Big Data Technologies

Assignment 2

Overview

Part 1 Goals

- Revisit the concepts of the lectures and apply some of the information/lessons to *new situations*.

Part 2 Goals

- Apply a generalisation of the content in the lectures in practical settings.

Submission

- Answers to be submitted on SUNLearn.

Part 1: Big Data Concepts

Question 1: Big Data General [3]

1. In some scenarios high velocity data does not lead to high volume data. What data governance principle may be applied here to enforce/ensure that? [1]
2. Suppose you are ingesting sensor data from many IoT devices (delivering data at a high rate), which are susceptible to data corruption and loss. You are using a standardised data format from all devices and are collecting these data whilst retaining a significant history. Attribute a high or low prevalence to each dimension based on this problem description (e.g. Low Volume, . . .). [2]

Question 2: Data Lifecycle [4] In the lectures we discussed the concepts of DevOps, DataOps and MLOps. These approaches allow us to deliver code, data or models quickly and reliably.

1. Compare MLOps and DataOps by means of their characteristics/patterns/principles (e.g. how they approach teams, testing, etc.) and produce a comparison table. [4]

Question 3: Data Architectures - The Lakehouse [5] Your organisation has constructed a schema-on-read architecture by leveraging object storage in the cloud (e.g. Google cloud storage, AWS S3).

1. List three benefits of this approach over using schema-on-write systems (from a big data perspective)? [3]
2. Although this architecture works well for large unstructured data, it presents challenges when it comes to reporting and business intelligence operations,as direct queries to the data do not benefit from the optimisations you may have in your typical RDBMS systems. To this end, a *Data Lakehouse* has been proposed as a solution. What data access and storage simplications are introduced by the Lakehouse? [2]

Question 4: Tensorflow serving [4] In the lectures we demonstrated how a to package a simple scorer as a Flask application within Docker.

1. Describe what the following code block is doing. [2]

```
nohup tensorflow_model_server --rest_api_port=8500  --model_name=fashion_model
    --model_base_path="${MODEL_DIR}"
```

2. Why would you wrap this in a Docker container? [1]
3. Which base image would you use when compiling the docker image? [1]

Part 2: Map Reduce & NoSQL

Question 5: Minimal Map Reduce; Grep [9] Consider the *distributed grep* example provided in Dean et al..

1. Write pseudo code for the *mapper* and *reducer* (see Dean et al.). [2]
2. Suppose we wish to search a file that is distributed over three nodes, named A, B and C (`A:part-0000`, `B:part-0001`, `C:part-0002`). The text file is split as follows:

File	Contents/lines
`part-0000`	A book is made from a tree. It is an assemblage of flat, flexible parts (still called "leaves")
`part-0001`	imprinted with dark pigmented squiggles. One glance at it and you hear the voice of another person,
`part-0002`	perhaps someone dead for thousands of years. Books break the shackles of time — proof that humans can work magic

You wish to perform a distributed grep of the word `book` (assume it is case insensitive). Provide the output of the *map* tasks and how the consequences of these tasks are distributed over the nodes (e.g. A: (key1, value1), B: (key2, value2), ...). [3]

3. Assuming we are using Hadoop's map-reduce, how will these data be processed by reducers (e.g. R1: (key1, value1), etc.)? [2]
4. During this entire process, which data are written to disk (in a Hadoop implementation)? [2]

Question 6: Python Map Reduce; Term Vector [5] Consider the notebook and the term-vector/host problem description given in Dean et al.

1. Write a psuedo code implementation of the *map* and *reduce* functions. [2]
2. Implement these functions in the provided notebook and submit them on SUNLearn. [3]

Your notebook is required to run end-to-end in order to be a valid submission. Consider the discussion of word frequencies here for context on how they may be used in *search*.

Question 7: Word Frequency on NoSQL tools [10] You wish to search for relevant documents by leveraging the capabilities of ElasticSearch. It maintains an inverted index over documents and stores term-vectors for document contents. Furthermore, it may be used as a backend for Tensorflow and for storing unstructured data in general.

1. Describe and motivate two important properties that make it useful within a big data architecture? [2]
2. Which clustering approach does it take (e.g. master/minion)? Describe this architecture very briefly with specific reference to sharding and replicas. [2]
3. Use the notebook to demonstrate that you can observe the term vectors for words (submit your result and code block). Describe what these term-vectors are computed on (how it relates to the index, documents, and fields)? [2]
4. Identify a term that occurs seldom in all the texts (by querying ElasticSearch). Submit the word and notebook cell block code that allowed you to find it. [2]

5. The search function on ElasticSearch is already performing an inverted index, find a sentence in which the term you identified as rare, is present. Submit the response and the code block that generated it. [2]

Note:

- The server is setup for you within the notebook (ElasticSearch can run within Google Colab)
- You may use the language of your choice (you may also use `curl` and API instructions directly).
- A valid submission is a notebook that works *end-to-end*.
- Query more frequently occuring terms for interest's sake. You will notice the scores change and the response time remains incredibly fast.

Question 8: Capturing Unstructured Data to NoSQL via APIs [12] You wish to gather data on non-fungible tokens for later analysis of purchasing patterns on a specific collection on the OpenSEA platform. As this is still discovery work, it is not clear which fields may be of value. Furthermore, you do not want to request data each time you require a new field. Consequently, you have decided to store each API response as a document in a MongoDB collection. You have also decided to investigate a single collection at first (in this case CryptoPunks).

1. Produce code that reads from the events API and writes the responses as documents in a MongoDB collection. [3]
2. How many event transactions are contained in MongoDB? Demonstrate by querying the database. [1]
3. Query the database and find the token identifier and selling price for the ten most expensive punks. [2]
4. How many unique punks were sold in this period? [2]
5. On which day did the most punks sell (where `"event_type": "successful"`) [2]
6. What is the total value of Etherium traded on that day (using `"event_type": successful`)? [2]

Note:

- You will need to submit the answers and code for each question (and a working notebook).
- The benefit of using a document oriented database is that we can retain the entire JSON response from the API for later processing — if we discover that we wish to process more fields later, then we may query the database instead of the API.
- Use the time range [2021-09-01T00:00:00+02:00, 2021-10-01T00:00:00+02:00] in your query (the *occurred_before* and *occurred_after* values on the Event API)
- You can convert wei units to ether (`10^18`) — treat the value as a string at first and strip `17`, e.g. `float(total_price[:-17])/10`
- You may use the MongoDB query language directly if you are able. You may also use the R Mongo package if you are more familiar with R. We recommend the use of `pymongo`.