The Patentability of Event Streaming Systems

Adam Gluck
CSE 5821: Legal Topics for CSE
Professor Bryan Choi
April 26, 2025

Table of Contents

Introduction

Software systems have grown increasingly complex as modern life becomes more reliant on digital infrastructure. Today's applications must support billions of users around the world, driving demand for scalable and flexible computing solutions. Companies like Amazon have responded by offering cloud platforms that provide on-demand infrastructure designed to meet these demands.

Apache Kafka has emerged as a foundational technology for handling large volumes of real-time data. It enables organizations to move and process information quickly between systems, supporting critical functions from financial transactions to personalized recommendations. However, Warpstream reimagines this model for the cloud era. By rebuilding Kafka's core functionality using cloud-native services, Warpstream offers a system that is more scalable and easier to manage while being more cost-effective.

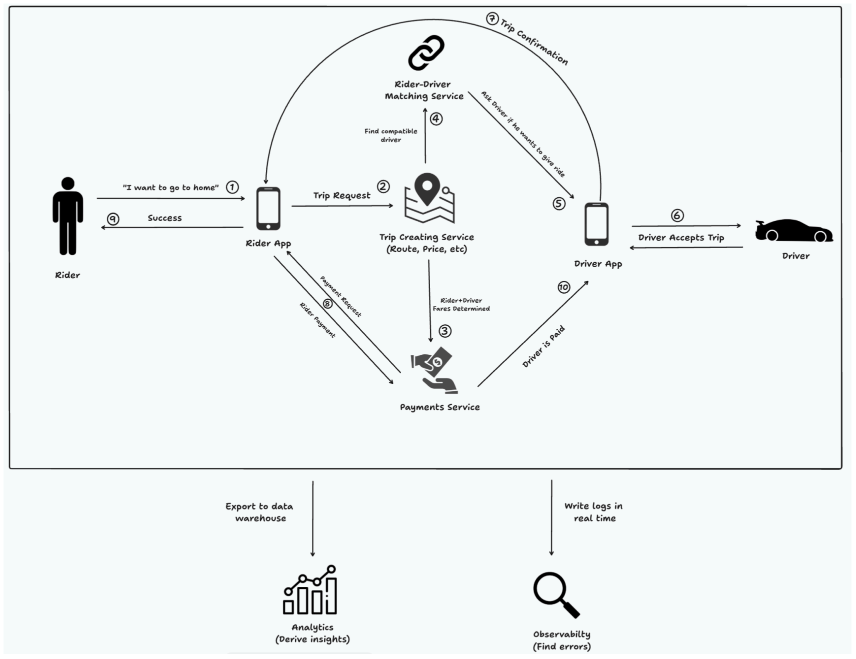
This paper explores whether these innovations are substantial enough to warrant patent protection. Using Warpstream as a case study, we will examine how U.S. patent law applies to modern, cloud-native software systems, particularly when they offer the same functionality as existing systems.

Technical Description

Event Streaming

Imagine you are building a ridesharing app like Uber or Lyft. When a customer requests a ride, the system must calculate the optimal GPS route, find an available driver, determine a price the driver will accept, and handle payment transactions between the rider and the driver. Beyond these core tasks, the system also requires observability, real-time monitoring to ensure each service is functioning correctly, and analytics to derive business insights from accumulated data.

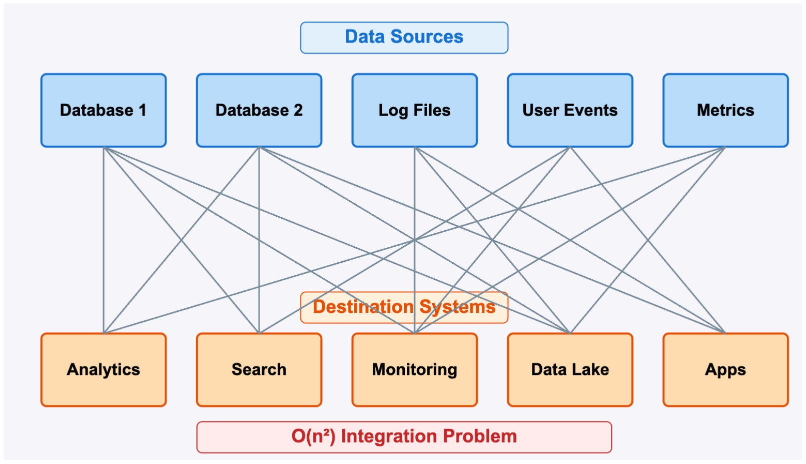
A common strategy to allow apps like this to scale to millions of users is to split each major function into an independent service, called a microservice. Since microservices operate independently, explicit communication between them is necessary to complete a workflow. The diagram below illustrates a simplified version of the communication involved in handling a single trip request:



In this system, when a rider initiates a trip request, the Rider App sends the request to the Trip Creating Service, which calculates details like the route and price. The Rider-Driver Matching Service then locates a compatible driver, and the Driver App prompts the driver to accept the trip. Once accepted, the system finalizes the trip booking, processes the driver's payment, and updates both rider and driver interfaces. Meanwhile, observability services log actions in real-time to detect errors, and analytics systems export data for business insights.

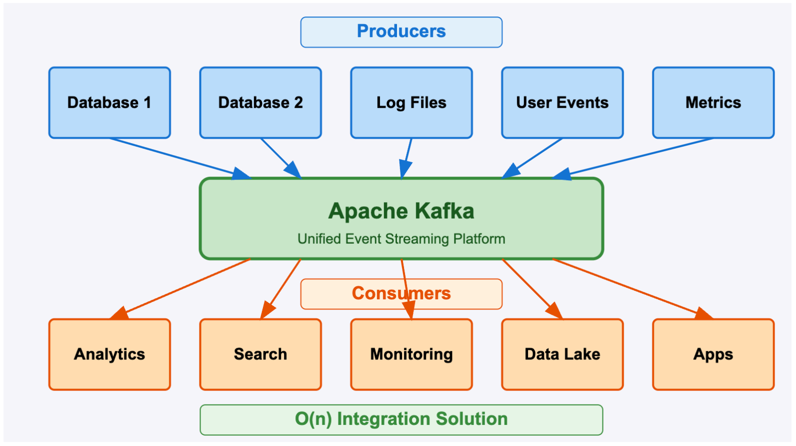
In real-world systems, the communication complexity grows even further. Features such as dynamic pricing based on traffic, premium service options, and live driver tracking require even more microservices and inter-service communication, increasing the system's complexity dramatically.

This growing complexity leads to a broader architectural problem: connecting many different data sources and destination systems. A naive solution would be to build custom pipelines for every pair of systems. However, as the system scales, the number of required connections grows quadratically, leading to an $O(N^2)$ integration problem, as shown below:



Every time a new service is added, it must be connected to every existing service individually, quickly making the architecture difficult to maintain and highly fragile. Even small changes in one service's data format or communication schema can ripple across the entire network and cause system-wide failures. This approach also introduces major performance bottlenecks, as actions may need to be broadcast separately to many different endpoints.

A more scalable and reliable solution is to use an event streaming platform like Apache Kafka to centralize communication. In an event-driven architecture, data sources (producers) and data consumers interact indirectly through Kafka. Producers send events to Kafka when something happens, and Kafka forwards those events to the relevant consumers. This architecture reduces the system's complexity from $O(N^2)$ connections to $O(N)$ standardized integrations, improving scalability, fault tolerance, and maintainability:

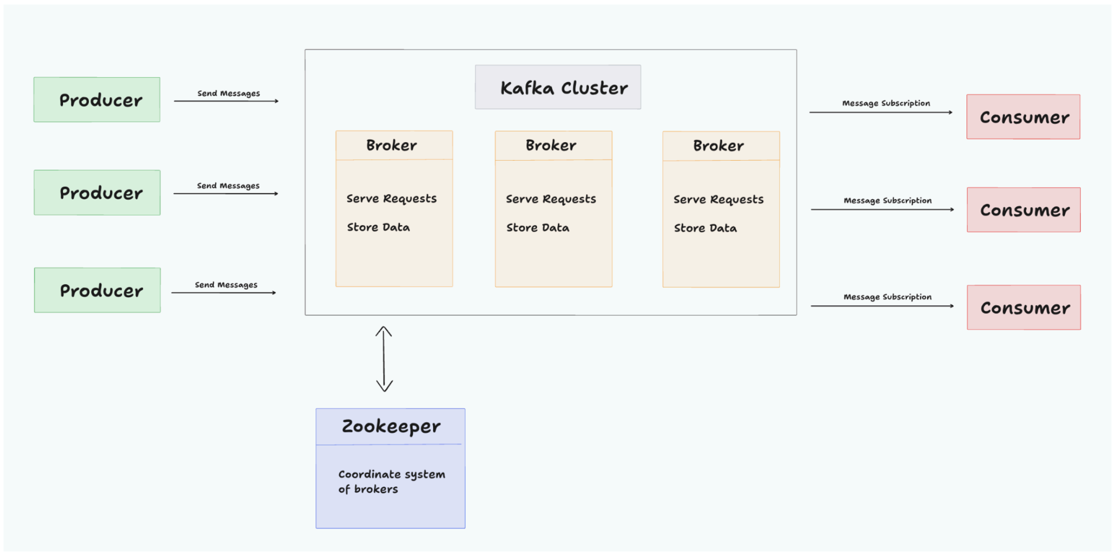


By introducing an intermediary platform like Kafka, new services only need to integrate once to the centralized event platform, rather than with every other service individually. This shift from direct pipelines to event streaming represents a fundamental architectural improvement for modern systems.

Apache Kafka

Now that we understand the need for event streaming platforms, we can dive into what Apache Kafka is and how it works. Apache Kafka is a popular open-source event streaming platform originally developed by LinkedIn in 2010. Today, it is used by over 80% of the Fortune 100[1] to move and process large volumes of real-time data efficiently.

Kafka operates on the concepts of producers and consumers. Producers create events, while consumers retrieve and process them. Between these two roles sits the Kafka cluster, made up of brokers: servers responsible for receiving, storing, and forwarding messages. To coordinate and manage the brokers, Kafka typically relies on a system like Apache Zookeeper, which ensures consistency and resilience even if individual brokers fail. The basic architecture is illustrated below:



Kafka scales horizontally by adding more brokers to the system. However, this approach introduces challenges. In order to prevent data loss during node failures, Kafka requires redundancy through data replication. The traditional formula dictates that to tolerate n broker failures, at least 2n + 1 replicas are needed. This redundancy, along with the need for close monitoring of node health and rebalancing workloads when brokers are added or fail makes running Kafka operationally intensive. Many organizations must dedicate specialized engineers solely to manage Kafka clusters. While Kafka has proven robust and reliable, this burden has opened opportunities for rethinking its architecture using newer technologies.

Object Storage

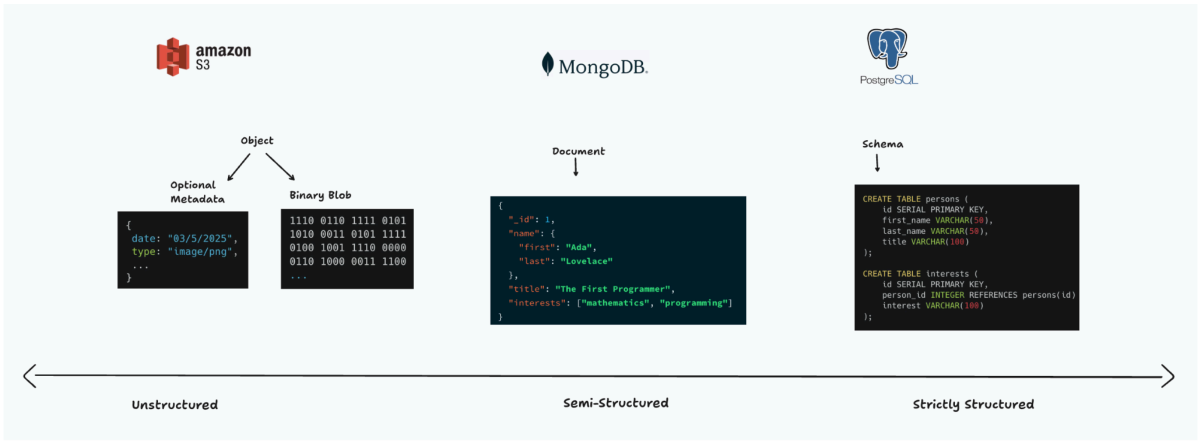
One technological advance that addresses some of Kafka's challenges is the rise of cloud-based object storage. Services like Amazon S3 (Simple Storage Service) offer a highly durable, low-cost method for storing large volumes of digital data as binary blobs, accompanied by optional metadata in a simple key-value format.

