International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Real-Time Data Streaming in Financial Services: Tools, Applications, and Implications

Pavan Kumar Mantha

Prinicpal Data Engineer Lead At Synchrony

Abstract

Financial services firms encounter unparalleled expectations for agility, personalization, and compliance throughout the digital transformation age. Real-time data streaming, facilitated by technologies like Apache Kafka, Apache Spark Structured Streaming, Apache Flink, and Apache Pulsar, has become crucial for handling high-volume, high-velocity data streams. This article assesses these technologies, their architectural intricacies, performance attributes, and practical applications in trading, fraud detection, credit scoring, and compliance. The analysis utilizes industry documentation and scholarly sources to assist practitioners in implementing strategic real-time streaming solutions.

Keywords: Real-Time Data Streaming, Apache Kafka, Apache Spark Structured Streaming, Apache Flink, Apache Pulsar, Financial Services, Fraud Detection, Credit Scoring

I. Introduction

Historically, financial services relied on overnight batch processing for risk reporting and transaction monitoring. This methodology is currently insufficient for institutions requiring sub-second processing.High-frequency trading (HFT) exemplifies the significance of latency: a 1-millisecond edge might translate to $100 million per year for trading businesses [1]. Retail banking requires immediate responsiveness for consumer transactions and alerts. Additionally, legal frameworks such as MiFID II and GDPR underscore the necessity for immediate transaction auditability and data transparency [2]. IDC forecasts a 21.5% compound annual growth rate for the stream processing industry from 2022 to 2028, underscoring the increasing usage of real-time streaming.

II. Use Cases and Business Impact

A. Algorithmic Trading

Real-time streaming enables high-frequency trading (HFT), as platforms assimilate real market information while sustaining latency in the microsecond to millisecond range. A prominent exchange decreased cross-region synchronization latency from 2 seconds to below 50 ms utilizing Apache Kafka, markedly improving market efficiency and liquidity [3].

B.Fraud Detection

Streaming frameworks provide real-time transaction pattern analysis, substantially lowering fraud losses. Capital One utilized Kafka to realize an annual savings of $150 per customer by detecting

fraudulent transactions in real-time [4]. Likewise, Flink's Complex Event Processing (CEP) enabled a European bank to detect fraud with 20% greater efficacy, resulting in yearly savings of €11 million [5].

C. Real-Time Credit Scoring

Dynamic credit scoring incorporates real-time income and transactional information, enabling prompt loan choices. Experian's streaming-based models significantly enhanced underwriting precision for thin-file clients [6].

D. Customer Analytics

Real-time analytics via Spark Structured Streaming enhanced retention and revenue by instantaneous, tailored interactions. A U.S. credit institution lowered turnover by 28% and tripled marketing ROI by promptly addressing at-risk consumers.

E. Adherence to Regulatory Standards

Streaming enables expedited compliance reporting. ING Bank deployed Flink for real-time anti-money laundering detection, significantly improving its compliance capacity [7].

III. Overview of Streaming Technologies

Several open-source platforms have emerged as the de facto choices for building streaming data infrastructure. This section presents a comprehensive architectural analysis of four prominent technologies: Apache Kafka, Apache Spark Structured Streaming, Apache Flink, and Apache Pulsar, examining their fundamental design, advantages, and constraints in relation to the requirements of financial services.

A. Apache Kafka - The Messaging Backbone

Kafka is a scalable, distributed messaging system that can handle millions of messages per second with millisecond latency, making it a suitable central data backbone. Kafka's design fundamentally revolves around the concept of topics, which are categories of messages distributed across a cluster of brokers. Every partition functions as an append-only log file on disk, wherein new messages are recorded progressively. Kafka's architecture intentionally prioritizes sequential disk I/O and operating system-level enhancements (like page caching and zero-copy transmission) to attain elevated throughput [15]. Producer writes are aggregated and immediately stored in the OS cache (and then flushed to disk according to configurable policies), so avoiding costly fsync calls for each message. Consumers sequentially read messages, monitoring their own offsets (positions in the log),which decouples consumer performance from the broker – multiple consumers can read at their own pace without impacting each other. This log-centric architecture enables Kafka to consistently process millions of messages per second on commodity hardware. Its main limitation is the lack of built-in processing, requiring external analytical frameworks.

