SYNERGEX

Distributed Computing

Transforming DBL Applications

WHITE PAPER

Contents

Executive summary

As your business expands, your applications need to manage additional users, a greater volume of data, more integrations, and broader geographic coverage. Distributed computing is one way a modern software system can extend its capabilities and better process data, offering scalable, reliable, and efficient handling of complex tasks. This white paper explores the fundamentals of distributed computing—its key characteristics and benefits and influential industry trends—as well as how Synergex's Streaming Integration Platform, which uses the Apache Kafka protocol, specifically facilitates the transformation of a DBL application to a distributed system. Our goal is to show how transitioning to a distributed system can help your software meet new demands effectively and efficiently.

What is a distributed system?

A distributed system is a network of independent computers that function as a single coherent system to users. Tasks are distributed across multiple machines that communicate and coordinate with each other.

Benefits

Key characteristics and benefits of distributed systems include:

Concurrent components

Multiple parts operate across different computers, communicating via messages to seamlessly coordinate their actions.

Shared resources

Components share resources like databases and files, maintaining consistent states across the entire system.

Scalability

Distributed systems can expand by adding more nodes, giving them the ability to cover larger geographic areas and scale up or down as needed.

Fault tolerance

As a result of redundancy and robust failure handling, distributed systems remain operational despite failures.

Transparency

Users are able to interact with the system as if it were a single, cohesive unit, regardless of the distributed nature of its components.

Industry trends

Several key developments have significantly increased the adoption of distributed systems in the software industry:

Cloud computing. Platforms like AWS, Azure, and Google Cloud have made distributed computing more accessible, allowing companies to deploy and manage applications across multiple locations affordably.

Microservices architecture. A microservices approach breaks applications into smaller, independent services that communicate over a network, enhancing scalability and flexibility.

Big data and real-time processing. The need to manage large data volumes efficiently makes distributed systems ideal for big data processing and real-time analysis.

Resilience and availability. The demand for better fault tolerance and availability can be met by spreading resources across multiple servers or locations. Distributed systems often allow data to stay geographically closer to the user.

Introducing Synergex's Streaming Integration Platform

The Streaming Integration Platform connects Synergex's ISAM DBMS to Kafka queues or "topics," facilitating the conversion of a DBL application to a distributed system.

Event streaming with Apache Kafka

Kafka is a highly acclaimed technology and protocol that the Streaming Integration Platform leverages to solve developer pain points related to running centralized systems:

Kafka **supports better application scalability** by allowing systems to expand across multiple servers and regions, efficiently handling more users and higher data volumes without performance loss. Load balancing and horizontal scaling ensure consistent performance as demand increases.

Kafka **reduces costly and disruptive production outages** by providing robust data replication and fault tolerance. If one server fails, Kafka ensures that data is still available from other servers, minimizing disruptions and maintaining continuous system operation.

Kafka **manages transaction complexity** by serving as a centralized platform that efficiently distributes large volumes of data across users and third-party applications. This setup ensures that all parties can access and process data simultaneously without bottlenecks, supporting high concurrency and integration flexibility.

Streaming Integration Platform advantages

The Streaming Integration Platform achieves the following:

- ✓ **Leverages existing code**. Code rewrites are generally not viable; they tend to be unsuccessful and cost-prohibitive. Existing, time-tested DBL code can be seamlessly integrated into this new solution.
- ✓ **Enables phased development and deployment**. Allows for incremental development and deployment, rather than requiring large-scale efforts that could disrupt the entire organization.
- ✓ **Preserves application speed for the end user**. Maintains the speed and efficiency of current applications to ensure users can continue their work effectively.
- ✓ **Provides user-friendly operations and diagnostics**. In a distributed system with multiple nodes, manually logging and checking files to diagnose issues would become cumbersome and ineffective. The Streaming Integration Platform provides diagnostic tools to handle the increased complexity.
- ✓ **Works on-premises and in the cloud**. The Streaming Integration Platform is versatile, functioning both on-premises and in the cloud.

Other building blocks

Synergex uses the following products in its implementation of Kafka.

HashiCorp Consul	Coordinates and configures services, facilitates resource locking between nodes, and distributes configuration data to individual nodes.
Redpanda	A faster, more efficient alternative to Apache's Kafka platform, using the Kafka protocol with regular Kafka clients. It requires less hardware, offers lower latency, and is easy to set up and manage.
docker	A platform that simplifies creating, deploying, and running applications using containers. (Containerizing is easy—it's just a Docker file away.)
WebSSH2	Integrates an SSH client directly into a web browser, modernizing access for users who rely on SSH or Telnet. It provides a browser-based, interactive SSH terminal for managing systems directly through the web.
SeaweedFS	Offers an S3 compatible API for on-premises organizations, functioning independently of AWS.

