can significantly influence system performance, energy consumption, and reliability. However, isolating the specific impact of these architectural "structures" or topologies from other factors remains a challenge, leaving a gap in empirical evidence.

The objective of this study is to quantify how communication topology and system size influence performance, resource usage, and energy consumption when workload and service logic are held constant.

To achieve this, the µBench framework was used to systematically generate and benchmark microservice systems. Six distinct topology families were instantiated—including sequential and parallel fan-outs, branched chains, hierarchical and probabilistic trees, and complex meshes across three system sizes (5, 10, and 20 services). Each configuration is executed ten times under a fixed CPU-bound workload generated by Locust. Performance and resource metrics are collected via Prometheus, while system-wide CPU package and DRAM energy are measured using EnergiBridge (Intel RAPL) in joules.

The results reveal statistically significant differences across the evaluated architectures. Densely connected topologies, such as meshes, exhibited lower throughput, higher latency, and increased energy consumption compared to hierarchical or fan-out structures. Visual analysis further identifies a strong interaction effect between topology and scale: the efficiency gap between simple and complex architectures widens disproportionately as the system grows. These findings provide empirical evidence that communication topology is a first-order determinant of microservice efficiency. Independent of the underlying code or workload, the architectural structure itself imposes measurable, scale-dependent effects on system performance and energy consumption.

ACM Reference Format:
Irena Ristova. 2026. Topology and Scale Effects on Performance and Energy in µBench Microservices . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Microservice architectures have become the standard for building large-scale, cloud-native applications due to their inherent modularity and scalability. By decomposing applications into independently deployable units, this style enables rapid evolution and flexibility. However, it also introduces complex network interaction patterns. These structural dependencies significantly influence non-functional system properties, including performance, reliability, and—crucially—energy consumption.

Energy-aware software engineering has been increasingly recognized as a first-order design concern, particularly for cloud-native systems where architectural choices directly affect operational energy costs and environmental impact [1].While the performance characteristics of microservices have been widely studied, the relationship between communication topology and energy consumption remains underexplored. Existing research has investigated performance characteristics of microservice-based systems, the relationship between architectural communication structure and energy consumption remains insufficiently explored. In particular, empirical studies often focus on individual systems or application-specific architectures, limiting the generalizability and reusability of their findings.

A key challenge in studying energy efficiency at the architectural level is the lack of systematic and reusable benchmarking datasets. Existing benchmarks typically evaluate fixed microservice applications or narrowly defined workloads, making it difficult to isolate the impact of architectural properties such as service dependency structure, degree of parallelism, or interaction complexity. Consequently, software architects lack the data required to reason about energy–performance trade-offs across different classes of microservice architectures.

This research addresses this gap by building a reusable benchmarking dataset containing energy and performance measurements for systematically generated microservice-based systems. Rather than focusing on a single application, the dataset comprises multiple families of systems that differ in their communication structure and size. Each system is generated and executed under controlled conditions, enabling direct comparison across architectural variants. The dataset is intended to support empirical research on energy-aware software architecture and to serve as a foundation for evaluating optimization techniques, analysis methods, and architectural design choices.

To construct the dataset, we use µBench, a framework for generating and benchmarking synthetic microservice systems. µBench provides predefined workmodels that encode distinct service interaction patterns. By analyzing the communication structure induced by these workmodels, we interpret each of them as a representative microservice communication topology. Systems are instantiated at

multiple sizes to capture scalability effects, and each configuration was benchmarked under strict experimental guidelines [2], with high-resolution timestamps synchronizing resource usage, power consumption, and service-level performance metrics.

The main contribution of this work is a publicly available dataset that enables controlled, topology-aware analysis of energy and performance characteristics in microservice-based systems. By systematically varying communication structure and system size, the dataset facilitates reproducible experimentation and comparative analysis, contributing empirical evidence to an area that is currently dominated by case-specific studies.[2].

2 RELATED WORK

Empirical research on microservice architectures primarily focuses on performance, reliability, and scalability. This body of work falls into three distinct categories: (i) benchmarking frameworks, (ii) analysis of production traces and dependencies, and (iii) theoretical assessments of architectural complexity and topology analysis.

Microservice Benchmarking Frameworks. Detti et al. introduced μBench, an open-source framework for generating configurable benchmark microservice applications on Kubernetes, enabling controlled experimentation on architectural features such as service mesh topology, workload characteristics, and internal service behavior [3]. μBench supports systematic construction of service dependency graphs, standardized workload injection, and automated metric collection. The authors used μBench to study performance and scalability effects of microservice architectures, including monolithic versus microservice deployments, sequential versus parallel downstream calls, and centralized versus distributed service meshes. Their results show that centralized meshes introduce bottleneck microservices that limit throughput and increase latency, and that parallel fan-out structures reduce cumulative latency by limiting sequential dependencies.

The importance of topology-aware and repeatable microservice benchmarking is further emphasized by Aderaldo et al. [4], who propose a set of architectural and DevOps requirements for microservice research benchmarks. Their requirements explicitly call for an *explicit topological view* of service dependencies and for *pattern-based architectures* representative of real-world microservice systems. They further stress the need for containerized deployment, orchestration support, and automation to enable reproducible empirical evaluation. This aligns directly with the topology-driven design of the present study, which systematically generates star, chain, tree, and mesh communication structures under controlled workload conditions using Kubernetes-based deployment.

While prior benchmark frameworks and requirement studies focus primarily on performance and scalability, they do not address energy behavior. In contrast, this study extends topology-aware benchmarking toward explicit energy measurement, enabling a systematic evaluation of how communication topology affects not only throughput and latency but also CPU package energy and DRAM energy.

Microservice Dependency and Call-Graph Characterization. Several large-scale production studies have shown that microservice call-graph structure strongly affects runtime behavior. Zhang et al. [5] characterize task dependencies in large data-parallel jobs in Alibaba Cloud and show that dependency patterns significantly influence execution time and scheduling behavior. Lue et al. [6] analyze microservice dependency graphs from Alibaba production traces and demonstrate that service fan-in, fan-out, and call-chain length are strong predictors of performance variation. Du et al. [7] further characterize microservice call graphs and their impact on runtime performance, showing that graph complexity and path length correlate with latency and throughput degradation. These studies provide empirical evidence that call-graph topology is a key determinant of microservice performance in real-world systems.

Architectural Topology and Complexity. Van der Laan [8] proposes using topology complexity metrics as indicators of architectural quality and argues that structural complexity correlates with maintainability and performance-related risks. Bakhtin et al. [9] analyze network centrality in microservice architectures and show that highly central services act as performance bottlenecks and reliability risks. These findings align with the observation that densely connected topologies introduce contention and coordination overhead, motivating the need to study how structural properties affect system behavior.

Performance Diagnosis and Dependency Management. Gan et al. [10] study dependency management and performance evaluation in cloud-native microservices, demonstrating that long call chains and excessive inter-service dependencies degrade performance. Hou et al. [11] propose causal analysis techniques for diagnosing microservice performance issues based on call-graph structure, highlighting the role of dependency paths in explaining latency and throughput anomalies. These works emphasize that microservice performance is strongly shaped by architectural dependency patterns.

Positioning of This Work. While prior studies have analyzed microservice dependencies using production traces or synthetic benchmarks, most focus primarily on performance and scalability, with limited attention to energy consumption. In contrast, this study systematically evaluates how communication topology affects not only throughput and latency but also CPU package energy and DRAM energy under controlled workload conditions. By combining μBench-based topology generation [3], benchmark requirements grounded in topology-aware design [4], automated large-scale experimentation, and energy measurement via RAPL counters, this work provides empirical evidence that architectural structure is one of the primary drivers of both performance and energy behavior. Moreover, unlike trace-based studies that observe existing systems, this study isolates topology effects by keeping service logic and workload constant, enabling causal attribution of observed differences to architectural structure.

2.1 Research Goal

The objective of this research is to benchmark and analyze microservice-based systems that conform to distinct communication topologies. The systems are instantiated using representative example workmodels provided by μBench and systematically configured and

scaled to enable comparative evaluation. Performance and energy-related metrics are collected under controlled workloads to construct a reusable benchmarking dataset for energy-aware software architecture research.

