Benchmarks

UNIVERSITY OF
COPENHAGEN

TU Delft

Why benchmarking?

Stress aspects of a system given a workload definition

Derive insights about [the (in)ability of] a system to

- Manage locks (e.g., in a contention workload)
- Partition and distribute data (e.g., with skewed record accesses)
- (Over)Usage of computational resources
- Stalls (coming from instruction misses)

Provide a clear comparison point for different systems

Why benchmarking transactional cloud applications?

Cloud programming systems exhibit large discrepancies

Variety of programming models and guarantees

Challenges on deciding for an ideal model

Entities, objects, workflows, durable, stateful, stateless, reliable, virtual, actors, services, and so on...

Source: shutterstock

Why benchmarking transactional cloud applications?

Data partitioning, access, and storage

Concurrent application logic execution

Fault handling

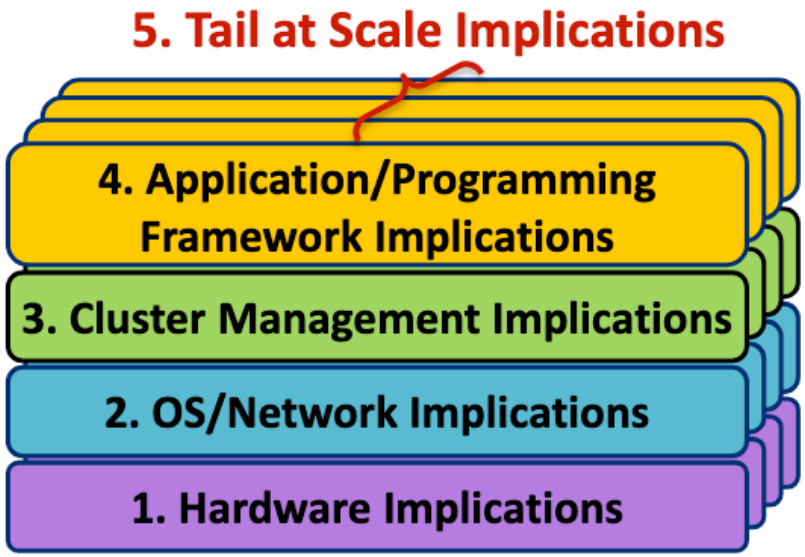
Upgrade support

Debugging

Guarantees during crashes and network partitions

Source: shutterstock

State-of-the-art microservice benchmarks



DeathStar benchmark

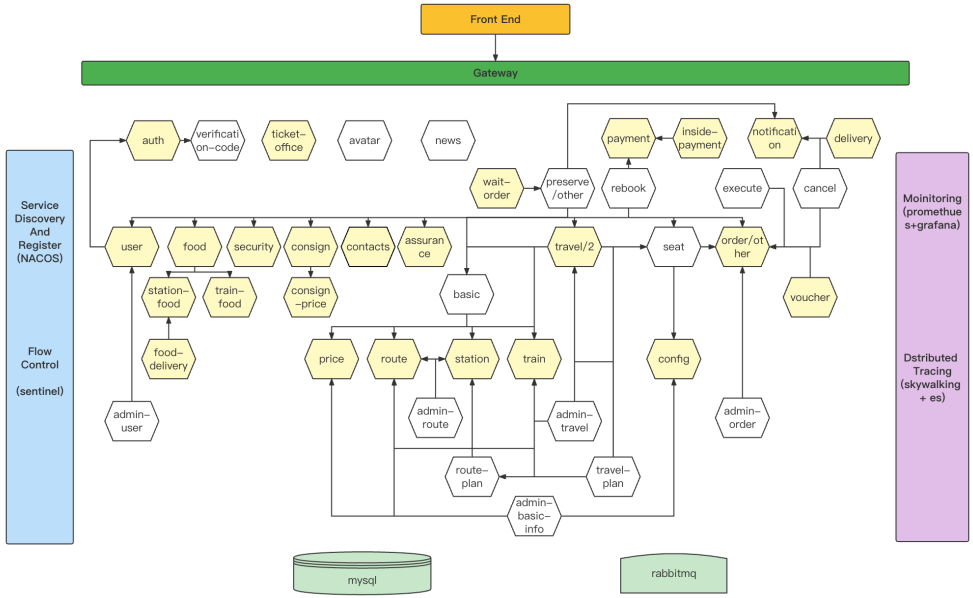
A suite of microservice applications

Effects of microservices on hardware and software in the system stacks

Focus on hardware, network, and operating systems

Y. Gan et al. “,An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems", ASPLOS 2019

State-of-the-art microservice benchmarks



- TrainTicket benchmark
- A complex and heterogenous microservice archietcture
- Focus on software engineering concerns
- Replicating industrial faults in microservices
- Support investigation of fault analysis and debugging

Xiang Zhou et al. ",Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study." IEEE Transactions on Software Engineering. 2018

What benchmarks are being used?

