GitHub repository:

https://github.com/TuFoZo/RippleD-LedgerSeqTime

Output from the app:

https://github.com/TuFoZo/RippleD-LedgerSeqTime/blob/master/output.txt

Plot constructed from the output using Gnuplot:

https://github.com/TuFoZo/RippleD-LedgerSeqTime/blob/master/Plot.JPG

How does the app work?

Summary: Script makes JSON RPC calls to Ripple public servers' every x seconds for x minutes of duration. After every request to the ripple server, response of time & validated_ledger.seq from Ripple public server is written to a local file. After polling for x minutes of duration, local output file is loaded into array and then calculated min, max, avg time of when ledger seq are incremented over given time.

Detail:

RippledServerInfo class - All global values are declared in the class RippledServerInfo for ease of use and clarity. i.e. static local file, polling interval, duration of poll, Ripple URL, JSON Request

Main Method –

- Main method which is the entry point to our app starts by initializing a file in the local system.
- A File class object *file* is created using the location from the global variable of MY_FILE.
- New PrintWriter object is created using the *file* object which also recreates the file if it already exists so every run has fresh data in the output file.
    - Using the PrintWriter class since we only need to write text format.
- A new *timer* is created from the Timer class so that we can schedule a task.
- The new *timer* creates a schedule process using our own class GetTimeSyncData which extended the TimerTask class with 0 delay for every POLL_INTERVAL seconds.
- Once this task is started, it runs in the background. After sleeping for SLEEP_DURATION amount, timer is canceled and purged.
- In other words, after we query the RippleD server for SLEEP_DURATION, the app cancels/purges all timers and then closes the file if not closed already.

Main Method - Calculating min, max, avg:

Assumption: Because sampling (polling) rate is 1 sec, every validated ledger sequence has equal weight in our sampling. In other words, if a validated ledger sequence number is found 5 times in our data output that would indicate our particular ledger seq took 5 seconds to validate and close.

- Using the output file created, file is scanned while looping through each line and splitting, to get only sequence numbers, and then add that to our array list.
- Next, as the code iterates through the array, we place each value in our hashmap. Once we get our data in a hashmap, we use collections class to calculate min, and max values.
- To calculate the average, we add all occurrences and then divide it by the number of sample size.
- This also allow us to extrapolate how long it took for each ledger to validate and close as well which I output to the console.

GetTimeSyncData - This class extends TimerTask class implementing Runnable interface so that we can override and run it using timer.schedule from our main.

Summary: This method sends a JSON request to the public Ripple server, parses the response and writes to the local file every time it is called.

- An Instance of *httpConn* URLConnection is returned from the *url* object created and then after setting a few methods such as setting the content type to JSON as well as setting this connection to be used for sending data to the server as well as setting the request method to a POST.
- After obtaining an OutputStream instance *os* to be able to write to the *httpConn* object, JSON request as bytes is written to the *httpConn* connection object.
- As the bytes are converted from the httpConn InputStream using the InputStreamReader to characters, they are buffered and stored in *br* BufferedReader object.
- At this point, if the HTTP Status is 200 meaning our connection's response from server is OK:
    - Modifiable *StringBuilder* class object is created so that the output from our *br* object can be stored in it and then converted to String text.
    - New JSONObject is created from the *resp* String text and then parsed the sequence from the retrieved JSON response as *intSeq* integer as well as the time as *strDateTime* string.
    - A few actions performed on the response:
        - Removed the miliseconds from the *strDateTime* string by removing anything after '.' from text
        - Formatted the *strDateTime* string as Calendar *cal* instance so that I could do analysis to calculate min/max/avg. It turned out I couldn't use this because I couldn't return an object for the run() due to the design.
    - At this point, concatenate the time and sequence from the response and write to our file while separating them with a comma delimiter
    - Making sure that the bufferedreader bj object and the httpconn are closed since we will call this GetTimeSyncData method many times using our task scheduler.

How did you decide on the polling interval?

- After I ran a preliminary test to observe historical data on how often the sequence number gets incremented, I realized that a new ledger is validated within a few seconds.
- Therefore, I chose 1 sec as the polling interval so that I have enough granularity to get the bigger picture. Anything less than that would unnecessarily DOS on the server as well as put strain on resources and bandwidth. Anything more, such as 1 minute, would miss the bigger picture.

- I should mention that time of the day of when the polling occurs makes a difference in the results. Therefore, to get the most accurate results, it would be good to extend this and have the system collect different times of the day.
    - At different hours over a week or month, depending on how accurate we would want the results. Also, I would think the XRP transaction volume due to x reasons would skew these results.
    - On the other hand, it is unnecessary as XRP metrics is already available by Ripple.

What do the results tell you?

- 45 Million+ ledgers have been closed since inception of 2012 and that's a remarkable achievement, considering no issues since ledger 1.
- Fast ledger validations signify how quick the settlement speeds are; in other words, network reaching consensus.
- Other popular cryptocurrencies such as Bitcoin and Ethereum payments cannot come close to XRP near real-time settlement times.
- It can be said how efficiently (minimal 'disagreement' delays) nodes agree with the majority to choose a final version of ledger.
    - In other words, not only the XRP Ledger Protocol is ahead of other cryptocurrencies but also there is still more room to push TPS.

What might explain the variation in time between new ledgers?

- The time it takes for majority of the nodes to agree to declare a consensus naturally varies in time due to Consensus Process of the XRP Ledger Protocol
- Not only majority of the nodes/validators need to agree on ledger but also need to validate which all add time variations to the dynamic nature of the ledger validation process.
- It could also be due to high transaction volume, network latencies/congestion
- One unlikely but possible reason: 'nefarious cartel' interferences to thwart consensus. (Of course, I have not seen this in any of the metrics I have observed but a possible scenario
    - Reference: https://ripple.com/files/ripple_consensus_whitepaper.pdf

---

Bonus Question #1:

Enhance your script to calculate the min, max, and average time that it took for a new ledger to be validated during the span of time captured.

- See github repository please.

App includes additional code that calculates the min, max and avg time it took for a new ledger to be validated during the span of time captures.

There are other (better) ways that you could use the rippled API to find how long each ledger took to close/validate. Using the API documentation, find and describe one of these methods (you don't need to actually implement it.)

Easiest way to find the answer would be to look at here ☺

https://xrpcharts.ripple.com/#/metrics

- Using the URL of *https://data.ripple.com*, Rippled public API can be accessed and queried easily.
- One of the available methods is *Get Stats* that retrieve statistics about the XRP Ledger, including ledger_count & ledger_interval.

  ledger_count : The number of ledgers closed during this interval.

  ledger_interval : The average number of seconds between ledgers closing during this interval.

- Example request: https://data.ripple.com/v2/stats/?start=2019-03-08&end=2019-03-09&interval=hour&family=metric&metrics=ledger_count,ledger_interval
- By providing start and end time as well as the interval, family of metrics, we can easily retrieve the ledger_count and ledger_interval.
- In summary, the API will just provide the information we need in JSON once we submit the fields.

We can easily wrap basic code around the response and parse the required information

start=2015-08-30, end=2015-08-31

interval=day

family=metric

metrics=accounts_created,exchanges_count,ledger_count,payments_count…

Visual Plot Constructed from output data:

Output from the app is imported into Gnuplot to construct a plot that visualizes how frequently the ledger sequence is incremented over time.

Image exported from the GNU Plot:

https://github.com/TuFoZo/RippleD-LedgerSeqTime/blob/master/Plot.JPG

Output used to construct the plot:

https://github.com/TuFoZo/RippleD-LedgerSeqTime/blob/master/output.txt