Following the Goal–Question–Metric (GQM) approach, the research goal is formalized as follows:

- **Purpose:** To evaluate and compare the energy efficiency of different microservice communication topologies.
- **Issue:** Limited empirical understanding of how communication structures influence non-functional properties such as energy consumption, latency, and throughput.
- **Object:** Microservice-based systems instantiated from selected µBench example workmodels, representing a range of communication patterns that differ in structural characteristics such as degree of centralization, fan-in and fan-out behavior, execution parallelism, path length, and connectivity density.
- **Viewpoint:** Researchers and software architects interested in architectural trade-offs and energy-efficient system design.

2.2 Research Questions

From the research goal above, the following research questions (RQs) are derived:

- **RQ1:** How does the communication topology of a microservice system affect its energy consumption under controlled workloads?
- **RQ2:** Are certain microservice communication topologies statistically more energy-efficient than others?
- **RQ3:** How do system performance characteristics (e.g., latency and throughput) relate to energy consumption across different communication topologies?

These research questions are addressed through systematic experimentation in which microservice systems are generated and deployed using µBench, while workload execution, measurement orchestration, and data collection are handled by an external Experiment Runner framework. Energy consumption is measured using hardware-level energy counters (RAPL), and performance metrics are collected under identical workload conditions across topology families and system sizes.

3 EXPERIMENT PLANNING

This section describes how the experiment was planned and structured prior to execution. It defines the rationale behind topology selection, the construction and scaling of experimental systems, and the experimental subjects, variables, hypotheses, and design decisions that guided the benchmarking process.

3.1 Rationale for Topology Selection

The selected topologies were chosen to represent a diverse set of microservice communication structures commonly observed in real-world systems. Rather than focusing on application-specific architectures, the selection targets fundamental interaction patterns that differ in centralization, execution order, branching behavior, and connectivity. These structural dimensions are known to influence system behavior and architectural complexity, and are therefore expected to affect energy consumption as well [8, 9].

The topology families considered in this study are derived from executable µBench workmodels, ensuring that each selected structure corresponds to a deployable and empirically grounded system configuration.

- **Structural Diversity:** The topology set spans centralized fan-out patterns, sequential chains with branching, hierarchical decompositions, probabilistic execution paths, and densely connected interaction graphs. This diversity enables comparison across architectures with contrasting fan-in/fan-out characteristics, degrees of parallelism, and interaction complexity [6, 7].
- **Architectural Significance:** Each topology corresponds to interaction patterns commonly encountered in microservice systems, including API Gateway–style orchestration [12], request processing pipelines [13], hierarchical and layered designs [8, 14], conditional execution paths, and highly coupled service networks [6, 9].
- **Empirical Grounding:** Empirical analyses of large-scale industry deployments reveal substantial variation in microservice dependency structures, including centralized hubs, long call chains, hierarchical trees, and dense interaction graphs [6, 7]. Studies on architectural complexity and centrality further show that these patterns recur across systems and can be characterized using graph-based metrics such as fan-in, fan-out, and path length [8, 9, 15].

3.2 Topology Construction and Scaling Approach

All microservice systems evaluated in this experiment are instantiated using predefined µBench workmodels from the Examples/ directory. Each workmodel encodes an executable service interaction pattern, which is interpreted and classified as a distinct microservice communication topology.

Rather than constructing abstract topology graphs from scratch, this study derives its topology families from these predefined workmodels to ensure executability, reproducibility, and alignment with µBench's intended usage. The selected workmodels were analyzed based on their induced service call graphs and execution semantics, and assigned descriptive topology names accordingly.

- Each topology was instantiated at three system sizes—5, 10, and 20 services—to study scalability effects. Sequential and parallel fan-out workmodels were extended to support larger configurations, while chain-, tree-, and mesh-based workmodels originally defined for larger systems were scaled down as needed.
- The **Chain with Branching** topology was lightly restructured to consistently enforce a linear execution path with forward branches across all system sizes, ensuring structural consistency.
- To isolate the impact of communication structure, all workmodels were standardized using a uniform, lightweight configuration. CPU-intensive workloads were enabled uniformly across all services, while memory and disk stress were disabled.

- Kubernetes resource limits, response sizes, and deployment settings were kept consistent across all systems to ensure that observed differences in performance and energy consumption are attributable primarily to topology structure and execution semantics rather than configuration variability.

3.3 Topology Definitions

In this study, microservice communication topologies are derived from µBench workmodels, each encoding an executable pattern of service interactions. By analyzing the induced service call graphs and execution semantics, each workmodel is interpreted as representing a distinct communication topology. Descriptive topology names are assigned based on structural characteristics to support clarity and comparability [1, 3]. Table 1 summarizes the selected topology families, their structural characteristics, and the expected energy implications hypothesized at design time.

Fan-out refers to the number of outgoing service calls initiated by a service, while **fan-in** refers to the number of incoming calls received by a service. These properties are widely used to characterize microservice dependency structures, architectural complexity, and service centrality [6, 8, 9].

- **Sequential Fan-Out:** A centralized hub-and-spoke structure with strictly sequential execution semantics. A single entry-point service invokes downstream services one after another within a single execution group. Although the communication graph resembles a star, the execution path is linear, reflecting orchestrated service pipelines [13].
- **Parallel Fan-Out:** A centralized fan-out structure with concurrent execution semantics. The entry-point service invokes multiple downstream services simultaneously, placing each invocation in a separate execution group. This pattern corresponds to parallelized orchestration commonly implemented in API Gateway designs [12].
- **Chain with Branching:** A primarily linear sequence of services in which each service invokes the next, augmented by forward branches to downstream services. This topology introduces alternative execution paths while preserving an overall sequential structure, reflecting observed service call chains in practice [13].
- **Hierarchical Tree:** A balanced tree structure with a single root service acting as the system entry point. Each non-leaf service invokes a fixed number of child services, forming multiple hierarchical levels. Requests propagate deterministically along parent–child relationships [8, 14].
- **Probabilistic Tree:** An extension of the hierarchical tree topology in which individual service invocations are associated with execution probabilities. Not all branches are traversed for every request, modeling conditional routing and heterogeneous workload behavior [6].
- **Complex Mesh:** A densely connected service interaction graph without a dominant hierarchical structure. Services may invoke many others, resulting in multiple overlapping execution paths and high fan-in and fan-out. This topology reflects tightly coupled or organically evolved microservice systems [9, 16**?**].

3.4 Subjects Selection

The experimental subjects are synthetic microservice-based systems generated using µBench, each representing a specific communication topology and system size. These systems are not derived from a single real-world application; instead, they are systematically constructed to isolate and study the impact of architectural structure on performance and energy-related behavior.

Each subject is defined by:

- a communication topology, implemented through a µBench workmodel, and
- a system size, corresponding to the number of services in the deployment.

Six topology families were selected, and each family was instantiated at three different sizes (5, 10, and 20 services), resulting in 18 distinct system configurations. All systems were deployed and executed on the same infrastructure using identical runtime and workload settings.

The use of synthetic systems enables a high degree of control and repeatability, allowing architectural effects to be studied independently from application-specific logic or workload variability, in line with established benchmarking recommendations.

3.5 Experimental Variables

This experiment follows a controlled design in which architectural structure is deliberately varied while all other factors are held constant.

Independent Variables. The independent variables are the factors that are intentionally varied during the experiment:

- **Communication topology:** The structure of service-to-service interactions, operationalized through µBench workmodels. Each workmodel induces a distinct communication graph characterized by differences in centralization, branching, parallelism, and connectivity.
- **System size:** The number of services in the deployment. Three sizes are considered: 5, 10, and 20 services. System size is used to study how architectural effects scale with increasing complexity.

Together, these variables define a family of systems rather than a single application instance.

Dependent Variables. The dependent variables are the metrics observed and measured during the experiment. These reflect the system's behavior under load and are expected to be influenced by the architectural structure:

- **Performance metrics**, including request throughput and latency, collected via load testing.
- **System-level resource metrics**, such as CPU and memory utilization, collected via monitoring.
- **Energy-related measurements**, collected at the system level during execution.

