KEVIN JOHNSON

MODULE 4 READING COMPREHENSION

1. **Discuss some of the problems with JSON, XML, and CSV file types. (2 Pts)**

   There is a lot of ambiguity around the encoding of numbers in JSON, XML, and CSV. In XML and CSV, you cannot distinguish between a number and a string that happens to consist of digits. JSON distinguishes strings and numbers but not does not distinguish integers and floating-point numbers and does not specify precision. This is a problem whem dealing with large numbers.

   JSON and XML do not support binary strings. People can get around this by encoding the binary data as text using Base64. However, this increases the data size by 33%. CSV does not have any schema, so it is up to the application to define the meaning of each row and column. If an application change adds a new row or column, the change has to be handled manually.

2. **What is Thrift and Protocol Buffer, and how is it different than Avro? (4 Pts)**

   Apache Thrift and Protocol Buffers are binary encoding libraries. Both require a schema for any data that is encoded. Thrift and Protocol Buffers each come with a code generation tool that takes a schema definition and produces classes that implement the schema in various programming languages.

   Thrift has two different binary encoding formats: BinaryProtocol and Compact Protocol. In BinaryProtocol, each field has a type annotation and a length indication (where required). It has no field names. Instead, the encoded data contains field tags, which are numbers. Those are the numbers that appear in the schema definition and act like aliases for fields. Compact Protocol packs the field type and tag number into a single byte and uses integer length integers. Protocol buffers does the bit packing differently but is otherwise very similar to Compact Protocol.

   Apache Avro is another binary encoding format. Avro uses a schema to specify the structure of the data being encoded. It has two schema languages: Avro IDL is intended for human editing and the other, based on JSON, is more easily machine readable. The schema in Avro has no tag numbers. The binary encoding is much more compact than Thrift and Protocol Buffer. To parse the binary data, you go through the fields in the order that they appear in the schema and use the schema to tell you the datatype of each field. This means the binary data can only be decoded correctly if the code reading the data is using the exact same schema as the code that wrote the data.

3. **In regards to RPC, how is a network request different than a local functions call? (2 Pts)**

   The RPC Model, or remote procedure call, tries to make a request to a remote network service look the same as calling a function or method in your programming language. However, a network request is very different from a local function call:

   - A local function is predictable and either succeeds or fails depending only on parameters that are under your control. A network request is unpredictable due to the request or response being lost due to a network problem or the remote machine may be slow or unavailable. Such problems are outside your control.
   - A local function call either returns a result, throws an exception, or never returns (because it goes into an infinite loop or the process crashes). A network request has another possible outcome: it may not return a result due to a timeout. In that case, you do not know what happened.
   - Every time you call a local function, it normally takes about the same time to execute. A network request is much slower than a function call, and its latency is also wildly variable.
   - When you call a local function, you can efficiently pass it references to objects in local memory. When you make a network request, all those parameters need to be encoded into a sequence of bytes that can be sent over the network.
   - The client and the service may be implemented in different programming languages, so the RPC framework must translate datatypes from one language into another.

4. **What are message brokers? Give some examples. (2 Pts)**

   Asynchronous message-passing systems are somewhere between RPC and databases. They are similar to RPC in that a client's request is delivered to another process with low latency. They are similar to databases in that the message is not sent via a direct network connection, but goes to an intermediary called a message broker, which stores the message temporarily. Some examples of message brokers that are becoming more popular are RabbitMQ, ActiveMQ, HornetQ, NATS, and Apache Kafka.

   There are several advantages to message brokers compared to RPCs:

   - It can act as a buffer if the recipient is unavailable or overloaded, and thus improve system reliability.
   - It can automatically redeliver messages to a process that has crashed, and thus prevent messages from being lost.
   - It avoids the sender needing to know the IP address and port number of the recipient.
   - It allows one message to be sent to several recipients.
   - It logically decouples the sender from the recipient.