아주대학교
세상의 A+가 되겠습니다

2023-2
AJOU SOFTCON

자율 주행 자동차 데이터셋의 ETL Process와 Database 벤치마킹

이 름 장승현 지도교수 오상윤

연구배경

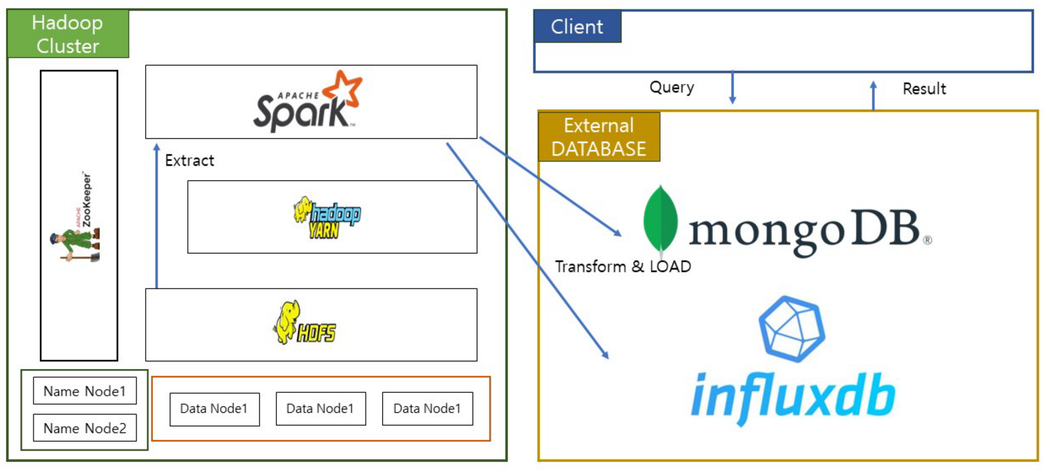
자율 주행 산업의 규모가 점차 커짐에 따라, 자율 주행 데이터를 수집하고 데이터를 연구에 적절히 활용해야 하는 요구도 매우 커진 상황이다. 하지만 자율 주행 데이터는 대규모 시계열 특성을 지니는 경우가 대부분이며, 이를 활용하기 위해서는 ETL(Extract, Transform, Load)과정이 필요하다.

본 연구에서는 NGSIM US-101 데이터 셋을 기존의 ETL process에서 범용적으로 활용되는 NoSQL 데이터 베이스인 MongoDB와 시계열 데이터에 특화되어 있는 InfluxDB를 벤치마킹하여 자율 주행 데이터셋에 특화된 ETL 처리 구조를 제안하고자 한다.

활용 데이터 셋

본 연구에서 활용된 NGSIM US-101 데이터 셋은 미국 US-101 freeway 옆 36층에 설치된 카메라로 획득한 데이터셋으로 획득 시간대 별로 데이터를 분리하여 정체 시간 데이터를 csv(raw)파일로 trajectory를 시간대 별로 정리하여 Vehicle ID부터 Time_HeadWay까지 18개의 column으로 정리된 데이터 셋이며 각 Global time(epoch time)이 기재되어 시계열 특성을 지닌다.

Architecture & ETL Process



클러스터 환경

	환경	스펙
Node	NameNode2, DataNode3	Amazon Ec2 Instance T2.Medium (30ssd)
DataBase	MongoDB, MongoDB TimeSeries Collection, InfluxDB	Amazon Ec2 Instance T2.Medium (30ssd)

본 연구에서는 하둡 클러스터 환경을 구축하였다. HDFS는 대용량 csv file을 데이터 블록(128MB)크기로 쪼개 DataNode에 분산시켜 저장시키고 YARN은 하둡 클러스터의 리소스를 관리해 병렬적으로 부하를 분산시키고, data block의 실제 위치를 조정하는 역할을 수행한다. Zookeeper는 각 node들의 상태를 관리하고, active상태인 namenode가 failover한 상황에 sub namenode에게 권한을 옮겨주어 실행을 지속시켜주어 high availability를 충족시킨다.

ETL Process
1. Extract
- Extract 과정은 Data Node 1,2,3에 분산되어 저장되어 있는 csv data chunk를 Apache Yarn를 통해 부하를 병렬적으로 분산시키고 Spark는 RDD객체의 중간결과를 Memory에 저장시키며 Batch Processing하여 Python의 Spark DataFrame형태로 Load시킨다.