These variables are not manipulated directly, but observed as outcomes of the chosen topology and system size under a fixed workload.

Controlled Variables. To ensure fair and comparable results, the following factors are kept constant across all runs:

- Workload configuration (same request rate, concurrency, and duration)
- Service configuration (identical µBench workload parameters and resource settings)
- Execution environment (same hardware, operating system, container runtime, and Kubernetes cluster)
- Benchmarking procedure (same deployment, measurement, and cleanup steps)

By controlling these factors, any observed differences in performance or energy-related behavior can be attributed primarily to the independent variables.

3.6 Experimental Hypotheses

Based on prior work on microservice architectures and architectural complexity, the experiment is guided by the following hypotheses:

- **H1:** Communication topology has a measurable effect on system-level performance and resource utilization under identical workload conditions.
- **H2:** Architectural structures with higher degrees of parallelism exhibit different performance–energy trade-offs than more sequential or centralized structures.
- **H3:** Increasing system size amplifies topology-related effects on performance and resource usage.
- **H4:** More structurally complex communication graphs (e.g., higher connectivity or branching) incur higher overhead than simpler structures under comparable load.

These hypotheses are formulated before observing results and focus on structural properties, rather than specific topology names.

3.7 Experiment Design

The experiment is designed as a fully automated, controlled benchmarking workflow executed on a remote server.

For validation purposes, the complete workflow was first executed using 18 runs (6 topologies × 3 sizes × 1 repetition) to verify correctness and end-to-end integration. Each run followed the same sequence of steps:

(1) Deploy the µBench application corresponding to the selected topology and size.
(2) Wait until all services are ready and reachable.
(3) Apply a fixed external workload to the system.
(4) Collect performance and system metrics during execution.
(5) Tear down the deployment and clean up resources.

The order of configurations is randomized to reduce temporal bias, and each configuration is executed in isolation to avoid interference between runs.

The final experimental setup extends this validated design with multiple repetitions per configuration to support statistical analysis. All data, analysis scripts, experiment code and figures required to replicate the results are available in the following replication package [17].

3.8 Data Analysis Plan

The data analysis focuses on comparing system behavior across different architectural structures and system sizes.

Planned analyses include:

- Descriptive statistics for performance, resource usage, and energy-related metrics.
- Cross-topology comparisons under identical workload conditions.
- Analysis of scaling effects as system size increases.
- Examination of variability across repetitions.

Rather than identifying optimal architectures, the analysis is intended to demonstrate how the resulting dataset can be used to study topology-related trade-offs in microservice systems.

4 EXPERIMENT EXECUTION

This section describes how the experiment was executed, including the hardware and software infrastructure, the toolchain used for deployment and measurement, and the execution workflow followed for each experimental run. Figure 1 provides an overview of the experimental infrastructure and execution flow, illustrating the interaction between the host machine, the Kubernetes cluster on the server, and the monitoring and measurement components.

4.1 Hardware and Execution Environment

The experiment was conducted using two machines with clearly separated responsibilities.

Host Machine. The host machine was used for experiment orchestration, workload generation, and data aggregation. It is an HP EliteBook 650 running Ubuntu 24.04.3 LTS, equipped with 16 GB RAM and a 512 GB SSD. The host executes the Experiment Runner framework and the Locust load generator, and coordinates deployments, measurements, and result collection.

Server Machine. All microservice systems were deployed and executed on a dedicated remote server (gl3). The server is equipped with two Intel Xeon processors (32 vCPUs in total), 384 GB RAM, and 36 TB of storage, and runs a Linux operating system. A single-node Kubernetes cluster was instantiated using Minikube on this server. The cluster was allocated 28 vCPUs and approximately 350 GB of memory, leaving sufficient resources for the host system and monitoring tools to ensure stable operation during benchmarking.

4.2 Toolchain

The experiment relies on a combination of established benchmarking and monitoring tools.

µBench. µBench is used to generate and deploy synthetic microservice-based systems based on predefined workmodels. The K8sDeployer component translates workmodel specifications into Kubernetes resources. Each experimental run deploys a fresh µBench application instance in an isolated Kubernetes namespace.

Experiment Runner. Experiment Runner is a Python-based framework used to orchestrate the full experimental workflow. It manages deployment, controls measurement phases, queries monitoring tools, and aggregates all collected metrics into a structured run table.

Locust. Locust is used as the workload generator. For each run, Locust executes a headless load test with 100 concurrent users over a fixed duration of 10 minutes, sending HTTP requests to the system entry point exposed by the gateway. Performance metrics such as throughput, response latency, and failure rate are recorded.

Prometheus. Prometheus is deployed within the Kubernetes cluster and continuously scrapes resource usage metrics from all pods. After each run, the Experiment Runner queries the Prometheus API to retrieve CPU and memory usage metrics for the corresponding namespace.

EnergiBridge. EnergiBridge is used to measure system-wide energy consumption on the server using Intel RAPL counters. It runs as a privileged process directly on the server (outside Kubernetes) and records CPU package and DRAM energy consumption during the measurement phase of each run.

4.3 Network and Communication Setup

All interactions between the host and server machines are explicitly controlled. HTTP requests generated by Locust on the host are forwarded through SSH tunnels to the server, where they are routed to the NGINX gateway deployed as part of the µBench application. The gateway acts as the single entry point for all requests and dispatches them to the internal microservices according to the deployed topology.

Monitoring data is accessed through a separate SSH tunnel that exposes the Prometheus API to the host machine. This setup enables secure access to runtime metrics without exposing services publicly.

4.4 Execution Workflow

Each experimental run follows a fixed and fully automated execution protocol:

(1) **Initialization.** A configuration specifying the topology, system size, and repetition is selected, and a dedicated run directory is created.
(2) **Deployment.** A new Kubernetes namespace following the pattern mubench-{topology}-{size}-{replicate} is created, and the corresponding µBench application is deployed using the selected workmodel. The system waits until all pods are ready and the gateway becomes responsive.
(3) **Measurement Start.** EnergiBridge is started on the server, and synchronized measurement timestamps are recorded to align workload execution with energy measurements.
(4) **Workload Execution.** Locust executes the load test against the gateway for a fixed duration of 10 minutes.
(5) **Measurement Stop.** EnergiBridge is stopped, and energy data is retrieved. Prometheus is queried for CPU and memory usage metrics.
(6) **Data Aggregation.** Performance, resource usage, and energy metrics are parsed and aggregated into the run table.
(7) **Cleanup.** The Kubernetes namespace is deleted, and the system waits for a cooldown period of 60 seconds before the next run to reduce carry-over effects.

All configurations are executed in randomized order. Timeout mechanisms and gateway readiness checks are enforced before measurements start; runs that fail to deploy or stabilize within predefined limits are logged and excluded from analysis.

4.5 Collected Metrics

For each run, the following metrics are collected:

- **Performance metrics**: throughput, average latency, 95th percentile latency, and failure rate (from Locust).
- **Resource usage metrics**: average CPU and memory usage across all pods (from Prometheus).
- **Energy metrics**: system-wide CPU package and DRAM energy consumption in joules (from EnergiBridge).

All metrics are timestamped and stored in a structured format, enabling consistent post-processing and comparative analysis across topologies and system sizes.

