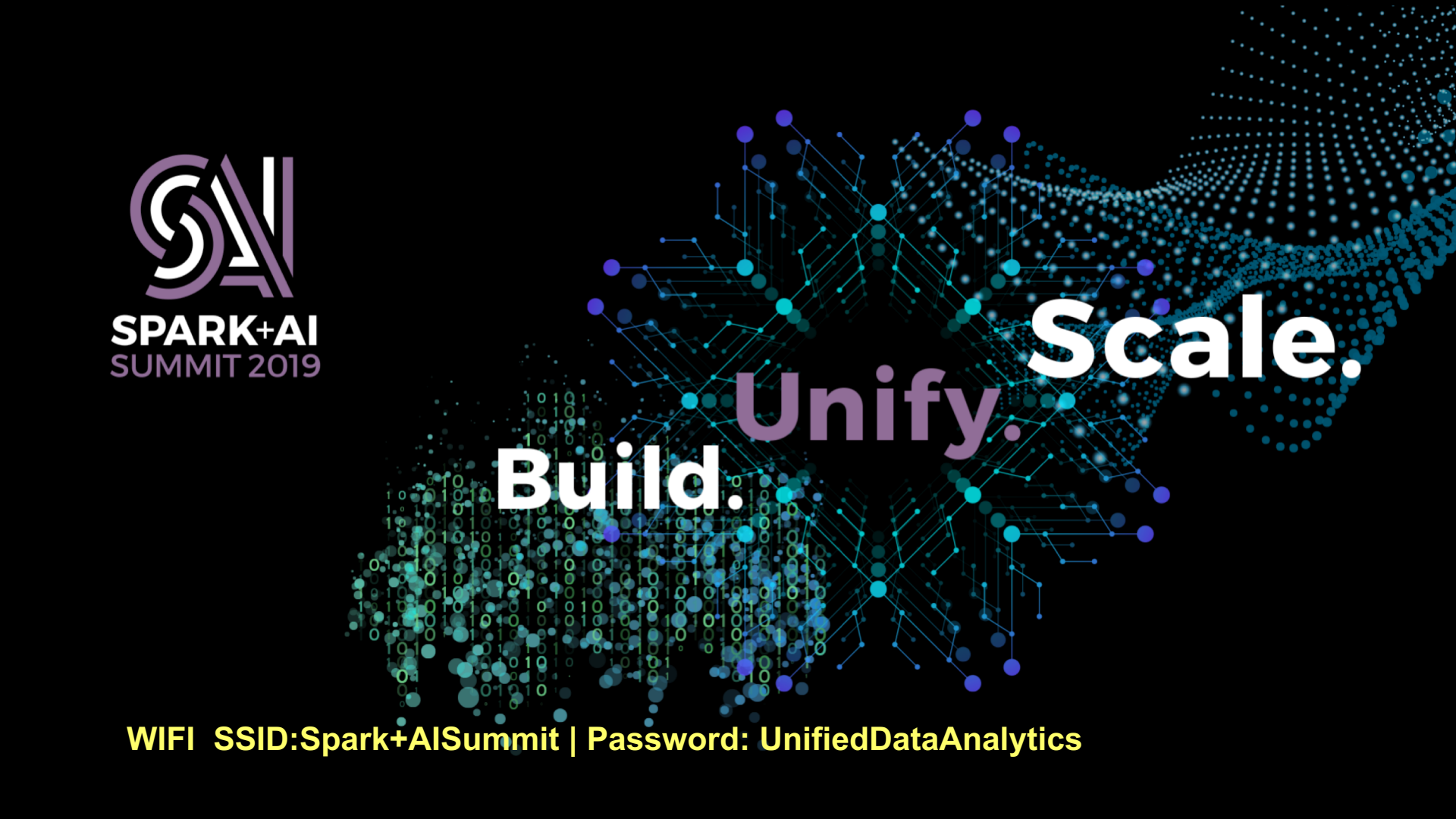


WIFI SSID:Spark+AISummit | Password: UnifiedDataAnalytics

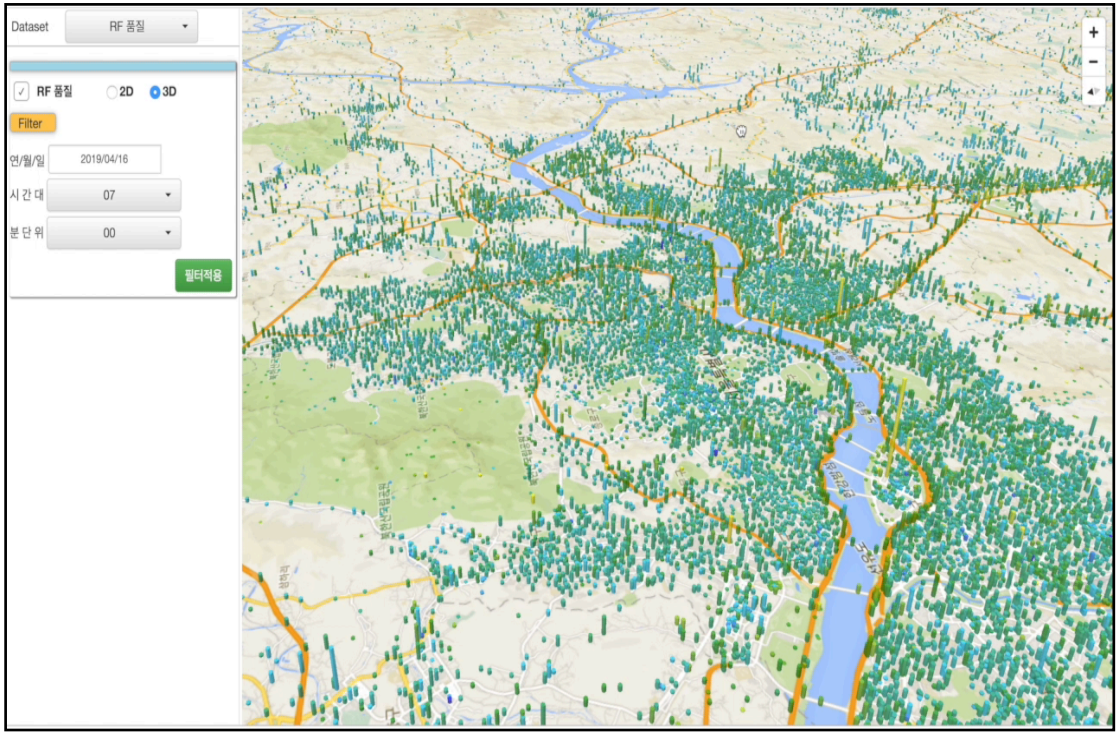
SPARK+AI SUMMIT 2019

Spark AI Usecase in Telco: Network Quality Analysis and Prediction with Geospatial Visualization

Hongchan Roh, Dooyoung Hwang, SK Telecom

#UnifiedDataAnalytics #SparkAISummit

Network Quality Visualization Demo



- Demo shows
  - Visualize RF quality of cell towers
  - Height : connection count
  - Colors : RF quality

- Data source
  - 300,000 cell tower network device logs

- Resources
  - 5 CPU nodes with Intel Xeon Gold 6240

youtube demo video link:

https://youtu.be/HpDkF3CxEow

This doesn't exactly reflect real network quality, we generated synthetic data from real one

Contents

- **Network Quality Analysis**
- Geospatial Visualization
- Network Quality Prediction
- Wrap up

Network Quality Analysis

SK telecom : The largest telecommunications provider in South Korea

- 27 million subscribers
- 300,000 cells

Target Data: Radio cell tower logs

- time-series data with timestamp tags generated every 10 seconds
- geospatial data with geographical coordinates (latitude and longitude) translated by cell tower's location

Network Quality Analysis

Data Ingestion Requirements

- Ingestion 1.4 million records / sec, (500 byte of records, 200~500 columns)
- 120 billion records / day, 60 TB / day
- Data retention period: 7 ~ 30 days

