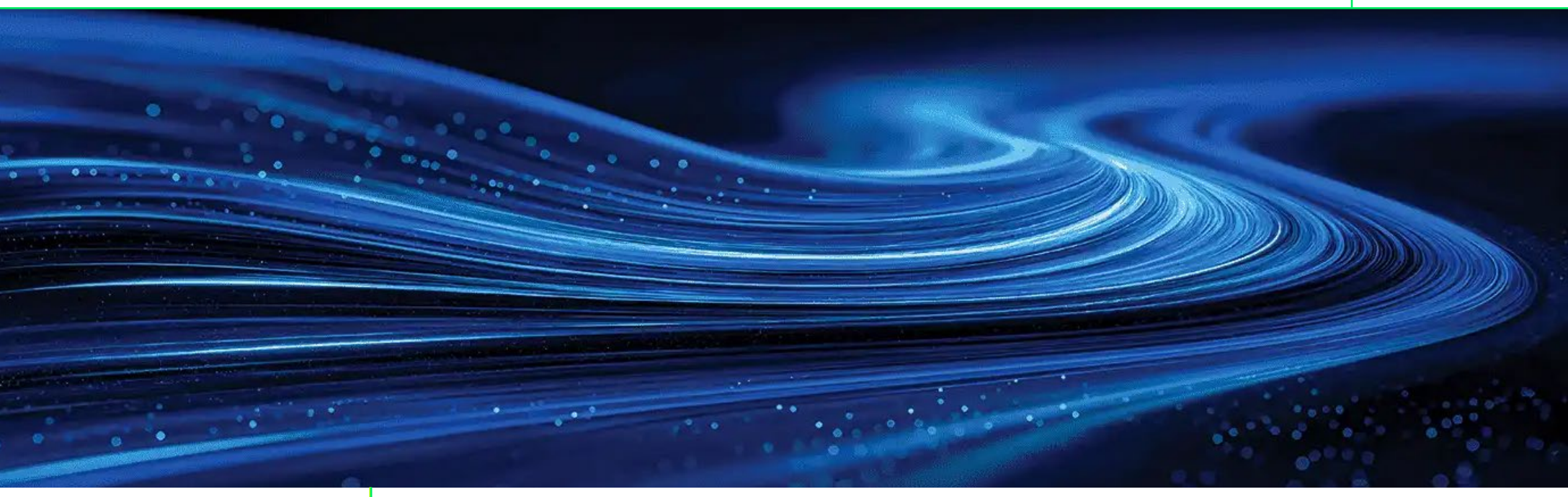
icon solutions × MongoDB

Scale Without Sacrifice:

How Icon Solutions and MongoDB Power Always-On Payments

Instant Payments Benchmark 2026 Report

Table of Contents



Scale Without Sacrifice

The account-to-account payments ecosystem is undergoing a rapid shift. Banks are increasingly operating in a 24x7, always-on world, where customers expect payments to complete immediately, not just domestically but internationally as well. Payment processing remains one of banking's most critical functions and must meet the highest standards of resilience and availability.

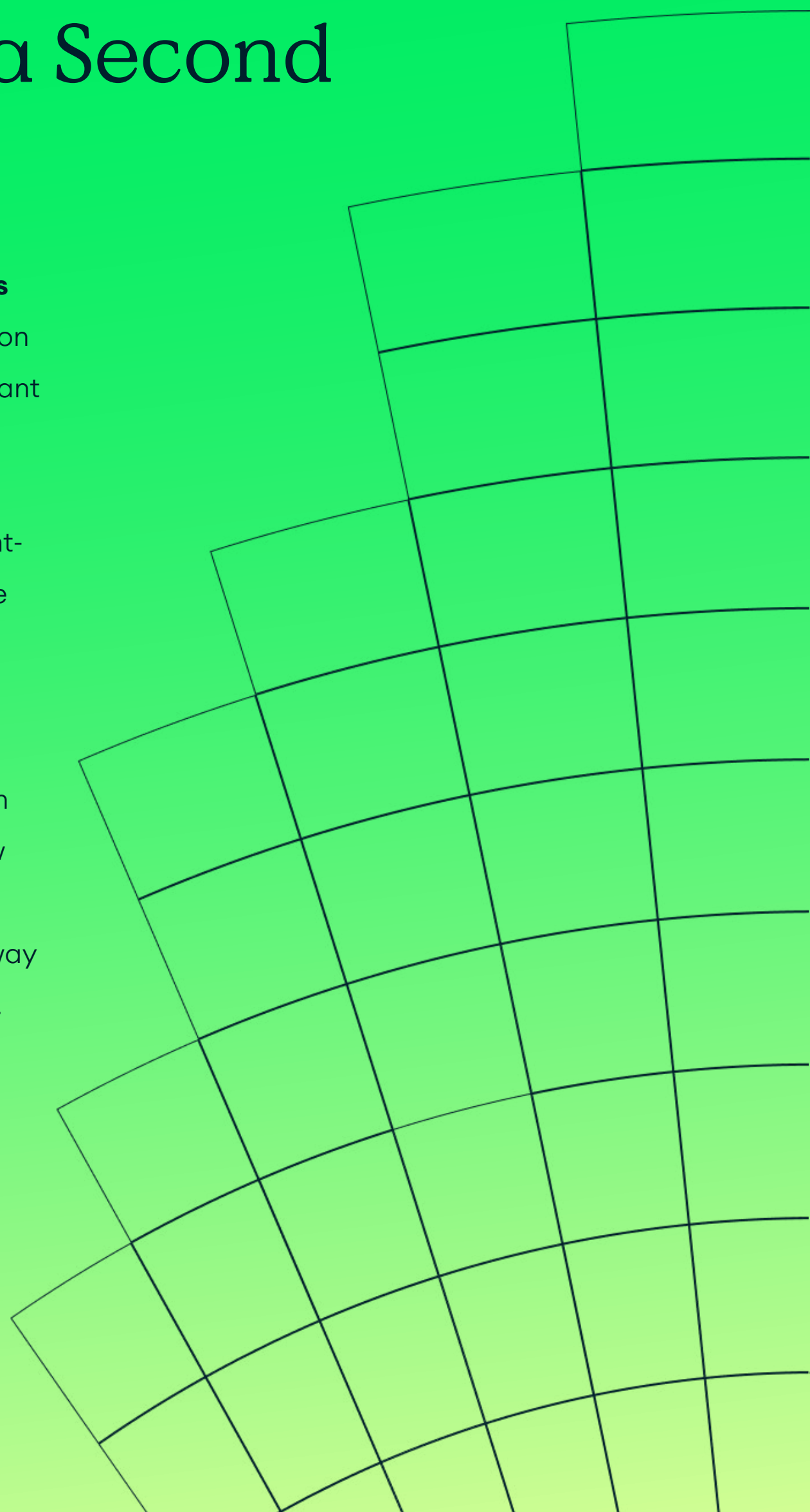
In this new ecosystem, account-to-account payment volumes, specifically instant payment volumes, are continuing to increase significantly. Instant payments are increasing because of movement within account-to-account payments, for example migration of traffic from net settlement schemes to instant schemes and direct debit volumes expected to partially move to request-to-pay with settlement done through instant schemes.