Cloudburst (VLDB 2020)

- Composition of functions
- machine learning prediction
- Retwis, a simple twitter clone

Boki (SOSP 2021)

- DeathStar benchmark
  - Movie reviews
  - Travel reservations
- Retwis

Snapper (SIGMOD 2022)

- Smallbank Benchmark
- TPC-C

Netherite (VLDB 2022)

- “hello world,” 5-task workflow
- Transfer across bank accounts
- MapReduce-style WordCount
- CollisionSearch

What benchmarks are being used?

SmSa (SoCC 2024)
- TPC-C
- YCSB
- Online Marketplace
- (more to come)

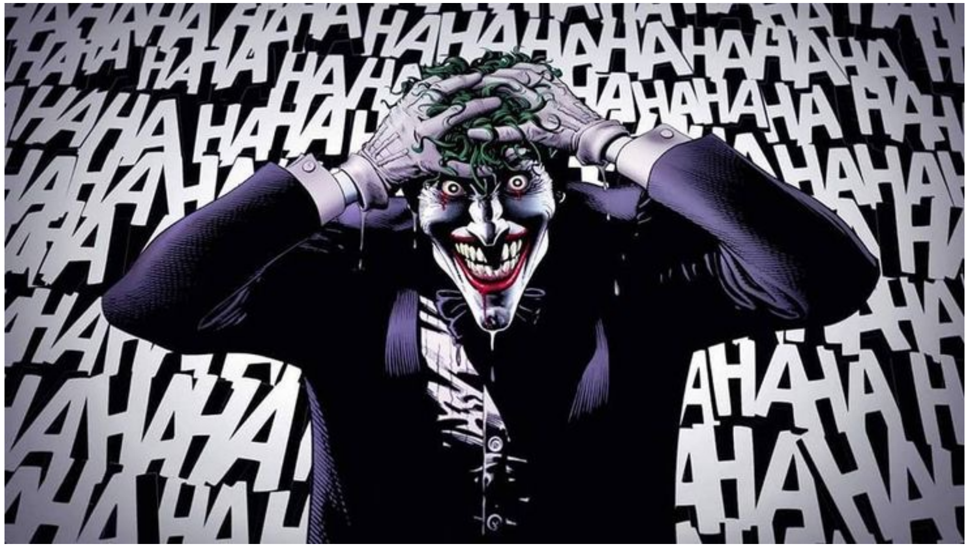
Styx (SIGMOD 2025)
- TPC-C
- YCSB
- DeathStar (microservice benchmark)

Histrio (DEBS 2025)
- banking application
- hotel reservation application

Portals (2023)
- Savina (actor benchmark)
- NexMark (stream processing)

Are we faithfully benchmarking transactional cloud applications?

- Benchmarking is conducted in an ad-hoc fashion
- Disparate benchmarks used
- Missing key aspects of data management in the cloud
- Disparate design and system models
- Challenging direct comparison across systems



Source: DC

Refreshing real-world data management challenges

Netflix (Sarma, 2020): “Rethinking service interactions as streams of **event exchanges**—**as opposed to** a sequence of **synchronous requests**”

Wix (Silnitsky, 2022): “[..] developers noticed that the inventory service **database does not reflect** the actual **inventory levels** of some products. Was the *OrderCreated* or *PaymentCompleted* **event produced correctly**?”

Nubank (Ferreira and Wible, 2017): “you have **late-arriving events** [...] we need to time travel back, **re-play the events** and figuring out the **balance that does not invalidate the invariants** we have, that is fixing invariants”

B2W (Mayerhofer, 2018): “[...] in a microservice architecture there are multiple services, each one of them handling a single concern. [...] Often a **microservice requires data from another microservice to serve its request**.”

Crowdtap (Viennot et. al, 2015): “the **same data**, needed by different (micro)services, **must be replicated** across their respective DBs.”

