README.md

Course project - Big Data Processing

Team

- Dmytro Lutchyn
- Yarema Mishchenko

System design

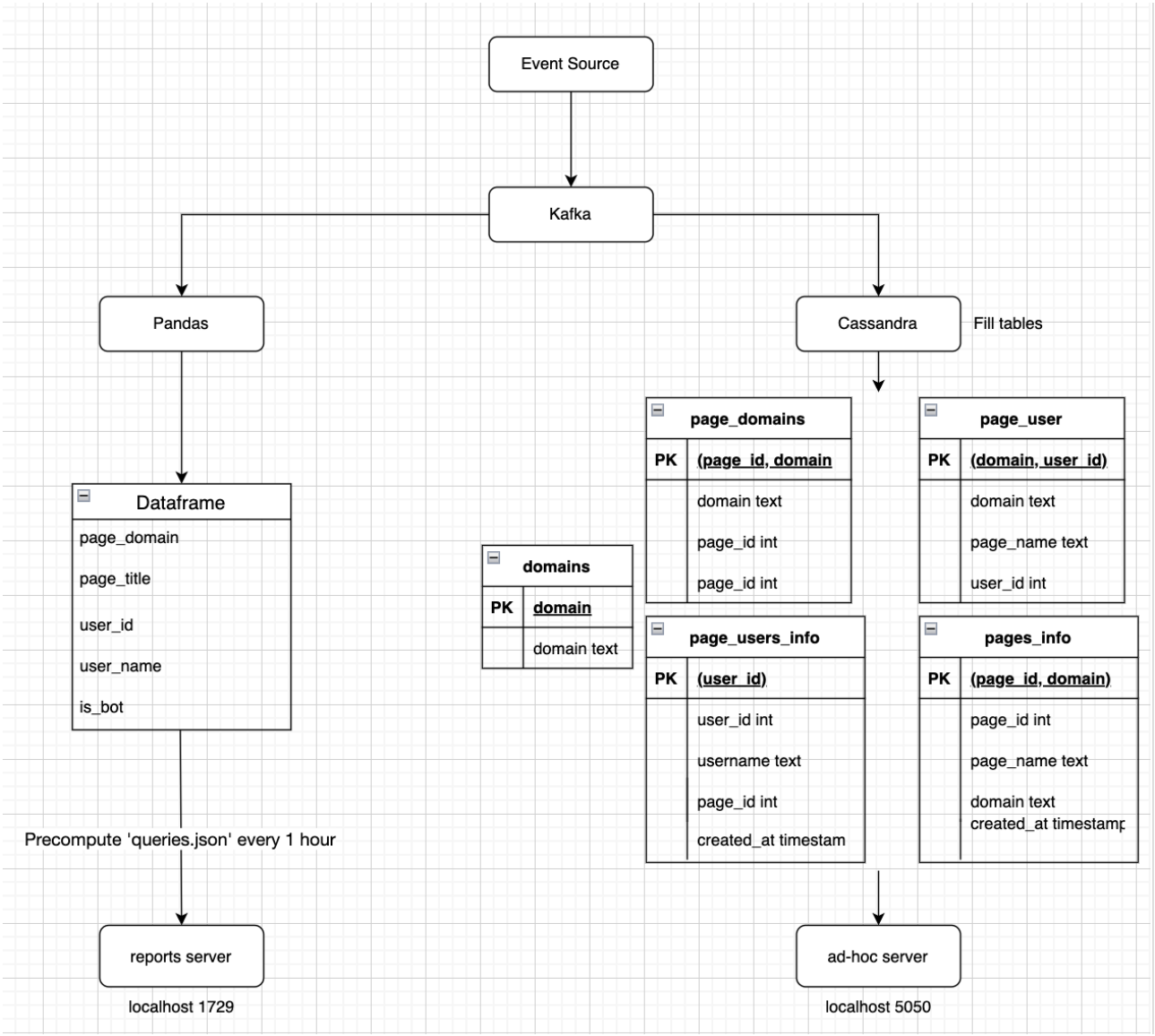
Our system consists of 4 independent services, as well as Kafka, Zookeeper, and Cassandra instances.

These 4 services are, as named in the project:

- `producer/producer.py` - responsible for fetching the Wikipedia data from the stream and passing it forward
- `consumer/consumer.py` - responsible for processing the data, filling Cassandra tables and maintaining a Pandas dataframe
- `precomputed-reports/reports_server.py` - responsible for maintaining a JSON file with precomputed outputs and serving it upon request
- `ad-hoc-queries/ad_hoc_server.py` - responsible for maintaining a Cassandra client connection and serving query results upon request.

In the diagram below, you can see how all those components create a coherent system.

Diagram


Event Source
Kafka
Pandas
Cassandra
Fill tables
Dataframe
page_domain
page_title
user_id
user_name
is_bot
Precompute 'queries.json' every 1 hour
reports server
localhost 1729
page_domains
PK
(page_id, domain
domain text
page_id int
page_id int
page_user
PK
(domain, user_id)
domain text
page_name text
user_id int
domains
PK
domain
domain text
page_users_info
PK
(user_id)
user_id int
username text
page_id int
created_at timestam
pages_info
PK
(page_id, domain)
page_id int
page_name text
domain text
created_at timestamp
ad-hoc server
localhost 5050

Kafka - producer & consumer

As events come into the system, they pass through a Kafka MQ. This is done for scalability as well as for separating services with different purposes. Preprocessing events in the producer allows the consumer to focus on the important logic, which is writing records to a Cassandra DB and precomputing reports.

