Keshav Singhal
IMT2018511

ScenariosDB

Project Elective

2022-05-06

1 Objective of the project

I worked with Autonomous Systems and Control team at siemens on ScenariosDb project. My objective was to understand the autonomous vehicle dataset(Nuscenes and kitti) and load these datasets in MongoDB and InfluxDB databases and optimize the query time from these datasets.

2 Datasets

2.1 Kitti

Kitti contains a suite of vision tasks built using an autonomous driving platform. The full benchmark contains many tasks such as stereo, optical flow, visual odometry, etc. This dataset contains the object detection dataset, including the monocular images and bounding boxes.

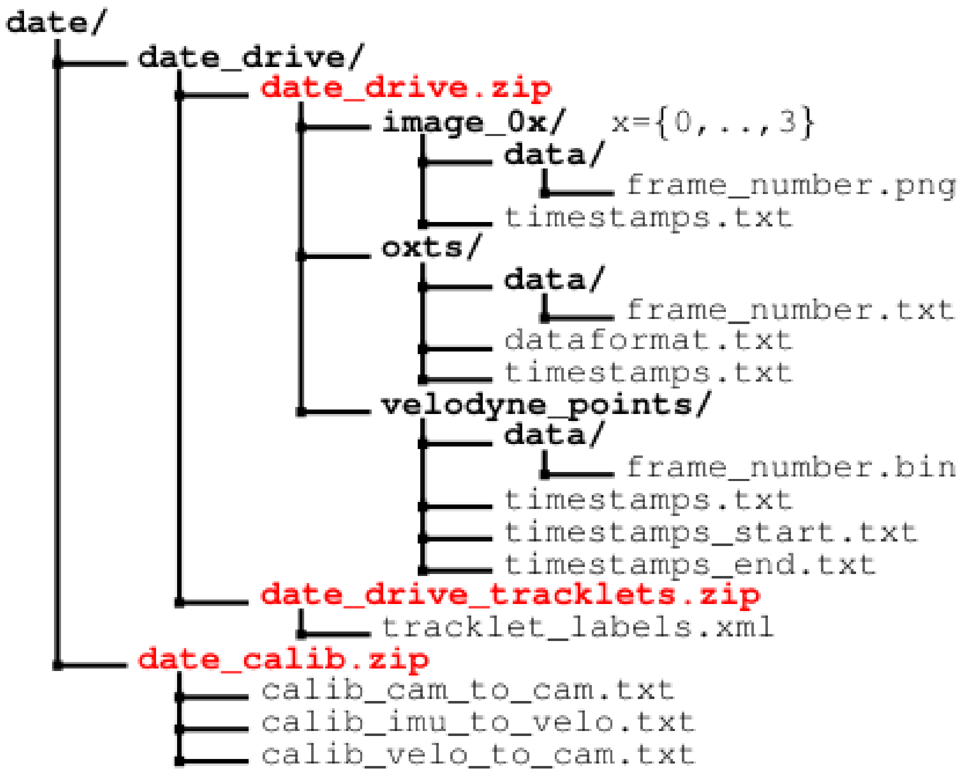


Figure 1: Kitti dataset

2.2 Nuscenes

The nuScenes dataset is a large-scale autonomous driving dataset with 3d object annotations. It features: Full sensor suite (1x LIDAR, 5x RADAR, 6x camera, IMU, GPS) 1000 scenes of 20s each 1,400,000 camera images 390,000 lidar sweeps Two diverse cities: Boston and Singapore Left versus right hand traffic Detailed map information

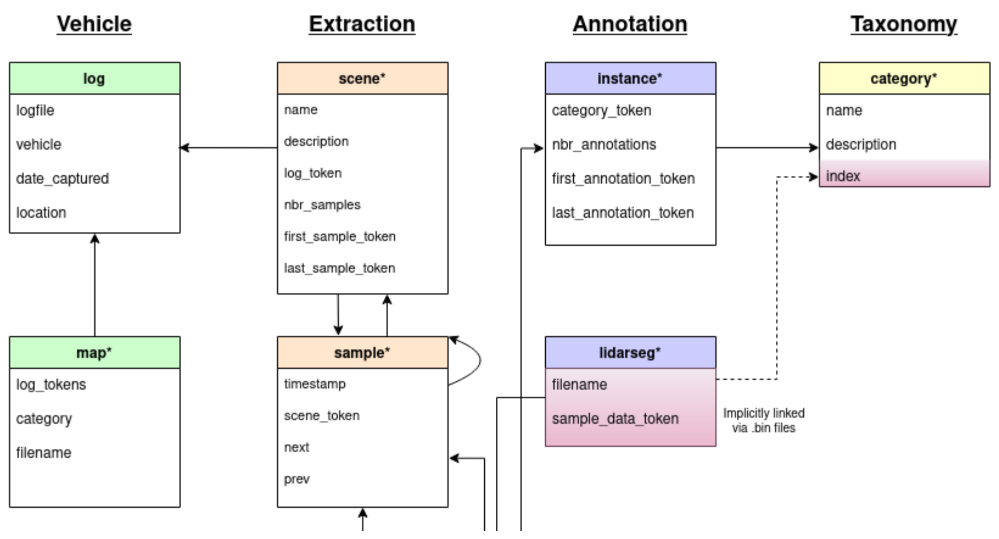


Figure 2: Nuscenes

3 Sprint Report

Work done in each week.