As this shift continues to materialise, there will be significant peaks in payment traffic that banks need to respond to, where the proportion of payment traffic will increasingly be bound to strict SLAs of several seconds. That means banks need payment systems and databases that are able to deal with the throughput, the peaks, the processing SLAs and importantly remain ultra-resilient.

In response, Icon Solutions and MongoDB have worked together on a benchmark designed to evaluate the scalability, latency, and resilience needed for this new payments environment.

The Proof: What Half a Million Operations a Second Looks Like

The objective of the benchmark was clear: demonstrate that IPF running on MongoDB Atlas can scale to significant throughput, remain highly resilient, and keep transaction processing within representative instant account-to-account payment scheme service levels. To do that, we ran the system through a “ladder” test: a series of stages, each raising the target payment volume a notch higher than the last, so we could see exactly how throughput, latency, and database load behaved at every level on the way up rather than only at the final peak.



The solution processed 6,000 payment transactions per second,

that's 6,000 end-to-end payments per second, with each payment including 12 individual steps such as validations and calls to simulators representing bank systems like the core banking system for booking requests.

Latency stayed well within instant payment scheme service levels.

Persistence latency held under 30 milliseconds even at peak, and the mean end-to-end latency at 6,000 transactions per second was 0.51 seconds, a fraction of the typical 10-second window that instant payment schemes allow.

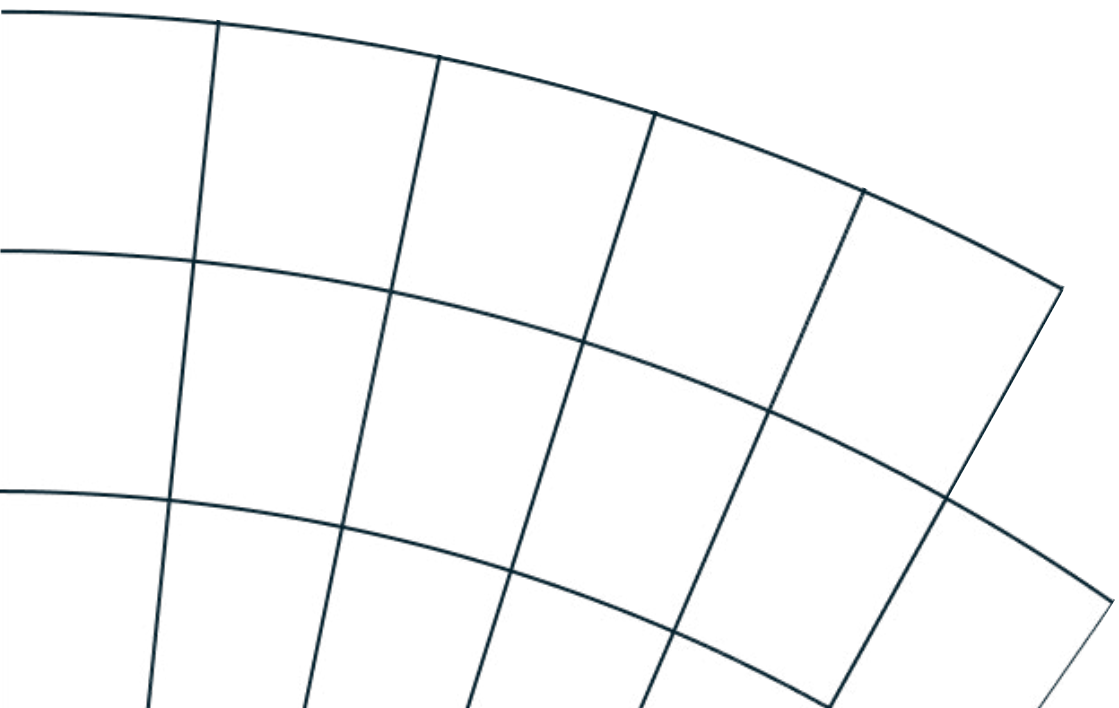
Scalability was close to linear

from the first rung of the test to the last, with meaningful headroom still in reserve at peak load.