For reports from Category A, we are using Pandas instead of Apache Spark. The reason is twofold:

- The amount of data is small enough to be suitable for a non-distributed system. We are keeping track of all records from the last 8 hours, which (at the rate of ~2/sec) there are about 57k. Given that one entry usually weighs no more than 500 bytes (pessimistic estimate, for long titles), we can easily fit our records in less than 30 MB.
- We are more comfortable with Pandas :D While often overlooked, the amount of developer's experience in a framework is often extremely important for a streamlined development process.

Cassandra - DB for storage

Our Cassandra storage has 5 different tables for each ad-hoc queries.

- domains `-> (domain text), PIRMARY KEY (domain)`
- page_user `-> (domain text, page_name text, user_id int, page_id int,) PRIMARY KEY (user_id, domain, page_id)`
- page_domains `-> (domain text, page_id int) PRIMARY KEY (domain, page_id)`
- pages_info `-> (page_id int, page_name text, domain text, created_at timestamp) PRIMARY KEY (page_id, domain)`
- page_users_info `-> (user_id int, username text, page_id int, created_at timestamp) PRIMARY KEY (user_id, created_at, page_id)`

This way the use of tables becomes easier and more efficient when doing SELECT queries.

You can check out `cassandra-client/create-tables.cql` for keyspace initialization.

Precomputed reports server - Category A

Every hour, the consumer recomputes records for the last 6 full hours, excluding the last hour. These requests, in a JSON form, are then sent via a POST request to our precomputed server. Here, the server stores `queries.json` to give out on a GET request, which you can do as a client via port forwarding.

Ad hoc server - Category B

Consumer creates CassandraClient that updates all existing tables as soon as the new message comes to Kafka. CassandraClient has separate methods for inserting data to different tables (`insert_to_page_user` , etc.), and methods for getting answers to all queries (`select_from_pages_info` , etc.)

To get SELECT results we have separate script that gets requests from the server and executes corresponding SELECT queries.

`client_demo.py` sends these requests via POST request to the server.

Results

All of our results are stored in the `project-results` directory. These results are generated using the `client_demo.py` script. As per the duration requirement, the system was running for ~8h 27m before making the requests.

Category A

For category A, the JSON file with query results is saved to `project-results/queries.json` .

The file is extremely large because of the 3rd query: the top author created over 20000 pages.

First query:


```
"first_query": [
    {
        "time_start": "01:00",
        "time_end": "02:00",
        "statistics": {
            "ar.wikipedia.org": 197,
            "arz.wikipedia.org": 99,
            "ban.wikipedia.org": 1,
            "bg.wikipedia.org": 2,
            "bjn.wikipedia.org": 1,
            "ca.wikipedia.org": 1,
            "ceb.wikipedia.org": 1,
            "commons.wikimedia.org": 1010,
            "cy.wikisource.org": 1,
            "da.wikipedia.org": 14,
            "de.wikipedia.org": 4,
            "en.wikibooks.org": 1,
            "en.wikinews.org": 4,
            "en.wikipedia.org": 179,
            "en.wikiquote.org": 1,
            "en.wikisource.org": 121,
            "en.wiktionary.org": 34,
            "es.wikipedia.org": 34,
            "es.wikiquote.org": 1,
            "es.wikisource.org": 4,
            "es.wiktionary.org": 1,
            "eu.wikipedia.org": 1,
            "fa.wikipedia.org": 34,
            "fi.wikipedia.org": 2,
            "fr.wikipedia.org": 14,
            "fr.wikisource.org": 18,
            "gu.wikipedia.org": 1,
            "hi.wikipedia.org": 1,
            "hr.wikipedia.org": 1,
            "ht.wikipedia.org": 2,
            "hu.wikipedia.org": 2,
            "id.wikipedia.org": 4,
            "incubator.wikimedia.org": 7,
            "is.wikipedia.org": 2,
```

Second query:


```
],
"second_query": {
    "time_start": "01:00",
    "time_end": "07:00",
    "statistics": [
        {
            "domain": "ar.wikipedia.org",
            "created_by_bots": 392
        },
        {
            "domain": "as.wikipedia.org",
            "created_by_bots": 17
        },
        {
            "domain": "bn.wikipedia.org",
            "created_by_bots": 2
        },
        {
            "domain": "ca.wikipedia.org",
            "created_by_bots": 1
        },
        {
            "domain": "commons.wikimedia.org",
            "created_by_bots": 3407
        },
        {
            "domain": "de.wikipedia.org",
            "created_by_bots": 10
        },
        {
            "domain": "en.wikipedia.org",
            "created_by_bots": 65
        },
        {
            "domain": "es.wikipedia.org",
            "created_by_bots": 2
        },
        {
            "domain": "eu.wikipedia.org",
            "created_by_bots": 7
```

Third query:


```
},
"third_query": {
    "time_start": "01:00",
    "time_end": "07:00",
    "top_users": [
        {
            "uid": 963971,
            "uname": "Vojtěch Dostál",
            "num_created": 21460,
            "titles": [
                "Q112453917",
                "Q112453918",
                "Q112453919",
                "Q112453920",
                "Q112453921",
                "Q112453922",
                "Q112453923",
                "Q112453924",
                "Q112453925",
                "Q112453926",
                "Q112453927",
                "Q112453928",
                "Q112453929",
                "Q112453930",
                "Q112453931",
                "Q112453932",
                "Q112453933",
                "Q112453934",
                "Q112453935",
                "Q112453936",
                "Q112453937",
                "Q112453938",
                "Q112453939",
                "Q112453940",
                "Q112453941",
                "Q112453942",
                "Q112453943",
```

Category B

As for category B, the output is displayed directly when running `client_demo.py`.

The arguments were chosen to fit well in a screen.