B. Apache Spark Structured Streaming - Micro-Batch Analytics

Apache Spark Structured Streaming is the streaming component of Apache Spark, a unified analytics engine. Spark employs micro-batch processing, appropriate for situations where a modest increase in latency (100–500 ms) is permissible. Incoming data is perceived as new rows added to an unbounded table, while users write queries (in SQL or DataFrame API) as if executing on a static table; the Spark engine subsequently turns this into an incremental execution that updates results in brief time intervals (batches). By default, Spark Structured Streaming aggregates data for a short interval (e.g., 100 milliseconds or 1 second), processes it as a mini-batch, outputs the findings, and then reiterates the process[8]. This gives the illusion of continuous query processing while leveraging the rich optimization techniques of Spark's Catalyst engine and the fault-tolerance of batch processing. Although Spark excels in analytics, it is less suited for ultra-low latency applications.

C. *Apache Flink — True Stream Processing*

Apache Flink is a distributed stream processing engine engineered with a native event-driven architecture for low-latency, high-throughput processing. Unlike Spark's micro-batching approach, Flink performs computations as a continuous flow of events, establishing it as an actual streaming engine. The Flink runtime comprises JobManagers (masters) and TaskManagers (workers) that collaboratively execute a directed acyclic graph of operators (such as map, filter, window, join, etc.) on streaming data. Upon deployment of a Flink task, records commence streaming through the operators instantaneously upon arrival, devoid of artificial batching. Flink facilitates real continuous event-driven streaming with latencies often under 10 ms, essential for low-latency financial applications [9]. Its intricacy, however, poses operational difficulties.

D. Apache Pulsar — Unified Streaming and Messaging

Apache Pulsar is a contemporary distributed messaging system, open-sourced by Yahoo in 2016, that is becoming recognized as a complement or alternative to Kafka. Pulsar's architecture is unique in that it separates the messaging serving layer from the storage layer. It comprises stateless brokers that manage producers and consumers, as well as routing, and a cluster of Apache BookKeeper nodes, referred to as "bookies," which offer durable log storage for messages. Pulsar's architecture decouples messaging from storage, providing fast throughput, low latency (~5ms), multi-tenancy, and geo-replication, advantageous for varied financial applications [10].

E. Other Emerging Technologies

In addition to the four principal platforms mentioned, several developing technologies in the streaming sector are being monitored by financial institutions.

- Redpanda (Vectorized Inc.): Redpanda is a contemporary streaming data platform that is API-compatible with Kafka serving as a direct substitute for Kafka clients, while is developed in C++ with an emphasis on performance and simplicity. It omits ZooKeeper and employs a thread-per-core architecture utilizing the Seastar framework. Redpanda exhibits remarkably low latency (single-digit milliseconds) and great throughput, and due to its compatibility with the Kafka

protocol, it can be utilized as an alternative without necessitating modifications to application code. Financial institutions seeking to mitigate Kafka's operational complexities have been exploring Redpanda for applications like as high-frequency trading data distribution, due to its potential for ultra-low latency. Nonetheless, as a more recent product, it has not yet achieved the same level of adoption as Kafka or Flink.

- Kafka Streams and ksqlDB:As part of the Kafka ecosystem, Kafka Streams (a Java library) and ksqlDB (a SQL layer on Kafka Streams) facilitate processing without the need for a separate cluster deployment, such as Spark or Flink. Kafka Streams is a lightweight library designed for integration into applications, enabling data processing from Kafka topics with exactly-one semantics. Numerous banks have employed Kafka Streams for transformations or stream joins, as it leverages the existing Kafka cluster.

  Although not an independent platform, it is a crucial component in the ecosystem, frequently utilized alongside Kafka to construct comprehensive streaming pipelines solely on the Kafka infrastructure.

- Cloud-Native Streaming Services: Cloud providers deliver fully-managed streaming services, such as Amazon Kinesis Data Streams, Google Cloud Pub/Sub and Azure Event Hubs. These are proprietary yet frequently exhibit Kafka-like semantics. Certain financial institutions, particularly smaller banks or fintechs lacking the inclination to operate their own clusters, utilize these services for specific applications (e.g., streaming stock quotes through. Managed service). Nonetheless, apprehensions around data residency and governance frequently restrict their application in heavily regulated contexts. After evaluating the solutions, Table I presents a concise comparison of Kafka, Spark, Flink, and Pulsar based on essential technical criteria pertinent to financial streaming applications.