Towards a benchmark for transactional cloud applications

Requirement	Criteria
Functional Decomposition	Isolation of Resources
Event Processing	Event Processing Order and Delivery Guarantees
Distributed Transactions	All-or-nothing Atomicity; Isolation Levels
Data Invariants	Data Invariant Enforcement
Data Replication	Data Caching and Replication Consistency
Query Processing	Query Processing Consistency

Towards a benchmark for transactional cloud applications

Requirement	Criteria	DeathStar	TrainTicket	?
Functional Decomposition	Isolation of Resources	Yes	Yes	Yes
Event Processing	Event Processing Order and Delivery Guarantees	No	No	Yes
		No	No	Yes
Distributed Transactions	All-or-nothing Atomicity; Isolation Levels	No	No	Yes
Data Invariants	Data Invariant Enforcement	No	No	Yes
Data Replication	Data Caching and Replication Consistency	No	No	Yes
Query Processing	Query Processing Consistency	No	No	Yes

Online Marketplace: A benchmark for data management in microservices

Home > ACM Journals > Proceedings of the ACM on Management of Data > Vol. 3, No. 1 > Online Marketplace: A Benchmark for Data Management in Microservices

RESEARCH-ARTICLE | OPEN ACCESS

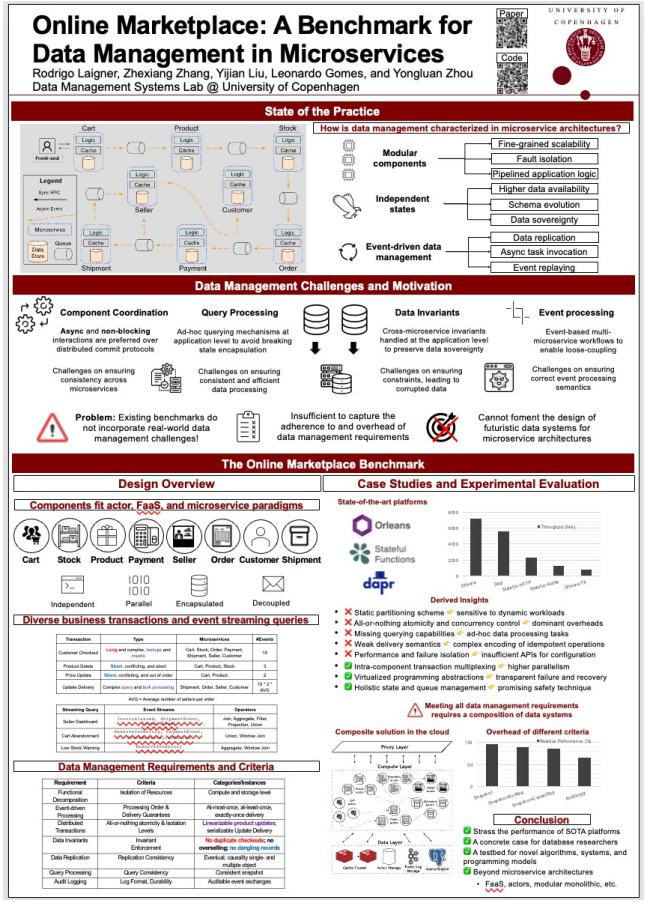
Online Marketplace: A Benchmark for Data Management in Microservices

Authors: Rodrigo Laigner, Zhexiang Zhang, Yijian Liu, Leonardo Freitas Gomes, Yongluan Zhou | Authors Info & Claims

Proceedings of the ACM on Management of Data, Volume 3, Issue 1 • Article No.: 3, Pages 1 - 26
https://doi.org/10.1145/3709653