Query Requirements

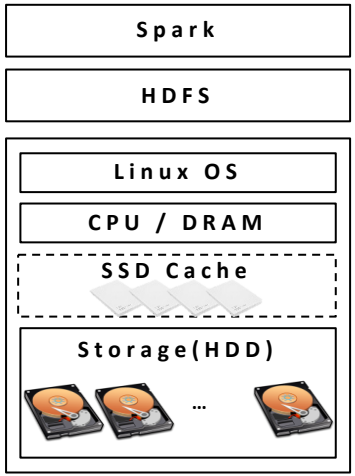
- web dashboard and ad-hoc queries: response within 3 sec for a specific time and region (cell) predicates for multi sessions
- daily batch queries: response within hours for long query pipelines having heavy operations such as joins and aggregations

Problems of legacy architecture

Legacy Architecture: Spark with HDFS (2014~2015)

- Tried to make partitions reflecting common query predicates (time, region)
- Used SSD as a block cache to accelerate I/O performance
- Hadoop namenode often crashed for millions of partitions
- Both ingestion and query performance could not satisfy the requirements

Legacy Architecture



New In-memory Datastore for Spark (FlashBase)

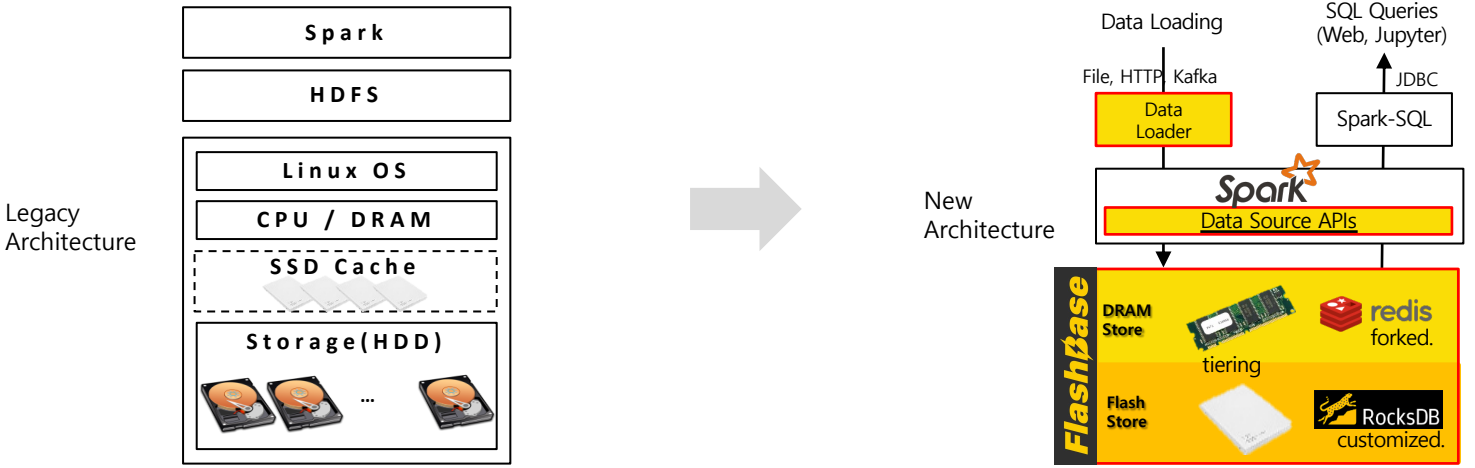
New in-memory (dram/SSD) datastore for Spark (2016)

- Tried to design a new data store for Spark (FlashBase) to support much more partitions

Best open source candidates to assemble

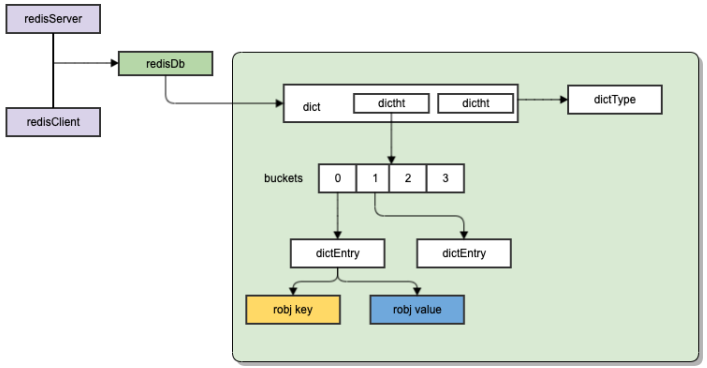
- Spark for query engine, redis for DRAM key-value store and RocksDB for SSD key-value store
- SSDs as main storage devices for small-sized parallel I/O with short latencies?

If we assemble these with some glue logics, can it be ?



Initial avengers didn't get along

Problem 1 - Redis Space Amplification Problem



Redis occupied DRAM Space 4 times of input data

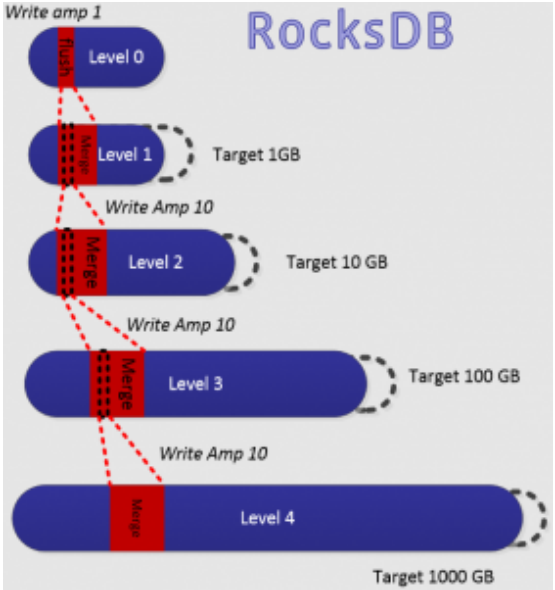
- At least 72 bytes for data structures
- 24B for dictEntry
- 24B for key redisObject and 24B for value redicObject
- 85 bytes for 12 byte key and 1 byte column value

DRAM is still expensive!

FlashBase reduced DRAM usage to 1/4 of original Redis

- custom data structure called column value array
- store data in column-wise
- gather column values in different rows (c++ vector style)
- data layout is similar to Apache arrow

Problem 2 - Rocksdb Write Amplification Problem



Rocksdb wrote 40 times of input data to SSDs

- Rocksdb consists of multi-level indexes with sorted key values in each level
- Key values are migrated from top level to next level by compaction algorithm
- Key values are written multiple times (# of levels)
  with larger update regions (a factor of level multiplier)

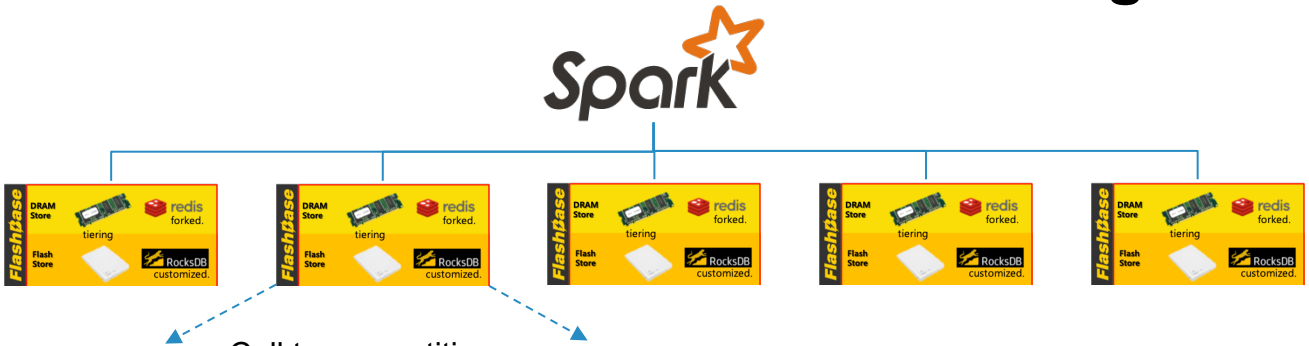
The more writes, the faster SSD fault!

