XMPRO

Engineering for Enterprise Scale

December 2025 | Author: XMPro Engineering Team

Contents

Coming in 2026: The distributed connector architecture described here is planned for an upcoming XMPro release. This document shares our engineering findings and the architectural approach we're taking to support enterprise-scale deployments.

1 Who This Is For

This article is written for **technical decision-makers and architects** evaluating platform scalability—CTOs, solution architects, enterprise architects, and technical leads responsible for selecting or scaling industrial operations platforms.

You'll find this relevant if you're:

- Evaluating XMPro for large-scale deployments
- Planning capacity for growing user bases
- Architecting data-intensive operational dashboards
- Comparing connector architecture approaches across platforms

For implementation details and deployment guidance, see our technical documentation.

2 Summary

The Challenge: XMPro's existing connector architecture—like most SaaS platforms—executes data retrieval within the application process. This works well for moderate concurrent user loads but creates bottlenecks at higher scale where hundreds of concurrent users access data-intensive dashboards, causing non-linear performance degradation.

What We Tested: We stress-tested two architectural approaches: the industry-standard in-process model (connectors execute within the application server) versus a distributed model (connectors execute on dedicated stream hosts via message broker).

The Outcome: The distributed architecture delivered:

- $\sim$**5**$\times$ **higher concurrent user capacity** at enterprise scale
- $\sim$**50**$\times$ **more page loads** under stress conditions
- **Near-linear scaling** characteristics instead of threshold-based degradation
- **Predictable performance** with complex, multi-data-source dashboards
- **Consistent developer experience** even under production load

What This Means: These findings are shaping an **enterprise-tier architecture** planned for 2026—extending XMPro's capabilities for customers requiring higher concurrent user capacity to scale predictably and cost-efficiently.

3 Why We Conducted This Study

As XMPro deployments grew larger—more concurrent users, more data sources per dashboard, larger payloads—we observed patterns that prompted deeper investigation:

- Performance degraded non-linearly as load increased
- Pages with multiple data grids performed disproportionately worse than simple pages
- Design mode (where developers build pages) was more affected than runtime mode

Rather than apply incremental fixes, we asked a fundamental question: *is the current architecture optimised for the workloads our enterprise customers actually run?*

The answer led us to explore an alternative: distributed connector execution.

3.1 Important Context: In-Process Connectors Are the Industry Standard

The in-process architecture we tested is the standard approach across the industry. Most SaaS platforms—including major BI tools, CRMs, and integration platforms—execute connectors within the application process. It's simpler to deploy, easier to operate, and performs well for moderate concurrent user loads.

For XMPro deployments with moderate concurrent users and standard dashboard complexity, the in-process model works well and requires no additional infrastructure.

The question we investigated wasn't "is our architecture broken?" but rather: **what do customers need when they require higher concurrent user capacity?**

4 Test Methodology

4.1 Environment

We stress-tested both architectures on constrained infrastructure to surface architectural differences clearly:

Component	Configuration
Application Server	Below minimum recommended (2 vCPUs, limited RAM)
Database	Basic tier
Load Tool	Grafana k6
Test Duration	∼10 minutes per scenario, staged ramp-up

Table 1: Test environment configuration

Important: We deliberately used constrained resources—below typical production sizing—to stress-test architectural limits. The improvement ratios we report are consistent across infrastructure sizes, but absolute capacity numbers depend on your specific deployment configuration.

Testing was conducted on Azure, but findings apply to equivalent resources on AWS, Windows Server, or other deployment targets.

4.2 What We Measured

Each simulated user performed complete page loads—not individual API calls. A single page load triggers multiple API calls: authentication, metadata, configuration, and data retrieval. This matters because architectural bottlenecks compound across request sequences.

Tests included automatic threshold detection:

- **p95 response time exceeding acceptable limits** = test ends (unacceptable user experience)
- **Error rate exceeding threshold** = test ends (system instability)

This gave us a consistent measure of *practical capacity*—the load level where the system remains usable, not just functional.