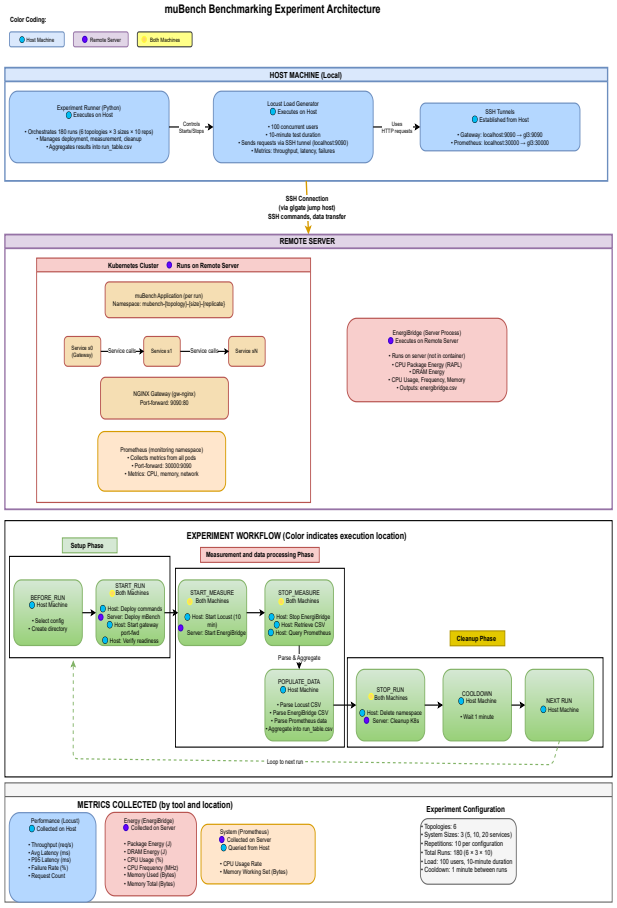

Figure 1: Overview of the experimental infrastructure and execution workflow

5 RESULTS

This section reports the empirical results obtained from benchmarking the generated microservice systems. First, descriptive statistics are presented to summarize overall performance, resource usage, and energy behavior across all configurations. Subsequently, exploratory visual analysis using boxplots is used to examine differences across communication topologies and system sizes. Finally, non-parametric hypothesis testing is applied to assess whether observed differences are statistically significant.

5.1 Descriptive Statistics

Table 2 summarizes descriptive statistics for all performance, resource usage, and energy metrics collected across the evaluated microservice systems. The table reports measures of central tendency and dispersion, including mean, standard deviation, median, minimum, maximum, and coefficient of variation.

Across all experimental runs, performance metrics exhibit substantial variability. Throughput values span a wide range, indicating notable differences in request-handling capacity across configurations. Similarly, both average and tail response times show large dispersion, with latency values extending from a few seconds to several tens of seconds.

The coefficient of variation highlights marked differences in metric stability. Failure rate and latency-related metrics show high relative variability, suggesting sensitivity to execution paths and runtime conditions. In contrast, CPU and memory usage exhibit lower coefficients of variation, indicating more stable resource utilization across runs.

Energy-related metrics display moderate variability compared to performance metrics. Total CPU package energy consumption remains within a relatively constrained range across runs, while DRAM energy consumption follows a similar pattern. This indicates that, although execution behavior differs across configurations, overall system-level energy usage remains comparatively stable.

While descriptive statistics provide an overview of metric distributions and variability, they do not establish whether observed differences across system configurations are statistically significant. To assess whether communication topology and system size induce systematic effects on performance and energy consumption, exploratory visual analysis using boxplots is presented next.

5.2 Visual Comparison Across Topologies and System Sizes

To provide an exploratory overview of how architectural structure and system size influence system behavior, boxplots were generated for key metrics across all configurations. Three complementary visualizations are used: a combined topology–system size view for performance and energy metrics, a size-only stacked view to illustrate global scaling trends, and a topology-only summary view for energy consumption.

Figure 4 shows combined boxplots for throughput, average response time, and CPU package energy consumption, grouped by topology and system size. Each box represents the distribution of metric values across ten independent repetitions for a single topology–size configuration. This representation enables direct comparison across the 18 experimental configurations (6 topologies × 3 system sizes) and reveals both main effects and interaction effects between topology and scale.

The combined plots reveal clear differences in median performance and energy behavior across architectural structures. For throughput, topologies with higher degrees of parallelism exhibit higher median request-handling capacity than more sequential or densely connected structures, whereas more complex topologies such as the *complex mesh* show lower throughput and greater dispersion. Quantitatively, at system size 20, the median throughput of the *probabilistic tree* topology is approximately 16.1 requests per second, whereas *complex mesh* degrades to about 8.3 requests per second, representing nearly a 50% reduction in request-handling capacity. At size 5, by contrast, most topologies achieve median throughputs in the range of approximately 18–24 requests per second, indicating that topology-related performance gaps widen significantly as system size increases.

For latency, similar structural effects are observed. Topologies with concurrent execution semantics show lower median response times, while more sequential and densely connected structures, including *chain-with-branching* and the *complex mesh*, exhibit higher latency and larger dispersion. These patterns are visible across all system sizes, although the magnitude of differences varies by topology. At size 20, the median average response time for *probabilistic tree* is approximately 4,760 ms, while *complex mesh* reaches about 10,240 ms, more than doubling the response time. At size 5, by contrast, median latencies range from approximately 2,100–3,400 ms across topologies, confirming that latency penalties of densely connected structures become disproportionately severe as system size increases.

Energy consumption patterns are also shown in Figure 4. Across all topologies, energy usage increases with system size from 5 to 20 services. However, the magnitude and pattern of this increase differ by communication structure, indicating interaction effects between topology and scale. More structurally complex topologies exhibit higher median energy consumption and steeper scaling trends, whereas fan-out and tree-based structures show comparatively lower energy usage at smaller system sizes. At size 20, the median CPU package energy consumption is approximately 38,676 J for *probabilistic tree* and 39,128 J for *complex mesh*. When aggregating across all topologies, median energy increases from approximately 30,200 J at size 5 to about 38,900 J at size 20, confirming a strong scaling effect of system size on energy consumption.

Additional metrics reinforce these structural effects. Failure rates are near zero for tree-based and fan-out topologies across all system sizes, whereas *complex mesh* and *chain-with-branching* reach median failure rates of up to approximately 4.8% at size 20, indicating that architectural complexity increases system fragility under load. CPU usage shows smaller but consistent topology-dependent differences: at size 20, median CPU utilization is approximately 52% for *probabilistic tree* and about 58% for *complex mesh*, reflecting higher coordination overhead in densely connected structures. DRAM energy consumption follows similar trends to CPU package energy, increasing from approximately 5,200 J at size 5 to about 6,800 J at size 20 when aggregated across all topologies, with *complex mesh* consistently exhibiting the highest median DRAM energy usage.

To provide a high-level overview independent of system size, Figure 3 summarizes CPU package energy consumption aggregated by topology across all sizes. This view highlights systematic differences in overall energy usage across architectural structures while masking size-specific effects.

Figure 2 presents a vertically stacked boxplot illustrating how system size affects throughput, latency, energy consumption, and CPU usage when aggregated across all topologies. These plots show a general scaling trend: as the number of services increases from 5 to 20, throughput tends to decrease, while latency and energy consumption increase. CPU usage exhibits smaller but consistent

increases with system size, indicating modest growth in resource utilization relative to the observed performance and energy effects.

The boxplots also reveal differences in variability across metrics. Latency-related metrics exhibit higher dispersion, indicating sensitivity to execution paths and runtime conditions. In contrast, CPU usage shows more stable distributions across repetitions. Energy-related metrics display moderate variability, with relatively constrained ranges compared to performance metrics.

While these visual differences suggest that communication topology and system size influence performance and energy behavior, boxplots alone do not establish whether the observed differences are statistically significant. These magnitude differences are not only statistically significant but also practically substantial: at size 20, a twofold increase in median latency and a 50% reduction in throughput would be directly observable by end users and would translate into materially higher operational costs and degraded quality of service. To assess the statistical significance of these effects formally, non-parametric hypothesis testing is applied in the following section.