[1] https://kafka.apache.org/powered-by

Unlike traditional databases, object storage systems are optimized for storing and retrieving entire files rather than performing fine-grained updates or fast indexed queries. They offer three primary operations: PUT(key, object, metadata) to store data, GET(key) to retrieve data, and LIST() to view stored keys. The simplicity of the interface provides scalability and parallelism, with services like Amazon S3 offering durability guarantees of 99.999999999% and availability of 99.99%[2].

However, this simplicity comes at a cost. Object storage has higher latency than traditional databases[3] and struggles with workloads requiring frequent small updates[4]. Additionally, it lacks features like indexing or complex query languages. Despite these limitations, with appropriate techniques such as batching multiple small reads and writes into larger operations, many systems can still leverage object storage effectively for cost savings and scalability.

The relationship between object storage and traditional databases is illustrated below:



```
{
 date: "03/5/2025",
 type: "image/png",
 ...
}
```

```
1110 0110 1111 0101
1010 0011 0101 1111
0100 1001 1110 0000
0110 1000 0011 1100
...
```

```
{
  "_id": 1,
  "name": {
    "first": "Ada",
    "last": "Lovelace"
  },
  "title": "The First Programmer",
  "interests": ["mathematics", "programming"]
}
```

```
CREATE TABLE persons (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    title VARCHAR(100)
);

CREATE TABLE interests (
    id SERIAL PRIMARY KEY,
    person_id INTEGER REFERENCES persons(id)
    interest VARCHAR(100)
);
```

At the unstructured end is object storage, optimized for flexibility and scalability but with limited querying capabilities. Moving toward the structured end, document databases like MongoDB and relational databases like PostgreSQL offer increasingly sophisticated query models at the cost of more rigid data models and higher storage costs.

Warpstream

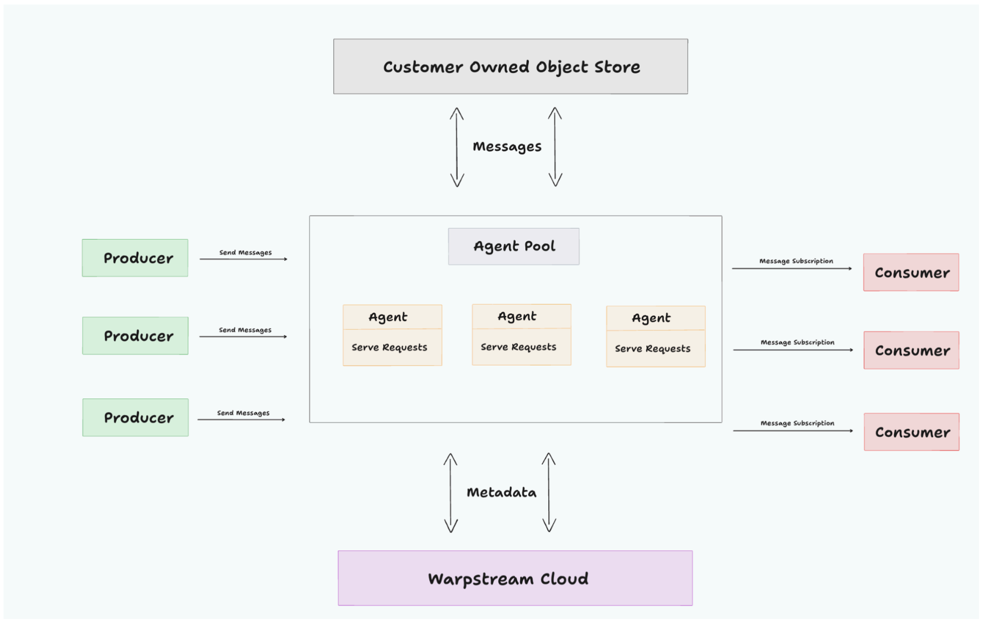
Warpstream is a Kafka-compatible event streaming platform built specifically for the cloud era, leveraging object storage as its core storage layer rather than traditional broker servers. Released in 2023 and acquired by Confluent just 13 months later for $220 million[5], Warpstream claims to deliver similar functionality to Kafka while being 5–10 times cheaper and significantly easier to scale.

