KEVIN JOHNSON

MODULE 8 READING COMPREHENSION

1. **Describe the publish/subscribe model as it pertains to events, messaging systems, producers, consumers, and message brokers. (2 Pts)**

In a stream processing context, a record is known as an event. An event is a small, self-contained, immutable object containing the details of something that happened at some point in time. In streaming, an event is generated once by a product (*publisher*s) and then potentially processed by multiple *consumers* (*subscribers*). In a streaming system, related events are usually grouped together in a *topic* or *stream*. In principle, a file or database is sufficient to connect producers and consumers: a producer writes every event that it generates to the datastore, and each consumer periodically polls the datastore to check for events that have appeared since it last ran. However, when moving toward continual processing with low delays, polling becomes expensive if the datastore is not designed for this kind of usage. The more often you poll, the lower the percentage of requests that return new events, and the higher the overheads become. Instead, it is better for consumers to be notified when new events appear.

A common approach for notifying consumers about new events is to use a messaging system, where a producer sends a message containing the event, which is then pushed to consumers. Within the publish/subscribe model, different systems take a wide range of approaches and there is no one right answer for all purposes. To differentiate the systems, two questions are helpful:

- What happens if the producers send messages faster than the consumers can process them?
- What happens if nodes crash or temporarily go offline-are any messages lost?

An alternative is to send messages via a message broker. A message broker is a kind of database that is optimized for handling message streams. It runs as a server, with producers and consumers connecting to it as clients. Producers write messages to the broker, and consumers receive them by reading them from the broker. By centralizing the data in the broker, these systems can more easily tolerate clients that come and go (connect, disconnect, and crash), and the question of durability is moved to the broker instead.

2. **What are the two main patterns of messaging when using multiple consumers? (2 Pts)**

When multiple consumers read messages in the same topic: two main patterns of messaging are used:

- *Load Balancing*: Each message is delivered to one of the consumers, so the consumers can share the work of processing the messages in the topic. The broker may assign messages to consumers arbitrarily. This pattern is useful

when the messages are expensive to process, and so you want to be able to add consumers to parallelize the processing.
- *Fan-out:* Each message is delivered to all the consumers. Fan-out allows several independent consumers to each "tune in" to the same broadcast of messages, without affecting each other.

3. **Explain the concepts of change data capture and event sourcing, and how they differ. (2 Pts)**

Change data capture is the process of observing all data changes written to a database and extracting them in a form in which they can be replicated to other systems. Change data capture makes one database the leader and turns the other into followers. Database triggers can be used to implement change data capture by registering triggers that observe all changes to data tables and add corresponding entries to a changelog table.

Event sourcing compared to data capture has one major difference. In data capture, the application uses the database in a mutable way, updating and deleting records at will. In event sourcing, the application logic is explicitly built on the basis of immutable events that are written to an event log. The event store is append-only, and updates and deletes are discourage or prohibited.

Event sourcing involves storing all changes to the application state as a log of change events and is a powerful technique for data modeling. From an application point of view, it is more meaningful to record the user's actions as immutable events. Event sourcing makes it easier to understand after the fact why something happened and guards against application bugs.

4. **What are Complex Event Processing and Stream Analytics, and how do they differ? (2 Pts)**

Complex event processing is an approach that allows you to specify rules to search for certain patterns of events in a stream. These systems use a high-level declarative query language such as SQL to describe the patterns of events that should be detected. These queries are submitted to a processing engine that consumes the input streams and internally maintains a state machine that performs the required matching. When a match is found, the engine emits a complex event with the details of the event pattern that was detected. In CEP engines, queries are stored long-term, and events from the input streams continuously flow past them in search of a query that matches an event pattern.

Another area in which stream processing is used is for analytics on streams. Compared to CEP, stream analytics is less interested in finding specific event sequences and is more oriented toward aggregations and statistical metrics over a large number of events. Statistical metrics are usually computed over fixed time intervals, known as windows. Stream analytics systems sometimes use probabilistic algorithms. Probabilistic algorithms produce approximate results, but have the advantage of requiring significantly less memory in the stream processor than exact algorithms.

5. **What are some machine learning algorithms that use greedy algorithms? (2 Pts)**

One machine learning application that uses greedy algorithms is feature selection in linear regression. In stepwise feature selection, a greedy algorithm can be implemented to evaluate the explanatory variables and find the optimal formula to fit to the predictor.

Greedy algorithms can also be used as layer-wise pretraining for neural networks. Pretraining is based on the assumption that it is easier to train a shallow network rather than a deep network. As an optimization process, a greedy algorithm divides the training process into a succession of layer-wise training processes. This leads to an aggregate to locally optimal solutions, which in theory should lead to a globally optimal solution.