Published: 11 February 2025 Publication History

2 345 PDF eReader



Online Marketplace design goals

Coverage

- Fully incorporate data management challenges

Industry strength

- Domain that widely adopts microservices

Low adoption barrier

- Ease understanding by non-expert experts

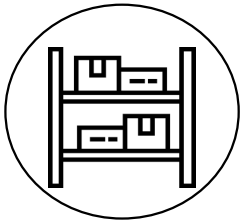
Generalizability

- Can be applied to multiple, different use-cases

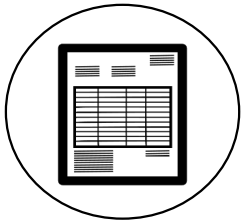
Online Marketplace application template

Cart

Product

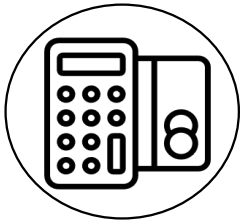


Stock

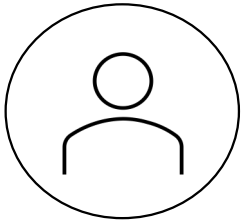


Order

Seller

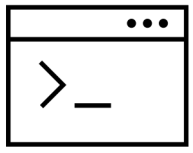


Payment



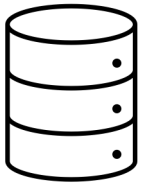
Customer

Shipment



Independent

Pipelined



Encapsuled

Decoupled

Online Marketplace workload

Transaction	Type	Microservices	#Events
Customer Checkout	**Long** and complex, lookups and inserts	Cart, Stock, Order, Payment, Shipment, Seller, Customer	10
Product Delete	**Short** and conflicting	Cart, Product, Stock	3
Price Update	**Short**, conflicting, and out of order	Cart, Product	2
Update Delivery	Complex query and bulk processing	Shipment, Order, Seller, Customer	10 * 2 * AVG

AVG = Average number of sellers per order

Streaming Query	Event Streams	Operators
Seller Dashboard	`InvoiceIssued, ShipmentEvent, PaymentEvent`	Join, Aggregate, Filter, Projection, Union
Cart Abandonment	`ReserveInventory, PaymentEvent, StockReservationFailed`	Union, Window Join
Low Stock Warning	`ReserveInventory`	Aggregate, Window Join

Online Marketplace synthesis

Requirement	Criteria	Categories/Instances
Functional Decomposition	Isolation of Resources	Compute and storage level
Event-driven Processing	Processing Order & Delivery Guarantees	At-most-once, at-least-once, exactly-once delivery
Distributed Transactions	All-or-nothing atomicity & Isolation Levels	**Linearizable product updates**; serializable Update Delivery
Data Invariants	Invariant Enforcement	**No duplicate checkouts**; no overselling; **no dangling records**
Data Replication	Replication Consistency	Eventual; causality single- and multiple object
Query Processing	Query Consistency	Consistent snapshot
Audit Logging	Log Format, Durability	Auditable event exchanges

Online Marketplace artifacts

Benchmark Driver

Management of the life-cycle of an experiment

Data generation, ingestion, and workload submission

Statistics collection and performance metrics report

Workload correctness

- Dynamic-and-decentralized states
- Asynchronous systems

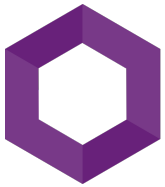
Highly-configurable and extensible APIs

Online Marketplace artifacts

Platform implementations

- Microsoft Orleans
- Apache Flink Statefun
- Microsoft Dapr
- Spring
- vMODB
- Styx?

Online Marketplace cases studies & evaluation



Component	Key	Design Rational
Cart	`customer_id`	Customer's cart management
Customer	`customer_id`	Customer data and stats processing
Product	`seller_id`, `product_id`	A product's state and operations
Stock	`seller_id`, `product_id`	A stock item's state and operations
Order	`customer_id`	Unit of order processing for a customer
Payment	`customer_id`	Unit of payment processing for a customer
Shipment	*h*(`customer_id`)	Unit of shipment processing for a disjoint set of customers
Seller	`seller_id`	Seller data and stats processing