TABLE I – COMPARATIVE FEATURES OF STREAMING PLATFORMS

Feature	**Apache Kafka**	**Spark Str. Streaming**	**Apache Flink**	**Apache Pulsar**
Processing Model	Log-based pub/sub (pull)[2]	Micro-batch (structured)[3]	Native event stream (continuous)[9]	Pub/Sub + Queues (segmented log)[12]
Typical End-to-End Latency	~5–15 ms(at p99 under load)[11]	100–500 ms (depends on batch interval) [3]	~5–10 ms (event-time processing)[9]	~5 ms publish latency (single msg); <10 ms p99[13]
Throughput	High: Millions of msgs/sec (10+ MB/s per partition) scales with brokers/partitions. [2]	High: Millions of msgs/sec, but throughput can drop with very low latency settings [3].	High: Millions of events/sec; proven at billions/sec at scale[14]	High: Comparable to Kafka; 3.2 GB/s observed (60% >Kafka in one benchmark)[13]
State	External to Kafka	Checkpointed state in	Keyed state with	Managed cursors and

Feature	Apache Kafka	Spark Str. Streaming	Apache Flink	Apache Pulsar
Management	(use Kafka Streams or consumer state stores) [2].	memory or disk; uses Spark engine's fault tolerance [3].	in-memory or RocksDB backend; checkpointed to durable storage for exactly-once[9].	Pulsar Functions state (stored in internal topics) for lightweight processing[12].
Fault Tolerance	Replication of logs (configurable); client retries for consumption [2].	Checkpointing and replay (micro-batch replay on failure); exactly-once output with transactional sinks [3].	Checkpointing + stream replay enable exactly-once processing. Automatic failover of tasks via JobManage.	Replication of data in BookKeeper (configurable quorum); cursors track acknowledgments for guaranteed delivery[12].
Strengths	Ubiquity, ecosystem, very high throughput, durability, log replayability [2].	Unified batch/stream API, rich analytics (SQL, ML), integration with data lakes/warehouses [3].	Ultra-low latency, event-time handling, complex event processing, large state with exactly-once [7].	Low latency + high throughput, multi-tenancy, geo-replication, unified pub-sub & queue model[12].
Limitations	No built-in processing; needs external compute. Operations (partitioning, scaling) add complexity [2].	Higher latency due to micro-batch; not ideal for sub-50 ms needs. Requires Spark cluster (heavyweight for simple tasks) [3].	Steeper learning curve; requires careful tuning. Smaller ecosystem than Spark. Typically used with Kafka (needs an external source) [7].	Newer project (smaller ecosystem); ops complexity of broker+bookkeeper; less tooling than Kafka (fewer connectors)[13].

IV. COMPARITIVE PERFORMANCE AND BENCHMARKING

Performance is essential for financial streaming platforms. Selection of technology frequently depends on its capacity to satisfy the latency and throughput demands of a certain application. We will evaluate the systems based on several critical performance metrics, referencing benchmarks when applicable.

A. Latency

As mentioned earlier, Apache Flink and Pulsar typically provide superior performance in achieving minimal end-to-end latency. Flink's event-driven engine experiencenegligible delays beyond network transfer, with numerous users reporting latencies in the single-digit millisecond range for Flink pipelines doing moderate tasks. Pulsar has also exhibited publish latencies in the 5 ms range at p99 for high-throughput applications, attributed to its distributed log and caching architecture. Kafka can have remarkably low latencies (about 5–20 ms) for message production and consumption, particularly with

suitable optimization (e.g., effective use of batching and compression). In a direct comparison conducted by Confluent, Kafka consistently exhibited lower p99 latencies than Pulsar at equivalent throughput levels (Kafka p99 ~5 ms against Pulsar ~25 ms @ 200k msg/s).