Full recovery from database node failure on average

5 seconds

for the resiliency test, with zero data loss and minimal latency impact.



That business throughput was underpinned by about

430,000

database operations per second

across two MongoDB-backed data stores at peak: roughly 236,000 operations per second on the IPF Event Journal and roughly 196,000 on the IPF Operational Data Store (ODS).

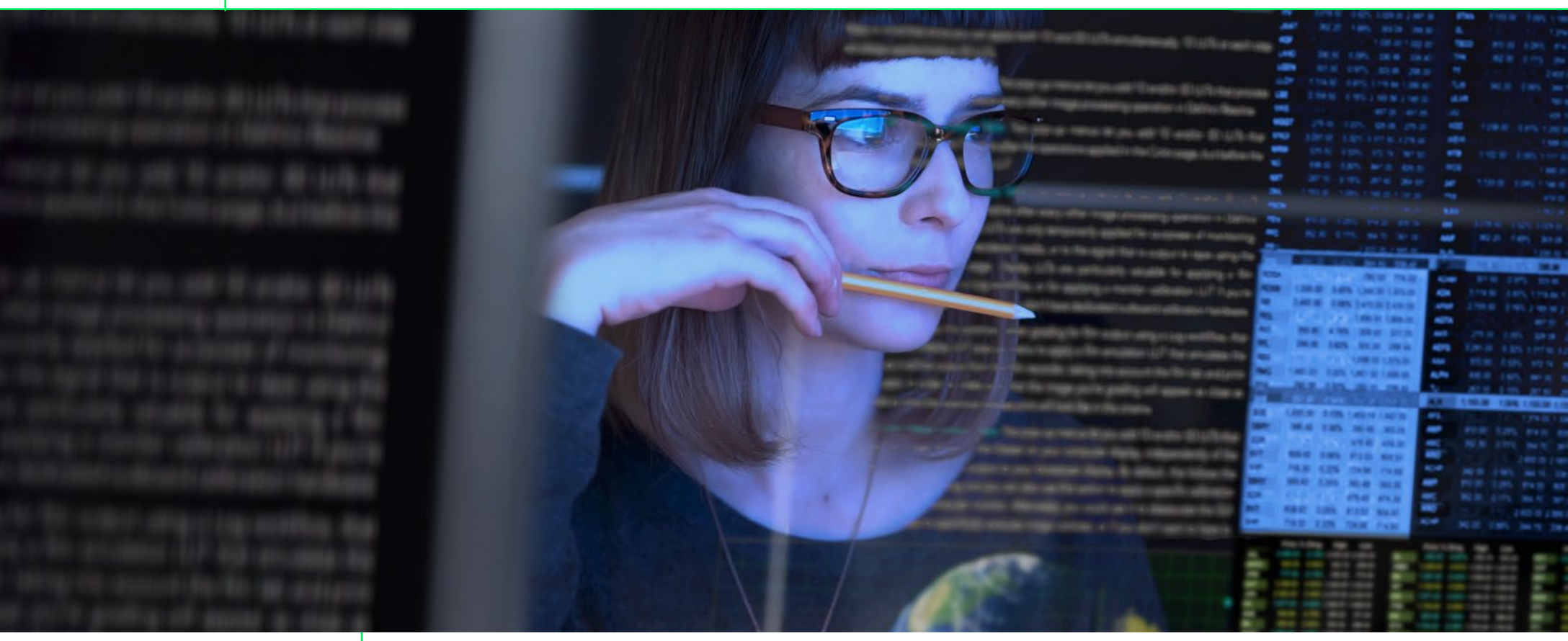
Seamless recovery from application node failure in

<90 seconds

for the resiliency test, with zero data loss and minimal latency impact.

Across the full ladder (from 500 payment TPS to 6,000 payment TPS), both the application and the database scaled in near-linear lockstep with load: throughput increased in line with each step of the test ladder, while per-shard CPU and insert rates climbed in matching, proportional increments step after step, with latency staying essentially flat until the very top of the range. That kind of predictable scaling matters as much as the headline number, it means capacity can be forecast with confidence, and growth can be planned as a function of infrastructure rather than a leap into the unknown.

THE FOLLOWING WALKS THROUGH WHAT WAS TESTED AND WHY IT MATTERS.



