NEEVE RESEARCH

Why Use A Distributed Architecture for Real-Time Big Data

Author: Girish Mutreja

This document introduces the reader to techniques and requirements leveraged by a distributed architecture for providing real-time processing on big data.

Neeve Research LLC 1148 Brace Avenue, San Jose CA 95125

Table of Contents

The Modern Enterprise

The world of enterprise applications has undergone a dramatic change. Drivers such as globalization, social computing, Omni channel access, round-the-clock computing, and domain-specific changes have pushed data and transaction processing requirements to levels that were once considered extreme but are now commonplace. At the same time, application architectures and development practices have evolved to keep pace with the highly dynamic business and social ecosystem in which these applications operate. This has given rise to significant computing trends such as cloud computing, micro-service architectures, and FaaS; each of which are enabling increasingly agile and resilient application delivery. These trends, in combination with multi-core processing and more reliable networks, are shifting applications from being monolithic to being more network-centric, multi-agent and distributed. Enterprises that used to be driven by heavyweight wheel-and-spoke architectures are now enabled by lightweight collaborative agent architectures where business functionality is developed, deployed, and managed in an agile, flexible, and continuous manner. X Platform™ is a next generation, memory-oriented platform that not only enables such application architectures but takes them to the next level of performance, agility, resilience, and scalability. X Platform™ combines in-memory computing with microservices architecture and next-generation transaction processing to super charge your data for real-time intelligent transaction processing.

Organic evolution is synonymous with survival in a continuously changing environment.

Multi-Agent Systems

Multi-agent systems are comprised of interconnected, reusable software agents. Each agent performs a specific task and collaborates with other agents to accomplish the overall business function. These agents are essentially absorbing messages from their surroundings and reacting to them with their own messages. Financial trading engines listen for new orders from an order manager, processes the orders, and dispatch them to the appropriate markets via exchange connectivity agents. Apps on our phones listen for messages from our friends and business associates, which we in turn, process and respond to in the form of additional messages. E-commerce engines both listen to and serve pricing requests, while simultaneously propagating informational events to analytical engines for dynamic pricing and yielding. All of these examples have one thing in common. Their function is driven and accomplished by collaborating software agents.

Why Enterprises Are Moving to Multi-Agency

This paradigm of decomposing business tasks into multiple collaborating agents is rapidly becoming the de facto standard for enterprise software. Such multi-agent systems have several significant benefits:

Operational Concerns

- *Time to market* – They require less time to develop due to functional decomposition, contract-based collaboration and agent lifecycle independence.
- *System resilience* – They exhibit higher system resilience due to fault isolation.
- *Scalability* - Inherently more scalable due to higher concurrency.

While achieving the above concerns provides significant benefits, the real driver of the multi-agent paradigm is *agility* and *organic growth*. Businesses are increasingly dynamic to a point that enterprise systems need to possess the ability to *organically* evolve with, and adapt to, the functions they serve. Organic evolution is synonymous with survival in a continuously changing environment. Being able to evolve organically will allow enterprise systems to not only keep pace with the ultra-dynamic nature of modern enterprises, but to become even stronger as they evolve to meet growing needs.

Systems with these capabilities go far beyond the technical and operational goals of *time to market, system resilience,* and *scalability*. Instead, they start serving the enterprise by allowing for (and even encouraging) continuous adaption - the first step towards *enterprise intelligence*. It is these properties that foster innovation.

Realizing the Full Potential of Multi-Agency

Achieving this level of adaptability is not easy. In addition to putting in place the required developmental practices and fostering the right culture, the technology platform that underpins such systems needs to possess one overarching capability: *it needs to make it trivial for developers to author components and snap them in or out without adversely affecting the whole system*. It is this simple yet powerful capability that truly enables a dynamic architecture.