Nonetheless, that disparity occurred under particular situations (such as entirely durable writes), and advocates of Pulsar have demonstrated instances when Pulsar's latencies are comparable to or lower than those of its competitors due to its asynchronous writing. Spark Structured Streaming, because to its micro-batch architecture, typically exhibits increased latency. A typical design, such as a 100 ms batch interval, results in approximately 100–300 ms latency, with aggregation windows or backpressure potentially increasing that latency further. Spark's continuous mode can theoretically achieve latencies of approximately 1 to 10 milliseconds, nonetheless, its functionality is constrained.

In conclusion, for sub-10 ms latency requirements, Flink (or maybe Pulsar Functions/Kafka Streams) is preferred, while Spark is generally utilized when a latency of approximately 0.1–1 second is permissible. Kafka's function is primarily to transport rather than to process; hence, Kafka introduces minimal latency (a few milliseconds) to the pipeline, whereas the selection of the processing engine (Spark versus Flink) significantly influences the end-to-end delay.

D. Throughput

Each of the four platforms is engineered for elevated throughput; nevertheless, their limitations and scalability characteristics vary. Kafka is recognized for its capacity to manage substantial message volumes, capable of ingesting millions of messages per second. Published testing indicate that Kafka clusters can achieve write speeds above 1 GB/s, equivalent to around 1 million 1KB messages per second, and exhibit linear scalability with the addition of brokers. Kafka's throughput is enhanced through batching, as producers and consumers aggregate data to mitigate overhead; however, excessive batching may introduce latency, necessitating a careful balance.streamnative.io Pulsar throughput is comparable; one benchmark demonstrated 1.6 GB/s write throughput on a 3-node Pulsar cluster without journaling (about 0.8 GB/s with full durability), in contrast to Kafka's approximately 1.1 GB/s on equivalent hardware. This suggests that Pulsar can marginally exceed Kafka in overall throughput when set for comparable durability, mostly owing to its capacity to concurrently write to many BookKeeper nodes and its more assertive client-side batching. Flink and Spark, as processing engines, generally depend on some underlying messaging/logging system for data ingestion, commonly Kafka or Pulsar, hence their performance is frequently constrained by Kafka/Pulsar.

Nonetheless, the engine must manage the event processing pace independently. Flink has demonstrated remarkable scalability, exemplified by Alibaba's processing of 4 billion events per second, equating to an extraordinary throughput of 32 Gbps, achieved with a substantial cluster[14]. Flink can accommodate extremely high input rates due to its efficient event loop and network stack; nevertheless, the user's job parallelism and logical complexity may limit practical throughput. Spark can manage elevated rates; however, if the input exceeds the batch interval significantly, it may queue batches, resulting in increased delay. Typically, Kafka and Pulsar are evaluated at rates exceeding millions of events per second on moderate clusters, whereas Flink and Spark can achieve comparable throughput, contingent upon sufficient resources (CPU, memory) and appropriate scalability (parallelism, partitioning).

Numerous financial applications necessitate a throughput ranging from thousands to hundreds of thousands of events per second, a capacity that all these systems can readily accommodate; however, it is in the extreme scenarios, such as tick-by-tick market feeds in extensive marketplaces or enterprise-wide event aggregation, when meticulous benchmarking becomes essential.

Scalability and Elasticity: Kafka achieves scalability by incorporating additional brokers and redistributing partitions, so relocating data to new brokers for balance. The method is manual or semi-manual and can become burdensome when data volumes are substantial, particularly when transferring partition logs. Pulsar's architecture has greater elasticity, allowing for the independent addition of bookies to expand storage and brokers to enhance client management; it can also dynamically distribute topic ownership. In cloud contexts, Pulsar may manage incremental growth more effectively, and it features tiered storage for older data, facilitating near-infinite log retention without overloading hot storage. Spark and Flink achieve scalability by augmenting the number of worker nodes and enhancing job parallelism (for Spark, this may involve expanding the number of shuffle partitions; for Flink, it entails boosting task parallelism).

Flink facilitates the scaling of stateful jobs through a "savepoint" mechanism: one captures a snapshot and subsequently redeploys the job with increased parallelism from that snapshot - somewhat intricate yet achievable. Spark streaming operations can be grown by halting and restarting with increased parallelism, as the new execution will resume from the last checkpoint.

