Ripple IE Tech Challenge Write Up - Craig Sim – 11th April 2020

GitHub Contents

Please find enclosed, within the Github repository, two Python 3 scripts:

- ripple.py – the analysis and visualisation script:
  - Produces visualisation, Fig. 1, and creates XRPLedger_ServerInfoLedgerCloseFrequency.csv
  - Please find and example of XRPLedger_ServerInfoLedgerCloseFrequency.csv enclosed with the Github repository
- ripple_visualisation_only.py – visualisation only script:
  - Loads XRPLedger_ServerInfoLedgerCloseFrequency.csv, expects CSV to be in the present working directory, and re-produces visualisation, Fig. 1

To run both scripts type: python ripple.py or python ripple_visualisation_only.py at the command line.

(PLEASE NOTE: The scripts use Python libraries: requests, pandas, time, datetime, matplotlib and seaborn. Depending on your python instillation you may need to download one or more of these libraries using pip or conda).

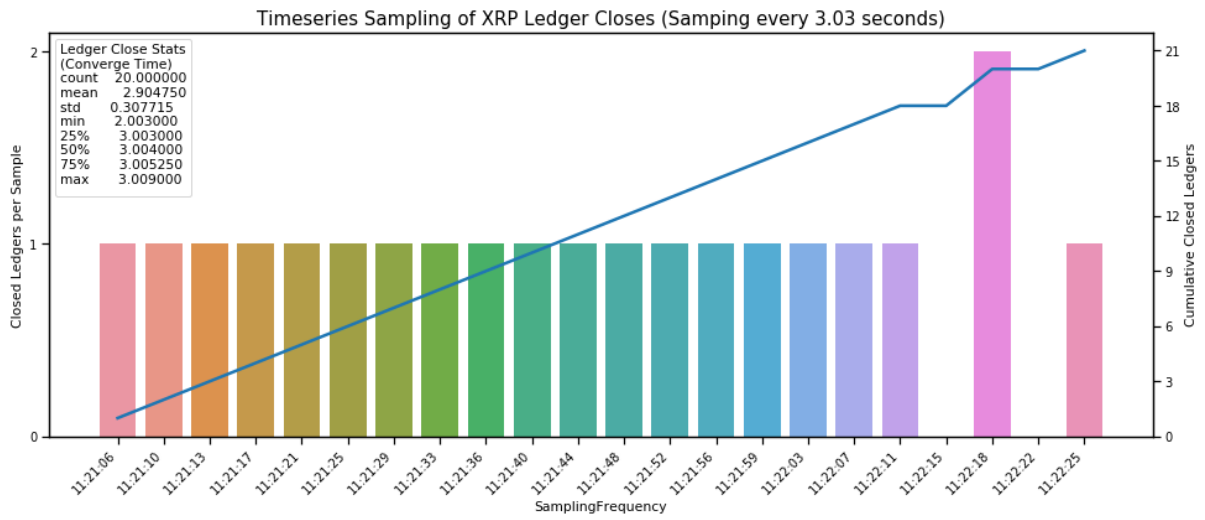

Fig 1. Example Visualisation: Bar plot shows 'Closed Ledgers per Sample", overlaid Lineplot shows "Cumulative Closed Ledgers", legend shows Ledger Close statistics.

How Does ripple.py Work?

The script is a Python 3 script that calls the "server_info" function on the Ripple XRPLedger WebSocket API server URL s1.ripple.com:51234. Analyses and extracts Ledger Close information, relating to validated ledger sequence number and current time, and generates a visualisation of the timeseries analysis sampled during the runtime of the script. The visualisation demonstrates how the ledger close times can vary through time.

Ripple.py has three main functional component's:

- The class XRPLedgerWebSocketAPI
- The pandas dataframe results
- The seaborn and matplotlib visualisation

Class XRPLedgerWebSocketAPI

The class has three methods:

- __init__ - the parameterised constructor which expects a string for the WebSocket API URL connection
- getRequest – which mimics a cURL command. It expect the WebSocket API command in JSON format. getRequest will call the WebSocket API using the URL provided to the __init__ constructor. It returns the JSON array which has been returned from the WebSocket API
- getServerInfoLedgerCloseFrequency – which takes two parameters:
    - Iterations – number of times to call the WebSocket API
    - sample_frequency – the time to "sleep" between each iterations call to the WebSocket API