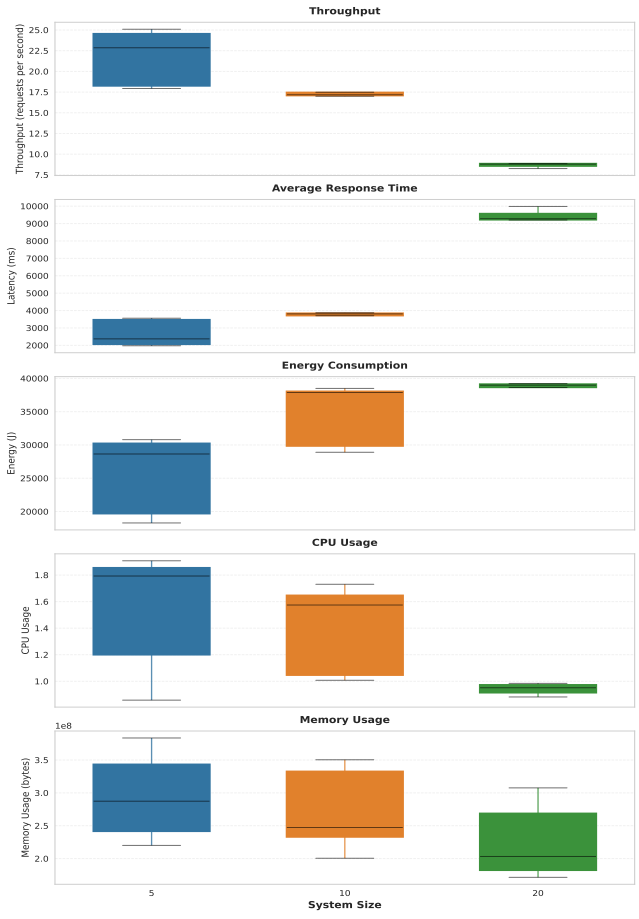

Figure 2: Stacked boxplots showing how system size affects throughput, latency, energy consumption, and CPU usage.

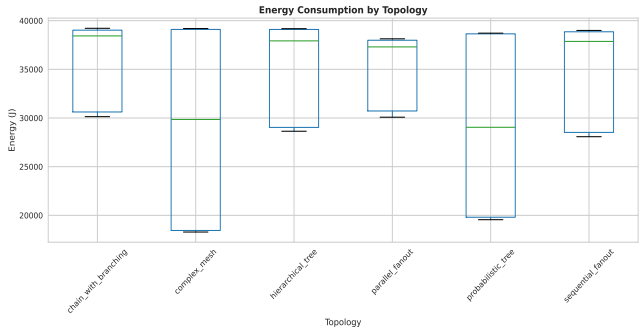

Figure 3: Energy consumption aggregated by topology (all sizes combined). Highlights structural energy differences.

5.3 Hypothesis Testing

Because the collected performance and energy metrics exhibit skewed distributions, heteroscedasticity, and non-negligible dispersion across configurations (as visible in the boxplots), parametric test assumptions such as normality and homoscedasticity cannot be safely assumed. Moreover, each topology–size configuration contains only ten repetitions, which further limits the robustness of parametric tests. For these reasons, non-parametric tests were selected.

Specifically, Kruskal–Wallis tests were used to assess overall differences across topology groups for each metric, as they do not require normality assumptions and are robust to unequal variances. When the Kruskal–Wallis test indicated a statistically significant effect, pairwise Mann–Whitney U tests were applied as post-hoc comparisons to identify which topology pairs differed. This testing strategy follows standard recommendations for non-normally distributed experimental data in empirical software engineering [2]. A significance level of $\alpha = 0.05$ was used for all tests.

To evaluate whether the observed differences across communication topologies are statistically significant, non-parametric hypothesis tests were applied to all collected metrics. For each metric M, the hypotheses are defined as follows:

Null hypothesis (H_0): the distribution of M is identical across all communication topologies (i.e., any observed differences are attributable to random variation).

Alternative hypothesis (H_1): at least one topology exhibits a different distribution of M.

Across the evaluated metrics (throughput, average response time, 95th percentile response time, failure rate, CPU usage, memory usage, CPU package energy, and DRAM energy), the Kruskal–Wallis tests indicate statistically significant differences between topologies (all $p < 0.05$; see Appendix A). These results provide evidence that communication topology has a measurable effect on both performance and energy-related behavior under identical workload conditions.

Post-hoc Mann–Whitney U tests reveal multiple statistically significant pairwise differences between specific topologies (Appendix A). In particular, densely connected architectures differ significantly from more centralized or hierarchical structures across latency-related metrics and energy consumption. These statistically

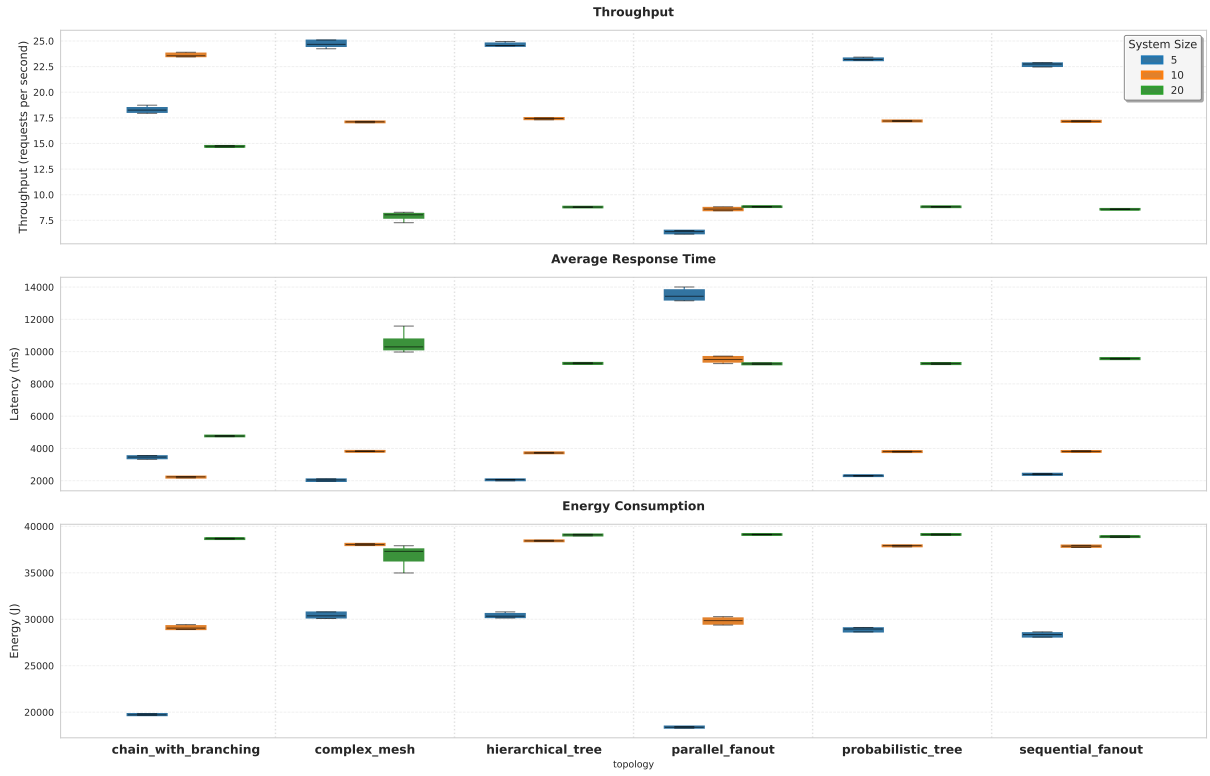

Figure 4: Combined boxplots of throughput, average response time, and energy consumption grouped by topology and system size. Interaction effects are visible.

significant contrasts are consistent with the median shifts and separation patterns visible in the combined topology–and–size boxplots (Figure 4).

Overall, the hypothesis testing results support the claim that topology influences energy and performance outcomes (supporting H1), and they provide inferential confirmation that the visual differences observed in Section 5.2 are unlikely to be explained by random variation alone. The interaction effects visible in Figure 4 further provide partial support for H3, indicating that topology-related effects become more pronounced with increasing system size, although the magnitude of these effects varies across topologies.

Detailed test statistics and p-values are reported in Appendix A.

6 DISCUSSION

The experimental results provide empirical evidence that communication topology significantly influences both performance and energy characteristics of microservice systems. This section interprets the findings in relation to the research questions and hypotheses and situates them within the controlled architectural and workload assumptions enforced by the benchmarking framework.

6.1 Topology Effects on Performance and Energy (H1, RQ1, RQ2)

The statistical test results provide robust support for H1, indicating that communication topology has measurable and statistically significant effects on system behavior. Across all eight evaluated metrics, the Kruskal–Wallis tests reject the null hypothesis of identical distributions ($p < 0.05$ for all metrics, with most $p < 10^{-10}$). This establishes that architectural structure is a primary determinant of performance, energy consumption, and reliability under identical workload conditions.