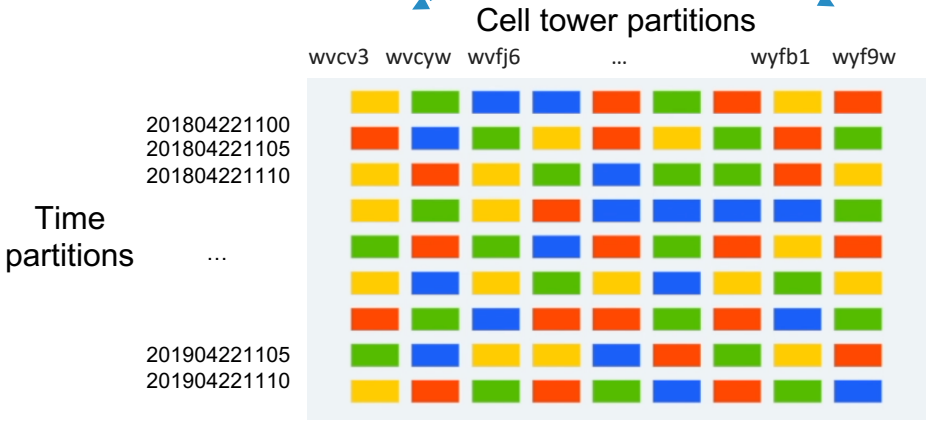
- SSDs have limitation on the number of drive writes (DWPD)
- SSD fault causes service down time and more TCO for replacing SSDs

FlashBase reduced writes to 1/10 of original Rocksdb

- Customized rocksdb for compaction job to avoid next level update at most
- 5 times better ingestion performance, and 1/10 TCO for SSD replacement

Query Acceleration 1 - Extreme Partitioning





A partition combination for network quality analysis

- 300K (cell towers) X 100K (time slots) = 30B (total partitions)

FlashBase

- Up-to 2 billion partitions in a single node
- Needs 15 nodes to store 30 billion partitions

Hadoop File System

- Up-to 1 billion partitions in a single cluster
- 300GB DRAM for 1 billion partitions
  (150 byte DRAM for a file or a block)

Node spec: E5 2680 v4 CPU, 256GB DRAM, 20TB SSDs

Query Acceleration 2 - Filter Push Down

- Custom relation definition to register redis as Spark's datasource by using data source API v1 (Redis/Rocksdb Relation -> R2 Relation)
- Filter / Projection pushdown to Redis/Rocksdb Store by using prunedScan and prunedFilteredScan

```
package com.skt.spark.r2

case class
R2Relation (
  identifier: String,
  schema: StructType
    )(@transient val sqlContext: SQLContext)
  extends BaseRelation
  with RedisUtils
  with Configurable
  with TableScan
  with PrunedScan with PrunedFilteredScan with InsertableRelation
  with Logging {

  def buildTable(): RedisTable
  override def buildScan(requiredColumns: Array[String]): RDD[Row]
  def insert( rdd: RDD[Row] ): Unit
}
```

Query Acceleration 2 - Filter Push Down

Spark Data Source Filter pushdown

- And, Or, Not, Like, Limit
- EQ, GT, GTE, LT, LTE, IN, IsNULL, IsNotNULL, EqualNullSafe,

Partition filtering

- Each redis process filters only satisfying partitions by using push downed filter predicates from Spark
- prunedScan is only requested to the satisfying partitions

Data filtering in pruned scan

- Each pruned scan command examines the actual column values are stored in the requested partition
- Only the column values satisfying the push downed filter predicates are returned

Network Quality Analysis Example

Network quality analysis query for one day and a single cell tower

- 0.142 trillion (142 billion) records in ue_rf_sum[1] table (7 day data, 42TB)
- 14,829 satisfying records

[1]user_equipment_radio_frequency_summary table

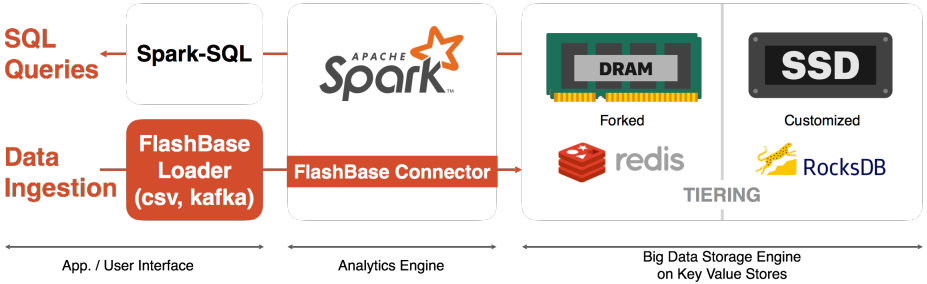
```
select * from ue_rf_sum
    where event_time between '201910070000' and '201910080000' and cell_tower_id = 'snjJlAF5W' and rsrp < -85;
```

Spark with HDFS		Spark with FlashBase	
Partition filtering with time	1/10080	Partition filtering with time and cell tower	1/(10080 * 30000)
Half an hour		Less than 1 sec	

HDFS Cluster: 20 nodes (E5 2650 v4 CPU, 256GB DRAM, 24TB HDDs)

FlashBase Cluster: 16 nodes (E5 2680 v4 CPU, 256GB DRAM, 20TB SSDs)

Ingestion Performance and other features



Features	Details
Ingestion performance	500,000 records/sec/node
In-memory datastore	DRAM only, DRAM to SSD Tiering
Massively Parallel Processing	100 redis processes per a single node
Extreme partitioning	Up-to 2 billion partitions for a single node
Filter acceleration	Using fine-grained partitions and push downed filters
Column-store	Column-store by default (row-store option)
Column value transformation	Defined by java grammar in schema tbl properties
Compression	Gzip level compression ratio w/ LZ4 level speed
Vector processing	Filter and aggr. acceleration (SIMD, AVX)
ETC	Recovery, replication, scale-out

Node spec: E5 2680 v4 CPU, 256GB DRAM, 20TB SSDs

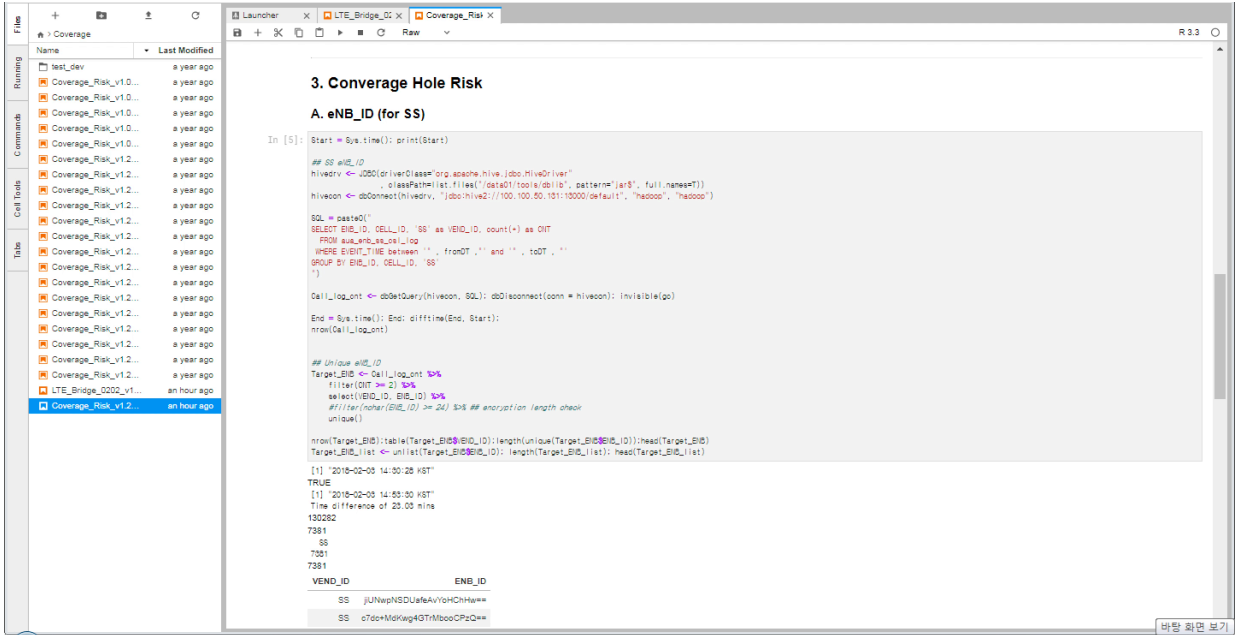
Seven Production Hells

Network OSS Deploy – Web Dashboard (2017)

- Deployed to Network Operating System in 2017
- Web Dashboards queries with time and region predicates
- Wired/Wireless Quality Analysis
- Mobile device quality analysis : Display mobile device quality data per hierarchy

Network OSS Deploy – Batch Queries (2017)

- Batch queries with jupyter via R hive libraries
- Analysis for coverage hole risk for each cell



Contents

- Network Quality Analysis
- **Geospatial Visualization**
- Network Quality Prediction
- Wrap up

Why Geospatial Visualization?

