OMBench: Taming Data Management Requirements in Cloud-Native Applications

Rodrigo Laigner
University of Copenhagen
rnl@di.ku.dk

Xikun Jiang
University of Copenhagen
xikun@di.ku.dk

Boris Düdder
University of Copenhagen
boris.d@di.ku.dk

Yongluan Zhou
University of Copenhagen
zhou@di.ku.dk

ABSTRACT

Function-as-a-Service, actor runtimes, and microservice frameworks have emerged as popular platforms for scalable cloud applications. However, understanding the data management trade-offs of these solutions, not to mention their programming abstractions and architectural design, can be cumbersome for practitioners and researchers. Furthermore, existing benchmarks fail to capture the data management requirements sought after by practitioners. Our recently developed Online Marketplace Benchmark (OMBench) fills these gaps by addressing key data management challenges developers face, including distributed transaction processing, consistent data querying and replication, event processing, and data integrity constraint enforcement. We present OMBench through a decision-and-evaluation toolkit with rich visualization features. Users can navigate a set of candidate platforms to meet data management requirements and make informed decisions about their guarantees and resource implications. Users can easily use the toolkit to evaluate performance across competing candidates to pinpoint the trade-offs of cloud programming platforms.

PVLDB Reference Format:
Rodrigo Laigner, Xikun Jiang, Boris Düdder, and Yongluan Zhou. OMBench: Taming Data Management Requirements in Cloud-Native Applications. PVLDB, 19(12): 4506 - 4509, 2026.
doi:10.14778/3827998.3828051

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/diku-dk/OnlineMarketplaceBenchmark.

1 INTRODUCTION

The advent of cloud computing has led software developers to rethink the architecture of data-intensive applications for cloud-native deployment. There is an emerging interest in replacing traditional three-tier architectures with event-driven microservice architectures (EDMA), Function as a Service (FaaS), and actor-based runtimes, all of which provide highly modularized stateful components that communicate via asynchronous events. By prescribing

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828051

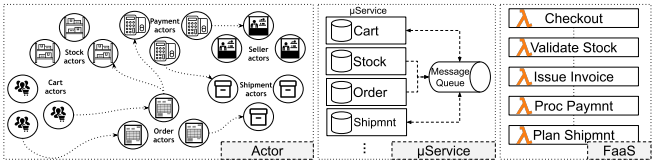


Figure 1: Cloud Application Architectures Differences

the functional decomposition of an application, these emerging paradigms enable different software teams to use specialized programming languages, frameworks, and database systems for each component, thereby better aligning technology choices with specific requirements. Moreover, the independent asynchronous components can be deployed, scaled, and evolved separately, helping to isolate failures. Overall, they facilitate the capitalization on the inherent characteristics of cloud computing.

Despite the cost savings and software engineering benefits, data management remains a significant burden in cloud applications [6]. Developers face a myriad of challenges [5], including maintaining data invariants in the presence of late-arriving and out-of-order asynchronous events, ensuring transactional guarantees across distributed and heterogeneous components, querying consistent data from multiple components, replicating data respecting ordering semantics, and ensuring event delivery guarantees.

However, data management issues in cloud applications have not been captured by existing benchmarks [6], jeopardizing the understanding of the unmet needs of cloud applications by database researchers and, consequently, hindering the advancement of data management systems to properly address these challenges.

Furthermore, state-of-the-art platforms for cloud applications, such as Microsoft Orleans [3], Dapr [1], Flink Statefun [4], and AWS Lambda [2], differ significantly, exhibiting large discrepancies in their programming models, data consistency and event processing guarantees, and architectural designs (e.g., compute/storage disaggregation). Fig. 1 highlights key differences: whereas actors co-locate state and computation, microservices and FaaS typically follow a stateless design, delegating state management and concurrency control to a database. Their distinct execution models also influence how components are modeled and parallelized.

When architecting a cloud application and choosing which platform to implement it on, users often face significant challenges assessing how these choices affect application performance and properties. Realizing this too late and choosing an unsuitable system can be detrimental to an application's success.