The fundamental difference between Kafka and Warpstream lies in architecture. While Kafka brokers are responsible for both transmitting and storing event data, Warpstream

[2] https://docs.aws.amazon.com/AmazonS3/latest/userguide/DataDurability.html
[3] S3 Express is a new AWS offering that allows faster latency with tradeoffs
[4] Appends were recently added to S3 with some limitations
[5] https://pitchbook.com/profiles/company/534965-14

separates these concerns. Event data is stored directly in customer-owned cloud object storage, while lightweight, stateless compute agents handle the ingestion and delivery of events. These agents do not maintain persistent local state, allowing storage and compute resources to be scaled independently and flexibly.

Additionally, Warpstream separates data from metadata. Event data, potentially containing sensitive information, is stored in the customer's object store. Metadata, which tracks offsets, consumer positions, and other pieces of data used for handling data transmission are managed separately through Warpstream's own cloud service. This separation not only improves system design but also simplifies regulatory compliance for customers, especially for data privacy standards like HIPAA. Warpstream's architecture is depicted below:



By offloading storage to cloud infrastructure and simplifying compute through stateless agents, Warpstream significantly reduces operational complexity compared to Kafka. Storage expansion requires only purchasing more cloud storage, while compute scaling involves deploying new agents. This architectural shift illustrates how cloud-native design principles can lead to more efficient, scalable, and cost-effective event streaming systems.

Legal Analysis

Now that we understand Warpstream's architecture and underlying technology, we can assess whether it would be eligible for patent protection under U.S. patent law. To qualify, an invention must meet several requirements: it must fall within patentable subject matter and not be an abstract idea (§101), it must be novel and not disclosed in prior art (§102), it must be non-obvious to a person having ordinary skill in the art (§103), and it must be adequately described in the filed claims (§112).

Patent Claim

Since no official patent has been issued for Warpstream, this analysis uses a hypothetical claim shown below. We will assume that these claims pass (§112) for this analysis.

A distributed event streaming system comprising:

- An interface compatible with Apache Kafka for producing and consuming event data
- An object storage system used as the primary storage for all event data, optionally owned by the customer
- A metadata service, operated by the system provider, that manages the coordination and tracking of event data streams
- Stateless compute agents that handle data ingestion and delivery without using local disk storage, relying entirely on the object storage and metadata service.

Patentable Subject Matter (§101)

For a software product to be under patentable subject matter, it must pass the Alice/Mayo Framework established in Alice Corp. v. CLS Bank (2014). This framework consists of two steps:

1. Is the Claim Directed to an Abstract Idea?

2. Does the Claim Include an Inventive Concept?

Mayo Step 1

One argument in favor of Warpstream satisfying Step 1 is that it addresses a concrete technical problem, scaling Kafka-like systems in a simpler and more cost-effective way through a specific system architecture. This approach is supported by DDR Holdings v. Hotels.com (2014), where a system that solved a business-technical problem in a non-conventional manner was found patent-eligible. Similarly, Warpstream restructures event streaming by eliminating traditional broker architecture and relying on cloud-native infrastructure to achieve cost and scalability advantages.

However, there is also a strong counterargument that Warpstream simply applies known cloud infrastructure techniques to an existing system without fundamentally changing the underlying concept of event streaming. Under Parker v. Flook (1978), simply implementing an abstract idea in a new technological environment is not enough. If Warpstream's improvements are viewed as straightforward adaptations of conventional cloud tools, a court could characterize the system as merely an abstract idea constrained to a particular technological field.

Mayo Step 2

Under Step 2, the inquiry is whether Warpstream's architecture includes an inventive concept that transforms it into patentable subject matter. Supporters would argue that the combination of stateless compute agents, customer-controlled object storage, and a separate metadata service reflects a technical improvement over prior event streaming systems, much like the database architecture improvements found patentable in Enfish v. Microsoft (2016).

Opponents could argue that Warpstream's changes, while practically useful, merely represent conventional modernization of known components. Mayo v. Prometheus (2012) reinforces that using known techniques to make existing processes cheaper or more efficient does not necessarily constitute an inventive concept. By analogy, replacing a traditional motor with a more efficient one in an existing ceiling fan design would not, by itself, make the fan patentable.

