International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I4P115

Volume 73 Issue 4, 108-113, April 2025

Original Article

Comparative Analysis of Spark Connect and REST APIs: Architecture, Performance, and Use Cases

Ajinkya Potdar

Senior Technical Program Manager, Texas, USA.

Corresponding Author : ajinkya.potdar@gmail.com

Received: 16 March 2025 Revised: 15 April 2025 Accepted: 23 April 2025 Published: 30 April 2025

Abstract *- This paper presents a comparative analysis of Spark Connect and REST APIs, two prominent technologies facilitating the communication between applications and data sources. Spark Connect, a relatively new feature in Apache Spark, introduces a client-server architecture that enhances several key tenets of the overall solution while supporting multiple programming languages. On the other hand, REST APIs, a traditional solution, remain the backbone of web services, offering a standardized, scalable, and stateless approach to data exchange. This research contributes to the field by exploring the architectural differences, communication protocols, data handling mechanisms, scalability, security implications, and performance trade-offs of both technologies. Spark Connect excels in interactive data analysis and remote Spark development, whereas REST APIs dominate web services, microservices, and mobile app integration. By virtue of the contextual analysis and information, we assess the strengths and limitations of both solutions, enabling organizations to make informed decisions to select the appropriate technology for their use cases. This analysis emphasizes the importance of adopting the right solution to ensure cost and performance optimization in a secure environment.*

Keywords *- Big data, Data analytics, Spark Connect, REST API, Data access methods.*

1. Introduction

In today's big data and distributed computing ecosystem, seamless and efficient communication between applications and data sources is quintessential. Spark Connect [1] and REST APIs are two leading technologies that address this need.

Spark Connect is a new feature in Apache Spark [7] that offers a client-server architecture for interacting with Spark clusters. On the other hand, REST APIs are the cornerstone of web services to facilitate this data exchange.

Given that Spark Connect is new, the objective of this paper is to provide a comparative analysis of Spark Connect and REST APIs, taking a deep dive into their strengths, weaknesses, and use cases, which would help the users choose the right data access method to achieve optimum performance.

2. Spark Connect

Spark Connect is a protocol that enables client applications to interact with a remote Spark server. Similar to how applications connect to a database via a JDBC driver, this protocol allows clients to request operations from a Spark server. This decoupled approach renders many benefits.

2.1. Key characteristics of Spark Connect [3]

2.1.1. Stability

By decoupling client programs from the Spark driver, Spark Connect provides higher fault tolerance. If a client program encounters a memory issue or crashes, it will not affect the entire cluster or other applications running on it.

2.1.2. Upgradeability

Spark drivers can be upgraded without requiring customer application changes to support forward compatibility as well as simplify the maintenance period. Consumers will therefore, benefit from improved features and innovations from Spark without stopping running software.

2.1.3. Debuggability

Through Spark Connect, programmers are authorized to debug code interactively in development and directly from the preferred development environments. It will simplify the development process as well as improve the speed to identify where faults lie with simplicity.

2.1.4. Language Support

Spark Connect supports clients written in other languages, that as Python GO and also allows JVM-based languages to communicate with Spark as well as expose

Spark to more developers. Spark Connect marks a major shift from Spark's traditional monolithic driver architecture to a more scalable and flexible client-server model. This new design makes it easier to deploy and manage Spark applications, particularly in cloud environments and multi-tenant scenarios. Instead of requiring a full Spark installation on the client machine, Spark Connect allows applications to connect to a remote Spark cluster using just a lightweight client library.

This simplifies deployment, making it especially useful for applications running in environments with limited resources.Here is a simplified explanation of how Spark Connect works:

- The client application establishes a connection with the spark server using gRPC, a high-performance open-source framework for remote procedure calls.
- When the client application submits a data frame query, Spark Connect converts it into an unresolved logical plan; this plan outlines the desired operations without specifying how they should be executed; think of it as the high-level blueprint for the query.
- This unresolved logical plan is then encoded using protocol buffers, a language-agnostic mechanism for serializing structured data, and sent to the Spark Server.
- The Spark server receives the plan, optimizes it using sparks catalyst optimizer and executes it on the cluster.
- The results of the query are streamed back to the client application as Apache Arrow record batches a standardized columnar format for in-memory data representation this allows for efficient data transfer and processing.
- The Spark Connect is best suited for many use cases, including Interactive Data analysis Spark embedded into text editors and notebooks and remote development without SSH; it also offers improved utilization of resources and lower client application startup times.