Online Marketplace cases studies & evaluation

Workload sensitiveness to program design

Performance and failure isolation

Weak delivery -> idempotent operations

Query processing features

Opaque data models; data invariants

Native synchronization primitives

Deployment and event processing model

Online Marketplace cases studies & evaluation

Virtualized abstractions

Holistic management of state and queue operations

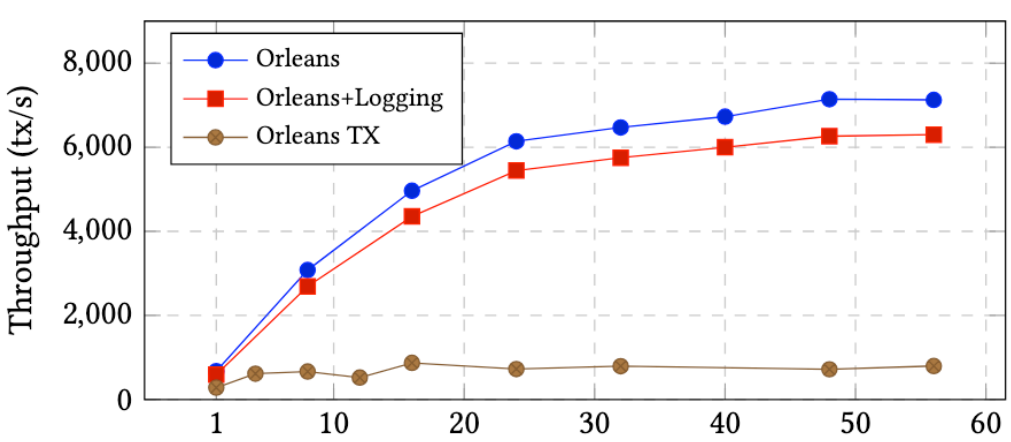
Performance techniques

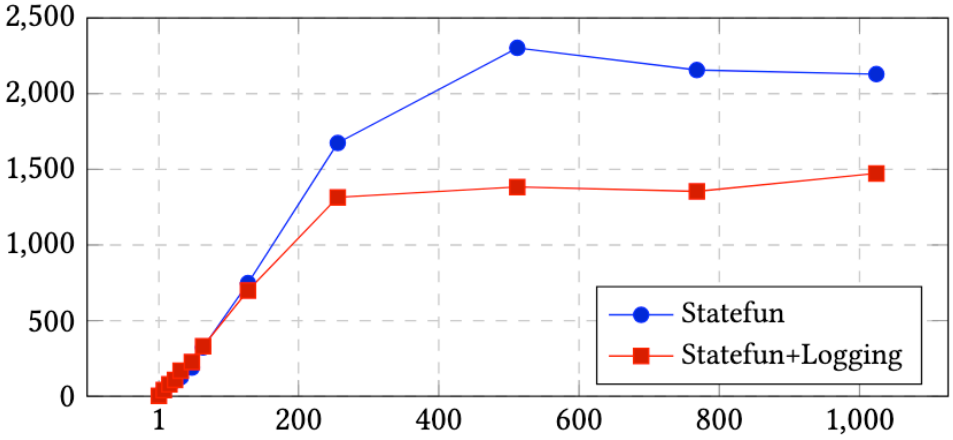
Multiplexing non-conflicting transactions

Batch-oriented execution in skewed workloads

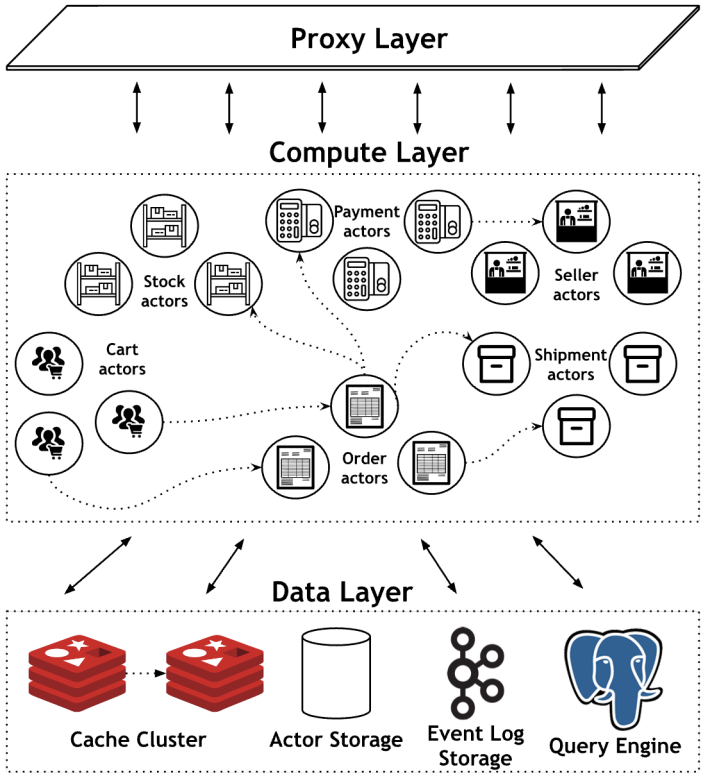
Composability -> careful co-design

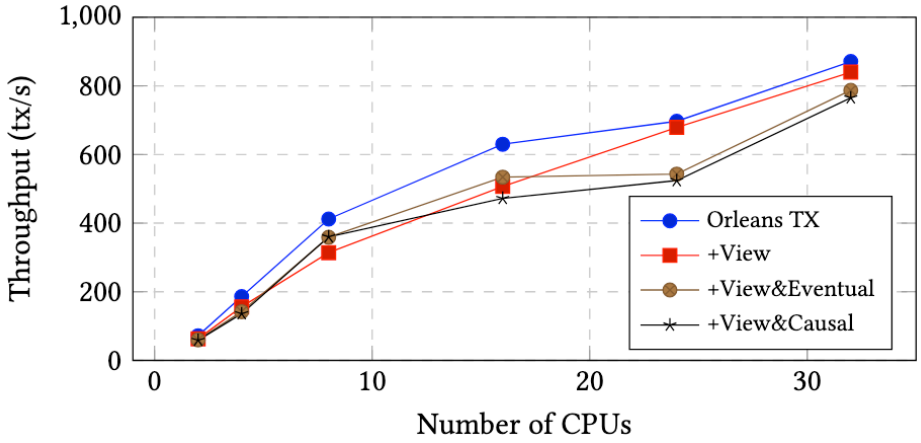
Online Marketplace cases studies & evaluation