Harmony Core support

The Streaming Integration Platform has a number of features that are underpinned by Synergex's Harmony Core, a proven, open-source framework for creating RESTful web services APIs. For example, Harmony Core is used to wrap and isolate legacy code that contains a lot of global and static variables so it can run in multi-threaded environments. Developers can also use Harmony Core to incorporate third-party APIs to enhance functionality and support new logic. This integration ensures a more seamless and efficient handling of data across different system components.

Distributed Computing with the Streaming Integration Platform

Before taking advantage of distributed computing, applications generally run from a single node as in Figure 1. Users interface with the application via SSH and/or a RESTful web service, and the application has direct access to the data files.

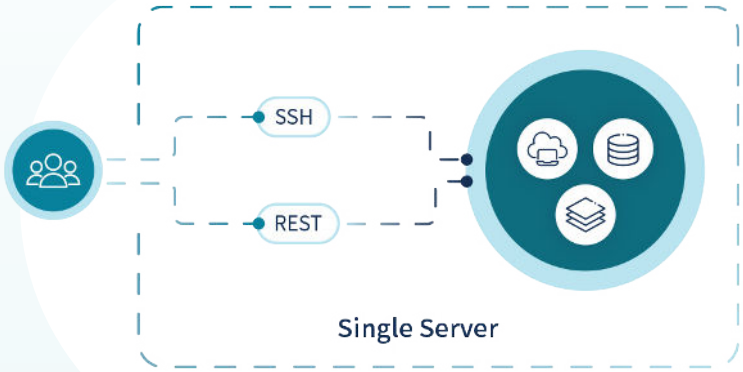


Figure 1: Single-node architecture (before distributed computing)

Phase 1: Leader with replicas

In the first phase, the developer implements a leader node and one or more follower nodes. All nodes have the same application, data files, and Harmony Core RESTful web service. The leader node can perform both read and write operations, while the followers can only perform reads. Developers will need to implement some kind of routing between the users, the leader, and the followers since some operations can only be serviced by the leader.

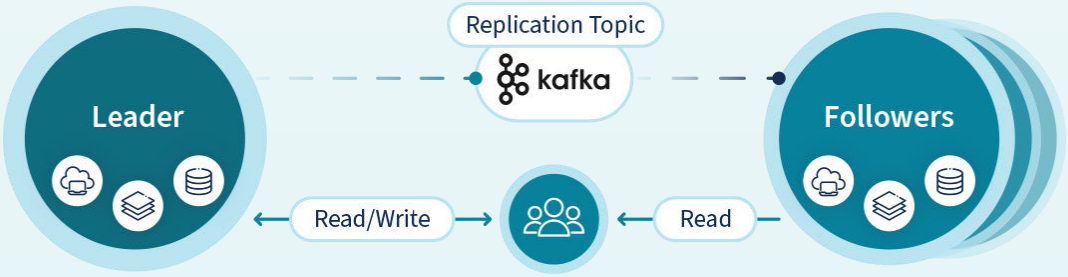


Figure 2: Phase 1: Distributed architecture – Leader with replicas

Access to the application is primarily via SSH, allowing users to interact directly with the leader node. The application has direct file system access, enabling operations on ISAM files that are stored on the disk of each node. Write operations on the leader are monitored through I/O hooks, which capture data activity and send it to Kafka using a replication agent.

The operation is then processed by Kafka and sent to the follower nodes, which stay synchronized with the leader through their own replication agents. Unlike the leader, the follower nodes primarily handle read operations, as they don't manage ISAM file locks. This configuration allows the follower nodes to serve data requests that don't require write access.

This process is similar to (and a major enhancement over) Synergex's SQL replication solution, where applications use I/O hooks to replicate their ISAM data in an RDBMS. In the new approach, the locks on ISAM files are maintained separately on each node, which can significantly reduce lock contention between nodes. Using Kafka as an intermediary for managing data updates ensures that the ISAM files on the follower nodes are almost always up to date, except for a brief period during data transit. This approach ensures that data across nodes remains consistent and highly available. This method offers better scalability by decoupling the data replication process from the RDBMS server's direct handling. By leveraging Kafka's robust messaging and queuing capabilities, the system can handle larger volumes of data changes more efficiently, allowing the replication process to scale as data volume grows.