However, this dynamicity must *not* come at the cost of stability, performance, reliability or any other quality of service (QoS) concern of the application. The system must be dynamic and still satisfy stringent service demands. To enable this level of agility, the platform must:

1. *Completely* take away the QoS burden from the developer

2. Make it *easy* to author application logic
3. Make it *risk-free* and *non-intrusive* to deploy business functionality

Completely take away the Quality-of-Service (QoS) burden

Quality of service complexities are more pronounced in multi-agent systems due to their distributed nature. Some of the key areas that become more complex and have more stringent SLAs are *performance*, *storage, collaboration, transaction management*, *error handling,* and *system monitoring*. Why and how each of these areas become less forgiving in distributed systems is covered later in this paper. For now, it suffices to say it's very common for development teams to spend more than 50% of their time working on issues that have nothing to do with the business requirements. *As these non-functional aspects become harder, developers will spend even more time on them*. Application developers should be focused on the *delivering business functions*. More focus on domain-specific problems result in systems keeping pace with the business, more innovation, more revenue, and more robust systems. The platform needs to provide the full gamut of QoS leaving the developer to author and innovate on relevant and salient aspects of the applications.

Make it *easy* to author application logic

Actors in multi-agent systems have application logic, are stateful and are collaboration-aware. This means that agents send and receive messages to-and-from other agents and remember their state between messages. Application logic is triggered by inbound messages, state is queried and updated by the application logic, and additional messages are sent out for other agents to process. In current systems, *including cloud applications*, developers have to bolt on a messaging technology, a storage technology, a transaction management technology and a monitoring technology to the application logic (and configure them to work together) when developing an agent. This is not easy, and severely hampers the agility and robustness of the agents authored. This is particularly true due to the much higher QoS requirements brought on by distributed systems. Just as higher-level programming languages were developed to make it easier to write software programs, multi-agent platforms need to make it easy for developers to author fully integrated stateful and collaborative behavior into agents. Essentially, the platform should commoditize how agents integrate application logic, state management, and collaboration.

Make it *risk-free* and *non-intrusive* to deploy business functionality

The ability to quickly test out new business functionality without requiring a "stop the world" release methodology or completely impacting the overall system is paramount. Organizations can no longer suffer from weekend releases and rollbacks that impact production operations, customer experience, or data integrity. Enterprise systems need to be available 24x7 – requiring a new way for maintaining healthy systems.

There *cannot* be any resistance to deploying or rolling back functionality from production systems. If there is, the resistance will grind the agility of the system a halt. The platform must provide the tooling (or integrate with existing tooling) that makes deployments continuous, smooth, easy, and risk-free.

In-Memory Multi-Agent Platform

Once it is easy to add, modify and test business functionality, remove QoS aspects from development concerns, and it is risk-free to deploy components, then the true promise of multi-agent systems has been realized. X Platform™ *is such a platform*. It is powered by a key innovation that centers around the use of *in-memory computing* in a unique manner that eliminates the boundaries between application logic, messaging, state management, and transaction processing. X Platform™ integrates these in a manner that allows developers to author application logic, manage state and communicate with other agents using "plain old Java objects". Developers author applications using Java, manage state using POJOs, and *memory is durable*. They communicate with other agents using POJOs in a *"fire-n-forget"* manner with *exactly once delivery semantics*. By taking ownership of the complexities of state management and messaging, X Platform™ allows the developer to focus exclusively on application logic. This makes it extremely simple to author stateful, collaborative, fully reliable, scalable, and extremely performant applications using Java.

Makes it easy to develop and "snap" agents in and out of the system.

Systems built with X Platform™ regularly process hundreds of thousands of transactions-per-second in single to double-digit microsecond latencies with linear scalability while running in high-availability mode. Agents built with X Platform™ are running in full availability mode and are in no danger of failures from process, machine, network, or data losses. As an enterprise grade system, X Platform™ is not just for developers. It provides comprehensive

monitoring and management capabilities with unprecedented insight into both individual components and your overall system.

An X Platform™ App

To provide some context to the above, let's illustrate how easy it is to build an application using X Platform™. For this illustration, we will build a simple application that receives a message containing a number from an upstream agent, adds the received number to a counter that it maintains in its state and sends a message outbound to a downstream agent with the resulting sum.

When authoring such an application using X Platform™, a developer performs the following functions

1. Model state
2. Model messages
3. Author logic using Java
4. Build, run, control and monitor the application

Model State

First, the developer defines the agent's state such as the following:

```
<model>
    . . .

    <entities>
        <entity name="Repository">
            <field name="counter" type="Long"/>
        </entity>
    </entities>
</model>
```

At build time X Platform™ will convert this to POJOs.