Our recently developed *Online Marketplace* benchmark (OMBench), published in SIGMOD [5], fills these gaps. It models the core data

management requirements in modern cloud applications, including data invariants, functional data partitioning, distributed transaction and query processing, data replication, and event processing. For direct comparison across competing systems, it prescribes criteria that a cloud platform should meet, including resource isolation, transactional and query consistency, application data integrity, and event processing order and delivery guarantees. The workload generation relies on a novel method that accounts for the decomposed and asynchronous nature of application components in the cloud. Moreover, it provides insights into four modern cloud programming platforms: Microsoft Orleans (the actor model), Apache Flink Statefun (stateful function as a service), Dapr (microservice frameworks), and a composite platform that weaves multiple data systems together, a common EDMA design in practice.

In this work, we demonstrate OMBench with OMBench Toolkit, a novel interactive decision and evaluation toolkit that allows users to easily navigate the landscape of cloud application platforms by offering: **(1) Platform Recommendation:** users express their data management requirements, which are matched against a curated set of features to convey recommended platform and related configuration & architectural design needed; **(2) Tuning and Architectural Configuration:** for each platform, users can dynamically vary data management criteria (e.g., isolation levels and event delivery guarantees) and adjust its configuration (e.g., number of worker nodes) and architectural design (e.g., compute/storage disaggregation) to assess their implications for the platform execution model and cloud computational resources; **(3) Evaluation and Comparison:** in addition to these insights, users can experimentally compare the performance impact of data management criteria across different platforms, configuration & architecture options using the implementation repositories and automatic deployment scripts provided by the toolkit, thereby pinpointing the trade-offs of cloud platforms with respect to data management requirements.

2 OMBench OVERVIEW

In this section, we summarize the OMBench specification [5].

Application Domain & Workload. OMBench models an *Online Marketplace*, a class of popular internet-scale applications that often employ cloud-native architectures, such as microservices, for loose coupling, resource and fault isolation, and high scalability [5]. Application components, each with distinct characteristics and data management tasks, form *Online Marketplace*, such as Cart, Stock, Product, Order, Payment, and Shipment. A subset of their interactions across platforms is shown in Fig. 1. Besides, business transactions and queries typical for marketplaces are modeled, such as **customer checkout** (long-running transaction involving several components and data invariants to be maintained), **price update** (which are streamed across different components for both caching and querying purposes), **product delete** (deletes existing products regardlessly whether they are currently involved in order processing), **delivery update** (batch updates several deliveries), and **seller dashboard** (streaming and pull-based complex analytical queries).

Data Management Criteria. OMBench data management criteria are summarized in Table 1. It is common to find multi-component workflows in trending cloud application architectures, such as microservices and FaaS. As cloud developers often seek to ensure the completeness of such workflows, business transactions, such as *Customer Checkout*, shall ensure all-or-nothing atomicity. Moreover, as operations span across components, it is essential to preserve each component's data invariants. Typical invariants include ensuring stock items always refer to existing products and no overselling.

Table 1: Synthesis of OMBench Data Management Criteria

Requirement	Criteria	Instances
Functional Decomposition	Isolation of Resources	Compute and storage level
Event Processing	Processing Order	Order and payment events must precede shipment events
	Delivery Guarantees	At-most-once, at-least-once, and exactly-once delivery
Distributed Transactions	All-or-nothing Atomicity; Isolation Levels	Linearized product updates; Serializable Update Delivery and Stock operations
Data Invariants	Data Invariant Enforcement	No duplicate payments; No overselling of items
Data Replication	Replication Consistency	Eventual; causality at the object level and across multiple objects
Query Processing	Query Consistency	Seller Dashboard; Eventual or Consistent Snapshot
Audit Logging	Durability Guarantees	Auditable customer checkout traces across every component

Multi-component interactions often rely on durably stored asynchronous events to decouple producers and consumers in time. The event-processing semantics offered by cloud programming platforms can directly affect the above correctness criteria. For example, while exactly-once processing can ensure all-or-nothing atomicity, in practice, it is challenging to ensure such semantics across components built with disparate technologies and transactional properties. Conversely, weaker delivery guarantees require implementing idempotent operations to avoid corrupting states.

Furthermore, the correctness semantics for replicating product data to the Cart rely on eventual and causal replication. For consistent data processing, the two queries that make up the seller dashboard must reflect the same snapshot of the application state. For event processing, events can be processed either in an unordered fashion or in causal order. For example, payment-related events must occur before shipment events to preserve causality.