- Looked at Kitti and Nuscenes dataset.
- Learnt about Influxdb database(time series database).
- Learnt about MongoDB database.
- Loaded the GPS and IMU sensor dataset in influxdb database.
- Loaded the annotations data in MongoDB database.
- Read and understood the NCLS(Nested Containment List) Paper.
- Started the implementation of NCLS on our dataset using Influxdb and MongoDB.
- Completed the implementation of Joins of both the dataset using NCLS.
- Learnt about Google's BigQuery.
- Worked on implementation of loading data in BigQuery using influxDb and MongoDb.
- Found about BigTable and started working on it as well.
- Wrote a script to connect Bigtable to local terminal and load data into bigtable by querying on Influxdb and Mongodb.
- Worked on the final presentation and final report.

4 Final summary of sprints

I worked on understading the datasets and was able to understand the databases the team was working on. I was able to load and process datasets in the databases. I was able to optimize the query time on these databases by implementing the join of databases using NCLS. I was also able to load data in bigtable but we needed to pay to load the data and for every query.

5 Code details

I wrote the code for loading the datasets, and working on NCLS and BigTable in python and could be found here.

6 NCLS

Interval query are slow because the overlapping intervals for any given query may not be contiguous in standard indexing. Therefore, the database query cannot stop at the first non-overlapping interval, but must scan the rest of the database. It is caused solely by the intervals that are contained within other intervals, i.e. x:startiy:startiy:stopix:stop.

If a list of intervals L is ordered on start and its intervals have no containment relation, then it is also sorted on stop. The practical value of this observation is that if we subdivide the database by moving all intervals that are contained in a given interval into a separate sublist, the time complexity of overlap query is reduced to O(log N + n), since now we can guarantee that no more results are present once we encounter the first non-qualifying interval. The subset of intervals that overlap the query are guaranteed to be contiguous in any given sublist. Thus, we only need to search the sublists of these overlapping intervals recursively.

The running time of NCList scales linearly as a function of result set size. The running time scales as the logarithm of the database size. The slope of this log-linear trend is low; a 100-fold increase in the database size resulted in only about a 30NCList is over 5–500-fold faster than existing interval query indexing methods. The test data show increasing speed advantages for NCList (relative to other methods) for increasing database size and query width.
NCList database construction was 100-fold faster than the two methods with reasonable query scalability (binning and R-tree indexing).

6.1 Algorithm

```
$R \leftarrow \emptyset$
$subend \leftarrow H[sublist].start + H[sublist].length$
$i \leftarrow$ BinarySearchEnd($L$, $H[sublist]$, $start$)
while $i < subend$ and $L[i].start < stop$ do
    $R \leftarrow R \cup \{L[i]\}$
    $R \leftarrow R \cup$ Overlap($L$, $H$, L[$i$].$sublist$, $start$, $stop$)
    $i \leftarrow i+1$
end while
Return $R$
```

Figure 3: NCLS Algorithm

7 BigQuery

BigQuery is a fully-managed, serverless, enterprise data warehouse that enables scalable analysis over petabytes of data with all of its implications of complex grading facilities. It is a platform as a Service (PaaS) that supports querying using ANSI SQL. It is a fully managed platform- no server, no resources deployed. It can be accessed through Web UI, REST API, Client SDK. Google BigQuery has built-in machine learning capabilities. It is a highly-scalable, highly-distributed, low-cost analytics OLAP data warehouse capable of achieving a scan rate of over 1 TB/sec.

Google BigQuery was designed as a "cloud-native" data warehouse. It was built to address the needs of data driven organizations in a cloud first world. It is deeply integrated with GCP analytical and data processing offerings, allowing customers to set up an enterprise ready cloud-native data warehouse.

7.1 Architecture

BigQuery's serverless architecture decouples storage and compute and allows them to scale independently on demand. This structure offers both immense flexibility and cost controls and we don't need to keep our expensive compute resources up and running all the time. This is very different from traditional node-based cloud data warehouse solutions or on-premise massively parallel processing (MPP) systems. This approach also allows us to bring our data of any size into the data warehouse and start analyzing it using Standard SQL without worrying about database operations and system engineering.

- Compute is Dremel, a large multi-tenant cluster that executes SQL queries.
- Storage is Colossus, Google's global storage system.
- Compute and storage talk to each other through the petabit Jupiter network.
- BigQuery is orchestrated via Borg, Google's precursor to Kubernetes.