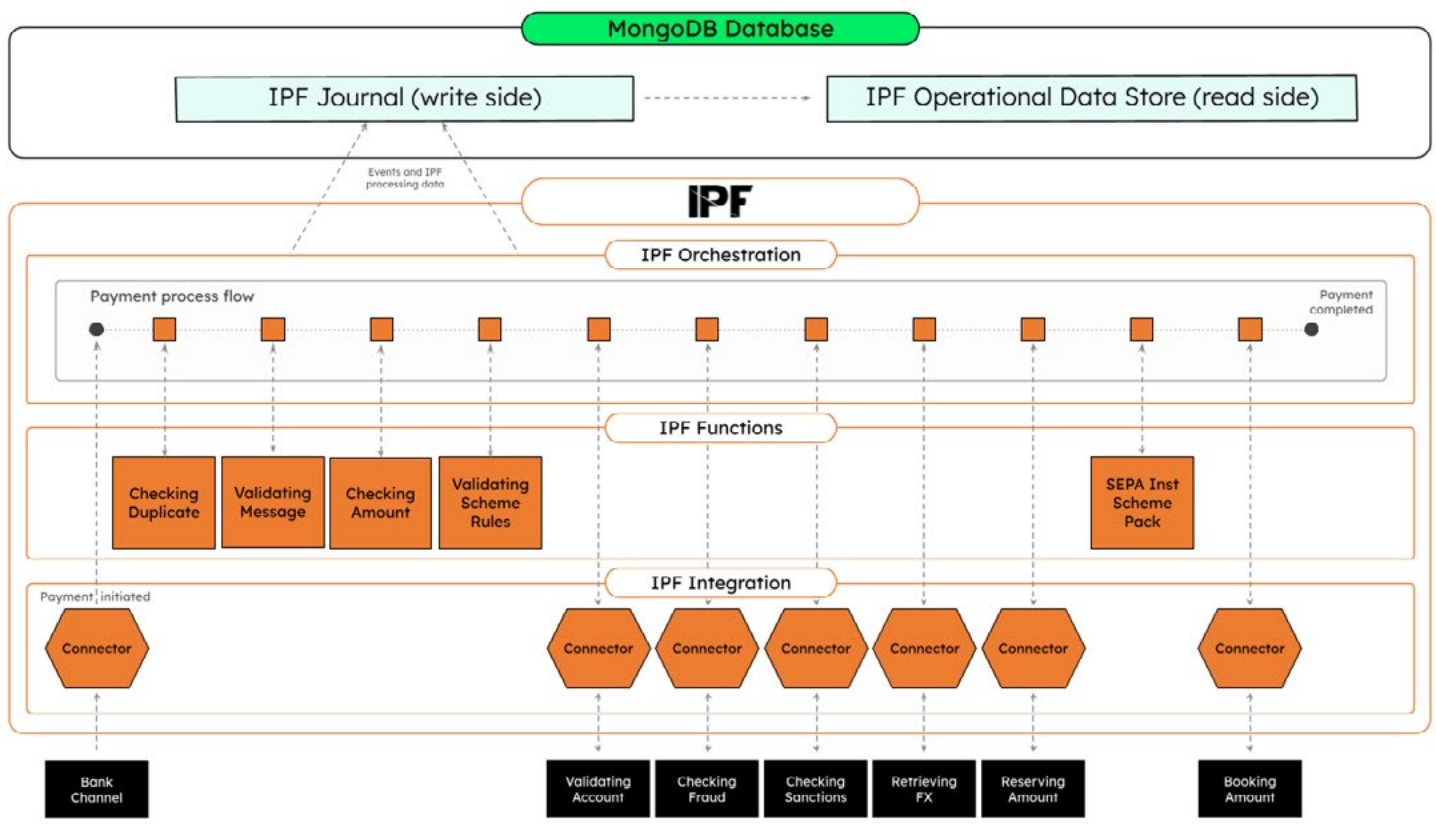
Inside the Benchmark: How We Put IPF and MongoDB to the Test

Account-to-account payment processing consists of numerous steps, including validations, checks like fraud and sanctions, accounting interactions like funds reservation and funds booking, and finally scheme processing (scheme validation and generation of the scheme messages). IPF (the Icon Payments Framework) is a framework-based payment solution, used to implement payment processing use cases across any account-to-account payment type and across the payment value chain (order management, payment execution, and clearing and settlement).

With IPF, at a high level, payment processing is a combination of payment orchestration (process flows), payment functions (e.g. duplicate check, payment routing etc.) and integration with bank systems via IPF Connectors (e.g. to Fraud, Sanctions and Accounting). Data (events and other message data) is persisted to the IPF Event Journal, backed by the MongoDB database. The processing data is streamed across to the IPF Operational Data Store, again backed by the MongoDB database.

In payment architectures based on event-driven flows like this one, the database is not just storing records after the fact, it participates directly in the system's ability to scale, recover, and operate efficiently, especially when the workload is overwhelmingly driven by inserts.

Figure 1.
Functional Coverage



The diagram represents our sample IPF solution, a reference SEPA Instant payment solution, with the outbound payment flow consisting of 12 distinct steps, also referred to as events. These are a combination of calls to IPF functions (e.g. duplicate check) and calls to external systems via IPF Connectors, for which simulators are used (e.g. the accounting simulator). So, when we talk about a payment transaction, we mean a transaction that spans the entirety of this flow and completes all 12 functional steps.