5 Key Findings

5.1 Concurrent User Capacity

Metric	Distributed vs In-Process
Concurrent user capacity	∼**5**× **higher**
Page loads under stress	∼**50**× **more**
Median page load time	∼**3**× **faster**
Success rate	Higher reliability

Table 2: Concurrent user capacity comparison (small payload, single data source, runtime mode)

The in-process architecture reached its threshold at a certain concurrency level, after which response times exceeded acceptable limits. The distributed architecture handled approximately 5× more concurrent users while maintaining acceptable response times.

Context: For many XMPro deployments, the in-process capacity is entirely sufficient. The distributed architecture extends capacity for organisations with larger user bases, peak load requirements, or particularly data-intensive dashboards.

5.2 The Compounding Effect: Multiple Data Sources

Real dashboards aren't simple. They display multiple data grids, charts, and forms. We tested pages with increasing data source counts:

Data Sources	Relative Performance
0 (baseline)	Both architectures equal
1	Distributed ∼175× more throughput
2	Distributed ∼600× more throughput
3+	Distributed orders of magnitude better

Table 3: Performance by data source count

The baseline (0 data sources) confirms both architectures perform identically when connectors aren't involved. The rendering pipeline isn't the bottleneck.

With multiple data sources under heavy load, the in-process model showed significant degradation while the distributed model maintained consistent throughput.

Why this matters for enterprise deployments: Simple dashboards with 1–2 data sources perform adequately with in-process execution—and that covers many use cases. However, enterprise operations dashboards often consolidate multiple data sources: asset status, production metrics, alerts, work orders, maintenance schedules. The distributed architecture is designed specifically for these data-intensive scenarios where multiple connectors execute concurrently.

The performance difference stems from resource isolation: in the distributed model, connector execution happens on dedicated hosts rather than competing for resources within the application process.

5.3 Large Payload Handling

Payload Size	Distributed Advantage
Small (∼10KB)	∼50×
Medium (∼128KB)	∼65×
Large (∼1MB)	∼25×
Very Large (∼5MB)	∼7×

Table 4: Distributed advantage by payload size

Both architectures degrade with larger payloads, but the distributed model maintains its advantage across all sizes tested. Even at very large payloads—well beyond typical use—it still delivers meaningful improvement.

5.4 Developer Experience: Design Mode

Design mode (where developers build pages) showed even larger differences than run mode—often more severely affected by architectural bottlenecks.

Metric	Distributed Advantage
Concurrent users at threshold	∼**13**× **more**
Page loads completed	∼**200**× **more**
Median load time	∼2× faster
Success rate	Significantly higher

Table 5: Design mode performance comparison

The gap widens dramatically with page complexity:

Data Sources	Distributed Advantage
1	∼200×
2	∼1000×+
3	Orders of magnitude (in-process reached threshold early)

Table 6: Design mode scaling by data source count

Why design mode is more affected:

- **Interactive patterns** — Developers iterate repeatedly, amplifying per-request overhead
- **Metadata overhead** — Design mode loads additional configuration alongside content
- **Shared infrastructure** — Design and production workloads compete for the same application server resources

Developer productivity depends on responsive tooling. Slow page loads during iterative design work accumulate into significant lost time. The distributed model doesn't just make individual developers faster—it allows more developers to work concurrently without degrading each other's experience.

6 Why the Distributed Model Performs Better

6.1 Architectural Difference

In-Process Architecture

This approach optimises for **simplicity and operational ease**—no additional infrastructure required. Connectors load on-demand within the application process:

```
User request -> Application Server
                  |-- Load connector (cached or fresh)
                  |-- Execute query
                  '-- Return results
```

For moderate concurrent user loads, this model performs well. At higher scale with many concurrent users, the shared process becomes a constraint.

Distributed Architecture (Enterprise Tier)

This approach optimises for **throughput at scale**. Pre-loaded connectors on dedicated hosts handle execution, freeing the application server:

```
User request -> Application Server
                  '-- Publish request to message broker
                                |
                                v
                      Stream Host Collection
                        |-- Connectors pre-loaded
                        |-- Execute query
                        '-- Publish results
                                |
                                v
                Application Server receives results
```

The trade-off: additional infrastructure (message broker, stream hosts) in exchange for higher capacity and isolation.