Neither is inherently autoscaling by default; however, both are compatible with resource managers (Kubernetes, YARN) to facilitate elasticity. Regarding dynamic workload surges, Kafka and Pulsar possess mechanisms for backpressure. Kafka clients employ a pull model and can intentionally decelerate consumers; Pulsar's broker will offload messages to disk if consumers exhibit sluggishness. Spark may postpone batches if overloaded (processing time beyond batch interval), resulting in increased latency.

Flink incorporates inherent backpressure monitoring; if sources generate data too rapidly, it transmits backpressure upstream to prevent out-of-memory errors. However, persistent backpressure signifies a requirement for scaling or improvement of the pipeline.

Selecting a streaming platform in finance typically necessitates evaluating performance attributes in relation to specific use case needs. A trade pricing system requiring microsecond latency may opt for a custom in-memory solution or FPGA instead of Kafka or Pulsar. Conversely, for millisecond-level latency with substantial throughput, Flink on an optimally configured Kafka pipeline may be adequate; certain trading firms utilize Flink for real-time analytics on marginally delayed data, while core matching is executed in specialized systems. For comprehensive event collection across the organization, such as monitoring every customer interaction and transaction, Kafka's established durability and extensive connection ecosystem may surpass the advantages offered by Pulsar's more recent features. A cloud-native fintech may choose Pulsar to unify queues and publish-subscribe mechanisms, leveraging its functionalities for rapid processing but sacrificing Kafka's maturity for more operational flexibility.

Ultimately, it is recommended to benchmark against one's specific workload. Industry benchmarks provide a comprehensive overview: Kafka and Pulsar are comparable, with each excelling in distinct measures (Kafka frequently demonstrates marginally lower tail latency in evaluations conducted by its vendor)[13].

Pulsar demonstrates superior aggregate throughput in tests conducted by its provider. The ongoing arguments between Spark and Flink constantly demonstrate Flink's superiority in latency and efficiency for streaming applications, while Spark excels in providing a unified engine and user-friendliness for current Spark user. Numerous financial organizations utilize various tools concurrently, leveraging their respective advantages (e.g., Kafka as a resilient buffer and integration hub, Flink for real-time analytics, and Spark for extensive batch analytics on the stored data).

V. Real-World Case Studies

To substantiate the comparison with tangible results, we emphasize various real-world implementations of streaming technologies in the financial services sector. These case studies demonstrate the problems encountered, the selected platform, and the achievements attained:

A. *Kafka in Real-Time Market Data Aggregation*

A leading global stock exchange encountered a 2-second delay in synchronizing its order book data among its data centers in New York, London, and Tokyo. This latency was obstructing arbitrageurs and compromising total market equity. The exchange utilized Apache Kafka as a foundational element for real-time trade intake and cross-site replication. Trade events and order changes are published to Kafka topics and concurrently consumed in each region, with Kafka's broker replication ensuring that all sites receive complete data. The outcome was a significant enhancement — they attained sub-50 ms cross-region data propagation, so establishing a worldwide integrated order book in near real-time. This enhancement not only eradicated prospects for latency arbitrage but also augmented the exchange's appeal to high-frequency traders.

Currently, Kafka is utilized by numerous exchanges and banks for the delivery of market data and the aggregation of ticks, offering a dependable, high-velocity conduit that supplies pricing engines and analytical models.

B. Spark Structured Streaming in Real-Time Customer Analytics

A U.S.-based credit card issuer with millions of digitally engaged customers observed an increase in customer attrition, particularly among younger, mobile-centric users. To address this, they invested in a real-time customer analytics platform utilizing Spark Structured Streaming. Data from mobile app interactions, website clicks, and transaction history is incessantly processed through a Spark pipeline. The pipeline integrates these streams with static customer data (demographics, product holdings) and use machine learning models (trained using Spark MLlib) to calculate a churn risk score in real time for each user session. Should the score surpass a predetermined threshold, the system initiates an instant retention action, such as dispatching a targeted offer or notifying a sales representative.