- **Geospatial Analysis**

  - Gathers, manipulates and displays geographic information system (GIS) data

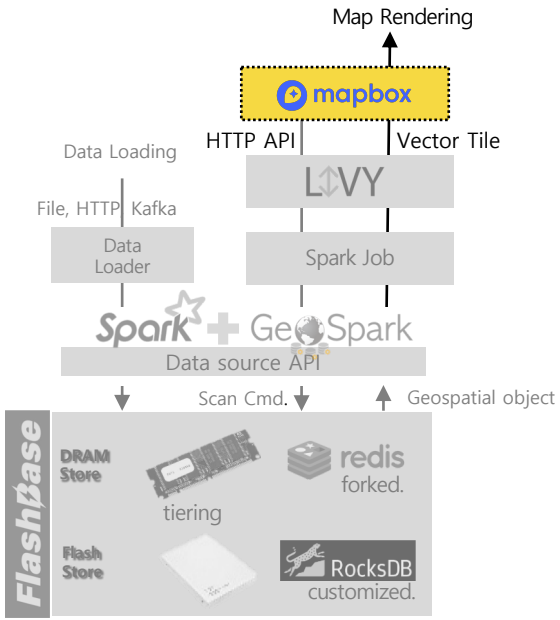
  - Requires heavy aggregate computations

→ **Good case to demonstrate real-time big data processing**

  - Some companies demonstrated geospatial analysis to show advantages of GPU database over CPU database

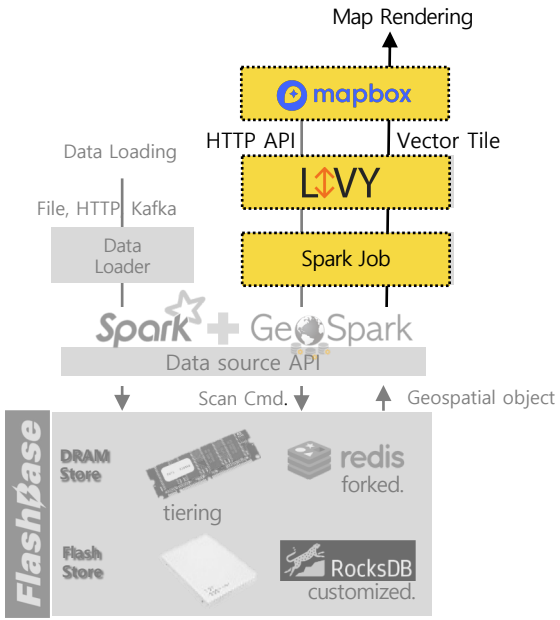
→ **We have tried it with Spark & FlashBase based on CPU**

Architecture of Geospatial Visualization



- Front-end : MapBoxJS
  - MapBox uses VectorTile to render on overlay layers.
  - Get VectorTile via HTTP API

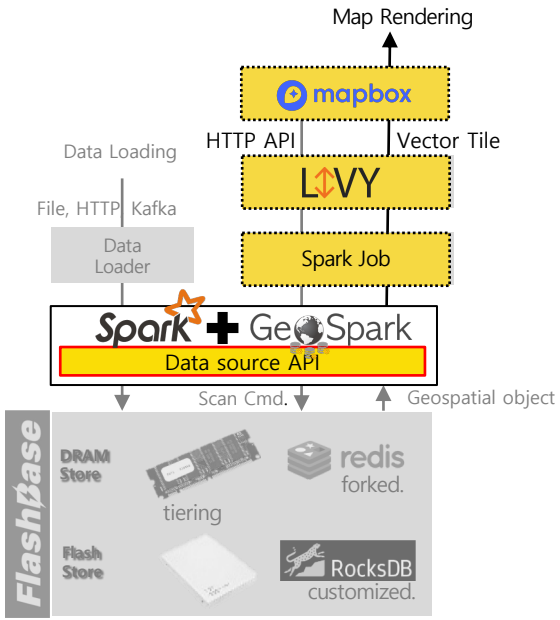
Architecture of Geospatial Visualization



- Front-end : MapBoxJS
  - MapBox uses VectorTile to render on overlay layers.
  - Get VectorTile via HTTP API

- Back-end Web server
  - Build VectorTiles with Spark Job.
  - Apache LIVY
    : manipulate multiple Spark Contexts simultaneously

Architecture of Geospatial Visualization

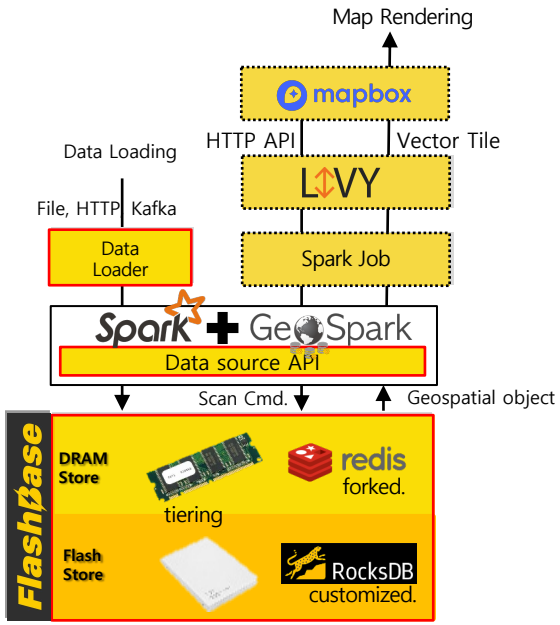


- Front-end : MapBoxJS
  - MapBox uses VectorTile to render on overlay layers.
  - Get VectorTile via HTTP API

- Back-end Web server
  - Build VectorTiles with Spark Job.
  - Apache LIVY
    : manipulate multiple Spark Contexts simultaneously

- Spark Cluster & Geo-Spark
  - Geo-Spark : support Geospatial UDFs and Predicates

Architecture of Geospatial Visualization



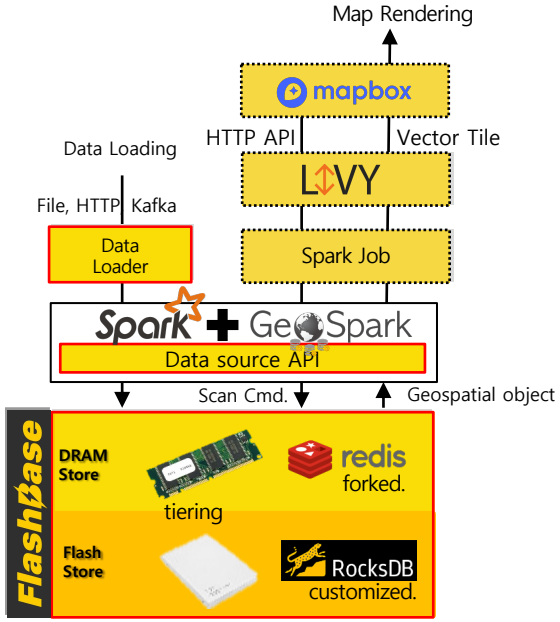
- Front-end : MapBoxJS
  - MapBox uses VectorTile to render on overlay layers.
  - Get VectorTile via HTTP API