Online Marketplace cases studies & evaluation





Online Marketplace cases studies & evaluation

Stress the performance of state-of-the-art platforms

- Reveal performance and functionality issues

A concrete example for database researchers

- A testbed for novel algorithms, systems, and programming models

Trending cloud application architectures

- Beyond microservice architectures
- FaaS, actors, modular monolithic, etc.


RESEARCH-ARTICLE | OPEN ACCESS

Rethinking State Management in Actor Systems for Cloud-Native Applications

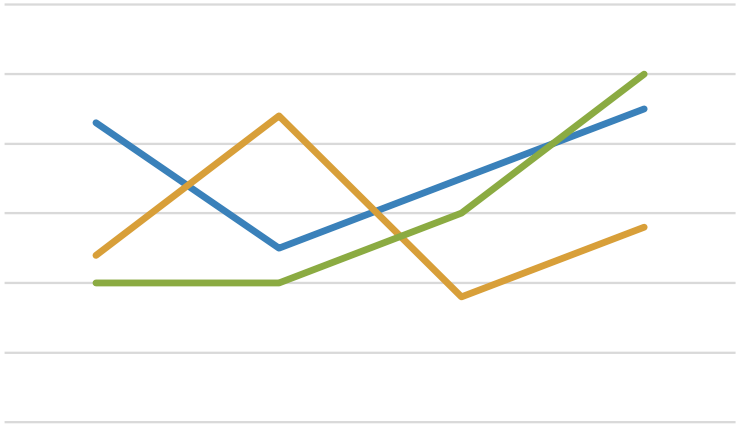
Authors: Yijian Liu, Rodrigo Laigner, Yongluan Zhou

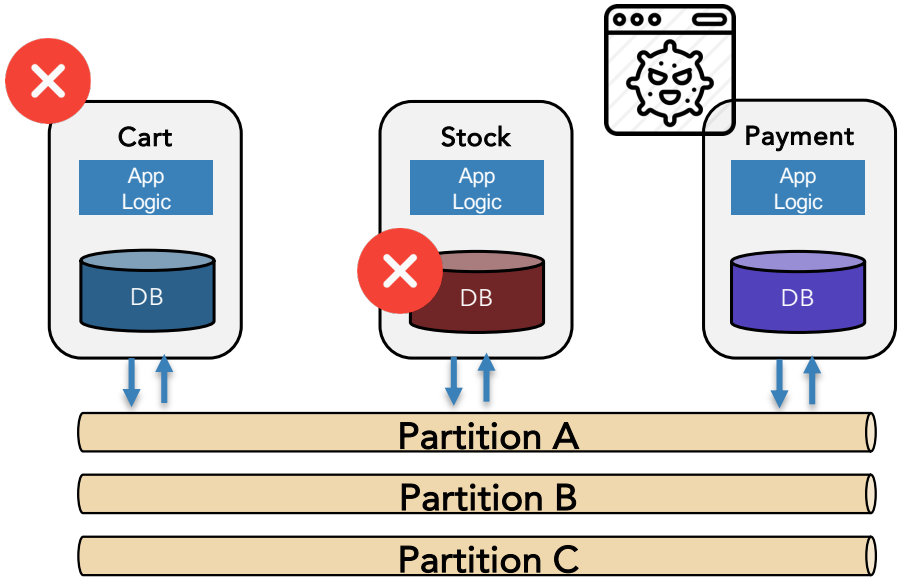
Claims

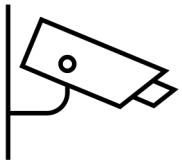
SoCC '24: Proceedings of the 2024 ACM Symposium on Cloud Computi

Pages 898 - 914 • https://doi.org/10.1145/3698038.3698540

Are we done?




Cart
App Logic
DB
Stock
App Logic
DB
Payment
App Logic
DB
Partition A
Partition B
Partition C



Benchmarks