Compare and contrast the various programs available for you to use in real-time.

Answer the following questions in your written response:

- What programs behave similarly?

- How do these programs differ from each other?

- If you could only pick one program, which one would you choose and why?

**Answer:**

The various programs available for use in real-time are :

* **Kafka**
* **Flume**
* **Spark Streaming**
* **Storm**
* **Flink**

**Storm :**Storm is the hadoop of Streaming world. It is the oldest open source streaming framework and one of the most mature and reliable one. It is true streaming and is good for simple event based use cases.

Advantages:

* Very low latency,true streaming, mature and high throughput
* Excellent for non-complicated streaming use cases

Disadvantages

* No state management
* No advanced features like Event time processing, aggregation, windowing, sessions, watermarks, etc
* Atleast-once guarantee

**Spark Streaming :**

Spark has emerged as true successor of hadoop in Batch processing and the first framework to fully support the Lambda Architecture (where both Batch and Streaming are implemented; Batch for correctness, Streaming for Speed). Continuous Streaming mode promises to give sub latency like Storm and Flink, but it is still in infancy stage with many limitations in operations.

Advantages:

* Supports Lambda architecture, comes free with Spark
* High throughput, good for many use cases where sub-latency is not required
* Fault tolerance by default due to micro-batch nature
* Simple to use higher level APIs
* Big community and aggressive improvements
* Exactly Once

Disadvantages

* Not true streaming, not suitable for low latency requirements
* Too many parameters to tune. Hard to get it right.
* Stateless by nature
* Lags behind Flink in many advanced features

**Flink**:

Flink is also from similar academic background like Spark. The implementation is quite opposite to that of Spark. While Spark is essentially a batch with Spark streaming as micro-batching and special case of Spark Batch, Flink is essentially a true streaming engine treating batch as special case of streaming with bounded data. Though APIs in both frameworks are similar, but they don’t have any similarity in implementations. In Flink, each function like map,filter,reduce,etc is implemented as long running operator (similar to Bolt in Storm)

Flink looks like a true successor to Storm like Spark succeeded hadoop in batch.

Advantages:

* Leader of innovation in open source Streaming landscape
* First True streaming framework with all advanced features like event time processing, watermarks, etc
* Low latency with high throughput, configurable according to requirements
* Auto-adjusting, not too many parameters to tune
* Exactly Once
* Getting widely accepted by big companies at scale like Uber,Alibaba.

Disadvantages

* Little late in game, there was lack of adoption initially
* Community is not as big as Spark but growing at fast pace now
* No known adoption of the Flink Batch as of now, only popular for streaming.

**Kafka Streams :**

Kafka Streams , unlike other streaming frameworks, is a light weight library. It is useful for streaming data from Kafka , doing transformation and then sending back to kafka. We can understand it as a library similar to Java Executor Service Thread pool, but with inbuilt support for Kafka. It can be integrated well with any application and will work out of the box.

Due to its light weight nature, can be used in microservices type architecture. There is no match in terms of performance with Flink but also does not need separate cluster to run, is very handy and easy to deploy and start working .

One major advantage of Kafka Streams is that its processing is Exactly Once end to end. It is possible because the source as well as destination, both are Kafka .

Advantages:

* Very light weight library, good for microservices,IOT applications
* Does not need dedicated cluster
* Inherits all Kafka good characteristics
* Supports Stream joins, internally uses rocksDb for maintaining state.
* Exactly Once ( Kafka 0.11 onwards).

Disadvantages

* Tightly coupled with Kafka, can not use without Kafka in picture
* Quite new in infancy stage, yet to be tested in big companies
* Not for heavy lifting work like Spark Streaming,Flink.
* Low latency , High throughput , mature and tested at scale

**Flume :**

Apache Flume is an open-source distributed system for ingesting online stream data from different sources to Hadoop HDFS or HBase. It is a highly available, reliable, and easy to use the system.

Apache Flume provides support for different sources and sinks. Apache Flume caters to high throughput with lower latency. Despite its disadvantages, Flume’s advantages outweigh its disadvantages.

Advantages:

### Open-source

Apache Flume is an open-source distributed system. So it is available free of cost.