Further, the system uses a Consul service for configuration management and node role coordination, effectively deciding which node acts as the leader. This setup facilitates seamless dynamic configuration adjustments and node role transitions

Application transparency

Applications interact with ISAM files stored locally on the disk. Whether the application is running on a leader or a follower node, it accesses these files directly, just as it would in a non-distributed environment. The application has no need to fetch data over a network or through external servers using xfServer.

I/O hooks are implemented to intercept file operations transparently. These hooks capture write operations performed by the application on the leader node and package these changes, forwarding them to Kafka topics. This process is invisible to the application, thus maintaining transparency to the users. The approach minimizes changes to the existing application code, which is crucial for systems with extensive codebases. Instead of rewriting large portions of the application, I/O hooks allow for a non-intrusive integration of new data handling and replication mechanisms. This method ensures that enhancements to data handling and availability can be implemented without disrupting the core functionality of the application or requiring significant code modifications.

Once updates are pushed to Kafka, they are replicated across the system. Replication agents on each node then replay these operations into their respective local ISAM files. Applications on these nodes, whether they are performing read operations or accessing data in other ways, interact with up-to-date ISAM files as if they were interacting directly with a local database.

Change Data Capture

The process described above is an implementation of Change Data Capture (CDC), a technique used to detect and capture changes in a database, enabling real-time data updates across systems. CDC provides powerful real-time data processing capabilities.

This integration offers significant benefits to legacy systems, which can be updated instantly across platforms. By capturing changes directly from the application, CDC minimizes the performance impact on the source database. Kafka's scalable architecture efficiently handles large volumes of data changes, ensuring data consistency by synchronizing changes immediately. This reduces discrepancies, improves scalability, and integrates legacy systems into modern, event-driven architectures without requiring a complete overhaul.

Phase 2: Refactoring write operations

The second phase of setting up a distributed application addresses coordination challenges and includes a method to manage write operations more efficiently. Instead of directly writing to ISAM files, the application creates a command object for each write operation and places it in a queue. This object encapsulates the specific instructions for the operation. This more structured approach centralizes the handling of write operations.

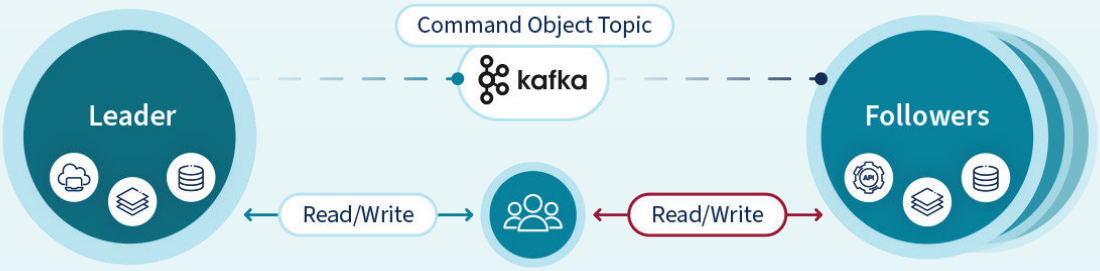


Figure 3. Distributed architecture – Refactoring write operations

The queued commands are executed by whichever node is the current leader. This ensures that all write operations are centralized and managed by a single, authoritative source at any given time. With this separation of concerns, the standard application can run on both leader and follower nodes without modifications.

The regular operational flow of the application remains unaffected by the underlying changes in how write operations are handled. The mechanism for executing write operations is transparent to the users. They interact with the application as usual, unaware of the behind-the-scenes processing.

Coordination in a distributed system

In a distributed system, the node coordination process primarily focuses on leadership management. First, using distributed locks, Consul elects a leader node from the available nodes. If that leader later fails or becomes unavailable, the system needs a protocol to transition smoothly to another leader node.

When a new node is granted leadership, the first steps are to ensure it has the latest data and has fully synchronized with the replication queue. This ensures the new leader has all necessary information to handle requests accurately.

Once a follower node becomes the leader, it starts processing new write operations. It also handles pending operations, ensuring they are executed and then replicating the updates back to the Kafka topic. This creates a continuous loop of data processing and replication across the system.

