Here is the task we would like you to consider. You should be able to make reasonable assumptions if some of the requirements are not precise enough and we will be able to discuss during the discussion. If you do not have any parts of the requirements implemented, please, provide your design thoughts, if you do not have them yet, please, let us know as well. It's not all or nothing, try to do what you can picking the items you feel more comfortable with.

Allowed time: 24 hours
Delivery: Push to a git repository

Problem

Design and implement in Scala/Java the following System.

The System consists of a Server, a Reader and a Writer. All of them expose some basic stats about the requests they issue or receive etc via JMX or an HTTP server. The components in the System connect via TCP and exchange data encoded as byte strings. All the components handle connection failures, log any failures, and retry to connect periodically. They also send separate KeepAlive messages (pick your format for those) and the other side disconnects if the KeepAlive did not arrive in time. We need a working prototype but please, also write unit tests for the critical parts of the System.

Writer

The Writer opens a connection to the Server and sends a 0 byte to greet the Server. The exchange that follows proceeds by the Server issuing a request by sending an offset and an integer containing the number of requested integers for the Writer to send to the Server. The Writer always sends sequential integers from the requested offset onwards. So if the Server request was (5,3), the Writer will reply with 5,6, and 7 and wait for the next request for numbers. There is no acknowledgement sent back to the Writer from the Server on receiving a new subsequence successfully as it is down to the Server to handle the case when the reply has not been received in full.

Reader

The Reader's job is to maintain up to 1000 concurrent 'sequences' that get processed and replaced by new ones as fast as possible like this. Initially, it has no sequences so it creates them one by one by generating a new UUID for each of them and connecting via (one or more connections) to the Server and sending a pair (UUID,0). The Server will then start sending back sequential integer numbers as soon as it has them available, which may not be immediately (in actual fact, the Server will send exactly 10 numbers--as per the spec for the Server below--for any sequence denoted by this UUID but the Reader does not know this). For each sequence, the exchange proceeds in lockstep: the Server replies with one number by sending a pair (UUID,Number) and the Reader requests the next number by sending the UUID for this sequence. The Reader checks all the integers that each one is Previous+1 and logs and reports the count and description of any errors. Once it has received a termination message (UUID,-1) from the Server, it knows that the sequence is complete, it then replies with the same UUID one final time and the sequence is considered to be complete. The Reader has 1000 sequences with unique

UUID’s in circulation at all times once 1000 are initiated and immediately starts a new sequence with a new UUID once all numbers for a given sequence are received. It is your decision whether to use one connection or many. If the Reader detects that the Server is down, it will terminate all the currently in-flight sequences and restart creating them and consuming them from scratch. The Reader exposes the number of completed sequences and sequences in-flight via JMX and possibly an HTTP server.

Server

The Server keeps consuming the integer numbers from the Writer at a pace that is actually required, not storing too much data for no reason. It first persists in a database or data store all the data it receives from the Writer, so that if it is stopped and restarted, it will know where it is with respect to the Writer. When a new request for sequence arrives from the Reader, if the capacity is not full (i.e., the number of active sequences is <=1000), the Server will allocate a new sequence internally and copy up to 10 numbers sequentially from the data received from the Writer—they will happen to be sequential integer numbers but the Server does not verify that—it is down to the Writer to ensure that the data it sends are always incrementing. The Server then ships all the unsent data for any sequences to the Reader. Once the sequence data (all 10 integers) is fully sent, the Server sends a termination message (UUID,-1), liberates that slot and becomes ready to accept a new request for a sequence.