Workload Generation and Metric Collection. The asynchronous nature of the components makes it a distinct challenge for OMBench to generate the workload and collect metrics correctly compared to other database benchmarks. For example, if a cart contains product information that significantly lags behind the latest updates, it can contain deleted products or products with outdated prices. Both cases can distort the workload distribution and, hence, lead to inaccurate benchmarking results. Besides, querying the components' states to form coherent transaction inputs can incur prohibitive overhead due to expensive remote synchronization between OMBench and the platform.

OMBench maintains a consistent mirror of the components' states, preventing remote synchronization on forming transaction inputs. As there are multiple concurrent readers and writers of the product states, the OMBench workload generator creates a new version on each update to avoid synchronization among them.

A customer session has a lifecycle that begins with adding one or more cart items and ends with a checkout request. To prevent

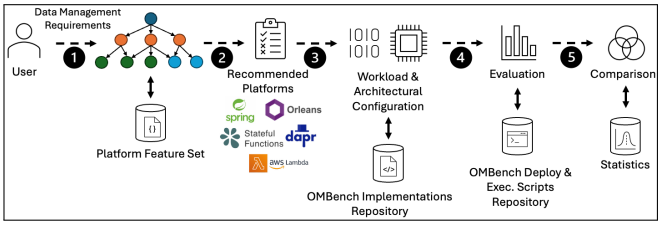


Figure 2: OMBench Toolkit Pipeline Overview

concurrent threads from modifying the same cart and affecting workload distribution, OMBench adopts the actor model, and each customer session is managed by a single actor.

Furthermore, the asynchrony among components in a multi-component transaction creates challenges in matching transaction requests to their corresponding results. OMBench assigns unique IDs to transaction inputs decoupled from their submission to avoid thread coordination. The IDs flow across components through exchanged events, enabling transparent tracing and metric collection.

3 OMBench TOOLKIT

Fig. 2 depicts the pipeline of the OMBench Toolkit, which allows for a natural progression in exploring, experimenting, and comparing cloud programming platforms. The toolkit offers a web-based GUI developed with TypeScript via the React framework and is packaged with repositories of OMBench implementations (i.e., source code and configurations for OMBench implementations across various platforms), scripts to deploy and execute them, and a knowledge dataset specifying the data management features supported by the platforms. The repositories, deployment scripts, and knowledge datasets are structured to be easily extended to cover additional platforms in the future. Given its low overhead, users can host the toolkit with the OMBench workload generation on the same host.
Steps ① & ②: Because application requirements vary widely across application domains, the OMBench toolkit provides a rule-based, explainable expert system that leverages knowledge curated from trusted sources [5] and empirical studies [6] to guide users. Users are presented with an interactive survey whose objective is to assign scores to candidate platforms while constructing a decision tree to convey the recommended platform and related configuration & architecture needed. We explore these further in scenario § 4.1.
Step ③: For each recommended platform, users can dynamically vary data management requirements and criteria (e.g., isolation levels and event delivery guarantees) and modify its configuration (e.g., the number of worker nodes) and architectural aspects (e.g., compute/storage disaggregation) through a visualization tool to assess their implications in the execution model and cloud resources.

For example, achieving stronger query consistency often requires specialized query engines, since cloud programming platforms typically lack rich query-processing features. The visualization helps users understand how data, often encapsulated through programming abstractions (e.g., actor private states), must flow into the query engine to achieve the above. This often adds overhead to the transaction path, and users must make a trade-off between performance and consistency. This is explored in scenario § 4.2.

Step ④ Based on the information gained through the steps above, users can then compare the performance impact of data management criteria across different configuration & architecture options. The toolkit executes the deployment scripts and integrates with the OMBench workload generator to populate data and generate coherent inputs (§ 2).
Step ⑤ The collected statistics of the experimental results are visualized through a benchmark dashboard and stored for historical access, enabling comparisons across competing platforms with various configurations and pinpointing the trade-offs of state-of-the-art cloud platforms with respect to data management requirements. We explore these in scenario § 4.3.