2.2. Limitations of Spark Connect [8]

While Spark Connect offers a more flexible and decoupled approach to interact with Spark clusters, it is essential to consider its limitations when planning its adoption.

2.2.1. Limited API Support

Even though Spark Connect works with many PySpark and Scala APIs, support for some, are still missing. For example, key components like SparkContext and RDD aren't supported in any version of Spark Connect.

2.2.2. Performance Overhead

The client-server model introduces network latency, especially when handling large datasets. Serialization and deserialization add computational overhead, potentially slowing down processing.

3. REST API

REST APIs [5] place an architecture constraint of a standard upon designing web services. REST APIs apply HTTP to communicate and typically mean the exchange of JSON or XML data between a client and a server. A REST API, however, strictly adheres to the principles of REST architecture fundamentals, such as statelessness and decoupling of the server and the client. This affects the scalability and processing of data. REST APIs, by adhering to these principles, are generally more scalable and efficient in handling large datasets.

3.1. Key Characteristics of REST APIs

3.1.1. Uniform Interface

Any request passed on by a REST API must follow the constraints of that API structure. As such, when a client is issuing the request, it must put each piece of information in the position where any other client would put it. A good example is the URL or uniform resource locator for accessing resources using HTTP, which is only one of the several types of uniform resource identifiers with wider scopes.

3.1.2. Client-Server Separation

REST APIs require client and server applications to be completely independent of each other. The client should know nothing more than the complete name of the resource it desires in the virtual space provided by the API, other than that the client and server know nothing of each other except as communicated by API transactions.

3.1.3. Statelessness

Each client request must include all information necessary to handle it and the server does not need to retain any information about the request once it receives it. REST API design does not allow to storage of session data between requests.

3.1.4. Capability

As opposed to the server's per-client statelessness, resources need to be cacheable somewhere in or between the client and server. For the server, for instance, when a particular resource has already been served, it is expected to be accessed again by some source.

3.1.5. Layered System Architecture

REST APIs are constructed based on a layered architecture. This means that the system is separated into different layers, with each layer having a specific set of functionalities and responsibilities. This layering offers several benefits, such as improved scalability, maintainability and security. The layered architecture promotes loose coupling between different components of the system. Each layer speaks only to the layers next to it and does not care about the internals of the other layers. This segregation of concerns enhances the modularity and ease of maintenance of the system. The data access layer changes, for example,

would not necessarily affect the business logic layer or presentation layer if the interfaces between layers remain unchanged. Layered architecture also contributes to improved scalability because it places intermediaries like load balancers or API gateways in the presentation, which can distribute traffic on many servers and handle increased demands without having any effect on the lower-level business logic or data access layer. Further, the layered structure is more secure. It enables the implementation of security controls like authentication and authorization inside specific layers to protect sensitive resources and data without affecting other parts of the system.

3.1.6. Code on Demand

While REST APIs can and do often simply return data to be used by the client, it is necessarily common for code to be delivered to run on the client (For example, Java objects or web applications in JavaScript). If this is the case, then such code may only be run on demand by the client. REST APIs have wide usage in web services integration in mobile app development data access and microservices.

3.2. Limitations of REST APIs

Despite their strengths, REST APIs have some inherent disadvantages:

3.2.1. Over-Fetching and Under-Fetching of Data

REST API will often return more data than the client's needs (over-fetching) or even require multiple requests to retrieve all the needed data (under-fetching). This can lead to increased bandwidth use and slower response times.

3.2.2. Possible Versioning Challenges

As the REST APIs evolve, changes in the API may cause incompatibility with existing applications. This requires careful versioning and management of changes in the API so that existing integrations are not disrupted.

3.2.3. Limitations in Dealing with Real-Time Data

REST APIs are not well equipped to deal with applications requiring real time updates of data. Their request-response paradigm can introduce latency and, hence, are not suited for real-time applications.