Ruling

Overall, Warpstream's technology likely qualifies as patentable subject matter under §101. Its architecture addresses a specific technical problem, improving the scalability and cost structure of event streaming systems, through a concrete combination of stateless agents, separated metadata services, and customer-controlled object storage. These features, when properly claimed, would distinguish Warpstream from simply abstracting an existing idea.

However, the strength of Warpstream's patent eligibility depends on how narrowly the claims are drafted. A claim closely tied to the particular technical implementation, emphasizing the specific separation of storage and compute, the stateless nature of agents, and the role of metadata coordination, would likely survive §101 scrutiny. In contrast, a claim that broadly attempts to cover any cloud-native adaptation of an event streaming system risks being invalidated as merely applying conventional technology to a known concept.

The line for patentable subject matter, therefore, falls between protecting Warpstream's particular technical solution and trying to monopolize the general idea of cloud-optimized event streaming. While the design outlined in the example patent used for this analysis is specific enough, making it more abstract (such as by replacing object storage with "any cloud storage") would make it fail §101.

Novelty (§102)

For Warpstream to be eligible for a patent, it must be novel under §102. This means the claimed invention must not be disclosed in any single prior art reference. To assess this, we evaluate several existing technologies to determine whether they anticipate Warpstream's design.

Kafka Tiered Storage[6]

Kafka Tiered Storage is a relatively recent feature introduced to help Kafka scale by splitting data into two tiers: hot data and cold data. Hot data, which is frequently accessed, remains stored on traditional Kafka brokers. Cold data, which is less frequently needed, can be offloaded to external object storage systems to reduce costs, at the price of increased query latency.

The overlap with Warpstream lies in the use of object storage for cost savings. However, unlike Kafka Tiered Storage, Warpstream offloads all event data to object storage, eliminating the need for brokers to store hot data. Additionally, Warpstream introduces additional engineering optimizations to minimize the latency impact of accessing object storage. Therefore, although Kafka Tiered Storage shares some conceptual similarities, it does not anticipate Warpstream's full system architecture.

Apache Pulsar[7]

Apache Pulsar is another event streaming platform that can emulate the Apache Kafka interface. Pulsar splits Kafka's monolithic broker cluster into two specialized clusters: one for stateless communication with producers and consumers, and one for persistent storage, coordinated by Apache Zookeeper.

Like Warpstream, Pulsar separates compute and storage. However, Pulsar's storage cluster consists of traditional servers rather than cloud-based object storage. Moreover, Pulsar does not separate data from metadata in the way Warpstream does, Warpstream allows customers to retain full control over their raw event data, while the system provider manages metadata separately. These architectural differences make Warpstream distinct from Pulsar in both design and function.

Redpanda[8]

Redpanda is a Kafka-compatible event streaming platform designed for higher performance and lower latency. It achieves these improvements primarily through reimplementing Kafka's architecture in C++, using high-performance techniques like advanced caching, zero-copy IO, and performance-oriented tiered storage (as covered by Patent 11743333[9]).

Although Redpanda and Warpstream both aim to improve upon Kafka, their approaches diverge significantly. Redpanda prioritizes speed and performance enhancements within a broker-centric architecture, while Warpstream focuses on simplifying the architecture by removing broker persistence entirely and relying on cloud object storage for all data durability. While both systems aim to reduce costs and complexity, Warpstream's architectural philosophy sacrifices some speed for ease of scaling and lower operational overhead, fundamentally different engineering tradeoffs from Redpanda's optimization strategy.

[6] https://cwiki.apache.org/confluence/display/KAFKA/KIP-405%3A+Kafka+Tiered+Storage
[7] https://pulsar.apache.org/
[8] https://www.redpanda.com/
[9] https://patents.google.com/patent/US11743333B2/en?oq=11743333

Patent 20110196848 A1[10]

Patent Publication 20110196848 A1 discusses strategies for separating data from metadata within distributed systems. This reference shows that the general concept of decoupling metadata from raw data was already known in the field. However, while the idea of a data-metadata split existed, Warpstream's specific implementation, using customer-owned object storage for data and a centralized cloud metadata service for coordination, offers architectural and operational differences. Therefore, although this prior patent narrows Warpstream's ability to claim novelty solely on the idea of separating data and metadata, Warpstream's full system remains distinct.