4 DEMONSTRATION SCENARIOS

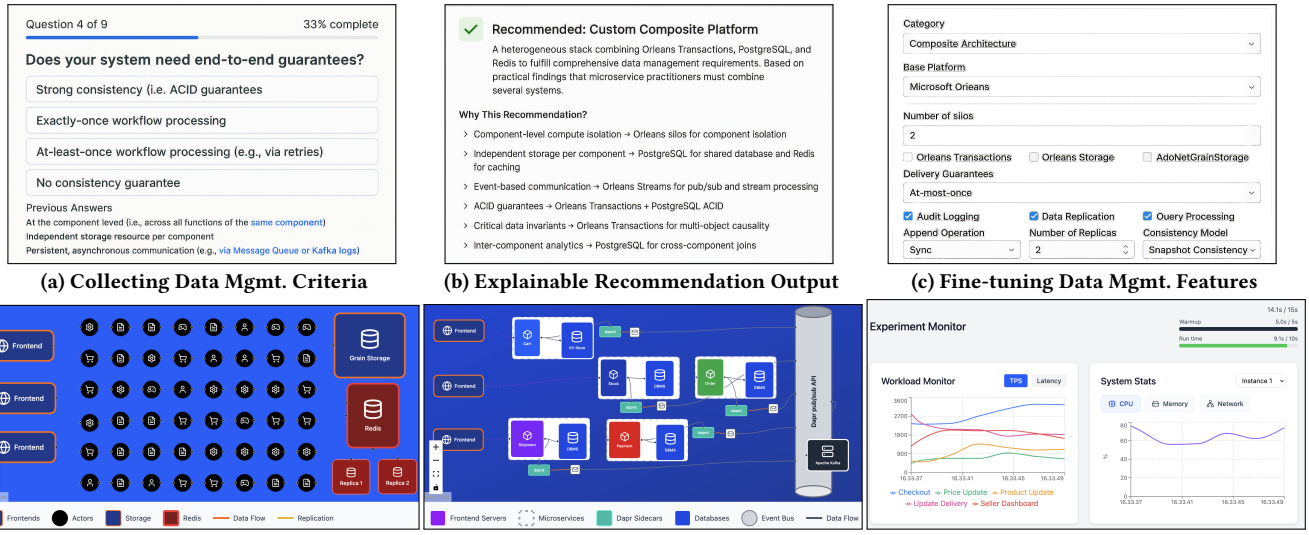
4.1 Platform Recommendation

This scenario explores how users can begin navigating the landscape of scalable cloud applications. Attendees will interactively express their application data management requirements (e.g., data invariants and transaction isolation), which will be matched against a curated set of features offered by state-of-the-art cloud platforms. In line with the data management criteria (Table 1), Fig. 3a presents a subset of these questions. From compute resource isolation, data processing and consistency properties, to the required audit logging durability guarantees, users are equipped with an effective framework to align platforms with their data management requirements.

At the end, a recommended platform is presented, along with its corresponding decision tree and the scores for each competing platform, aiding explainability (Fig. 3b). Note that **an unfit criterion does not necessarily prune a platform, as the assigned scores are balanced to determine the final recommendation**. As shown in Fig. 3b, albeit jeopardizing component-level resource isolation, the user's advanced query processing and replication requirements lead to a composite platform, a popular approach in practice, backed by Orleans for ACID transactions, PostgreSQL for advanced query capabilities, and Redis for multi-object causal replication. Users are encouraged to explore a broad range of scenarios and to learn which platforms are suitable for each.

4.2 Workload & Architectural Configuration

From the recommendation, users can quickly switch to the experimental configuration, where they can fine-tune workload and resource parameters, such as the scale factor, transaction ratio, and system model, and visualize the platform architecture across a cloud provider's resources. Fig. 3c shows a typical configuration space for defining diverse data management features, including transactional and event-delivery guarantees, data-replication and query-processing semantics, and audit logging. Parameters related to cloud resources and data management criteria may alter the application topology, which users can assess through interactive visualization. Fig. 3d shows the visualization panel of the aforementioned composite platform, composed of stateless proxies, two Orleans silos, a Redis cluster, and PostgreSQL, which helps attendees understand the resource and synchronization overheads of meeting complex data management requirements on cloud platforms. Attendees can explore competing platforms and gain insights into architectural and programming model trade-offs. Fig. 3e, for



(a) Collecting Data Mgmt. Criteria (b) Explainable Recommendation Output (c) Fine-tuning Data Mgmt. Features