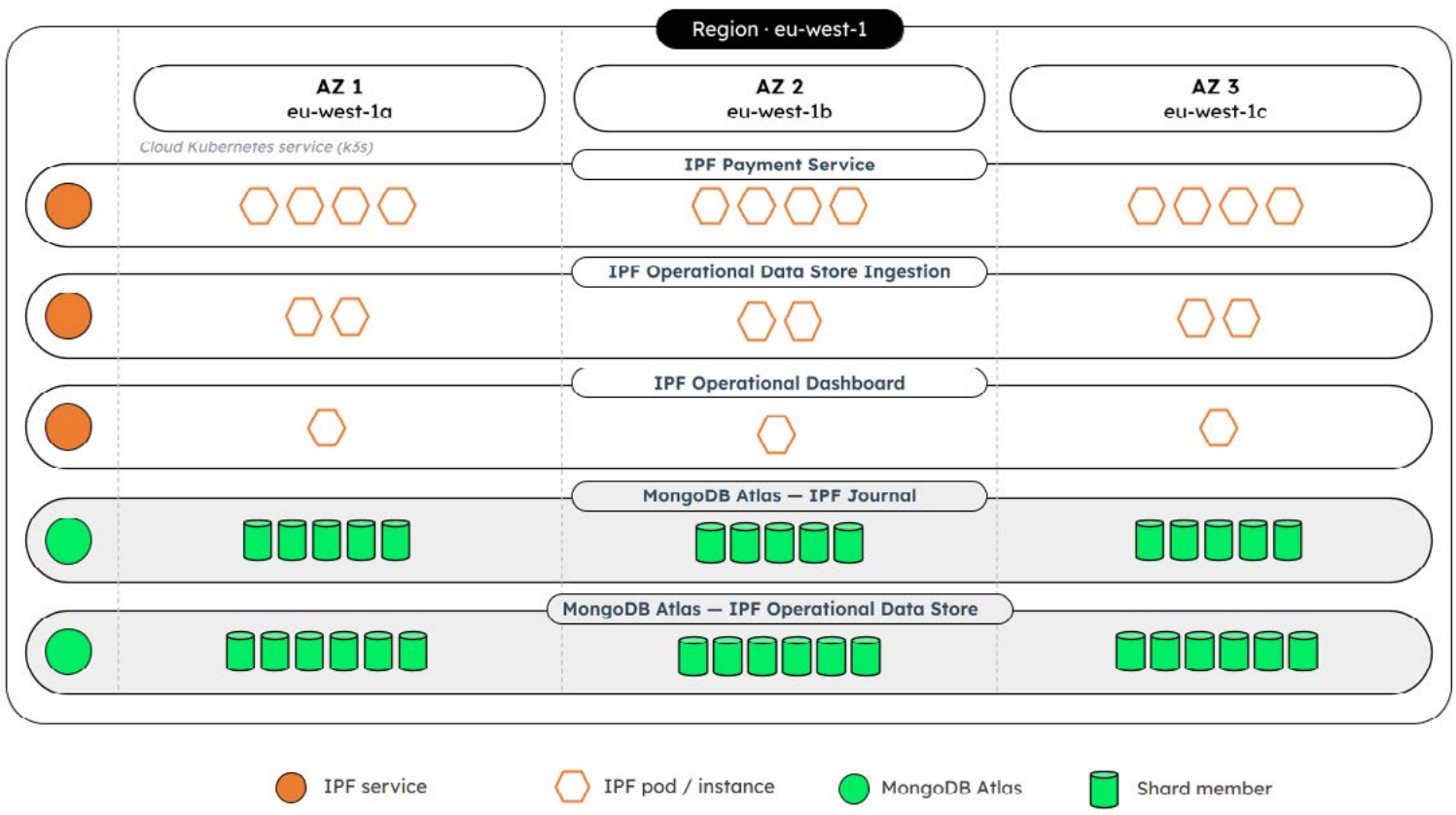


Figure 2.
Deployment setup

For the deployment, the IPF application ran across 12 application pods, alongside the Kafka service for the IPF Processing Data and the IPF Operational Data Store service. For the 6,000 TPS test, the IPF Payment Service pods were 44 vCPU each, operating at roughly 60% utiliation. The MongoDB database was configured as sharded Atlas clusters, scaled through a ladder test from 500 to 6,000 target payment transactions per second: a five-shard cluster backing the IPF Event Journal and a six-shard cluster backing the IPF Operational Data Store. Each shard was built on an M80-class tier – 32 vCPUs and 128 GB of RAM, with storage configurable from 750 GB up to 4 TB – sized for exactly this kind of high-throughput, write-intensive operational workload.

Engineered for the Extreme:

Meeting the Toughest Non-Functional Requirements

Sustaining several thousand payments per second, and the hundreds of thousands of database operations per second beneath them, while maintaining resilience, availability, and low latency is not easy. It's something that is often underestimated by banks when they do a self-build, and for traditional vendors with legacy platforms, trying to retro-architect them for always-on, highly scalable payment processing is a significant challenge.

IPF and MongoDB are modern solutions, built on modern technology and modern architectural patterns. Non-functional requirements were never an afterthought – they were baked in from the very beginning, and both Icon and MongoDB continue to invest heavily in furthering support for the highest NFRs.

Besides leaning heavily on MongoDB as the core data platform, several technologies and patterns did specific work in this benchmark:

CONTINUED ON NEXT PAGE

Technologies

- **Akka** underpins IPF's concurrent, fault-tolerant processing model – the same code that runs on a single core scaled across the 12-pod cluster used in this test.
- **The IPF Connector Framework**, built on Alpakka's back-pressured streams, kept integrations with the fraud, sanctions, and accounting systems (simulators in this test) from overwhelming downstream services as load climbed.
- **Kafka** streamed processing data from the flows to the Operational Data Store continuously throughout the run.
- **Kubernetes** orchestrated the deployment across regions and availability zones (Figure 2), providing the elastic, self-healing infrastructure layer.

Patterns

- **Event Sourcing –** instead of storing just the current state of data, the full sequence of events that led to that state is stored, so the current state can always be derived by replaying those events.
- **Command Query Responsibility Segregation (CQRS) –** efficient and non-blocking writes, efficient read-side queries with complex view models, without impacting the command-side, and independent scaling of teams and technology.

Why MongoDB is central to the benchmark

MongoDB

MongoDB is central in the modern **event-driven payments architecture of IPF**, the data layer is directly tied to whether the platform can scale predictably, recover cleanly, and keep latency within the bounds expected of instant payment schemes. In the tested solution, MongoDB backed both the IPF Event Journal and the IPF Operational Data Store, supporting **two distinct high-throughput workload profiles at the same time.**

That dual role matters. A platform built for instant processing must do more than persist transactions. It must absorb **continuous high-ingest activity**, support fast access to **richly indexed operational and BI data,** and provide a resilient foundation for event-driven processing so the data layer keeps pace with the application rather than becoming the bottleneck that limits scale.