Ruling

Overall, while individual elements of Warpstream's design, such as object storage use, compute-storage separation, Kafka compatibility, and data-metadata separation, are present in prior technologies, no single prior art reference discloses the full combination of these elements as employed by Warpstream. Additionally, Warpstream's architecture results in different performance characteristics, scalability advantages, and operational efficiencies compared to any of the cited prior art. Therefore, Warpstream's system as a whole appears to satisfy the novelty requirement under §102.

Non-Obviousness (§103)

For Warpstream to be patentable, it must meet the non-obviousness requirement of 35 U.S.C. §103. Specifically, the invention must not have been an obvious variation of existing technology to a person having ordinary skill in the art ("PHOSITA") at the time of invention. The Supreme Court in Graham v. John Deere (1966), established the controlling framework for this analysis, which examines four key factors:

1. The scope and content of the prior art,

2. The differences between the prior art and the claims,

3. The level of ordinary skill in the pertinent art, and

4. Secondary considerations such as commercial success, long-felt but unsolved needs, and the failure of others.

Each of these factors must be weighed to determine whether Warpstream's architecture represents a non-obvious invention or merely an aggregation of known concepts yielding predictable results. The argument for Warpstream being non-obvious rests primarily on the technical barriers present at the time of its development. Object storage systems, such as Amazon S3 or Google Cloud Storage, historically exhibited high latency and were not

[10] https://patents.google.com/patent/US20110196848A1/en

considered suitable for event streaming as they typically demand low-latency, high-throughput access patterns. Designing a scalable, performant event streaming platform on such storage systems would have required significant innovation and technical skill. Additionally, Warpstream's architecture required advanced engineering skills to effectively mask the latency penalties inherent in object storage, while still providing an interface compatible with Kafka clients. This level of technical sophistication could support a finding of non-obviousness, as it suggests that the solution was not readily apparent to those skilled in the field.

However, the argument for obviousness is strongly informed by the Supreme Court's reasoning in KSR International Co. v. Teleflex (2007). In KSR, the Court emphasized that when a combination of familiar elements according to known methods yields no more than predictable results, the combination is likely unpatentable. Warpstream's architecture of combining tiered storage concepts from Kafka, the separation of compute and storage from Apache Pulsar, cloud-optimized Kafka interfaces seen in Redpanda, and metadata separation techniques outlined in existing patents appears to follow this pattern. As explained in MPEP §2141 and §2143, an invention is considered obvious if a PHOSITA would have had a reason to combine prior art teachings with an expectation of success.

The scope and content of the prior art reveal that each of Warpstream's components was individually known. Kafka's Tiered Storage introduced the use of object storage for cold event data to reduce costs. Apache Pulsar demonstrated the value of separating the compute and storage layers to improve scalability and maintainability. Redpanda illustrated that Kafka-compatible systems could be re-architected for optimized performance and cost efficiencies. Separating data from metadata was also a known strategy in the art, as reflected in Patent Publication 20110196848 A1. Thus, the individual ideas behind Warpstream were not novel on their own and would have been accessible to a PHOSITA familiar with distributed system architecture.

The differences between Warpstream and the prior art primarily involve extent, not uniqueness. Warpstream extends the tiered storage model by moving all event data, not just cold data, to object storage. It furthers the separation of compute and storage by eliminating any persistent broker storage nodes entirely. While these changes required careful engineering, they would arguably have been logical, incremental steps for a skilled artisan seeking to optimize costs and scalability using cloud-native technologies.

The level of ordinary skill in the art at the relevant time was high. A PHOSITA would possess experience with distributed systems, event streaming platforms, cloud infrastructure, and data storage technologies. Given the growing trend toward cloud-first architectures and the documented use of object storage for cost savings in other areas, a PHOSITA could have reasonably envisioned an event streaming system that pushed further into cloud-native object storage, even if some latency compromises were involved.

Secondary considerations also fail to overcome the presumption of obviousness. Although Warpstream may have achieved commercial success, that success must be causally linked to the technical merits of the invention rather than market timing or cost leadership alone. Additionally, there was no demonstrable long-felt but unmet need that Warpstream uniquely solved; systems like Pulsar and Kafka's tiered storage were already evolving toward similar goals. There is also no evidence of widespread failure by others to integrate object storage into event streaming systems in a meaningful way.

Overall, although Warpstream's design reflects careful engineering and a clear understanding of distributed systems, it represents a combination of known elements yielding predictable results. Under the standards articulated in Graham and KSR, Warpstream's architecture would likely be deemed obvious to a PHOSITA and therefore unpatentable under §103.