6.2 Performance Benefits

Factor	In-Process	Distributed
Connector loading	Per-request	Pre-loaded
Resource contention	High (shared process)	Low (isolated hosts)
Failure isolation	Connector crash affects app	Connector crash isolated
Scaling model	Scale app servers	Scale stream hosts

Table 7: Architectural performance comparison

7 Implications for Enterprise Deployments

7.1 Capacity Scaling

The distributed model's advantage **compounds** at higher resource allocations. On larger infrastructure:

- In-process architecture scales sub-linearly (diminishing returns)
- Distributed architecture scales near-linearly (proportional improvement)

This means the $\sim$5$\times$ baseline advantage can grow to 5–8$\times$ or more on properly sized production infrastructure.

7.2 Infrastructure Efficiency

For deployments requiring enterprise-scale concurrent user support with multi-data-source dashboards:

In-Process approach:

- Requires significant horizontal scaling (many application server instances)
- Each instance carries connector loading overhead

Distributed approach:

- Fewer application server instances + stream hosts + message broker
- Estimated **5–8$\times$ lower infrastructure cost** for equivalent capacity

This comparison holds whether you're running on Azure, AWS, or on-premises Windows Server—the architectural efficiency translates across platforms.

7.3 Decision Framework

The distributed architecture is particularly valuable when:

Scenario	In-Process	Distributed
Moderate users, simple pages	Adequate	Lower infrastructure cost
Growing user base	May struggle	Recommended
Large concurrent user counts	Requires significant scaling	Well-suited
Multi-data-source dashboards	Constrained	Designed for this
Large payloads	Struggles	Handles well
Developer productivity critical	Limiting	Enables

Table 8: Decision framework for architecture selection

8 What This Means for XMPro's Direction

These findings validate an architectural direction we've been developing: **consolidating integrations into a distributed stream-based architecture**.

Historically, we maintained separate systems for application connectors and stream processing. The in-process connector model was designed for simplicity and ease of operation. As customer deployments continue to scale, we're unifying these systems to further optimize performance at higher concurrency levels.

By unifying these systems, we gain:

- **Single optimisation target** — Engineering focuses on one integration runtime
- **Shared infrastructure** — Stream hosts serve both real-time and interactive workloads
- **Predictable scaling** — Linear degradation instead of threshold collapse
- **Developer experience** — Design mode that keeps pace with production mode

This isn't just about making pages load faster. It's about building a foundation where enterprise-scale deployments perform predictably—where infrastructure investment delivers proportional capacity.

9 Limitations and Caveats

9.1 What We Tested

- Synthetic load (uniform page requests, not mixed workloads)
- Single cloud region (Azure; architectural findings apply to AWS and on-premises)
- Fixed query complexity (simple queries)
- Stress test conditions on constrained infrastructure

9.2 What We Didn't Test

- Mixed workload patterns (runtime + design mode + API + background jobs)

- Geographic distribution and network variability
- Complex queries (joins, aggregations, stored procedures)
- Failover and recovery scenarios
- Memory behaviour over extended periods

9.3 Production Considerations

Production introduces variables our tests controlled: concurrent workload types, payload variability, peak vs. sustained load, infrastructure sharing. The improvement ratios we measured should remain consistent, but absolute capacity depends on your specific infrastructure sizing. Treat these findings as directional guidance for architectural decisions.

10 Conclusion

Enterprise scalability isn't just about bigger servers—it's about architectures that scale efficiently. Our load testing revealed that connector execution location is a leverage point: moving it from shared application processes to dedicated stream hosts transformed scaling characteristics from threshold-based collapse to linear degradation.

The distributed connector architecture planned for our 2026 release represents this learning applied. For organisations running data-intensive applications at scale, the architectural pattern—separating computation-heavy operations into dedicated worker collections—offers a path to predictable, cost-efficient scaling.

For deeper exploration of the methodology, multi-data-source behaviour, design mode findings, and payload analysis, see our technical blog series on distributed architecture patterns.