Data flow

Apache Flume allows its users to build multi-hop, fan-in, and fan-out flows. It also allows for contextual routing as well as backup routes (fail-over) for the failed hops.

Reliability

In apache flume, the sources transfer events through the channel. The flume source puts events in the channel which are then consumed by the sink. The sink transfers the event to the next agent or to the terminal repository (like HDFS).

The events in the flume channel are removed only when they are stored in the next agent channel or in the terminal repository.

In this way, the single-hop message delivery semantics in Apache Flume caters to end-to-end reliability of the flow. Flume uses a transactional approach for guaranteeing reliable delivery of the flume events.

Recoverability

The flume events are staged in a flume channel on each flume agent. This manages recovery from failure. Also, Apache Flume supports a durable File channel. File channels can be backed by the local file system.

Steady flow

Apache Flume offers steady data flow between reading and writes operations. When the rate at which data is coming exceeds the rate of writing data to the destination, then Apache Flume acts as a mediator between the data producers and the centralized stores. Thus offers a steady flow of data between them.

Latency

Apache Flume caters to high throughput with lower latency.

Ease of use

With Flume, we can ingest the stream data from multiple web servers and store them to any of the centralized stores such as HBase, Hadoop HDFS, etc.

Reliable message delivery

All the transactions in Apache Flume are channel-based. For each message, two transactions are there – one for the sender and one for the receiver. This ensures reliable message delivery.

Import of Huge volumes of data

Along with the log files, Apache Flume can also be used for importing huge volumes of data produced by e-commerce sites like Flipkart, Amazon, and networking sites like Twitter, Facebook.

Support for varieties of Sources and Sinks

Apache Flume supports a wide range of sources and sinks.

Streaming

Apache Flume gives us a reliable solution that helps us ingesting online streaming data from different sources (such as email messages, network traffic, log files, social media, etc) in HDFS.

Fault-tolerant and scalable

Flume is an extensible, reliable, highly available, and horizontally scalable system. It is customizable for different types of sources and sinks.

Inexpensive

It is an inexpensive system. It is less costly to install and operate. Its maintenance is very economical.

Configuration

Apache Flume contains a very declarative configuration.

Documentation

Flume provides complete documentation with many good examples and patterns which helps its user to learn how Flume can be used and configured.

Limitations :

. Weak ordering guarantee

Apache Flume offers weaker guarantees than the other systems such as message queues in the event of moving data more quickly and for enabling cheaper fault tolerance. In Apache Flume’s end-to-end reliability mode, the flume events are delivered at least once, but with zero ordering guarantees.

. Duplicacy

Apache Flume does not guarantee that the messages reaching are 100% unique. In many scenarios, the duplicate messages might pop in.

. Low scalability

Flume scalability is often low because for any businesses, sizing the hardware of a typical Apache Flume may be tricky, and in most of the cases, it is trial and error. Due to this, Flume scalability aspect is often under the lens.

. Reliability issue

The throughput that Apache Flume can handle depends highly upon the backing store of the channel. So, if the backing store is not chosen wisely, then there may be scalability and reliability issues.

. Complex topology

It has complex topology and reconfiguration is challenging.

Its hard to pick one , but if I have to pick any, I’ll go with **Flink** ,

Because

Flink is fully real-time data streaming framework. Although no microbatches here, but it is faster than Storm, and in big data, being fast can save not only time, but also money. It is also fully scalable - thousands of nodes can be added if needed. It also has a very strong fault-tolerance and is built to ensure that we only process each data point once (sometimes called exactly-once processing). We can also utilize Scala with Flink, similar to Spark Streaming - but again with event-based real-time data rather than microbatches.

Flink is a run-time engine that can be run on top of a variety of different platforms, including Hadoop/YARN, AWS, and Google Cloud. It has two APIs that we can use to process either streaming or batch data.

* **DataStream API:** This includes something for event processing called CEP and something for querying, called Table.
* **DataSet API:** This helps deal with batch data, and includes FlinkML for machine learning, Gelly for graph processing, and Table for querying.

Flink connects to just about anything we can think of, including:

* HDFS
* Cassandra
* Kafka