Note: X Platform™ also provides an option for the application to work with non-generated POJOs. In that case, the above step is not needed.

Model Messages

Next, the developer defines the messages such as the following:

```
<model>
    ...
    <messages>
        <message name="Message">
            <field name="value" type="Long"/>
        </message>

        <message name="Event">
            <field name="result" type="Long"/>
        </message>
    </messages>
</model>
```

At build time X Platform™ will convert this to POJOs.

Author logic using Java

An X Platform™ agent is essentially a sophisticated, enterprise grade message processor (as are agents in multi-agent systems). Therefore, the application logic is essentially a bunch of message handlers. The following is the message handler for the agent that receives messages defined above, updates the counter in its state, and sends the resulting value downstream.

```
    @EventHandler
    public final void onMessage(Message message, Repository repository) {
        repository.setCounter(repository.getCounter() + message.getValue());
        Event event = Event.create();
        event.setResult(repository.getCounter());
        sender.sendMessage("events", event);
    }
```

Build

A developer brings all of this together using a build tool of their choice. X Platform™ provides build tool plugins to generate Java classes from the XML based state and message definitions. In the above, the `Repository`, `Message` and `Event` classes referenced by the message handler are generated by X Platform™ code generator plugins. The developer works with them in the message handlers as regular POJOs.

Run

Once built, X Platform™ provides advanced tooling to deploy and run the application. The platform provides extensive configuration and tuning for X Platform™ runtime.

Key Takeaways

- X Platform™ makes it easy to author stateful, collaborative application logic and seamlessly integrates state management, messaging, transaction processing and application logic.
- X Platform™ hides all quality-of-service complexities from the developer without compromising these capabilities:
    a. The generated repository is *fully durable*.
    b. The above code will be executed once and *exactly once* across process, machine or data center failures resulting in no message or data loss.
    c. The above code performs at hundreds of thousands of transactions per second with a processing latency in single-digit microseconds.
    d. The above code can be scaled horizontally for concurrent processing with user-defined data affinity.
    e. The code is *garbage free*.
- X Platform™ provides a dashboard from which a user can configure, control, monitor and even troubleshoot a running system.

QoS Challenges with Multi-Agent Systems

As we discussed before, there are some non-functional complexities that can be more pronounced in multi-agent systems. This is the key reason why the implementation of these capabilities needs to be delegated to the platform leaving the developer to focus on application logic. The important areas that one needs to keep in mind in multi-agent systems are as follows:

- Performance
- Storage
- Collaboration
- Error handling
- Transaction Management
- System Monitoring

Performance

Application performance is always important. However, in multi-agent architectures, the performance of an agent becomes of even greater consequence. This is because, in contrast with monolithic systems, a business transaction can and generally does span multiple agents. The more a multi-agent architecture meets its agility goals, the more agents get developed resulting in longer transaction pipelines. In other words, the more successful a multi-agent architecture

is from an agility standpoint, the worse the business transaction performance can become. This is particularly true with transaction latency and throughput if agents communicate using synchronous modes of communication. It is critical that individual agent performance be extremely fast. Generally, if individual agents perform in double-digit microseconds, then one would expect the business transaction performance should be in single-digit milliseconds. If agents perform in hundreds of microseconds to single-digit milliseconds, then transaction performance would be in the high tens to hundreds of milliseconds. If agents perform in double-digit to hundreds of milliseconds, then transaction performance would be expected to be in seconds. It is critical that agent performance be kept at the lowest possible latency to provide enough headroom to not compromise on the overall system's performance.

X Platform™ implements several key techniques such as cut-through serialization, off-heap data storage, transaction pipelining, and memory mapped IO to keep performance in the low microsecond range.

Storage

Storage becomes important for two primary reasons: transactions and logs. Agents need to store both their transaction changes and log messages where they can be independently examined for recovery. When dealing with high performant systems, bottlenecks can be devastating; therefore storage needs to be as fast and simple as possible. Private agent storage is highly recommended for maximum agility and performance.