This makes MongoDB more than background infrastructure: it was a core enabler of the benchmark outcome. The IPF Event Journal, supporting high-volume event persistence, reached about **236,000 database operations per second** at peak. The IPF Operational Data Store, built with many indexes to support BI and operational queries, still reached about **196,000 database operations per second** at the same test point–together, about **430,000 database operations per second** across the two clusters.

Another notable finding was that moving from

MongoDB Atlas 8.0 to 8.3 reduced persistence latency by about 50%.

For example, at 4,000 payment TPS, the persistence time reduced from **40 milliseconds to about 20 milliseconds.**

This observation shows how platform-level improvements can translate directly into better performance for high-volume, always-on payment workloads. It also suggests that newer MongoDB Atlas versions may continue to deliver further performance gains as the platform evolves.

Running on MongoDB Atlas 8.3, the platform handled that demanding, dual-workload profile comfortably, with clear capacity to spare. Even at the top of the test, hot-shard CPU utilisation sat at only about half, database ingest scaled smoothly step after step, the WiredTiger write queue stayed at zero, and replication lag remained low throughout.

For financial institutions evaluating data platforms for modern payments, that is a compelling proof point: the database actively powered payment modernisation goals – scale, resilience, operational query support, and simplicity – in a single managed environment, with room to grow well beyond what was tested.

Beyond 6,000 Payment TPS:

The Roadmap to Even Greater Scale

Without a doubt, there are banks and Payment Service Providers (PSPs) that need to process even greater volumes than this, and as volumes continue to increase over time, more institutions will fall into that category. The good news is that this benchmark points to a clear path to get there.

The first strategy is to look at how the solution can be broken up into different services, for example, separating bulk payment processing from individual payment processing, or separating geographically. By segregating the solution it's possible to get more throughput, since each service is capable of sustaining several thousand TPS on its own.

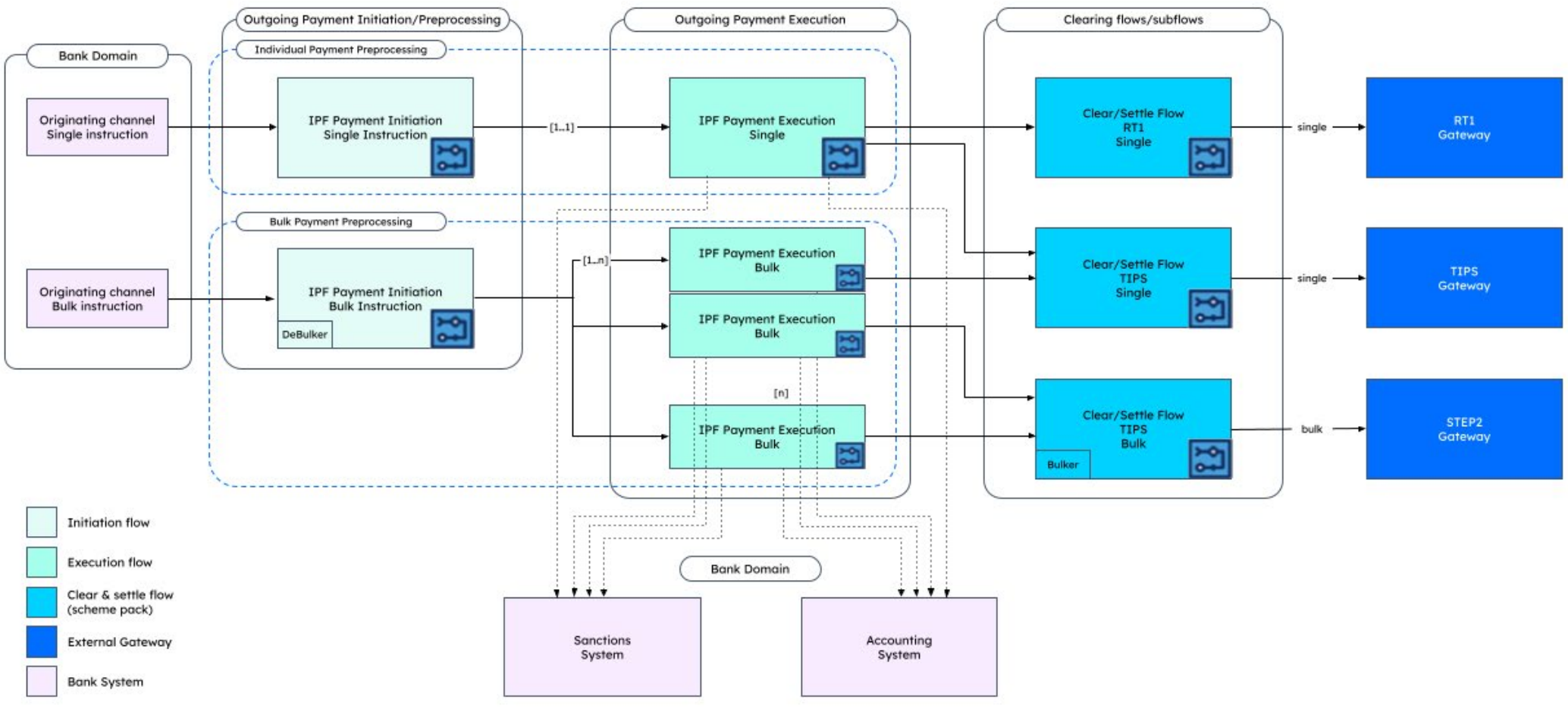
As shown in that diagram, the solution can also be segregated along the value chain (payment initiation, payment execution, and payment clearing and settlement), breaking the overall flow into multiple smaller flows to make the solution more supportable and functionally scalable.