2. Transform
- csv파일의 각 Row를 그대로 Load시키는 것이 아닌, projection과정을 거치고, 연구에서 활용한 데이터 셋의 timestamp를 US/Pacific시간으로 변환시킨다.

3. Load
- Spark를 사용해 Batch Processing하여 MongoDB에는 Document형식으로 , InfluxDB에는 measurement형식으로 각 Batch import시켰다.

DataBase BenchMarking

자율 주행 데이터 셋처럼 대규모와 시계열의 특징을 지니는 데이터는 ETL과정도 중요하지만, 대규모의 데이터를 어떤 크기로 저장하는 지와 시간에 따른 통계치를 위한 쿼리의 속도가 중요하다.

본 연구에서는 Raw csv file (1,000,000) Column의 data set을 load하여 NoSQL Database인 MongoDB의 일반 Collection과 timeseries Collection , open TSDB인 InfluxDB의 storage size, bulk insert performance, time query performance, time period query performance를 비교한다.

DataBase Spec

	환경	Index	Indexing
MongoDB	Version 6.0 (WiredTiger Engine)	(Time, lane_id) : Compound Index	B-Tree (no retention policy)
MongoDB(ts)	Version 6.0 (WiredTiger Engine) TimeSeries Collection	(Time, lane_id) : Secondary Compound Index	B-Tree (retention policy) with timeBlock
InfluxDB	Version 2.7.1	(Time & tag keys) : Compound Index	LSM-Tree(retention policy) with timeBlock

벤치마킹 결과 및 분석

	MongoDB	MongoDB(ts)	InfluxDB
Storage SIZE	48.04MB	43MB	23MB
Index SIZE	19.75MB	7.45MB	-
bulkInsert (50000)	41.23sec 1.9633sec(avg)	61.34sec 2.92sec(avg)	56.72sec 2.70sec(avg)
Query ($eq:time)	5ms(IXSCAN)	140ms(IXSCAN)	77ms
Query {$lt:time)	1094ms(IXSCAN)	3029ms (Clustered IXSCAN)	491ms
Query timeBetween	713ms (IXSCAN)	2191ms(IXSCAN)	322ms
Group by time period	112ms (IXSCAN)	3955ms(COLLSCAN)	95ms
Group by time period &lane_id	2544ms (COLLSCAN)	3257ms(COLLSCAN)	134ms

결과는 다음과 같았다. MongoDB는 내부적으로 B-Tree Indexing방식을 사용하고, 이를 BSON 형식으로 저장한다. MongoDB timeseries또한 Indexing방식은 같지만, 내부적으로 time으로 묶어서 Bucket으로 저장하며 timeseries collection은 별도로 time의 gap을 채워주기 위해 densification(time의 gap을 채워주기 위해 timefield만 가지고 있는 documen를 생성해서 채워주는 것)과 time-to-live(일정 시간 후에 data를 자동으로 삭제)해주는 차이가 존재한다. 하지만 벤치마킹 결과 storage Size나 Index size에서 확연한 성능 개선을 보인 반면 여러 time 쿼리에서 Bucket을 unpacking하는 과정에 시간이 많이 소모된 것을 확인할 수 있었다.

InfluxDB는 MongoDB와 같이 key-value store을 하지만 LSM-Tree Indexing 방식을 사용하고, time block으로 이를 나누어 저장한다. 또한 특이하게 tag-key와 time을 묶은 compound Index가 자동으로 생성되며 field key-value값은 중복이 있을 경우 따로 저장하지 않는다. 이러한 Indexing과 데이터를 중복해서 저장하지 않는 특성에 의해 data compression이 굉장히 개선되었고 Query 속도 또한 성능적인 측면에서 뛰어났다.

결론

성능과 data storage측면에서는 InfluxDB가 우월한 성능을 보였지만, 모든 벤치마킹이 그러하듯 세부사항에 그 결과가 영향을 미치기에 절대 절대적인 결과라고 할 수 없다. 또한 InfluxDB는 앞서 Indexing 방식과 성능 개선을 위한 방식에 의해 trade-off로 update와 delete성능이 떨어진다고 한다. 하지만, IOT나 자율 주행과 같이 read나 write연산이 더 많이 수행되는 환경이라면 충분히 사용하기 좋은 성능을 보인다고 생각한다.
또한, InfluxQuery의 편리함과 여러 UI interface를 제공한다는 점과 InfluxDB3.0부터 10배 빨라진 성능을 보인다고 하니 주목해볼만 하다.