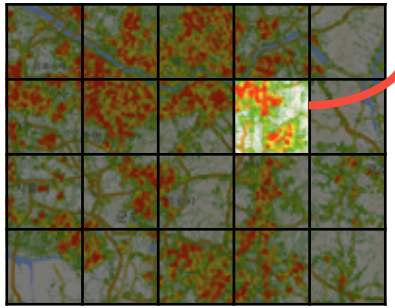
- Back-end Web server
  - Build VectorTiles with Spark Job.
  - Apache LIVY
    : manipulate multiple Spark Contexts simultaneously

- Spark Cluster & Geo-Spark
  - Geo-Spark : support Geospatial UDFs and Predicates

- FlashBase
  - FlashBase stores objects with latitude and longitude.
    Each data is partitioned by its GeoHash

Architecture of Geospatial Visualization





256
X
256 pix.

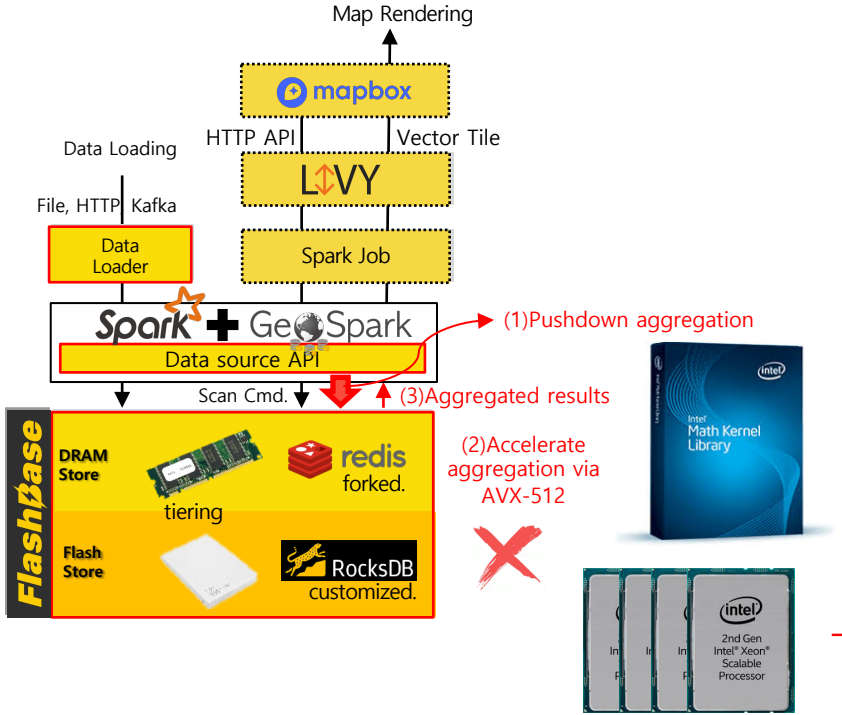
Building VectorTile requires AGG GROUP BY pixel.

- If web-client shows 20 Tiles,
→ 1.3 million (256 x 256 x 20) AGG operations are required.
→ Cause heavy computations & shuffle writing in Spark.
- If user scroll map,
→ all tiles should be re-calculated.

Problem

-Latency issue of HTTP API : Sluggish map loading !
-Building VectorTile needs
heavy computation & shuffling for aggregation.

Architecture of Geospatial Visualization



- Optimization of performance

1. Spark pushdowns aggregation to FlashBase
   FlashBase sends aggregated results to Spark
   → **Reduce Shuffle writing size and computation of Spark to 1/10.**

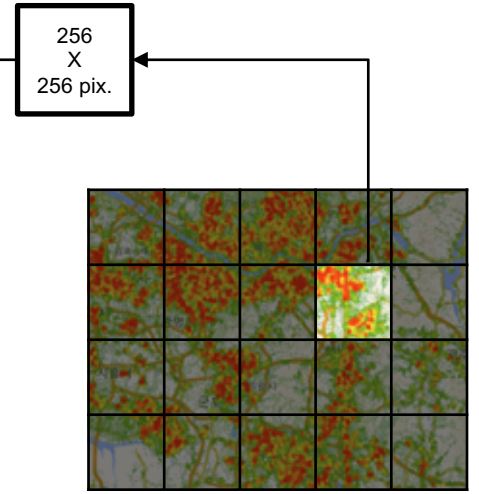
2. FlashBase accelerates aggregation with vector-processing via Intel's AVX-512 (Intel Math Kernel Library)
   → **2 x faster aggregation.**

→ **20 times faster than original GeoSpark**

Optimization Detail

The Query building features of VectorTile

SELECT * FROM pcell WHERE ST_VectorTileAggr('7,109,49', 'AVG')



1. ST_VectorTileAggr(arg1, arg2)
 - Custom predicate which contains aggregation information.
 - arg1 : zoom level of map & tile pos (x, y) in Globe
 - arg2 : aggregation type (SUM or AVG)

2. Define & Apply a custom optimization rule
 - Applied during optimization phase of query plan.
 - Parse aggregation information from predicate and pushdown it to FlashBase

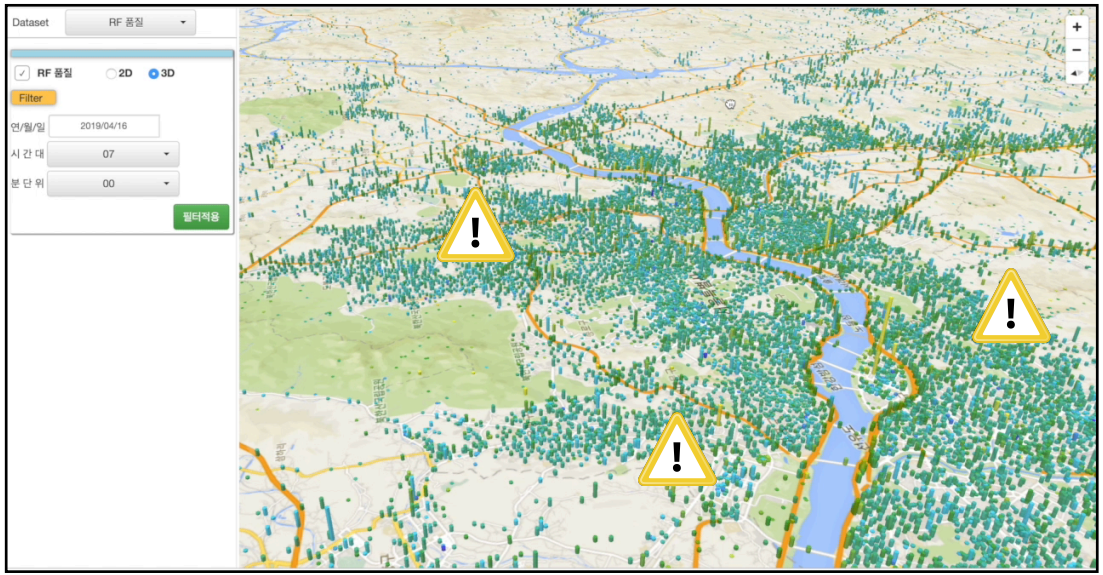
3. Aggregation in FlashBase
 - Parallelized computation by FlashBase process count (Generally 100 ~ 200 process / node)
 - Each process of FlashBase accelerates aggregation using Intel MKL.