4. Comparative Analysis

Table 1. Spark connect vs REST API

Feature	Spark Connect	REST API
Architecture	Client-Server	Client-Server
Communication Protocol	gRPC	HTTP
Data Format	Apache Arrow	JSON, XML, etc
Primary Use Case	Interactive data analysis, remote Spark development	Web Services, data exchange, microservices.
Language Support	Relatively new	Mature and widely adopted.

Both Spark Connect and REST APIs feature client-server design but differ with regard to their communication protocol, data representation and primary usage. Spark Connect is specifically designed to talk to Spark clusters, while REST APIs are purpose-agnostic and most frequently applied for an assortment of web services. It must be noted that not all REST APIs are in accordance with the REST architecture style and such non-RESTful APIs may be limited in scalability as well as in data management.

5. Detailed Comparison

Although the aforementioned table gives a bird's eye view comparison, let's discuss in detail the individual differences between spark connect and REST APIs.

5.1. Communication

Spark Connect uses gRPC, a high-performance framework with advantages in efficiency and language support. REST APIs are layered over HTTP, a mature protocol that is widespread but possibly less efficient in managing data communication.

5.2. Data Handling

Spark Connect leverages Apache Arrow as the native data format - a columnar format designed for efficient in-memory data processing and transfer. While REST APIs can support several data formats, like JSON and XML, depending on the use case, might need extra processing or conversion.

5.3. Job Submission

As Spark Connect can handle and resolve logical plans that are optimized and required by the server, it enables more dynamic and flexible job submission. REST APIs require more structured requests with predefined endpoints and parameters.

5.4. State Management

While REST APIs are intrinsically stateless, Spark Connect leverages Spark’s internal state management framework. This could have implications for applications that require session management or stateful interactions.

5.5. Scalability

Both Spark Connect and REST APIs are scalable but they're scalable in different ways. Spark Connect uses sparsely distributed processing capability for handling big datasets and large request volumes. The decoupled architecture helps client applications to scale independently from the Spark driver in order to prevent resources from competition and increase overall cluster stability.REST APIs scale since they are stateless and cached statelessness makes it possible for servers to serve requests without worrying about retaining session information. Caching also has the effect of increasing scalability because it reduces handling redundant requests.

5.6. Security Implications

Spark Connect relies on Spark's security features [6] for authentication, authorization and encryption. These include Kerberos authentication, SSL encryption for data transport, and Access Control Lists (ACLs). Web UI's correct configuration of these security features is necessary to protect Spark clusters and data features from unauthorized access. REST APIs have no built-in security and therefore the developers ought to implement security practices carefully themselves. They include:

- Transport Layer Security (TLS):
- Encrypting data in transit to protect sensitive data.
- Error handling
- Input data validation
- Rate Limiting
- API Gateways
- Authentication and Authorization: Verifying client identities and granting appropriate permissions.

5.7. Performance Implications

The performance of both Spark Connect and REST APIs are subject to various factors. Network latency and bandwidth will inevitably impact the responsiveness of applications using either method.

However, the efficiency of gRPC should make Spark Connect more robust in environments with high latency or limited bandwidth. The size and complexity of the data being transferred play a crucial role. The benefits of protocol buffers in terms of reduced size and faster parsing become more significant as data structures become larger and more intricate.

Spark Connect can improve performance by minimizing resource utilization. For the purpose of this research, simple and complex processing was done on the Google Cloud Platform for both Spark Connect and REST API to compare the execution times.

Table 2. Spark Connect vs REST API Execution time comparison

Operation	Spark Connect Execution Time	REST API Execution Time
Simple Query	~1-2s	~5-10s
Complex Query (multijoin)	~3-5s	~7-12s
Large Dataframe Processing	~5-8s	~15-20s

5.8. Adoption Rate

There is no reliable data in terms of percentage for adoption of Spark Connect as it is relatively new and is a developing trend within the Apache Spark ecosystem.On the other hand, the industry data shows that REST APIs are overwhelmingly dominant, with approximately 93.4% of API developers using them [9].

6. Use Cases

6.1. Spark Connect

6.1.1. Interactive Data Analysis

Data scientists and analysts can use Spark Connect to interact with Spark clusters from notebooks and IDEs, enabling efficient data exploration and analysis and also building interactive dashboards that display real-time or near real-time insights from large datasets.

6.1.2. Remote Spark Development

With Spark Connect, developers can build Spark applications on remote clusters by embedding Spark in text editors without needing SSH.

