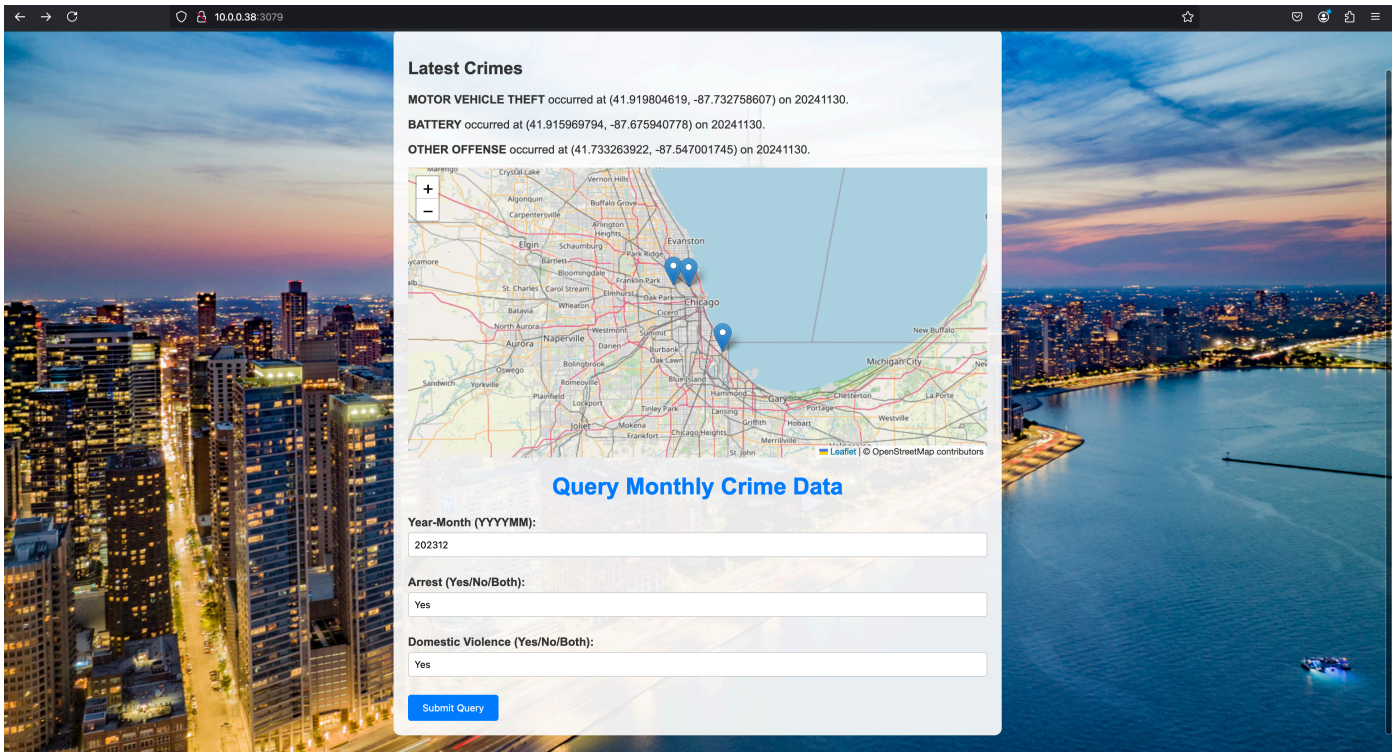
Chicago Crime Big Data Application

1. Project Overview

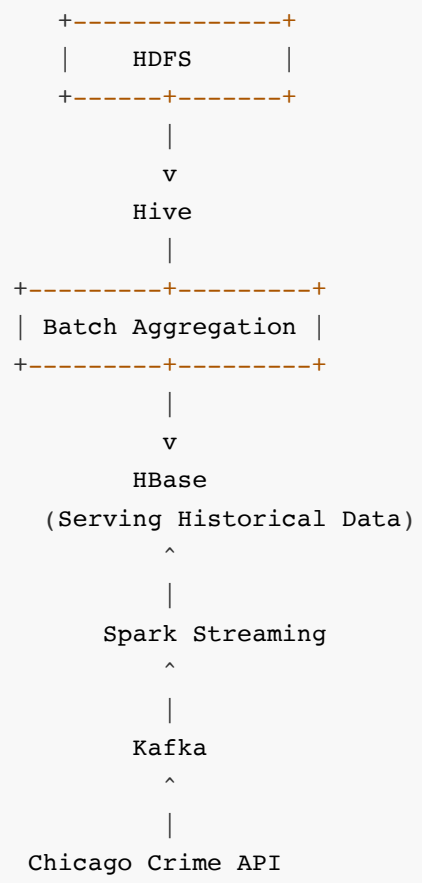
This project aims to build a comprehensive big data application that ingests, processes, and visualizes historical and real-time crime data from the City of Chicago. The system leverages a Lambda Architecture to handle both batch and speed (real-time) layers, ensuring the delivery of timely analytics and a reliable dashboard for end-users. The application integrates an array of big data technologies—HDFS, Hive, HBase, Kafka, and Spark Streaming—and provides a polished web dashboard to query, visualize, and interact with processed crime data.