The combined boxplots and post-hoc comparisons reveal consistent structural patterns. More densely connected topologies, most notably the *complex mesh*, exhibit significantly lower throughput and higher latency than more centralized or hierarchical structures. In contrast, fan-out and tree-based structures (*parallel fanout*, *probabilistic tree*) demonstrate higher throughput and lower latency. These results indicate that architectural organization alone, independent of service logic and workload, introduces systematic differences in request-handling efficiency.

These effects are not only statistically significant but also substantial in magnitude. At system size 20, throughput and latency differences between *probabilistic tree* and *complex mesh* differ by roughly a factor of two, as quantified in Section 5.2. These differences are directly observable at the application level and confirm that architectural structure has first-order performance consequences.

Energy consumption shows a similarly structured effect. While all topologies exhibit increased energy usage with system size, the magnitude of this increase varies substantially. *Complex mesh* and *hierarchical tree* show higher median CPU package energy consumption and steeper scaling trends, whereas *probabilistic tree* and *parallel fanout* exhibit more moderate growth. At system size 20, structurally complex topologies consistently incur higher energy overhead, as detailed in Section 5.2.

Because the workload is CPU-bound (each service computes 100 digits of π repeated 10 times per request), energy consumption primarily reflects cumulative CPU activity rather than network energy. The higher energy usage observed for *complex mesh* is therefore consistent with longer execution paths, increased coordination overhead, and greater cumulative service invocation per request.

Failure rate results further indicate that topology affects not only performance and energy efficiency but also system robustness. The Kruskal–Wallis test shows extremely strong evidence of differences ($p < 4.6 \times 10^{-22}$). *Complex mesh* and *chain with branching* exhibit significantly higher failure rates than tree-based structures, whereas fan-out and tree-based topologies remain near zero. These findings support RQ1 and RQ2, confirming that topology alone has a first-order effect on both efficiency and reliability.

6.2 Interaction Effects Between Topology and System Size (H3)

The combined topology–system size boxplots reveal interaction effects between architectural structure and system scale. The magnitude of topology-related differences varies across system sizes, indicating that architectural effects are scale-dependent.

For throughput, topology differences are moderate at small sizes (5 services), but widen substantially at larger scales. As system size increases to 20 services, tree-based and fan-out structures maintain relatively stable throughput, whereas *complex mesh* degrades sharply, as quantified in Section 5.2. This pattern indicates that structurally complex topologies become increasingly inefficient as systems grow.

Latency exhibits a corresponding interaction pattern. At small sizes, median latencies are relatively close across topologies, while at size 20 the divergence becomes pronounced. Tree-based structures maintain substantially lower latencies, whereas *complex mesh* exhibits more than a twofold increase, confirming that cumulative coordination and service-hop overhead grows with both topology complexity and system size.

Energy consumption shows the most consistent interaction effect. When aggregating across all topologies, median CPU package energy increases from approximately 30 kJ at size 5 to about 39 kJ at size 20. Structurally complex topologies consistently incur higher energy overhead at scale, while fan-out and probabilistic tree structures exhibit more moderate increases. This confirms that simpler communication patterns maintain better energy efficiency as systems grow.

Taken together, these results provide partial support for H3. Topology effects are observable at all system sizes, but their magnitude increases with scale. However, the evaluated size range (5–20 services) remains limited, and stronger or qualitatively different interaction effects may emerge at larger scales.

6.3 Performance–Energy Trade-offs (RQ3)

The joint analysis of throughput, latency, and energy consumption reveals a clear performance–energy relationship across architectural structures. Topologies that achieve higher throughput and lower latency also tend to exhibit lower energy consumption. Conversely, topologies with poorer performance characteristics exhibit higher energy overhead.

This relationship indicates that energy consumption in this setting is driven primarily by runtime behavior rather than static architectural complexity alone. Faster execution and reduced coordination overhead lead directly to lower cumulative CPU activity and therefore lower energy usage. These findings address RQ3 by demonstrating that topology-induced performance differences translate into corresponding differences in energy efficiency.

6.4 Variability and Reliability Considerations

Beyond median differences, the results reveal important variability patterns. Latency-related metrics exhibit the highest coefficients of variation (CV $\approx$ 0.62–0.65), indicating substantial variability across repetitions and sensitivity to runtime conditions and execution paths. This variability is particularly pronounced for *complex mesh* and *chain with branching*.

Energy metrics exhibit lower variability (CV $\approx$ 0.20–0.24), indicating more consistent energy consumption patterns across repetitions. This stability suggests that energy usage is more predictable than latency or throughput, making it suitable for capacity planning and cost estimation.

Failure rate exhibits the highest variability (CV $\approx$ 1.13), with some topologies showing near-zero failures while others reach failure rates of up to approximately 4.8%. This high variability, combined with statistically significant differences between topologies, indicates that architectural structure fundamentally affects system reliability, not only average performance.

It is important to note that these findings arise under the controlled workload and architectural assumptions enforced by µBench. All services perform identical CPU-bound computation, and all inter-service communication uses synchronous HTTP. As a result, the observed differences can be causally attributed to topology-induced variations in call structure, coordination overhead, and path length, rather than to service heterogeneity or workload imbalance. This controlled design strengthens internal validity but also constrains the scope of generalization.

7 THREATS TO VALIDITY

This section discusses potential threats to the validity of the experiment, structured according to the standard classification into internal, external, construct, and conclusion validity.

7.1 Internal Validity

All experimental runs were executed on the same dedicated server and Minikube cluster, with randomized run order, namespace isolation, and a fixed cooldown period between runs. These measures

reduce confounding effects related to execution order and cross-run interference.

However, Docker and the Minikube cluster were not restarted between runs, which may have introduced residual resource states or caching effects. CPU frequency scaling, thermal throttling, and background operating system activity were not explicitly controlled. One experimental run failed and was manually re-executed and reintegrated into the dataset, introducing a small risk of inconsistency. Pod readiness timeouts were also observed for larger topologies, although all deployments ultimately completed successfully. These factors may have introduced minor noise into the measurements but are unlikely to explain the large and systematic differences observed across topologies.

7.2 External Validity

The study evaluates six synthetic topologies and three system sizes (5, 10, and 20 services) under a standardized, CPU-bound workload. While this controlled design enables fair comparison, it limits generalization to real-world systems with heterogeneous services, variable workloads, autoscaling behavior, and complex deployment environments.

All services perform identical computation and communicate via synchronous HTTP, with no caching, persistence, or asynchronous messaging. The experiments were executed on a single-node Kubernetes cluster using Minikube, which does not reflect production-grade multi-node deployments. In real-world clusters, service placement, inter-node network latency, and cross-node resource contention may alter the observed performance and energy characteristics. Moreover, the evaluated size range is limited, and much larger systems may exhibit different topology effects. These constraints limit the extent to which the results generalize to production microservice systems.

7.3 Construct Validity

Communication topology is operationalized using synthetic service graphs generated by µBench, which capture structural differences but do not reflect the full complexity of real-world architectural patterns. Energy consumption is measured using Intel RAPL counters via EnergiBridge, capturing CPU package energy and DRAM energy at the system level. Network interface energy, storage energy, and power consumed by external infrastructure components are not included, which may underestimate the total energy footprint of communication-intensive topologies.

Energy is measured as total energy per run rather than per request, reflecting sustained system energy usage but not fine-grained energy efficiency. Performance metrics are measured at the client side using Locust, and the workload is synthetic and homogeneous. While this design isolates topology effects, it simplifies the constructs of performance, energy efficiency, and microservice topology relative to real deployments.

During pilot experimentation, an alternative approach based on manually generated service graphs was explored. Although these service graphs could be used to generate workmodels and later deployed and executed at larger scales, their structural and semantic properties did not fully satisfy the assumptions expected by the µBench workmodel generator. As a result, the generated workmodels did not reliably instantiate the intended request propagation and execution behavior, resulting in insufficient and inconsistent CPU utilization signals. To avoid conflating topology effects with modeling artifacts, this study therefore relies exclusively on validated µBench workmodels with well-defined execution semantics.