After more than a year of utilization, this streaming personalization solution resulted in a 28% decrease in monthly churn rate and enhanced marketing campaign ROI by 3.5 times, enabling the organization to intervene at optimal moments. They choose Spark because to their data science team's prior familiarity with it, enabling the reuse of existing batch models with little code modifications. Although the second-level latency was satisfactory for this application, the capacity to manage intricate joins and interface with their data lake for the preservation of historical information was essential. This scenario highlights that

streaming is not solely focused on speed, but also on facilitating new business processes, such as retaining clients in real-time instead of retrospectively analyzing their departure.

C. Flink for Fraud Detection in Cross-Border Payments

A European digital bank engaged in international transactions discovered that certain fraudulent patterns were evading their batch-based detection tools. For instance, identifying a planned fraud involving numerous low-value transactions dispersed across accounts and countries was excessively time-consuming with daily data. They developed a fraud detection engine utilizing Apache Flink and its CEP library. The engine processes transaction streams from all regions (via Kafka topics) and identifies patterns such as swift fund transfers between identical IP addresses across many accounts, or sequences of activities suggestive of mule account networks. Through real-time event analysis and state maintenance (including running counts of failures and geolocation discrepancies), the Flink job effectively identified suspicious activity within seconds and transferred it to intervention systems. The result was a 20% increase in detected frauds (previously overlooked) and an expected annual decrease of €11 million in fraud losses for the bank.

An illustrative pattern involved identifying when a single device attempted to log in to 10 distinct accounts during a 5-minute timeframe; this was detected, and the accounts were secured within approximately 2 seconds of the 10th attempt. Flink's exactly-once processing guarantees that, despite potential pipeline failures, events will neither be omitted nor double-counted. The efficacy of this solution has prompted the bank to evaluate Flink for more real-time risk management applications. The bank's engineers identified Flink's proficient management of out-of-order events and its capacity to sustain extensive state (for user session data) as primary factors in their decision.

D. Pulsar in Trade Reporting and Multi-Tenant Event Streaming

A prominent worldwide investment bank engaged in trading, retail banking, and wealth management aimed to consolidate its event streaming infrastructure. Various teams operated distinct Kafka and RabbitMQ clusters, resulting in redundancy and elevated operational overhead. The bank implemented Apache Pulsar to establish a cohesive event bus for many departments. Their interest in Pulsar stemmed from its multi-tenancy feature, allowing each team (tenant) to maintain separate namespaces and quotas, as well as its integrated geo-replication, which facilitated real-time replication of essential topics, such as risk alerts, between their primary data center and a disaster recovery site.

The initial application of Pulsar involved a regulatory transaction reporting pipeline, where trades from many desks (equities, fixed income, etc.) are published to Pulsar topics immediately upon occurrence. A collection of Pulsar Functions performs lightweight changes by supplementing data with reference information, after which the events are processed by a reporting service that collects and transmits them to regulators within the mandated 1-minute SLA.

Pulsar's durable storage (BookKeeper) instilled assurance that no messages would be forfeited, and the system adeptly managed the volatile dynamics of trading (significant surges at market open/close) without losing momentum – the separation of storage ensured that as long as bookkeepers could record the data, brokers could accommodate spikes by writing to disk and delivering to consumers as they reconciled.

The pilot demonstrated favorable outcomes: the consolidated Pulsar cluster managed three distinct workload types (market data pub-sub, internal queueing for middle-office tasks, and the trade reporting stream) simultaneously, achieving p99 latencies below 10 ms and throughput in the hundreds of thousands of events per second, all on a modest 5-broker, 10-bookie cluster. Operators observed that the implementation of a singular system for monitoring, equipped with an integrated dashboard for topic statistics, constituted a significant enhancement. The primary problem involved educating many teams on Pulsar's API and converting certain existing Kafka consumers, which was alleviated by employing Kafka-on-Pulsar protocol compatibility during a transitional phase. This scenario suggests that Pulsar's usefulness in large financial organizations may lie in its ability to simplify designs, particularly when diverse messaging patterns are required, all while satisfying performance requirements[13].

These case studies illustrate concrete advantages: ultra-low latency facilitating novel trading techniques, real-time analytics conserving resources and finances, and architectural consolidation enhancing efficiency. They emphasize that the selection of technology is contingent upon context—each organization opted for the tool that most effectively aligned with their inherent strengths and requirements (Kafka for efficient log distribution, Spark for integrated analytics, Flink for intricate real-time pattern recognition, Pulsar for versatile infrastructure). In numerous cases, these systems operate together rather than independently.