Key Objectives:

- Ingest and store Chicago crime data (historical and real-time) for the period from 2001 to the present.
- Implement a Lambda Architecture:
  - **Batch Layer:** Process historical data from Hive, generate monthly aggregates, and store them in HBase.
  - **Speed Layer:** Incorporate real-time updates via Kafka and Spark Streaming, writing recent crime events into HBase for immediate display.
  - **Serving Layer:** Expose the processed data through a web-based dashboard, allowing users to query aggregated metrics and visualize trends over time.
- Provide an interactive dashboard with charts and maps, allowing users to select time ranges (e.g., YYYYMM) and districts for analysis, as well as to see the latest crimes as real-time alerts.

High-Level Architecture Diagram:



---

2. Data and Sources

1. **Historical Crime Data (2001–present):**
   Comprehensive datasets including crime type, date/time, location, arrests, domestic violence involvement, district, and geographical coordinates.
2. **Police Station and District Data:**
   Additional datasets providing police station addresses, district boundaries, and other contextual information.

---

3. Lambda Architecture Design

Batch Layer (Historical Processing):

- Ingest historical CSV data into HDFS.
- Use Hive external tables to clean and transform the data.
- Perform monthly aggregations and write results into HBase.

Speed Layer (Real-Time Processing):

- A Kafka producer fetches recent crime data via an API, producing JSON messages to Kafka.
- A Spark Streaming job consumes from Kafka, processes the data, and writes recent crimes into HBase.

Serving Layer (HBase + Web Application):

- Pre-aggregated data and live-updates from HBase are served to a Node.js + Express web dashboard.
- Visualizations are done with D3.js (charts) and Leaflet.js (maps).

---

4. Detailed Implementation Steps and Operations

Below are the step-by-step instructions, integrated into the report narrative. All project files (JARs, scripts, code) are located in `/home/sshuser/xli0808` on the cluster.

Step 1: Accessing the Cluster

From your local machine, connect to the cluster:

```
ssh -i /Users/lixiang/.ssh/id_rsa sshuser@hbase-mpcs53014-2024-ssh.azurehdinsight.net
```

You are now on the cluster's master node.

Step 2: HDFS Setup and Data Ingestion

Check HDFS environment:

```
hdfs dfs -ls /
```

If not done yet, create your personal directory in HDFS:

```
hdfs dfs -mkdir /xli0808
```

From your local machine, upload the CSV files to the cluster:

```
scp -i /Users/lixiang/.ssh/id_rsa /Users/lixiang/Downloads/Chicago_Crime_2001_2024.csv
sshuser@hbase-mpcs53014-2024-ssh.azurehdinsight.net:/home/sshuser/xli0808/
scp -i /Users/lixiang/.ssh/id_rsa /Users/lixiang/Downloads/PoliceStations.csv
sshuser@hbase-mpcs53014-2024-ssh.azurehdinsight.net:/home/sshuser/xli0808/
scp -i /Users/lixiang/.ssh/id_rsa /Users/lixiang/Downloads/PoliceDistrict.csv
sshuser@hbase-mpcs53014-2024-ssh.azurehdinsight.net:/home/sshuser/xli0808/
```

Then, on the cluster, put the data into HDFS:

```
hdfs dfs -put /home/sshuser/xli0808/Chicago_Crime_2001_2024_fixed.csv /xli0808/crime_data
hdfs dfs -put /home/sshuser/xli0808/PoliceDistrict.csv /xli0808/
hdfs dfs -put /home/sshuser/xli0808/PoliceStations.csv /xli0808/
```

Step 3: Hive Operations for Batch Layer

Enter Hive:

```
beeline -u 'jdbc:hive2://10.0.0.50:10001/;transportMode=http'
```

Create external tables, cleaned tables, and run the SQL provided in the project's documentation to perform data cleansing and aggregation. For example:

```
DROP TABLE IF EXISTS xli0808_raw_chicago_crime;

CREATE EXTERNAL TABLE IF NOT EXISTS xli0808_raw_chicago_crime (
  id INT,
  ...
  longitude DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'wasbs://.../xli0808/crime_data';
```

After cleaning and transforming, create monthly aggregation tables and insert aggregated data into HBase external tables. For instance:

```
INSERT OVERWRITE TABLE xli0808_crime_monthly_hbase
SELECT CONCAT(yearmonth, '#', district), monthly_crime_count
FROM xli0808_crime_monthly_agg;
```

Step 4: Verifying HBase Data (Batch Layer)

Open the HBase shell:

```
hbase shell
```

Scan the HBase table to ensure data is correctly loaded:

```
scan 'xli0808_crime_monthly_hbase', {LIMIT => 10}
```

You should see rows with `yearmonth#district` keys and `monthly_crime_count` values.