7.4 Conclusion Validity

Non-parametric tests (Kruskal–Wallis and Mann–Whitney U) were used to accommodate skewed metric distributions and heteroscedasticity, and all collected data points were included in the statistical analysis. However, only ten repetitions per configuration were performed, which limits statistical power. Moreover, multiple testing correction was not applied despite performing eight Kruskal–Wallis tests and fifteen pairwise Mann–Whitney U tests per metric, increasing the family-wise error rate.

Effect sizes were not computed, and not all pairwise topology comparisons are statistically significant, even when visual differences are visible in boxplots. While most p-values are extremely small, some reported differences would not remain significant under strict multiple-testing correction. In addition, the evaluated size range is limited, and the observed interaction effects supporting H3 are scale-dependent and not definitive.

8 CONCLUSIONS

This study provides empirical evidence that communication topology is a fundamental determinant of microservice system behavior. Under identical workloads and service logic, different architectural structures exhibit statistically significant differences in performance, energy consumption, and reliability. Densely connected topologies show lower throughput, higher latency, higher energy usage, and higher failure rates, whereas hierarchical and fan-out structures achieve more favorable performance and energy profiles. The observed interaction effects between topology and system size further indicate that architectural decisions have scale-dependent consequences, with topology-related differences becoming more pronounced as systems grow. Together, these results confirm that topology alone, independent of workload and implementation details, has a first-order effect on system efficiency and robustness.

Several extensions of this experiment are promising. First, µBench can be utilized more effectively by manually generating topology families with controlled structural characteristics, such as star, chain, tree, cyclic, tiered (layered), and database-centric architectures. These manually generated topologies can explicitly control database access strategies, such as how many services access a shared database and at what rates. Second, future experiments should be executed on larger Kubernetes clusters rather than a single-node Minikube setup, enabling evaluation at substantially larger system sizes (e.g., 200+ services). Studying such scales would better reflect real-world microservice deployments, which commonly involve hundreds or thousands of services, as observed in large production trace studies from Alibaba [5–7].

Finally, future work could explore deployment, scheduling, or routing strategies that explicitly account for microservice call-graph structure. Prior studies have shown that dependency structure,

call-graph complexity, and service centrality strongly affect microservice performance and bottlenecks [6–9]. In addition, future work should further investigate the formal semantic constraints of μBench service graph specifications to enable reliable generation of large-scale custom topologies with validated workload behavior. Building on these insights, topology-aware mechanisms such as graph reinforcement learning for optimized deployment [18] and topology-aware scheduling [14] could exploit these observed performance and energy differences to improve system efficiency and robustness.

REFERENCES

[1] I. Malavolta, "Green Lab: empirical software engineering for energy efficiency," 2020.

[2] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2012.

[3] A. Detti, L. Funari, and L. Petrucci, "μbench: an open-source factory of benchmark microservice applications," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[4] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. IEEE, 2017, pp. 78–83.

[5] H. Zhang, Y. Zheng, and W. Wang, "Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2019, pp. 1–12.

[6] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the 12th ACM Symposium on Cloud Computing (SoCC '21)*. New York, NY, USA: ACM, 2021, pp. 412–426.

[7] H. Du, X. Zhang, J. Wu, and Y. Zhang, "Characterizing microservice call graph and runtime performance," vol. 33, no. 12, 2024, pp. 3901–3914.

[8] K. van der Laan, "Using topology complexity as a software architecture design quality assessment," Master's Thesis, University of Amsterdam, July 2023, supervisors: Z. Zhao and J.J. Kester.

[9] A. Bakhtin, M. Esposito, V. Lenarduzzi, and D. Taibi, "Network centrality as a new perspective on microservice architecture," *Symmetry*, vol. 17, no. 4, p. 525, 2025.

[10] Y. Gan, Q. Li, J. Ma, Y. Zhang, and X. Wu, "On the design of microservices for cloud-native applications: Dependency management and performance evaluation," vol. 92, 2019, pp. 224–234.

[11] R. Hou, C. He, F. Zhang, Y. Liu, H. Zhang, H. Li, and J. Xu, "Pdiagnose: Performance diagnosis of microservices with causal analysis," in *Proceedings of the 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 31–42.

[12] C. Richardson, *Microservices Patterns with Examples*, 1st ed. Manning Publications, 2020.

[13] Z. Hao, X. Zhang, J. Liu, and Q. Wu, "Service call chain analysis for microservice systems," *Journal of Internet Technology*, vol. 23, no. 6, pp. 1203–1211, 2022.

[14] Y. Li, L. Yang, Z. Wu, L. Yu, K. Ren, and Z. Chen, "Topology-aware scheduling for microservice applications at the edge," in *Proceedings of the 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 2022, pp. 1–10.

[15] B. Fındık, "Using topological features of microservice call graphs to predict the response time variation," Master's thesis, Utrecht University, 2024. [Online]. Available: https://studenttheses.uu.nl/handle/20.500.12932/45567

[16] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices anti-patterns: A taxonomy," in *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 111–117.

[17] I. Ristova, "Replication Package for "Topology and Scale Effects on Performance and Energy in μBench Microservices"," 2026, accessed: 2026-01-26. [Online]. Available: https://github.com/IrenaRistova/topology-scale-mubench-replication

[18] W. Lv, P. Yang, T. Zheng, C. Lin, Z. Wang, M. Deng, and Q. Wang, "Graph-reinforcement-learning-based dependency-aware microservice deployment in edge computing," *IEEE Internet of Things Journal*, vol. 11, no. 1, pp. 1604–1615, 2024.

Table 1: Summary of Selected Microservice Topologies and Hypothesized Energy Implications

Topology Name	Structural Description	Fan-Out	Fan-In	Expected Energy Implications (Hypothesized)
Sequential Fan-Out	Centralized hub-and-spoke structure with strictly sequential invocation of downstream services.	High (at central service)	High (to central service)	Expected to increase cumulative CPU activity due to serialized execution; the central service may become an energy hotspot.
Parallel Fan-Out	Centralized hub-and-spoke structure with concurrent invocation of downstream services.	High (at central service)	High (to central service)	Hypothesized to reduce total execution time through parallelism while increasing instantaneous energy draw due to concurrent service activity.
Chain with Branching	Linear sequence of services with forward branches creating alternative execution paths.	Low–Medium	Low–Medium	Expected energy usage to scale with path length; branching may introduce variability in CPU utilization and intermittent load spikes.
Hierarchical Tree	Balanced tree structure with deterministic parent–child service invocations.	Medium (internal nodes)	Medium (internal nodes)	Hypothesized load concentration at higher tree levels; cumulative energy overhead may increase with depth and branching factor.
Probabilistic Tree	Tree structure with probabilistic service invocations and variable execution paths.	Medium (internal nodes)	Medium (internal nodes)	Expected uneven resource utilization across requests; probabilistic paths may lead to fluctuating CPU activity and underutilized branches.
Complex Mesh	Densely connected service network without centralized control.	High	High	Hypothesized high coordination and communication overhead; dense interconnections may increase cumulative CPU usage and energy consumption.

Table 2: Descriptive Statistics of Performance and Energy Metrics

metric	mean	std	min	median	max	cv
Throughput	15.4802	6.4160	6.1576	17.1256	25.0961	0.4145
Average Response Time	5851.4647	3612.1338	1975.6409	3819.2533	14004.9078	0.6173
95th Percentile Response Time	10874.4444	7026.6144	3800.0000	7600.0000	28000.0000	0.6462
Failure Rate	0.0079	0.0089	0.0000	0.0048	0.0519	1.1319
CPU Usage	1.3134	0.3881	0.7990	1.1185	1.9075	0.2955
Memory Usage	259553295.3838	58320826.1390	171511092.8254	247309591.2727	383525319.1111	0.2247
Energy Consumption	33284.7422	6524.2931	18284.7270	37780.9735	39216.4740	0.1960
DRAM Energy Consumption	9070.9853	2212.5501	3274.0010	10457.1300	10810.7990	0.2439

A FULL STATISTICAL TEST RESULTS

This appendix contains the full list of statistically significant test results across all metrics and topologies ($p < 0.05$).