Contents

- Network Quality Analysis
- Geospatial Visualization
- **Network Quality Prediction**
- Wrap up

Introduction of Network Quality Prediction

- Predict Network Quality Indicators (CQI, RSRP, RSRQ, SINR, …)*
  for anomaly detection and real-time management
- Goal : Unify Geospatial visualization & Network Prediction On Spark



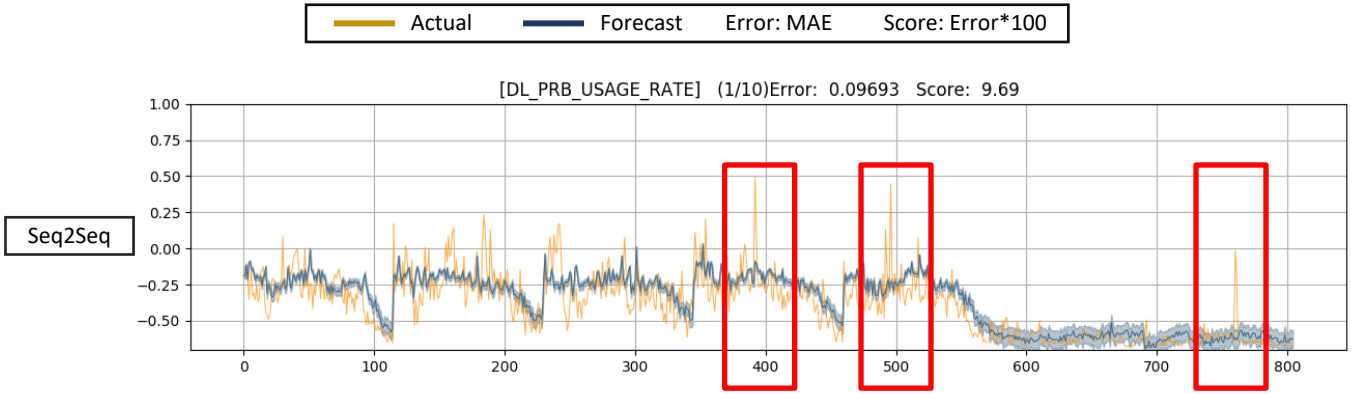
* CQI : Channel Quality Indicator
* RSRP : Reference Signal Received Power
* RSRQ : Reference Signal Received Quality
* SINR :Signal to Interference Noise Ratio

We focused on

1. Improving deep learning model for forecasting time series data
2. Improving architecture and data pipeline for training and inference

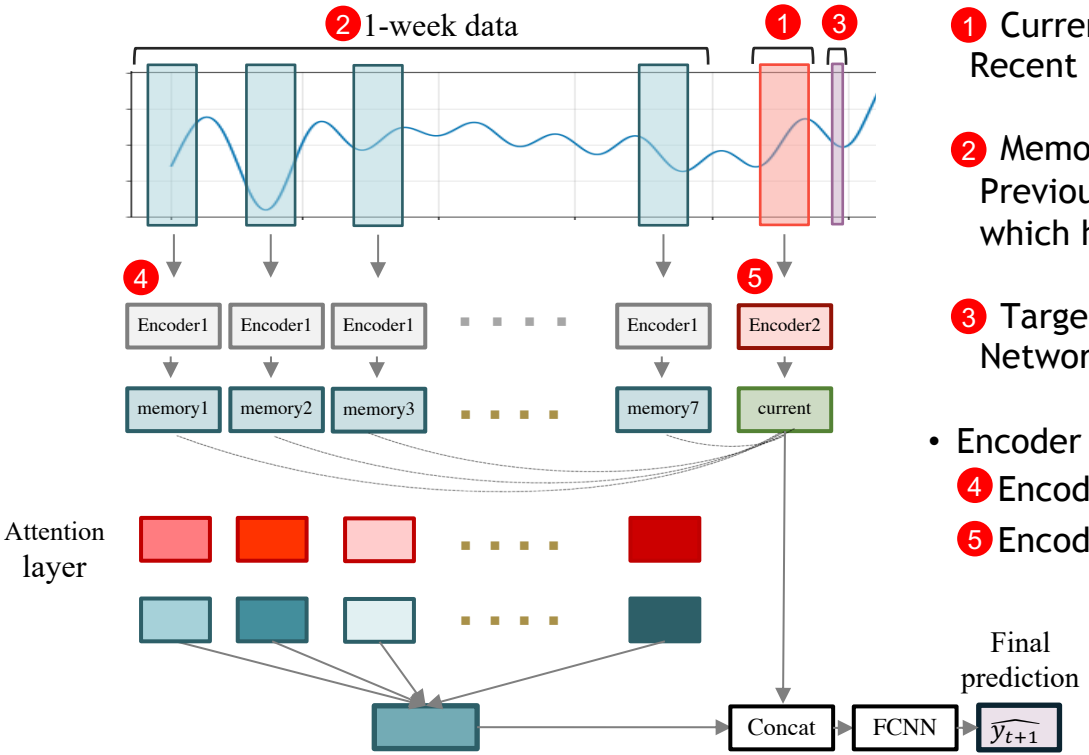
Model for Network Quality Prediction - RNN

- RNN type model(Seq2Seq) is common solution for time-series prediction. But not suitable for our network quality prediction.



Cannot predict sudden change!

Memory augmented model



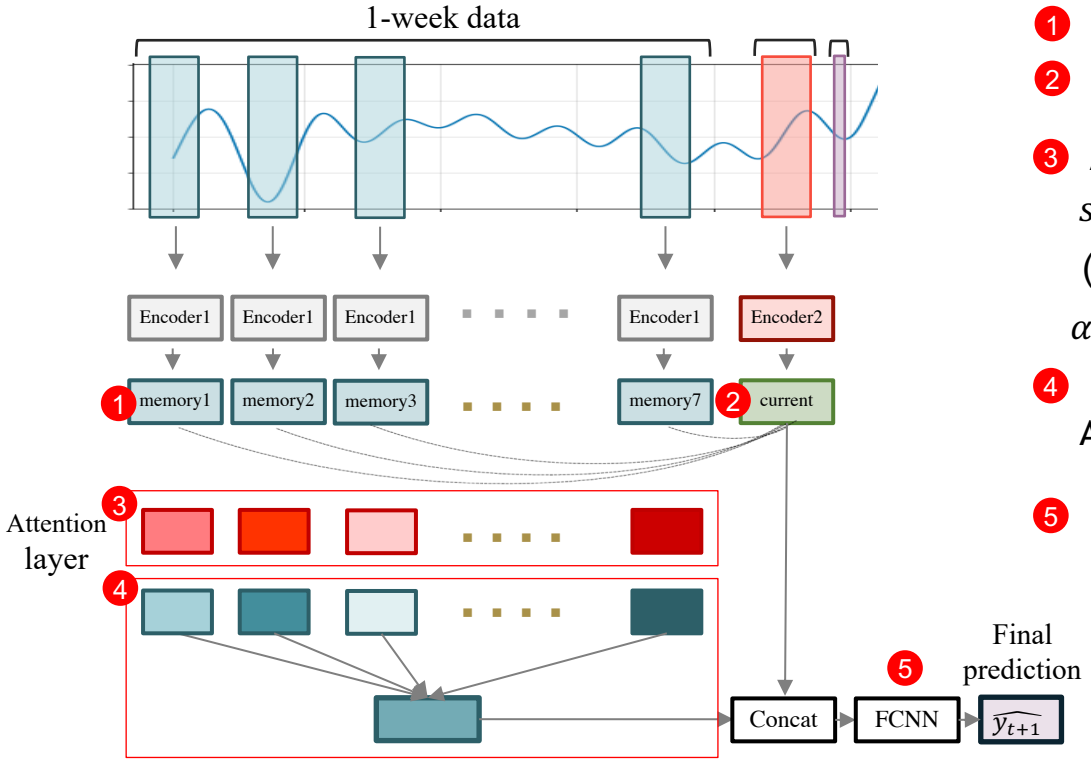
1 Current
Recent 50 min data with 5 min period

2 Memory
Previous 7 days historical data each of which has same time band with current and target.

3 Target
Network quality after 5 min

- Encoder : 1-NN (Autoregressive term)
  - 4 Encoder1 : $h_t = c + w_1 y_{t-ndays-1} + \ldots + w_{11} y_{t-n\,days-11}$
  - 5 Encoder2 : $h'_t = c' + w'_1 y_{t-1} + \ldots + w'_{10} y_{t-10}$