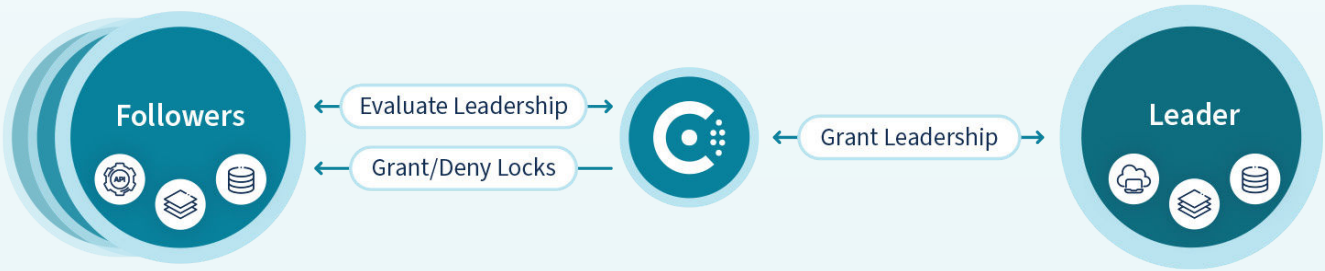


Figure 5. Coordinating leadership changes between nodes

Blue-green deployments

Distributed systems provide advantages to blue-green deployments, where multiple active versions are maintained in the same system. For example, developers need the ability to migrate data schemas live. Traditionally, adding a field to an ISAM file involved taking the system offline, using a migration tool on a single node, and then restarting the system with the new code to utilize the new field. However, in a distributed system with simultaneous versioning, these data migrations occur live. I/O hooks can now intercept read and write operations to perform transformations that previously required offline scripts. Additionally, developers can execute their migration scripts directly on the data local to a node, enabling the support of multiple schema versions concurrently.

Hardware considerations

This solution will likely require an increase in hardware, primarily to enhance redundancy. If an organization's current on-prem data center setup includes a backup node that's rarely used, it might now become part of the active hardware pool. If additional hardware is needed, these units might be smaller than traditional setups because the distributed system design allows for scaling across multiple, coordinated nodes rather than relying on single, large-scale hardware.

This approach not only makes the system more modular and potentially reduces the scale (and possibly the cost) of individual hardware units, but it also simplifies backup processes. Distributed systems can eliminate the need for expensive, high-availability solutions like storage area networks (SANs) traditionally used in DBL applications for maximizing availability. These systems often come with high costs due to the need for expensive synchronous geo-redundancy setups, such as those using dark fiber links between data centers. In our model, the reliance shifts from costly SAN replication to achieving availability through multiple nodes, which reduces the dependency on a single point of failure and expensive infrastructure.

The distributed nature of the application also allows for less sensitivity to latency, providing flexibility in managing latency without significant investment in telecommunications equipment.

Moreover, the system's design aims for high availability with minimal downtime. For direct applications accessed via RDP or SSH, users may need to reconnect if their specific node fails, but the overall system remains robust. The use of APIs and web applications can further obscure the disruption to end users, as the failover and node replacement are managed seamlessly in the background, enhancing user experience and system reliability.

CASE STUDY

Distributing the RCC application with the Streaming Integration Platform

The RCC Resort Management Solution (RCC) is a software application serving all aspects of the timeshare/resort management industry. The application was originally developed in the 1980s as a Unix solution by Resort Computer Corp. At its peak, over 500 timeshare resorts worldwide ran on RCC. Synergex purchased the application in 2010 with the goal of updating the application to serve the current needs of our clients and grow the client base.

Initial state

Before implementing the Streaming Integration Platform, the application had the following characteristics:

- It was designed to function on both Linux and Windows platforms.
- It already supported semi-distributed functionality. This architecture allowed each resort to operate its own server, in addition to a central server located at the home office, enabling a two-node setup.
- It used ISAM files to queue write operations. Within the application, “engines” processed these queues by pulling records from the ISAM files, executing the necessary instructions, and then updating the ISAM files locally on the disk.
- It had an API built on Harmony Core that interfaced with the engines. This API provided direct data access through OData and a blend of Traditional Bridge and native .NET functionalities, facilitating both logical operations and straightforward data access. (Traditional Bridge is a Harmony Core technology that allows traditional Synergy external subroutines and functions to be incorporated into and exposed as part of a Harmony Core web service.)
- It was accessible via SSH on Linux and through RDP on Windows.
- It managed resource coordination using a convention based on ISAM locks, which helped control access and maintain data integrity.
- It operated on .NET.