VI. Current Adoption Trends and Industry Statistics

Real-time streaming has transitioned from a specialized to a fundamental element of financial IT infrastructure. Several industrial trends and figures exemplify this expansion:

A. Widespread Adoption Across Subsectors

A 2022 Gartner report on data streaming in finance indicates that more than fifty percent of banks and insurance firms have implemented stream processing in production, with several others in pilot stages [7]. Gartner's analysis indicated that in financial services, banking constitutes approximately 45% of streaming analytics use cases, capital markets around 30%, insurance/insurtech roughly 15%, and fintech startups about 10% (based on deployment volume). This indicates that banks are at the forefront of adoption, perhaps owing to greater financial resources and the necessity for immediate customer engagement, but capital markets significantly influence this trend due to the latency-sensitive characteristics of trading applications. FinTech companies, although having a smaller market share, frequently implement streaming from inception, leveraging it as a competitive edge over established firms.

B. Market Growth

The market for streaming data technology is expanding rapidly. IDC projects the stream processing software market will expand at a compound annual growth rate of 21.5% from 2022 to 2028, reaching several billion dollars[16].This is more rapid than numerous other facets of analytics. Investments are driven by escalating data volumes (IoT, digital banking channels, etc.) and the strategic necessity to utilize data in real time. Numerous financial organizations identify "real-time analytics" as a primary technological need. A 2023 IDC survey of banks revealed that 67% of participants intended to augment expenditures on real-time data pipelines and event-driven systems.

C. *Vendor and Community Ecosystem*

The advancement of open source technologies like as Kafka and Flink has resulted in a robust ecosystem of vendors and cloud services. Confluent, the primary vendor of Kafka, conducted an initial public offering in 2021, indicating the demand for Kafka-centric streaming services. They reported more than 100% year-over-year growth in cloud service utilization, predominantly from financial sector clients employing Kafka in hybrid cloud situations[17]. Data Artisans (now Ververica, acquired by Alibaba) and AWS Kinesis Data Analytics, which utilizes Flink for SQL on streams, have rendered Flink accessible in a managed format, hence promoting its usage. Apache Pulsar lags in business usage however is experiencing growth; StreamNative, Pulsar's vendor, has observed increased utilization in the financial and telecom sectors where multi-tenancy is significant. A notable trend is convergence and cross-pollination: Kafka has included functionality previously found in Pulsar, such as tiered storage in newer versions, while Pulsar has introduced Kafka API compatibility to facilitate migration. This competition is catalyzing rapid enhancements, which benefit end users.

D. *Use Case Trends*

Initially, streaming in finance world was focused on trade and fraud detection. It has now broadened to encompass customer experience, regulatory technology, and internal operations. Hyper-personalization in banking, namely through real-time next-best offers, is a prominent trend. McKinsey and BCG have released numerous publications advocating for banks to leverage real-time data to customize services, highlighting substantial enhancements in cross-selling and customer satisfaction.

Real-time payments serve as an additional catalyst; as quick payment networks such as SEPA Instant, FedNow, and UPI gain prominence, banks are compelled to process transactions and conduct fraud assessments within seconds, necessitating the integration of streaming technology into their core payment systems. In financial markets, real-time risk management, such as intraday Value at Risk estimates and immediate margin calls, is increasingly anticipated, transitioning streaming into areas historically dominated by batch processing. Regulators occasionally require more frequent reporting, such as the Consolidated Audit Trail in the US equity market, which necessitates data transmission throughout the day. All these elements are establishing streaming capability as a baseline prerequisite.

VI. CONCLUSION

Real-time data streaming is transforming financial services by facilitating a transition from batch processing and delayed insights to instantaneous, event-driven operations. Utilizing sophisticated streaming technologies such as Kafka, Spark Structured Streaming, Flink, and Pulsar, financial organizations can eradicate superfluous latency, identify fraud in real-time, execute instantaneous trading decisions, and provide customers with hyper-personalized experiences immediately. Each of these technologies possesses distinct advantages: Kafka excels in durable high-volume data distribution, Spark offers unified analytics, Flink specializes in low-latency stateful processing, and Pulsar provides flexible multi-tenant messaging. When integrated with sound design and best practices, they constitute a formidable toolkit for addressing the industry's data challenges.