Memory augmented model



1. m_t = memory for step t
2. c = current state
3. Attention Layer (1 layer neural-network)

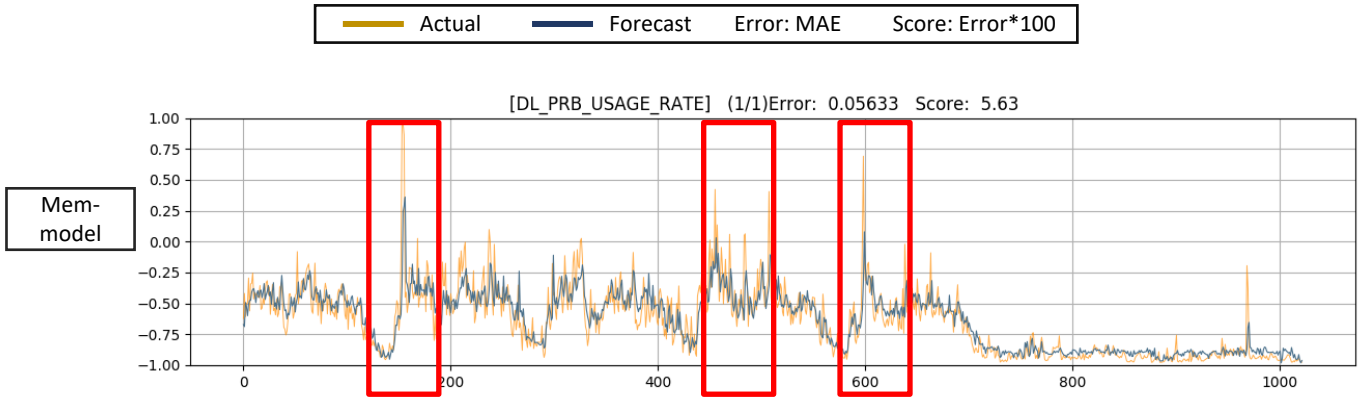
   $score_t = v^T \tanh(W_a[m_t; c])$

   $(v, W_a$: weight parameters)

   $\alpha_t = softmax(score_t)$
4. Attention Vector

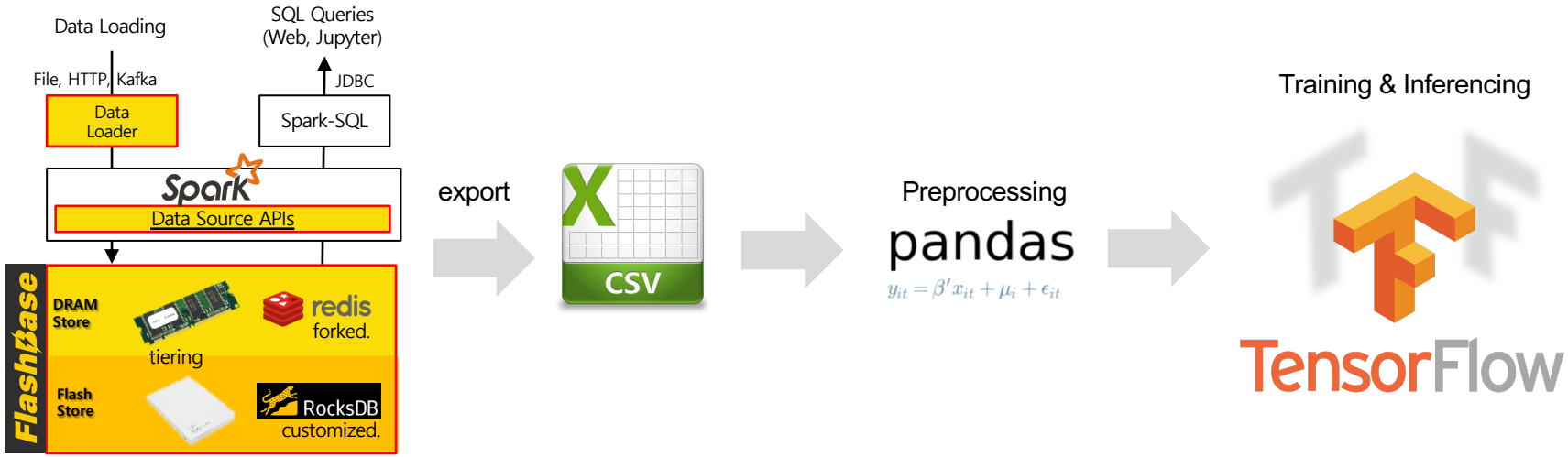
   Attention weighted summation of m_t
5. Fully connected neural-network

Memory augmented model - Test result



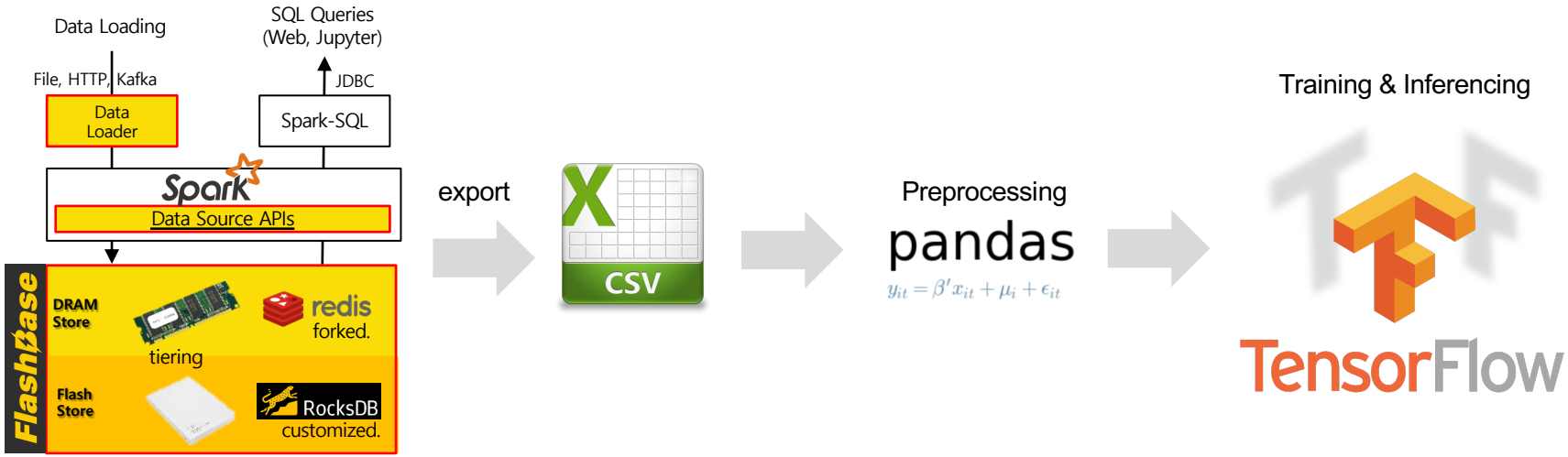
Improved predictions for sudden change!

Training & Inference Architecture - Legacy



1. Export data to CSV from Spark ThriftServer using Hive Client
2. Preprocessing with pandas.
3. Train or infer with TensorFlow CPU

Training & Inference Architecture - Legacy



Problem

1. No in-memory Pipeline between data source and Deep-Learning layer
2. Pre-processing & Inference & Training are performed in single server.