Beyond service segregation, the benchmark reinforces that an individual service can be scaled higher still through additional sharding. The result was achieved on sharded architectures – five shards for the IPF Event Journal and six shards for the IPF Operational Data Store – not by concentrating the workload on a single oversized machine – a practical and attractive scaling model, since institutions don't want future growth to depend entirely on moving to ever-larger machines, and there is also a practical limit to how far any single machine can scale, both technically and economically.

MongoDB's sharding model supports exactly that: by distributing the workload across multiple shards, the platform can increase payment throughput horizontally while preserving the operational characteristics that matter in production. For banks and PSPs, the benefit is not just more scale – it is a more flexible path to scale, with better room for growth, better infrastructure utilisation, and less dependence on pushing a single system component beyond its practical limits.

This makes the benchmark outcome more meaningful: it was achieved not by brute force on oversized infrastructure, but through a distributed data architecture designed to scale out – an important message for institutions planning payment modernisation programmes that need to grow over time, not just clear one fixed throughput target.

Figure 3.
Segregating the solution across the payment value chain



Built to Bend, Not Break: Resilience Under Real Failure Conditions

Throughput is one half of the story; the other is what happens when something breaks. To test that, we ran failure scenarios at 3,500 TPS to see how the MongoDB-backed IPF application responded in real time – not in theory, but under active transaction pressure. These failure scenarios included:

1. Graceful node shutdown (from 12 nodes to 11), with graceful bringing node back up (from 11 nodes back to 12)
2. Ungraceful kill / node failure (from 12 nodes to 11)

Failure scenario 1

GRACEFUL SHUTDOWN (WITH GRACEFUL BRINGING NODE BACK UP)

This scenario replicated a planned (therefore graceful) node shutdown, bringing the total nodes to 11 (from 12). We also demonstrated bringing that node back up, mimicking what happens in a rolling deployment scenario.

- **Throughput** | For the shutdown, there was no dip in throughput, the application continued to process at 3,500 TPS. When the node was brought back up again this had no impact on throughput.
- **E2E processing time** | Impact was confined only to the transactions on the departing node, all other transactions on the remaining 11 nodes were unimpacted. For the departing node (where the transactions needed to be rebalanced onto the remaining nodes), the P50 latency was flat, the P95 latency was flat, but the P99 latency was up by 19s. When the node was brought back up there was no impact.
- **Recovery time** | The recovery to full normal processing was roughly 60 seconds.
- **Data integrity** | All transactions were processed, there was zero data loss, each transaction reached a terminal state.

Failure scenario 2

UNGRACEFUL KILL / NODE FAILURE

This scenario replicated an unplanned node shutdown, i.e., a node failure, bringing the total nodes to 11 (from 12).

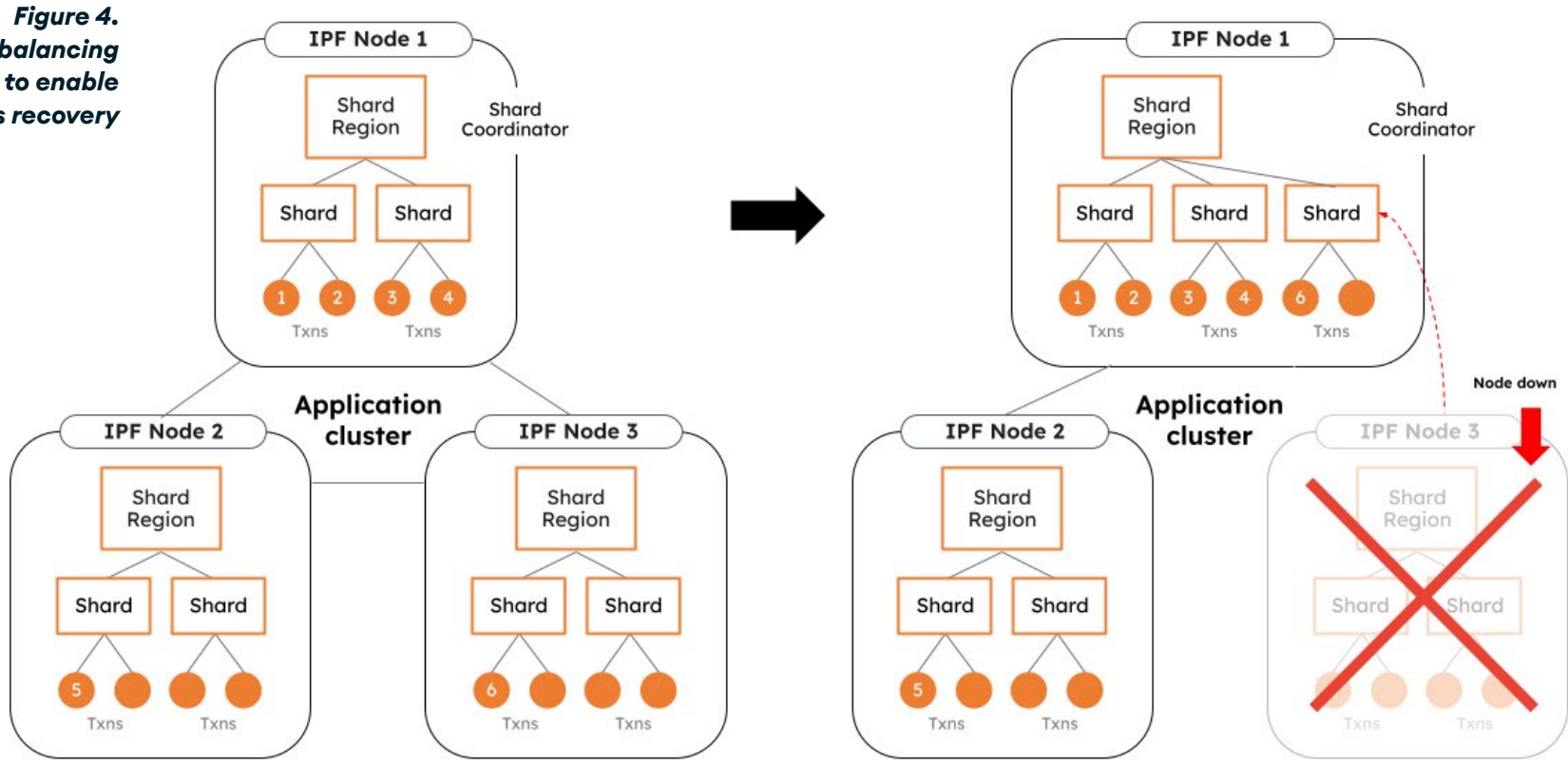
- **Throughput** | The node failure and the resulting recovery caused the throughput to temporarily decrease to roughly 1,000 TPS (from 3,500 TPS). Following the recovery the application returned to 3,500 TPS, with the interim traffic beyond 1,000 TPS queued up and subsequently evenly processed.
- **E2E processing time** | The impact was not confined, it affected the entire cluster. There was an increase in P50 latency, P95 latency and P99 latency (P99 latency was up by 45 seconds).
- **Recovery time** | The recovery to full normal processing was roughly 90 seconds.
- **Data integrity** | All transactions were processed, there was zero data loss, each transaction reached a terminal state.