Conclusion

This paper analyzed the patentability of Warpstream, a cloud-native event streaming system that reimagines Apache Kafka by separating storage and compute through the use of object storage and stateless agents. Under U.S. patent law, Warpstream likely qualifies as patentable subject matter under §101 and meets the novelty requirement of §102. However, it would likely fail the non-obviousness test under §103 because its design largely combines known elements in a way that would have been predictable to a person having ordinary skill in the art. While Warpstream's architecture delivers real technical and operational advantages, it is best understood as a skillful application of existing concepts rather than a fundamentally inventive breakthrough. As a result, obtaining broad patent protection on the overall system would be difficult, although the business and engineering innovations remain highly valuable.

Future Work

Looking ahead, Warpstream should adopt a focused intellectual property strategy that acknowledges both the opportunities and the limitations presented by its legal position.

First, Warpstream should protect its implementation primarily as a trade secret. Since Warpstream is closed-source, maintaining confidentiality over internal designs, performance optimizations, and system orchestration is critical. However, trade secrets come with the risk that competitors could independently discover similar techniques, and enforcement can be difficult if misappropriation is not clear or provable. Therefore, Warpstream must invest in access controls, NDAs, and other internal policies to minimize any risk of leakage or unintentional disclosure.

Second, Warpstream should consider limited use of copyright protection, but with caution. While copyright can protect the source code and technical documentation, Warpstream's close compatibility with Apache Kafka raises the question of whether it could be viewed as a "derivative work." If so, this could complicate or weaken Warpstream's ability to enforce its copyrights, especially given the open-source licensing terms of Kafka. Before pursuing copyright registration aggressively, Warpstream should conduct a careful legal review to evaluate whether its implementation is sufficiently original to avoid being classified as derivative.

Finally, while broad system patents are unlikely to be successful, Warpstream should still explore filing narrowly tailored patents covering specific technical innovations, such as latency optimization methods for object storage or novel metadata handling techniques. Even if individual components are incremental, well-drafted patents could strengthen Warpstream's negotiating position and provide limited but meaningful IP protection.

By relying primarily on strong trade secret protections, selectively pursuing narrowly scoped patents, and carefully assessing the viability of copyright claims, Warpstream can effectively use the U.S. legal system to protect its intellectual property.

Open Questions

An important unresolved issue is whether a system that takes an existing open-source project and redesigns it to be "cloud-native" should be eligible for patent protection. On one hand, a cloud-native redesign often involves significant engineering effort and can result in meaningful technical improvements, such as better scalability, lower costs, and easier management. On the other hand, if the core functionality remains the same and the innovation lies only in adapting to modern infrastructure, granting a patent might risk protecting what is ultimately a routine modernization. Under U.S. patent law, particularly following KSR v. Teleflex, applying known techniques to existing systems is often viewed as obvious unless the adaptation solves a previously unaddressed technical challenge in a non-trivial way. Therefore, to support patentability, a cloud-native redesign would likely need to incorporate additional new innovations, such as new storage strategies, coordination methods, security models, or other fundamental improvements to system behavior, beyond simply moving to the cloud.

Another open question concerns the extent to which a software product's interface can be protected by patents. This is particularly important for Warpstream's compatibility with Apache Kafka. Generally, software interfaces, such as APIs or wire protocols, can sometimes be patented if they present a novel, non-obvious technical solution rather than simply a functional necessity. The situation is similar to the legal questions raised in Apple v. Samsung, where Samsung was found to have infringed Apple's design and utility patents, including aspects of the iPhone's interface and overall look and feel. Although functionality and appearance are different areas of protection, the case showed that copying distinctive, protectable features, even if not copying internal workings, can be actionable. If Kafka's protocol had been patented, Warpstream's use of a compatible interface could have exposed it to potential liability, much like Samsung was liable for mimicking Apple's design language. Whether Warpstream infringes would depend on whether Kafka's patents, if they existed, claimed not just the internal operation but the external communication protocols that Warpstream reuses.

This comparison underscores a larger challenge: as software systems become increasingly reliant on standards set by open source projects, to what extent is it ethical for third-party companies to iterate on these projects to make and/or sell a patented successor without rewarding the original creators?