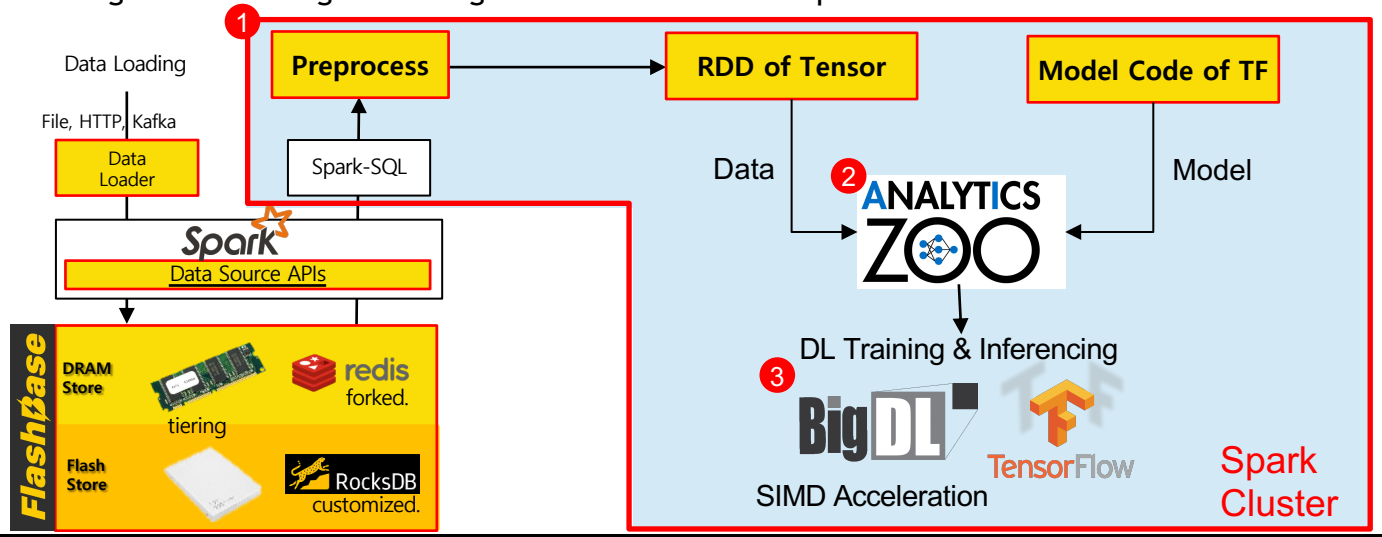
Training & Inference Architecture - New

1. Build In-memory Pipeline between FlashBase and Intel Analytics ZOO

   Data Layer And Inferencing & Training Layer are integrated into the same Spark Cluster
   Also share the same Spark session.
   Source Code : https://github.com/mnms/ARMemNet-BigDL

2. Intel Analytics Zoo : Used to unify TF model into Spark Pipeline seamlessly.
3. Intel BigDL : inference & training engine

   The processing of Inferencing & training can be distributed in Spark Cluster.



Comparison between two architectures

- Now only inference result : Also has a plan to run the distributed training later
- Test workload
  - 7,722,912 rows = 80,447 cell towers X 8 days X 12 rows (1 hour data with 5 minutes period)
  - 8 network indicators per row
  - → Input tensor (80,447 X 696) = current input (80,447 X 10 X 8) + Memory input (80,447 X 77 X 8)

	Pandas + TF	Spark + Analytics Zoo		
Data Export	2.3s	N/A		
Pre-processing	71.96s	local	2.56s	3 node Yarn 1.43s
Deep Learning Inference	1.06s (CPU) / 0.63s (GPU)	local	0.68s	3 node Yarn 0.18s

45 x faster

※ CPU : Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz
※ GPU : Nvidia-K80

Performed in a single node

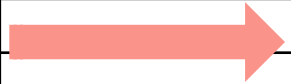
- Data and computations are distributed in 50 partitions.
- Preprocessing and inference are executed in a single Spark job

Comparison between TF CPU and Analytics Zoo

- Compare Inference Time
- Environment
  - CPU : Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz
  - Core : 36

Batch Size	Elapsed Time (TF-CPU)	Elapsed Time (Analytics Zoo Scala on Spark)
32	14.28068566	2.58333056
64	8.387897015	1.446166912
128	4.871953249	0.679720256
256	2.947942972	0.426830048
512	2.030963659	0.400032064
1024	2.012846947	0.395362112
2048	1.44268322	0.430505056

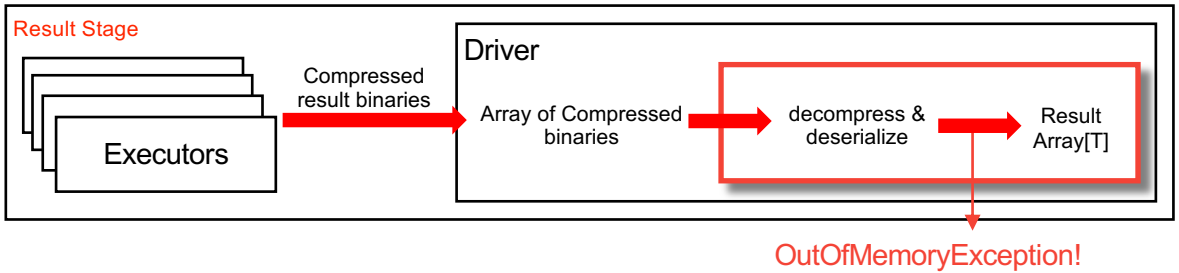
3~5 times faster



Appx. Memory Problem of Spark Driver

- collect() function of DataSet sometimes throws OOM while decompressing and deserializing result.
  → Job Fails and Spark Driver is killed.

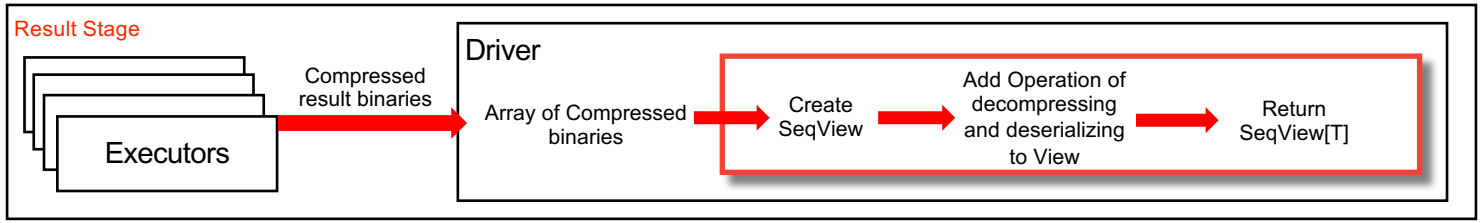
- Spark supports ‘spark.driver.maxResultSize’ config for this issue
  - it just reflects a compressed size
  - Actual result size would be 5x ~ 20x of compressed size.
  - It is difficult to adjust the config to protect driver from OOM.



Appx. Memory Problem of Spark Driver - Solution

- **Define collectAsSeqView function in DataSet**
  - Define SeqView which just holds compressed results and decompressing operations
  - Driver decompresses and deserializes according to each fetching.
  - Decompressed & deserialized results are collected as garbage after cursor moves to next.
  - Only compressed binary reside in memory : memory of job can be limited by 'spark.driver.maxResultSize'

  → **Completely protect the driver from OOM while collecting results**



Appx. Memory Problem of Spark Driver - Patch

- collectAsSeqView function only uses 10% ~ 20% memory compared to collect function.

- Create Spark Pull Request which applies this to thrift server.
  - PR : SPARK-25224 (https://github.com/apache/spark/pull/22219)
  - Review in progress
  - Create Spark Pull Request which applies this to thrift serve

Contents

- Network Quality Analysis
- Geospatial Visualization
- Network Quality Prediction
- **Wrap up**

Open Discussion

- More partitioning or indexing with less partitions
- Spark datasource v2 and aggregation pushdown
- Possible new directions of FlashBase for Spark ecosystem
- Efficient end to end data pipeline for big data based inference and training

How to use Spark with FlashBase

Free binary can be used (Not open sourced yet)

- Public Cloud: AWS Marketplace AMI (~19.12.31), Cloud Formation (~20.3.31)
- On-premise: github page (~20.3)

First contact to us if you want to try FlashBase and get some help

- e-mail: flashbase@sktelecom.com
- homepage (temporary): https://flashbasedb.github.io

Q & A

Visit Intel & SKT Demo booth

SPARK+AI SUMMIT 2019

DON'T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

Download on the App Store

GET IT ON Google Play