6.1.3. Building Language-Agnostic Spark clients

Spark Connect supports clients written in various languages, expanding the reach of Spark beyond JVM-based languages.

6.2. REST API

6.2.1. Web Services Integration

REST APIs ensure interoperability between various web services, allowing data to be seamlessly exchanged and integrated.

6.2.2. Mobile Application Development

Mobile applications can make use of REST APIs to gain access to server data and capabilities enabling connected mobile experiences to be created.

6.2.3. Data Access

REST APIs provide a standard way of accessing data from most sources, including databases, cloud storage and social networks.

6.2.4. Microservices

REST APIs are usually used to integrate and communicate between microservices within a distributed system.

6.2.5. Internet of Things (IoT) [10]

IoT devices use REST APIs to exchange data with servers, enabling remote monitoring, control, and data collection. This is imperative for applications like smart homes and industrial automation.

7. Industry-Specific Adoption of Spark Connect [11]

Joom, a global e-commerce site with tens of millions of affordable products for an enormous user base, relies heavily on Apache spark-based data analysis with over 1000 custom-built spark applications and a Kubernetes/EKS-based data platform; Joom's utilization of spark connect offers insightful feedback on its performance and problems in such a large production environment of this scale due to the size and

complexity of Joom's Spark infrastructure that their experience has high applicability in learning the weaknesses and strengths of Spark Connect in high-demand environments. The primary motivation for Joom to adopt Spark Connect was the desire for a more efficient and scalable Spark infrastructure. In moving toward a shared Spark Connect server, Joom aimed to move away from the limitations of maintaining a high volume of individual Spark applications, with the end goal of reducing resource consumption and optimizing utilization of their Spark cluster. Their main use case is to execute their large collection of internal Spark applications on a centralized Spark Connect server. In order to manage this diverse workload, Joom created a smart system that automatically determines whether to start a given application via Spark Connect or as a regular, stand-alone Spark application based on its historical resource usage patterns.

This decision-making automates considering metrics such as Total Task Time, Shuffle Write and Disk Spill for dynamic workload management. There have been multiple benefits for Joom since it adopted Spark Connect. It has recorded a significant reduction of client application memory footprint as these applications do not require their own Spark Driver. Also, the shared nature of the Spark Connect server has led to more efficient usage of executor resources as they could be shared dynamically and allocated to different concurrent client applications. Application startup times have also improved as a result of Spark Driver on the server already running and being instantly available for processing the client request. From the management perspective, Joom has simplified the deployment and maintenance of client applications by maintaining only a single JAR file for each application that can be used in both Spark Connect and traditional Spark modes.

Despite these advantages, Joom has also experienced several challenges in its implementation of Spark Connect certain spark features such as spark Session.sessionState.catalog and sparkSession.Sparkcontext.hadoop Configuration, was discovered to be incompatible with Spark Connect. Joom also had to avoid running very long-running and very resource-intensive applications on Spark Connect due to the stability limitations of the shared server. Another issue is managing very high concurrency loads on a Spark Connect server, particularly in light of the efficiency of the underlying Spark Task Scheduler's resource allocation. The use of third-party Spark components also had the associated risk of compatibility with Spark Connect architecture. Specifically, Joom experienced some instability issues with Spark Connect version 3.5, where client application jobs would randomly hang, necessitating the implementation of mitigation processes. Additionally, utilization of the useFetchcache=true property had the risk of increased disk space utilization on executors. Getting some Spark applications to function correctly as Spark Connect clients, especially those that are Third-party component-based and not designed with Spark Connect in consideration, was also a non-trivial task. These experiences indicate that while Spark Connect offers very important benefits, it requires diligent attention to application compatibility resource management and potential stability issues, especially in its early releases.

8. Future Trends

Legacy APIs, as much as they form the foundation of contemporary software development, face certain constraints in addressing the dynamic and increasingly complex requirements of data consumers. Such challenges are driving the development and adoption of new API protocols and interfaces that provide improved efficiency, flexibility, and intelligent data exchange. Protocols like GraphQL and gRPC are coming up as strong alternatives to REST. GraphQL enables clients to request specific data, thus minimizing issues of over-fetching and under-fetching, and operates through a single endpoint for all queries. The gRPC, on the other hand, is a high-performance, open-source framework that excels in environments where speed and efficiency are of foremost importance, particularly in microservices architectures and real-time systems - thanks to its support for bidirectional streaming and efficient communication.