---

5. Speed Layer: Kafka and Spark Streaming

Step 5: Kafka Topic Setup

Export Kafka broker addresses:

```
export KAFKABROKERS="wn0-
kafka.m0ucnnwuiqae3jdorci214t2mf.bx.internal.cloudapp.net:9092,wn1-
kafka.m0ucnnwuiqae3jdorci214t2mf.bx.internal.cloudapp.net:9092"
```

Create the Kafka topic:

```
kafka-topics.sh --create \
--topic xli0808_latest-crime-json \
--bootstrap-server $KAFKABROKERS \
--replication-factor 3 \
--partitions 1
```

Check the topic:

```
kafka-topics.sh --describe --topic xli0808_latest-crime-json --bootstrap-server
$KAFKABROKERS
```

All runnable jars are in `/home/sshuser/xli0808/`. For the producer:

```
cd /home/sshuser/xli0808/ChicagoCrimeKafkaProducer/target
java -jar ChicagoCrimeKafkaProducer-1.0-SNAPSHOT.jar
```

This fetches the latest crimes from the Chicago Data API and sends them to the Kafka topic.

Check messages using the Kafka console consumer if desired.

Step 7: Running the Spark Streaming Job (Scala)

Submit the Spark job to read from Kafka and write to HBase:

```
spark-submit --driver-java-options "-
Dlog4j.configuration=file:///home/sshuser/ss.log4j.properties" \
--class com.xiangli.ChicagoCrimes \
/home/sshuser/xli0808/KafkatoHbaseChicagoCrime/target/uber-KafkatoHbaseChicagoCrime-1.0-
SNAPSHOT.jar
```

This continuously listens to the Kafka topic, processes new crime events, and inserts them into `xli0808_speed_latest_crimes_table` in HBase.

Verify Speed Layer Data:

```
hbase shell
scan 'xli0808_speed_latest_crimes_table'
```

---

6. Serving Layer and Web Application

Step 8: Setting Up Proxy for Local Access

To view the Node.js dashboard locally, create an SSH tunnel:

```
ssh -i /Users/lixiang/.ssh/id_rsa -C2qTnNf -D 9876 sshuser@hbase-mpcs53014-2024-ssh.azurehdinsight.net
```

Configure local browser to use `localhost:9876` as a SOCKS proxy. This allows you to access the web interface on the remote cluster as if it were local.

Step 9: Running the Web Application (Node.js)

On the cluster:

```
cd /home/sshuser/xli0808/WebApp/
node app.js 3079 https://hbase-mpcs53014-2024.azurehdinsight.net/hbaserest $KAFKABROKERS
```

Open a browser on your local machine and navigate to `http://localhost:3079`. The dashboard will display:

- A map with the latest crimes.
- A query form for selecting year-month, arrest flag (Yes/No/Both), domestic (Yes/No/Both).
- Bar charts showing crime counts by district.
- The top banner alerts showing the newest crimes from the speed layer.

---

7. Example Data Flows

Historical Query Example:

- The user selects `YYYYMM = 202312`, `Arrest = both`, `Domestic = both`.
- The Node.js backend retrieves data from `xli0808_project2_crime_monthly_hbase` by scanning relevant row keys.
- Results are returned to the frontend, displayed in a ranked bar chart.

Real-Time Update Example:

- A new crime occurs, is published on the Chicago Data API.
- The Kafka producer retrieves this entry, sends it to `xli0808_latest-crime-json`.
- Spark Streaming consumes the topic and writes new entries into `xli0808_speed_latest_crimes_table`.
- On page reload, the latest crimes appear on the dashboard's map and alert banner.

---

8. Final Dashboard Features

- **Interactive Map:** Displaying the newest crime locations as markers.
- **Bar Chart Rankings:** Showing crime volume by district for a given month.
- **Time-Range Flexibility:** Users can query data from a large time range.
- **Real-Time Alerts:** Latest crimes are highlighted on the dashboard, reflecting new data ingested via the speed layer.

9. Conclusion and Future Enhancements

This Chicago Crime Big Data Application showcases a fully integrated Lambda Architecture solution. Historical batch processing ensures reliable analytics and stable reference data, while real-time streaming provides immediate insights into the latest crimes. The web dashboard makes the entire data pipeline accessible and understandable to end-users.

Potential Future Improvements:

- Advanced filtering options (e.g., by crime type, geographic area).
- Improved visualizations and UI enhancements.
- Integration of machine learning for crime trend predictions.
- Enhanced security and authentication for accessing the dashboard