The case studies and comparisons demonstrate that no singular technology excels in all aspects; instead, architects must choose the appropriate mix based on specific use case needs (throughput, latency, statefulness, etc.). The positive development is that the streaming ecosystem is evolving swiftly,

characterized by heightened convergence (e.g., enhanced interoperability and managed services) that reduces adoption obstacles. Analyst trends indicate that streaming is increasingly integral to contemporary financial architectures, akin to relational databases or core banking systems; envisioning a "bank of the future" without real-time event processing at its foundation is challenging.

Looking at on-going developments, the amalgamation of AI/ML with streaming, the advancement of analytics to the edge, and the rise of event meshes indicate increasingly responsive and intelligent financial systems. We may soon witness scenarios where credit models are updated in real-time with each client encounter, or decentralized risk assessments occur across worldwide offices in a federated manner, all facilitated by streaming technology. Competitive and regulatory challenges in finance will necessitate more rapid and intelligent data management, ensuring that the advancement of streaming technology aligns closely with the industry's increasing requirements.

In conclusion, financial institutions that adopt real-time streaming and integrate robust engineering methods are positioning themselves to address current demands and swiftly adjust to future opportunities. By analyzing the continuous stream of financial data, they acquire the capacity to act in real-time – a crucial factor that can determine whether they lead the market or fall behind.

REFERENCES

[1] M. R. Faulkner, "Clock Synchronization for Multi-Agent Systems," *arXiv preprint arXiv:1603.07322*, 2016. [Online]. Available: https://arxiv.org/pdf/1603.07322

[2] McKinsey & Company, “Real-Time Finance: A CEO’s Guide,” McKinsey Whitepaper, 2023.

[3] Confluent, "Real-Time Stock Market Data Streaming with Apache Kafka," Confluent Case Studies, 2023.

[4] K. Waehner, "Case Study: Apache Kafka at Capital One - Real-time Fraud Detection," Kai Waehner, 2023. [Online]. Available: https://www.kai-waehner.de/blog/2023/04/12/case-study-capital-one-apache-kafka-real-time-streaming-fraud-detection/.

[5] "Apache Flink Use Cases in the Financial Industry," Ververica, 2023. [Online]. Available: https://www.ververica.com/apache-flink-use-cases-in-the-financial-industry.

[6] Experian, “Streaming Credit Scoring Models – Improving Underwriting in Real Time,” Experian Insights Report, 2022.

[7] ING Bank, “Apache Flink for Real-Time AML Patterns,” in *Flink Forward 2022* (Industry Track), ING Presentation, 2022.

[8] Apache Spark, “Structured Streaming Programming Guide,” *Apache Spark Documentation* (v3.x), 2023. Available: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

[9] Apache Flink, "Apache Flink 1.19 Documentation: Stateful Stream Processing," Apache Flink Documentation

[10] Apache Pulsar Documentation – *Concepts and Architecture Overview*. Available: https://pulsar.apache.org/docs/en/concepts-architecture-overview/

[11] Confluent, "Why Kafka is the fastest messaging system," Confluent Blog. [Online]. Available: https://www.confluent.io/blog/kafka-fastest-messaging-system/

[12] Apache Pulsar Documentation – *Concepts and Architecture Overview*. Available: https://pulsar.apache.org/docs/en/concepts-architecture-overview/

[13] StreamNative, “Apache Pulsar vs. Apache Kafka: 2022 Benchmark Report,” StreamNative Blog, Aug. 2022.

[14] Alibaba Cloud, “Four Billion Records per Second – Apache Flink in Double 11 Festival,” Alibaba Cloud Blog, Dec. 2020.

[15] J. Kreps, N. Narkhede, J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of NetDB*, 2011.

[16] IDC, “Real-Time Data Trends in Financial Services,” 2023.

[17] Confluent Investor Relations, "Confluent Reports Fourth Quarter and Fiscal Year 2021 Financial Results," Confluent Inc., 2022. [Online]. Available: https://investors.confluent.io.