Table 3: Significant Statistical Test Results ($p < 0.05$)

Metric	Test	Statistic	p_value
Throughput	KW – Throughput (all topologies)	52.641	0.000000
Throughput	MWU (probabilistic_tree vs complex_mesh)	900.000	0.000000
Throughput	MWU (probabilistic_tree vs hierarchical_tree)	600.000	0.027086
Throughput	MWU (probabilistic_tree vs sequential_fanout)	600.000	0.027086
Throughput	MWU (parallel_fanout vs complex_mesh)	700.000	0.000225
Throughput	MWU (chain_with_branching vs complex_mesh)	810.000	0.000000
Throughput	MWU (chain_with_branching vs sequential_fanout)	600.000	0.027086
Throughput	MWU (complex_mesh vs hierarchical_tree)	60.000	0.000000
Throughput	MWU (complex_mesh vs sequential_fanout)	154.000	0.000012
Avg. Response Time	KW – Avg. Response Time (all topologies)	52.549	0.000000
Avg. Response Time	MWU (probabilistic_tree vs complex_mesh)	0.000	0.000000
Avg. Response Time	MWU (probabilistic_tree vs hierarchical_tree)	300.000	0.027086
Avg. Response Time	MWU (probabilistic_tree vs sequential_fanout)	300.000	0.027086
Avg. Response Time	MWU (parallel_fanout vs complex_mesh)	200.000	0.000225
Avg. Response Time	MWU (chain_with_branching vs complex_mesh)	84.000	0.000000
Avg. Response Time	MWU (chain_with_branching vs sequential_fanout)	300.000	0.027086
Avg. Response Time	MWU (complex_mesh vs hierarchical_tree)	837.000	0.000000
Avg. Response Time	MWU (complex_mesh vs sequential_fanout)	740.000	0.000019
95th Percentile RT	KW – 95th Percentile RT (all topologies)	59.582	0.000000
95th Percentile RT	MWU (probabilistic_tree vs parallel_fanout)	277.500	0.010806
95th Percentile RT	MWU (probabilistic_tree vs complex_mesh)	0.000	0.000000
95th Percentile RT	MWU (parallel_fanout vs chain_with_branching)	600.000	0.026874
95th Percentile RT	MWU (parallel_fanout vs complex_mesh)	207.500	0.000337
95th Percentile RT	MWU (parallel_fanout vs hierarchical_tree)	591.000	0.037288
95th Percentile RT	MWU (chain_with_branching vs complex_mesh)	95.000	0.000000
95th Percentile RT	MWU (complex_mesh vs hierarchical_tree)	863.500	0.000000
95th Percentile RT	MWU (complex_mesh vs sequential_fanout)	860.000	0.000000
Failure Rate	KW – Failure Rate (all topologies)	109.768	0.000000
Failure Rate	MWU (probabilistic_tree vs chain_with_branching)	44.000	0.000000
Failure Rate	MWU (probabilistic_tree vs complex_mesh)	4.000	0.000000
Failure Rate	MWU (probabilistic_tree vs hierarchical_tree)	206.000	0.000318
Failure Rate	MWU (parallel_fanout vs chain_with_branching)	150.000	0.000009
Failure Rate	MWU (parallel_fanout vs complex_mesh)	46.000	0.000000
Failure Rate	MWU (parallel_fanout vs sequential_fanout)	608.000	0.018503
Failure Rate	MWU (chain_with_branching vs complex_mesh)	172.000	0.000041
Failure Rate	MWU (chain_with_branching vs hierarchical_tree)	726.000	0.000046
Failure Rate	MWU (chain_with_branching vs sequential_fanout)	899.000	0.000000
Failure Rate	MWU (complex_mesh vs hierarchical_tree)	844.000	0.000000
Failure Rate	MWU (complex_mesh vs sequential_fanout)	900.000	0.000000
Failure Rate	MWU (hierarchical_tree vs sequential_fanout)	792.000	0.000000
CPU Usage	KW – CPU Usage (all topologies)	39.671	0.000000
CPU Usage	MWU (probabilistic_tree vs parallel_fanout)	291.000	0.019112
CPU Usage	MWU (probabilistic_tree vs chain_with_branching)	200.000	0.000225
CPU Usage	MWU (probabilistic_tree vs complex_mesh)	615.000	0.015014
CPU Usage	MWU (probabilistic_tree vs hierarchical_tree)	200.000	0.000225
CPU Usage	MWU (probabilistic_tree vs sequential_fanout)	200.000	0.000225
CPU Usage	MWU (parallel_fanout vs complex_mesh)	662.000	0.001767
CPU Usage	MWU (chain_with_branching vs complex_mesh)	742.000	0.000016

Continued on next page

Table 3 – continued from previous page

Metric	Test	Statistic	p_value
CPU Usage	MWU (chain_with_branching vs sequential_fanout)	600.000	0.027086
CPU Usage	MWU (complex_mesh vs hierarchical_tree)	147.000	0.000008
CPU Usage	MWU (complex_mesh vs sequential_fanout)	200.000	0.000225
Memory Usage	KW – Memory Usage (all topologies)	112.359	0.000000
Memory Usage	MWU (probabilistic_tree vs parallel_fanout)	0.000	0.000000
Memory Usage	MWU (probabilistic_tree vs chain_with_branching)	0.000	0.000000
Memory Usage	MWU (probabilistic_tree vs complex_mesh)	107.000	0.000000
Memory Usage	MWU (probabilistic_tree vs hierarchical_tree)	192.000	0.000141
Memory Usage	MWU (probabilistic_tree vs sequential_fanout)	300.000	0.027086
Memory Usage	MWU (parallel_fanout vs complex_mesh)	900.000	0.000000
Memory Usage	MWU (parallel_fanout vs hierarchical_tree)	825.000	0.000000
Memory Usage	MWU (parallel_fanout vs sequential_fanout)	834.000	0.000000
Memory Usage	MWU (chain_with_branching vs complex_mesh)	900.000	0.000000
Memory Usage	MWU (chain_with_branching vs hierarchical_tree)	800.000	0.000000
Memory Usage	MWU (chain_with_branching vs sequential_fanout)	804.000	0.000000
Memory Usage	MWU (complex_mesh vs hierarchical_tree)	304.000	0.031466
Memory Usage	MWU (hierarchical_tree vs sequential_fanout)	595.000	0.032651
Energy Consumption	KW – Energy Consumption (all topologies)	19.581	0.001497
Energy Consumption	MWU (probabilistic_tree vs parallel_fanout)	300.000	0.027086
Energy Consumption	MWU (probabilistic_tree vs chain_with_branching)	200.000	0.000225
Energy Consumption	MWU (probabilistic_tree vs hierarchical_tree)	276.000	0.010315
Energy Consumption	MWU (probabilistic_tree vs sequential_fanout)	300.000	0.027086
Energy Consumption	MWU (parallel_fanout vs chain_with_branching)	253.000	0.003671
Energy Consumption	MWU (parallel_fanout vs complex_mesh)	591.000	0.037782
Energy Consumption	MWU (chain_with_branching vs complex_mesh)	619.000	0.012732
Energy Consumption	MWU (chain_with_branching vs sequential_fanout)	599.000	0.028129
DRAM Energy Consumption	KW – DRAM Energy (all topologies)	28.670	0.000027
DRAM Energy Consumption	MWU (probabilistic_tree vs parallel_fanout)	224.000	0.000856
DRAM Energy Consumption	MWU (probabilistic_tree vs chain_with_branching)	203.000	0.000268
DRAM Energy Consumption	MWU (probabilistic_tree vs hierarchical_tree)	267.000	0.006972
DRAM Energy Consumption	MWU (probabilistic_tree vs sequential_fanout)	258.000	0.004637
DRAM Energy Consumption	MWU (parallel_fanout vs complex_mesh)	691.000	0.000377
DRAM Energy Consumption	MWU (chain_with_branching vs complex_mesh)	717.000	0.000081
DRAM Energy Consumption	MWU (complex_mesh vs hierarchical_tree)	283.000	0.013832
DRAM Energy Consumption	MWU (complex_mesh vs sequential_fanout)	254.000	0.003848