  The function initialises the first sequence number by calling the server_info WebSocket API function. It then for loops, number of loops determined by 'iterations' parameter. Each for loop starts with a sleep for a duration determined by the 'sample_frequency' parameter. The function then does the following:
    - Unpacks the server_info JSON array, returned from getRequest, extracts the sequence number and compares to the previous iterations sequence number to determine how many ledgers have closed in this iteration. Additionally it extracts the 'time' and the 'last_close' 'converge_time_s' and 'proposers'.
    - The for loop calculates the difference in sequence number 'seq_diff' and the cumulative total for the differences in sequence numbers 'cumulative_seq'.
    - The extracted and calculated results are stored in an list.

  After the for loop has completed the results are loaded into a Pandas dataframe which is returned to the calling function.

__main__ section

The __main__ is the controlling section of the script, it has the following functionality:

- It creates the XRPLedgerWebSocketAPI class and passes in the URL
- Determines the iterations and sample_frequency.
- Calls getServerInfoLedgerCloseFrequency to obtain the pandas results from calling server_info
- Saves the results pandas dataframe
  to XRPLedger_ServerInfoLedgerCloseFrequency.csv
- Analyses the 'Last Ledger Close' column from results to extract the min, max and mean using the pandas function describe(). This provides the answer to Bonus Question #1. The output of describe() is converted to a string to display as a legend on the visualisation.

- Create the visualisation using seaborn and matplotlib. The visualisation is the overlay of two plots, a bar chart of the 'Closed Ledgers per Sample' and a line plot of 'Cumulative Closed Ledgers. The bar chart y axis is on the left and the line plot on the right. Both plots have 'SamplingFrequency' as their x axis. Additionally, as described above, Ledger Close Statistics are displayed as a legend in the top left corner. Please refer to Fig 1.

How Does ripple_visualisation_only.py Work?

ripple_visualisation_only.py loads and XRPLedger_ServerInfoLedgerCloseFrequency.csv and re-runs the visualisation. There are no classes or functions defined only a __main__ section which loads XRPLedger_ServerInfoLedgerCloseFrequency.csv into a pandas DataFrame and then re-runs the legend creation and visualisations as per the functionality described in the previous section.

How did you decide on the polling frequency?

After running the visualisation several times the statistics created for the Ledger Close Statistics legend consistently defined the over 75% of the Ledger Closes took slightly more than 3 seconds, typically between 3 to 3.03 seconds, with 25% below 3 seconds. I decided to create a visualisation that explored how the ledger close can step out of sequence, setting the sample frequency to 3.03 seconds to explore how often the sequence changes within the sampling window.

What did the results tell me?

Choosing a sampling window of 3.03 seconds I'd expected 1 ledger close per sample, in most cases. Running the visualisation at a different times of the day resulted in slightly different outcomes, however they consistently showed that it was rare for 1 ledger close to per sample per minute. There was typically ledger close variation between 2 seconds and 3 seconds, 25% of cases, with occasional significant increases above 3.03 seconds. The number of ledger closes was typically 1 or 2 less than the 22 sampling iterations.

The results inform us that XRP Ledger does not to consistently close 20 Ledgers per minute.

What might explain the variation in XRP Ledger closes?

The XRP Ledger consensus algorithm can converge very quickly if all transactions included are not disputed. However if there are an unusual amount of disputes, and the transactions have to be removed, this may slow down the convergence. The visualisation suggests that disputes may happen 2 or 3 times every 20 ledger closes, or every minute of elapsed time.

Bonus Questions

Bonus Question #1

As mentioned in the section describing ripple.py, and the section describing the visualisation, bonus question #1 was answered by recording the 'last_close' : 'converge_time_s'. Holding the 'converge_time_s' results as a column within the results pandas dataframe, as column 'Last Ledger Close', allowed me to us the built in statistics method 'describe()' to extract min, max and mean.

Bonus Question #2

Assuming that the analysis we are undertaking is a visualisation of Ledger Closes from time A to time B, it may be better to "bulk down load" multiple contiguous ledger postings. This approach would require creating a list of ledger sequence index numbers and calling the WebSocket API 'ledger' method.

We'd then be able to produce a more complete analysis, as we would be confident we have the required dataset.