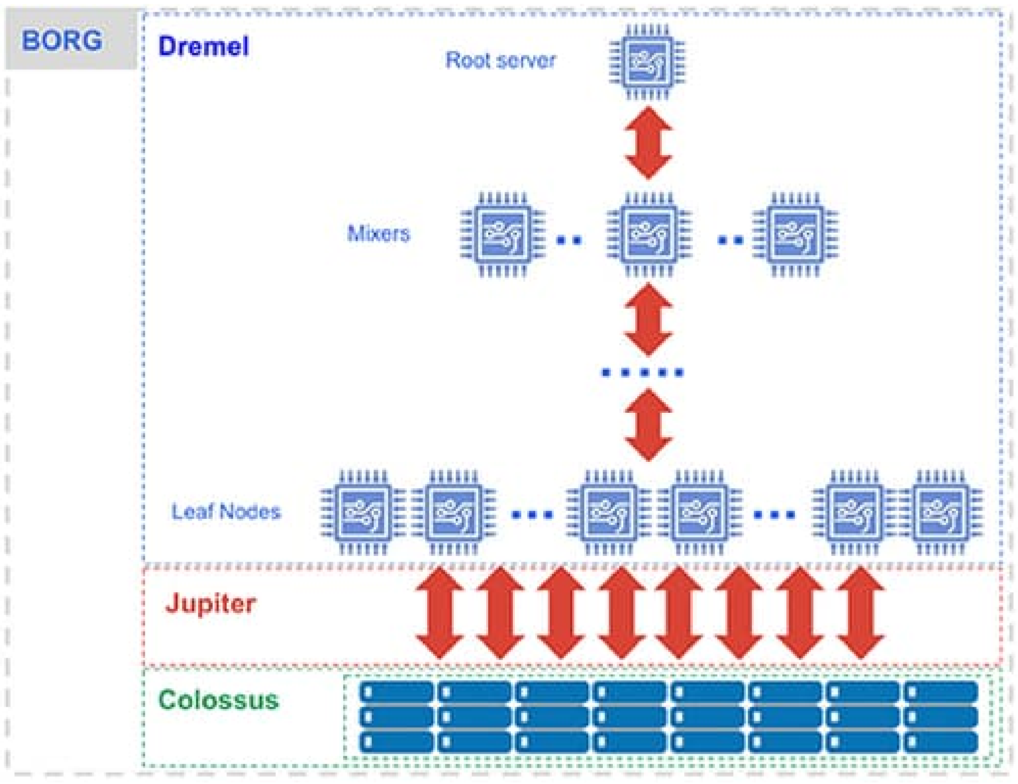
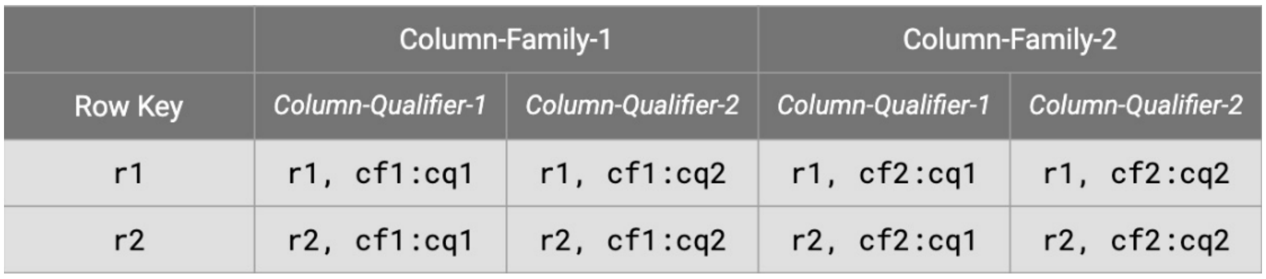


Figure 4: BigQeury Architecture

8 BigTable

Google Bigtable is a distributed, column-oriented data store created by Google Inc. to handle very large amounts of structured data associated with the company's Internet search and Web services operations. Bigtable was designed to support applications requiring massive scalability; from its first iteration, the technology was intended to be used with petabytes of data. Google Bigtable serves as the database for applications such as the Google App Engine Datastore, Google Personalized Search, Google Earth and Google Analytics.

Bigtable stores structured data in a simplified data model in comparison to a classical relational DBMS/data warehouse (no fixed schema, normal forms, etc.). Data is indexed using (uninterpreted) strings as row and column names. Data values are also stored as (uninterpreted) strings. That is, a table in Bigtable is nothing else but a sparse, distributed, persistent, multi-dimensional, sorted map.



	Column-Family-1		Column-Family-2	
Row Key	*Column-Qualifier-1*	*Column-Qualifier-2*	*Column-Qualifier-1*	*Column-Qualifier-2*
r1	`r1, cf1:cq1`	`r1, cf1:cq2`	`r1, cf2:cq1`	`r1, cf2:cq2`
r2	`r2, cf1:cq1`	`r2, cf1:cq2`	`r2, cf2:cq1`	`r2, cf2:cq2`

Figure 5: Bigtable