The mechanisms for ensuring seamless recovery

As per the above results, in both cases the recovery was complete in a very short period of time, with zero data loss and minimal (and in many cases zero) impact to the customer. So, how is this achieved?

If we take scenario 2 (the node failure), the node going down will be detected immediately and it will stop consuming transactions. The transactions that were being processed by the now failed node are redistributed amongst the remaining nodes in the cluster – this is referred to as a 'shard rebalancing'. These transactions that have been rebalanced will have their state rehydrated based on the persisted events, after which they can continue processing as normal. An illustration of this can be seen below.

Figure 4. Shard rebalancing mechanism to enable seamless recovery



As demonstrated by the results, this recovery process does temporarily impact the whole cluster and temporarily reduce the overall throughput (since the remaining nodes in the cluster need to receive transactions from the failed node). But, after 90 seconds once the recovery is complete, the whole cluster continues to process as normal and resumes at 3,500 TPS.

Regarding the latency, as visible the P99 latency for the transactions increased by roughly 45 seconds. The likelihood is that a bank / PSP will be processing a mixture of payment types, some instant, some batch/ACH, some RTGS etc. Instant payment schemes typically have strict processing time SLAs that are around 10 seconds, therefore any instant payments that were impacted by a 45 second latency increase as a result of a node failure would result in them being rejected. However, with IPF, there are multiple ways to mitigate this, including re-routing as non-instant payments (based on configurable conditions), re-executing as instant payments, or handling all the compensation (e.g., booking reversal if that has already taken place) so that the transaction is not left in an error state, therefore meaning no manual intervention is required.

MongoDB Atlas Failover Test

Separately, MongoDB Atlas failover was validated on the benchmark environment using Atlas's built-in primary failover test, in which the current primary is shut down, a new primary is elected and failover completes in about 5 seconds on average, with the original primary returns as a secondary.

This approach is meaningful from an operational perspective because it uses a standard Atlas capability that customers can run themselves through the Atlas UI or API to observe how their application behaves during replica set failover.

In the database failover testing, payment processing continued without payment loss or application restarts , confirming that the stack handled failover correctly through the combined behaviour of the database, driver, and application retry mechanisms.

The most important operational learning was that customer impact depended on the interaction between failover timing and the amount of operational headroom available in the database layer at the time – for example, whether cache pressure was low, write latency remained stable, and there was enough spare capacity to absorb the temporary retry wave triggered during the election. With sufficient headroom, failover was close to invisible at the application level; nearer the utilisation envelope, the same event could present as a temporary tail-latency wave while preserving processing continuity and data integrity, giving institutions a clear path to designing for graceful recovery under load.

For banks and PSPs operating in a 24x7, always-on environment, this is the proof point that matters most: the platform did not just process high volumes under ideal conditions, it kept processing payments through an unplanned failure, with the database layer absorbing the disruption rather than passing it on to the business.

Get the Full Benchmark Report

This post covers the headline story of the Icon Solutions x MongoDB instant payments benchmark. The complete report, including the full metrics ladder across every step from 500 to 6,000 payment transactions per second, plus further details on how the results were achieved, is available on request.

To get the full picture, reach out to your Icon Solutions or MongoDB representative, or contact the teams directly via:

ICON SOLUTIONS

MONGODB

WEB VERSION

PRESS RELEASE

About Icon Solutions

Icon Solutions is a fintech company that has been designing and implementing state-of-the-art payments systems since 2009. Its core product, the Icon Payments Framework (IPF), is an internationally proven payments development framework trusted by Tier 1 banks including Citi, NatWest, and BNP Paribas. IPF helps banks accelerate the transformation of their payments infrastructure by enabling them to build, test, and deploy payment processing solutions faster while maintaining control of timelines and costs.

About MongoDB

MongoDB, headquartered in New York, is focused on helping innovators create, transform, and disrupt industries with software. Its unified, intelligent data platform is built to power the next generation of applications, with integrated capabilities for operational data, search, real-time analytics, and AI-powered retrieval. Millions of developers and more than 67,000 customers across nearly every industry, including roughly 75% of the Fortune 100, rely on MongoDB for their most important applications.