The Ambar Approach to Data Streaming

Ambar Cloud Ltd.

January 8, 2024

Contents

1 Introduction **1**

2 Data Streaming Done Right **2**

3 Multi-Topic Operational Model **3**

4 Correctness **4**
- 4.1 Representation 4
- 4.2 Order Preservation 5
- 4.3 Basic Results 5
  - 4.3.1 Theorem: representation of composites 5
  - 4.3.2 Theorem: respective representation of composites . . . 5
  - 4.3.3 Theorem: order preservation of composites 6
- 4.4 Correctness Condition 6
- 4.5 At-Least Once vs. Exactly Once 6

5 Conclusion **7**

1 Introduction

Reacting to customer behavior in real time is the lifeblood of modern businesses, and doing so requires the use of stream processing software. But stream processing software lacks appropriate correctness guarantees out-of-the-box, whether open source or a proprietary cloud service. So engineering teams spend years implementing correctness guarantees themselves, but even the best teams retain error rates above 0.2%. In verticals such as finance, logistics, e-commerce, sports betting, and stock trading – where operating profits range from 1% to 10% of the total value flowing through the business

– even low error rates translate to significant profit losses. According to Gartner reports, this area of cloud computing is a \$70B+ market, projected to grow $>20\%$ YOY through to 2026.

The issue is that existing solutions adopt a myopic viewpoint. They focus on the problem of streaming the raw data itself, with no consideration for the ingress or egress of said data (a decidedly non-trivial task). At Ambar, our mission is to solve this problem holistically. This allows us to both lower the barrier to entry for real-time stream processing, and increase the reliability of end-to-end streaming solutions. We believe Ambar is not only the *easiest* way to get started with real-time stream processing, but is also the most *robust*.

Part of our holistic approach is the formal modeling of the entire stream processing problem, including both ingress and egress. This formal approach allows us to prove our algorithms correctly stream data to our customers, from its point of origin to its point of use, in a way that enables them to focus on application logic rather than infrastructure arcana.

In this white paper, we present our formal model of the data streaming problem, along with some preliminary proofs that give us concrete, sufficient requirements for the system we've implemented.

2 Data Streaming Done Right

What does it mean to stream data correctly? Answering this question precisely is at the heart of our efforts. From the application developer's perspective, the correctness requirement is that the system should allow them to consume streams in-order, and exactly once. Practical experience also tells us that the system needs to support partitioning if it has any hope of meeting equally important performance requirements. As with all interesting problems, the simplicity belies the complexity. We must be confident that as the data passes through our system, it undergoes only transformations which are guaranteed to preserve these requirements.

In the following sections, we give a high-level model of the architecture of an Ambar cluster built on a Kafka substrate, build the definitions required for stating our correctness condition, and prove the existence of certain sufficient conditions for the correctness of our cluster implementation in terms of this model.

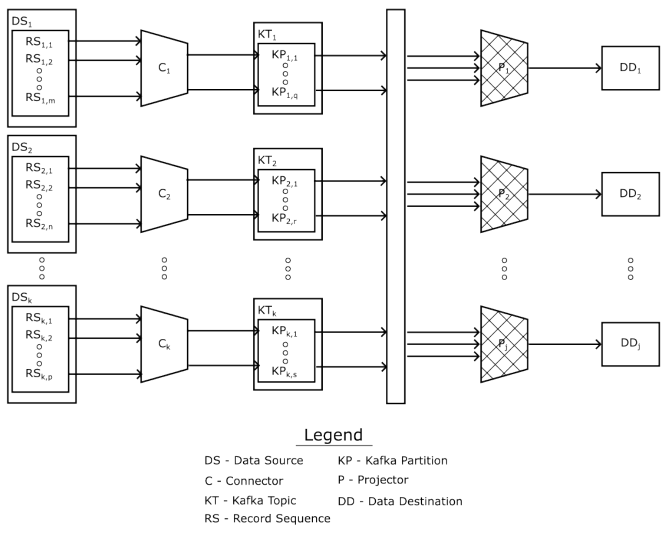

Figure 1: The architecture of an Ambar cluster.

3 Multi-Topic Operational Model

The model of an ambar cluster is shown in figure 1.

Generally, an Ambar cluster provides an in-order, at least once delivery service to customers. More specifically, it transports sequences of data items called *records* from sources to destinations, while preserving order, and providing aggregation and filtering capabilities.

A *data source* is a durable storage system which contains a set of *record sequences*. An Ambar cluster supports any number of data sources (e.g., MySQL, PostgreSQL, EventstoreDB).

A *connector* is a process which is responsible for reading record sequences from a data source, and producing them to an internal Kafka cluster. There is one connector for each data source. Each connector produces records to its own Kafka topic. In general, the number of partitions for each Kafka topic will be significantly fewer than the number of record sequences in the corresponding data source. The process of reading records and producing them to Kafka is referred to as *ingestion*.

A *projector* is a process responsible for transmitting record sequences from the internal Kafka cluster to the *data destinations*, which are idempotent HTTP endpoints owned by the client. There is one projector for each data destination, and any number of data destinations are supported. Projectors allow clients to specify *filters*, which select subsequences of records from the complete set of records stored in the system. Conceptually, these work by applying a predicate over an arbitrary merger of all record sequences, and submitting requests to the destinations for only those which match the filter.

4 Correctness

This section contains a correctness proof for an Ambar system, in terms of the model given above. Note that this is a proof of the system, rather than any specific implementation of said system. It is therefore analgous to a correctness proof for an algorithm as opposed to a function/routine. As a necessary precedent, this section also defines rigorously what it means for an ambar system to be considered correct.