(d) Actor-based Platform Dataflow & Blueprint (e) µService Platform Dataflow & Blueprint (f) Benchmark Monitor Dashboard

Figure 3: OMBench Toolkit GUI - Scenario I (a & b), Scenario II (c, d, & e), and Scenario III (f)

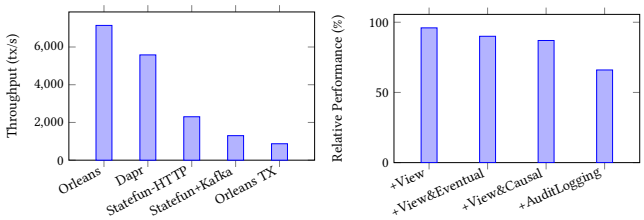


Figure 4: Evaluation across platforms and data mgmt. criteria

example, shows how a traditional microservices deployment differs drastically from an actor runtime in the cloud. Users will be invited to experiment with different data management features and criteria to understand their implications on the dataflow and architectural blueprint of each platform.

4.3 Evaluation & Comparison

In this scenario, we demonstrate how OMBench can pinpoint the impact of the programming abstractions and architectural designs offered by state-of-the-art cloud programming platforms on performance. In particular, we experiment with three popular consistency models found in multi-component workflows: eventual consistency, all-or-nothing atomicity, and transactional consistency. Fig. 4 (left) shows how performance varies across platforms, with eventual consistency (Orleans/Dapr) outperforming stronger models, such as Statefun (exactly-once processing) and Orleans Transactions (TX).

We also discuss the implications of data partitioning and scheduling strategies, the architecture, and the programming model of the underlying cloud platforms for performance. In this context, while the performance of actor-based systems is highly sensitive to how data items are mapped to fine-grained actors and to actor scheduling strategies that circumvent the single-thread design of the actor model, microservice frameworks tend to incur higher network and transaction overheads due to sidecar designs and concurrent state accesses by threads, respectively (Figures 3d and 3e, respectively).

Next, we demonstrate how OMBench pinpoints the overhead of two popular use cases in cloud application architectures: data replication and continuous queries. While cloud programming platforms currently do not support these natively, we leverage the aforementioned composite solution to analyze the effects of different correctness criteria. Fig. 4 (right) exhibits the relative performance difference of each use case. We discuss the trade-offs of eventual and causal replication and query consistency models on performance, as well as the overhead of audit logging.

Attendees will be able to explore the evaluation features of OMBench through a benchmark dashboard, as shown in Fig. 3f.

5 CONCLUSION

This work demonstrates OMBench through OMBench Toolkit, a novel interactive decision-and-evaluation toolkit for data management in cloud applications. It assists users in navigating cloud application platforms by providing recommendations based on data management needs, enabling configuration and architectural design tuning, and facilitating performance comparisons across platforms. OMBench and the toolkit are open-source and can be extended to new cloud platform implementations, and provide insights for devising novel data systems for cloud applications.

REFERENCES

[1] Dapr Authors. 2026. *Dapr.* Retrieved March 24, 2026 from https://dapr.io/

[2] Amazon AWS. 2026. *Lambda.* Retrieved March 24, 2026 from https://aws.amazon.com/lambda

[3] Tamer Eldeeb, Sebastian Burckhardt, Reuben Bond, Asaf Cidon, Junfeng Yang, and Philip A. Bernstein. 2024. Cloud Actor-Oriented Database Transactions in Orleans. *Proc. VLDB Endow.* 17, 12 (2024), 3720–3730. https://doi.org/10.14778/3685800.3685801

[4] Apache Flink. 2023. *Stateful Functions.* Retrieved April 27, 2026 from https://nightlies.apache.org/flink/flink-statefun-docs-master

[5] Rodrigo Laigner, Zhexiang Zhang, Yijian Liu, Leonardo Freitas Gomes, and Yongluan Zhou. 2025. Online Marketplace: A Benchmark for Data Management in Microservices. *Proc. ACM Manag. Data* 3, 1, Article 3 (Feb. 2025), 26 pages. https://doi.org/10.1145/3709653

[6] Rodrigo Laigner, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. 2021. Data management in microservices: state of the practice, challenges, and research directions. *Proc. VLDB Endow.* 14, 13 (Sept. 2021), 3348–3361. https://doi.org/10.14778/3484224.3484232