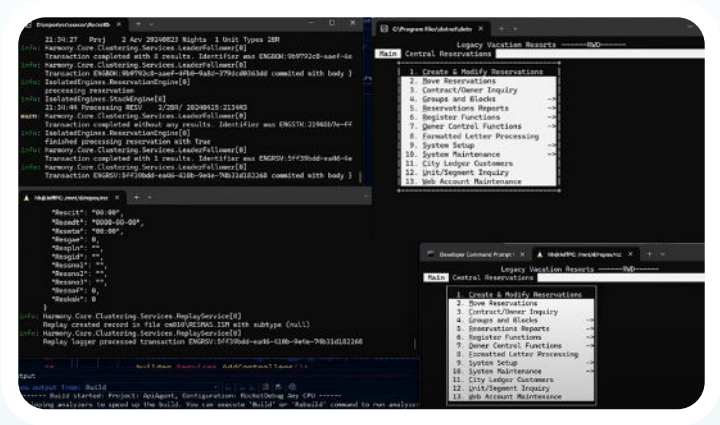
With the Streaming Integration Platform

The updated system includes the following enhancements:

- ✓ The entire system is containerized, with all components consolidated into manageable units.
- ✓ All configuration management is handled by Consul, eliminating manual setup steps for new nodes and automatically updating configurations on existing nodes.
- ✓ The traditional application is accessible through a web browser.
- ✓ Load balancing is enabled across multiple nodes to improve performance and reliability.
- ✓ Existing code from all engines is reused to maintain continuity and leverage proven functionality.
- ✓ Minimal changes were made to the application, to preserve its core functionality and reduce the risk of introducing errors.
- ✓ The transformation prioritized maintaining a very low-latency user experience to ensure that performance improvements were meaningful and would not compromise user satisfaction.

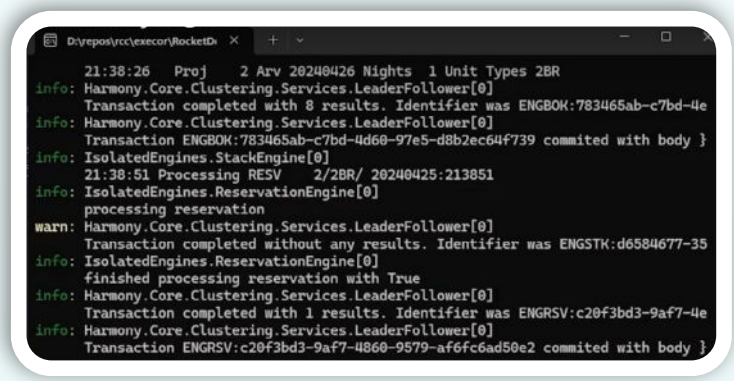
Transformation in action

We tested the new system by running the application on two nodes, one Windows and the other Linux.



We monitored the system via four windows, two showing the application running on each node and two showing log files from each node.

We created a new reservation on Linux, which happened to be our leader node. This created several events, which we viewed in our log files.



Agents on each node got the events and replicated the changes in their ISAM files. We confirmed that the new reservation made it to the follower node (Windows) by viewing it in the RCC application there.

Creating a reservation uses multiple backend engines, for example a booking engine and a reservation engine. In our updated system, these engines continue to do their normal work. The existing business logic doesn't know that we changed how we get the data. Also, the existing application code required minimal modification—since the original architecture already incorporated writing to queues for the engines, we could simply replace the subroutines handling these writes.

Learn more about RCC and how they previously implemented Harmony Core and RESTful web services

Read the Case Study

Conclusion

Distributed computing is fast becoming an essential element of modern system design, providing important advantages like scalability, reliability, resiliency, and efficiency. Synergex's Streaming Integration Platform facilitates transforming DBL applications to distributed systems by connecting ISAM DBMS to Kafka queues. You can leverage your existing code to take advantage of the latest, most effective technologies in a multi-phased approach.

Want to learn more?

For more information, contact Synergex at +1.916.635.7300 or synergy@synergex.com.

The Synergex Professional Services Group

Our Professional Services Group (PSG) can help you evaluate whether a distributed architecture is the right path for your application.

Learn about PSG

Synergex provides software development tools, application integration technologies, and expert consulting services to help enterprise application developers retain their software investment, keep up with advancing technologies, and bring their applications into the future. For over 45 years, Synergex technologies have been the foundation of applications that drive commerce around the world. Every day, millions of users interact with these systems in e-commerce, global logistics, manufacturing, healthcare, and other industries.