X Platform™ ensures *full* transactional consistency

X Platform™ implements private in-memory durable storage per agent. Each agent has a private instance of an in-memory "micro DB" with quorum based local persistence and asynchronous connectivity to shared enterprise stores. This enables independent schema management and store administration for maximum performance and agility with data integration to the rest of the enterprise.

Messaging & Collaboration

In a distributed platform it must be easy for agents to communicate using a variety of interaction patterns including synchronous request-reply, asynchronous request-reply, and event streaming. Messaging cannot become a bottleneck in the system either. This becomes especially important as the number of agents in an orchestrated transaction increases. Furthermore,

the mechanism by which one models collaboration between agents should be simple *and* flexible. For example, it should be simple to model a system with agents communicating using traditional services with a request-reply based APIs. Then at a later date add the emission of events from agents and a new set of agents to consume these events. This process should not take a re-design of the system, or even be considered as a technical challenge. Agents should be able to collaborate using event streaming over a rich user-controlled topic namespace, or purely via discoverable business functions (such as in a "function cloud".) Modeling collaboration between agents in a simple yet flexible manner is a key aspect of building a multi-agent system that can evolve in an organic manner.

X Platform™ has invented a simple and powerful abstraction to address these issues. It is not only possible to model *who* will communicate, but *how* they communicate as well. The fundamental abstractions for communications are *channels,* and these connect agents in a one-to-many fashion. Messages are sent and received as POJOs in a fire-n-forget manner through channels. Channels are simple but powerful and agile. They can be configured to allow for a system to be used in any of the manners described above and more.

X Platform™ implements a messaging provider abstraction layer where one can switch between providers in a plug-n-play manner without any code change. X Platform™ has adapters for many solutions including Solace, Kafka, ActiveMQ, JMS (generic), X Platform™ native messaging offering, and others.

Finally, X Platform™ messaging machinery is engineered for extreme performance. It implements techniques such as cut-through serialization and pipelining for extreme performance without compromise on reliability.

Error Handling & Transaction Management

Transactions are used to ensure the integrity of the data and the system in the face of exceptions and failures. In contrast to monolithic systems in which a business transaction is completely local to a single process, transactions spanning multiple agents are more complicated. Many solutions have been presented over the years, the most obvious is to use a distributed transaction management technique like two-phased commit. These techniques are technically feasible, but come with significant performance impacts and introduce developmental and operational complexities.

X Platform™ ensures *full transactional consistency* local to an agent with configurable policies governing how the system should behave in the face of exceptions and failures. X Platform™ relies on guaranteed-eventing and support compensating-transactions. With guaranteed eventing, the platform ensures the fire-n-forget semantic offered to an agent. This means that once a message is sent and the transaction in which the message was sent is deemed successfully complete, the platform will guarantee eventual delivery to the downstream agents interested in the event. In the event of a local transaction failure, the platform supports dispatching compensating transaction to upstream agents to roll back the changes made previously in that transaction. Both these techniques working in concert result in eventual consistency of the system in the face of exceptions and failures.

System Monitoring

A robust multi-agent system will encourage the development of additional agents. However, the more the moving parts in a system, the harder it is to manage, monitor, and troubleshoot issues. Therefore, it is critical to have a sophisticated, centralized management, monitoring and troubleshooting tool. Without such a tool, a multi-agent system can get out of control and become difficult to operate.

X Platform™ provides a centralized deployment management and monitoring tool. It is a sophisticated tool that allows for viewing of statistics, alerts, and lifecycle events. It also enables agent command-n-control, agent versioning, agent upgrade with zero service loss, trace log consolidation and drill down facilities for troubleshooting and debugging purposes.

Summary

Modern enterprises are rapidly moving from heavyweight, monolithic applications to light-weight, network-centric multi-agent applications. The shift is due to the increased agility, resilience and organic growth that these systems offer. This allows these systems to better keep pace with the business, develop intelligence for increased information flow and foster greater innovation in the enterprise. To finally harness the organic nature of multi-agent systems, the platform that underpins the systems needs to make it easy to author application logic by allowing a developer to seamlessly integrate business logic with messaging, state management and transaction processing. It must completely isolate the developer from all quality of service aspects of the application, while at the same time not compromise on them. The platform must enable simple, risk-free and non-intrusive deployment management. X Platform™ from Neeve Research is such a platform.