In the following subsections, we will assume the following definitions.

- Let R be a set of records.
- Let R^* be the set of all sequences formed from elements of R.
- Let f and g be total functions mapping between elements of R^*.
- Let m be a k-dimensional function from R^{*k} to R^*. That is, it maps a k-tuple of elements from R^* to a single element of R^*.
- For a sequence S, let $S[i:j)$ be the subsequence $S_i...S_{j-1}$.
- When convenient, we treat sequences as sets. In other words, for a sequence S, we say $r \in S$ iff $\exists i : S_i = r$.

4.1 Representation

We say that $f(S) = S^{'}$ is *representative* if

$$\forall r \in S, \exists r^{'} \in S^{'} : r^{'} = r$$

We say that f is representative if it is representative over its entire domain.

For an arbitrary predicate p over R, we say that $f(S) = S^{'}$ is *representative with respect to p* if

$$\forall r \in S : p(r) \implies \exists r^{'} \in S^{'} : r^{'} = r$$

We say that a function f represenative with respect to a predicate p if it is representative with respect to p over its entire domain.

We say that m is representative if it is representative in each of its arguments.

4.2 Order Preservation

We say that $f(S) = S'$ is *order preserving* if

$$S_i^{'} = S_j \implies \forall_{r \in S[1:j)} (r \in S^{'} \implies r \in S^{'}[1:i))$$

We say that f is order preserving if it is order preserving over its entire domain.

We say that m is order preserving if it is order preserving in each of its arguments.

4.3 Basic Results

4.3.1 Theorem: representation of composites

If f is representative, and g is representative, then $f \circ g$ is representative.

Proof: Let S be an element of R^*. Since g is representative, all of the elements of S appear in $g(S)$. Since f is representative, all of the elements of $g(S)$ appear in $f(g(S))$. Hence, all elements of S appear in $(f \circ g)(S)$. Therefore, $(f \circ g)(S)$ is representative.

4.3.2 Theorem: respective representation of composites

If f is representative with respect to some predicate p over R, and g is representative with respect to some predicate q over R, then $f \circ g$ is representative with respect to $p \wedge q$.

Proof: Let S be an element of R^*. Since g is representative with respect to q, $g(S)$ contains all the elements of S which satisfy q. Since f is representative with respect to p, $f(g(S))$ contains all the elements of $g(S)$ which satisfy q. Hence, $f(g(S))$ contains all the elements of S which satisfy $p \wedge q$, and $f \circ g$ is representative with respect to $p \wedge q$.

4.3.3 Theorem: order preservation of composites

If f is order preserving and representative, and g is order preserving and representative, then $f \circ g$ is order preserving and representative.

Proof: By contradiction. Suppose $f \circ g$ is not order preserving and representative.

If it is not representative, then for some sequence S, there exists an element r from that sequence which is not represented in $(f \circ g)(S)$. This would imply that either f is not representative, or g is not representative, either of which would be a contradiction.

On the other hand, suppose it is not order preserving. Let $(f \circ g)(S) = S'$ be a sequence for which it is not order preserving. Then there exists a pair of indices (i, j) such that:

$$S_i^{'} = S_j \wedge \exists_{r \in S[1:j)}(r \in S^{'} \wedge r \notin S^{'}[1:i))$$

Since g is order preserving and representative, we can conclude that S_j and r both exist in $g(S)$, and that the first appearance of S_j is prior to the first appearance of r therein.

Similarly, since f is order preserving and representative, we can conclude that S_j and r both appear in $f(g(S))$, and that the first appearance of S_j is prior to the first appearance of r therein. This contradicts our assumption that $f \circ g$ is not order preserving, and concludes our proof.

4.4 Correctness Condition

With the basic definitions above, we are ready to state our correctness condition, and show that it applies to our system model. We say that an Ambar system is correct if, for every data destination with associated filter predicate p, there exists an order-preserving function f, representative with respect to p, such that the sequence of records successfully submitted to the data destination's associated endpoint is the image of that function over the set of all record sequences.

Thus, if we can construct a system which transforms record sequences in accordance with a function having the above properties, we can be confident in its correctness.

4.5 At-Least Once vs. Exactly Once

The astute reader will recognize that our correctness condition provides *at-least once* delivery of each record, rather than exactly-once. This is an instance where our holistic approach informed our strategy. Idempotency is a very simple operation for customer endpoints to implement with the sort of data that is common to real-time streams. In concert with at-least once delivery, this idempotency provides exactly-once semantics, and provides us considerable breathing room in our implementation, which allows us to provide not only a more performant system, but also a more robust one.

5 Conclusion

In this white paper, we shared the formal model we use to architect the top level of our system, but it's important to note that the complexity doesn't stop there. Connectors, projectors, and the underlying message brokers must meet similar guarantees. Meeting those guarantees in the presence of practical roadblocks common to distributed systems like zombie fencing, server availability, data durability, load fluctuation, memory limits, and clock errors is a significant problem in its own right. We've done that too.

We've achieved guarantees for connectors to popular DBMSs (e.g., MySQL, PostgreSQL, EventstoreDB), which involved developing different algorithms for checkpointing and ingestion because each DMBS comes with its own internal mechanics and guarantees. We've also achieved guarantees for our projector (compatible with every connector) and for Kafka as an underlying message broker. In other words, we've achieved guarantees that provide end-to-end correctness to Ambar customers.

At Ambar, our goal is to democratize robust data streaming. Much like ACID brought robustness to databases, Ambar brings robustness to stream processing. As the first provably correct end-to-end data streaming solution, we empower engineers to build stream processing systems dependably, with less cognitive load, while eliminating errors that put businesses at risk.