In addition, Artificial Intelligence is radically changing the future of API development. AI-based tools are being created to automate various stages of the API lifecycle, including testing for bugs and inconsistencies, generating documentation through natural language processing, and streamlining deployment processes. Additionally, AI fortifies API security by analyzing usage patterns, identifying anomalies, and rapidly addressing potential threats. Looking forward, a shift towards more adaptable and intelligent interfaces are expected. AI interfaces, powered by advanced AI models, may transcend the rigid frameworks of traditional APIs, enabling them to comprehend and interpret natural language queries and respond dynamically based on the underlying intent of a request. This transformation would render data access more intuitive and accessible to a broader audience.

9. Conclusion

Spark Connect and REST APIs are valuable technologies for enabling communication and data exchange in distributed computing environments. Spark Connect offers a new approach to interact with Spark clusters, while REST APIs remain a popular choice for web services and data integration. Both technologies are expected to continue evolving and play significant roles in the future of big data processing and application development. This comparative analysis highlights the distinct strengths and use cases of Spark Connect and REST APIs. Spark Connect excels in stability, upgradability and debuggability, making it ideal for complex Spark applications and interactive data analysis. Its

decoupled architecture and language support enhance flexibility and resource efficiency. Conversely, REST APIs prioritize simplicity, flexibility, and cost-effectiveness, making them well-suited for web services, data exchange, and microservices.

Their widespread adoption and mature ecosystem ensure broad capability and ease of use. In conclusion, while both technologies have their own strengths and limitations, they would render a profound influence on core architectural tenets such as performance, security, and cost. This analysis underpins the importance of selecting the right technology for optimal outcomes and alignment with the future vision in distributed landscapes.

Conflicts of Interest

There is no financial or monetary gain for the author. The views expressed in this paper solely belong to the author and should not be attributable to his employer. This is a disclosure that the author's employer might have used Spark Connect.

Funding Statement

The research paper has been self-funded.

References

[1] High-Level Spark Connect Architecture, Apache Spark. [Online]. Available: https://spark.apache.org/spark-connect/#:~:text=Spark%20Connect%20is%20a%20protocol,JDBC%20driver%20%2D%20a%20query%20spark/

[2] The Spark Connect Overview - Spark 3.5.4 Documentation, Apache Spark. [Online]. Available: https://spark.apache.org/docs/3.5.4/spark-connect-overview.html

[3] Stefania Leone et al., Introducing Spark Connect - The Power of Apache Spark, Everywhere – Databricks, 2022. [Online] Available: https://www.databricks.com/blog/2022/07/07/introducing-spark-connect-the-power-of-apache-spark-everywhere.html

[4] What is REST API? IBM, 2025. [Online]. Available: https://www.ibm.com/think/topics/rest-apis#:~:text=A%20REST%20API%20(also%20called,transfer%20(REST)%20architectural%20style/

[5] The Mulesoft Website. [Online]. Available: https://www.mulesoft.com/api/rest/top-3-benefits-of-rest-apis/

[6] The Spark Documentation on Security. [Online]. Available: https://spark.apache.org/docs/latest/security.html

[7] Bill Chambers, and Matei Zaharia, *Spark: The Definitive Guide: Big Data Processing Made Simple*, O'Reilly Media, pp. 1-606, 2018. [Google Scholar] [Publisher Link]

[8] The Spark Documentation on Security. [Online]. Available: https://docs.cloudera.com/data-engineering/1.5.4/spark-connect-sessions/topics/cde-spark-connect-session.html

[9] J. Simpson, 20 Impressive API Economy Statistics, 2022. [Online]. Available: https://nordicapis.com/20-impressive-api-economy-statistics/#:~:text=17.,a%20staggering%2093.4%25%20adoption%20rate/

[10] How to use the IoT Central REST API to Control Devices, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-central/core/howto-control-devices-with-rest-api/

[11] Sergey Kotlov, Adopting Spark Connect, 2024. [Online]. Available: https://towardsdatascience.com/adopting-spark-connect-cdd6de69fa98/