metatron discovery

White Paper

Metatron:

A Druid-powered end-to-end solution

Contents

Figure list

Table list

1 Introduction

The development of information and communications technology has been accompanied by a rapid increase in the amount of data generated, highlighting the importance of efficient data collection, management, and utilization. However, RDBMS-based legacy tools are unable to process mass amounts of multidimensional data. This has led to the emergence of new methodologies and solutions aimed at satisfying the demand for big data.

Metamarkets, a technology startup based in Silicon Valley, launched a column-oriented distributed data store known as Druid in 2011, and open sourced it in October 2012. Many companies have turned to Druid for their backend technology because it offers various advantages, including fast and efficient data processing.

As a B2C telecommunications service provider, SK Telecom recognized the need to effectively manage and analyze the vast amounts of network data generated by its users every minute. Metatron, an end-to-end business intelligence solution with Druid as the underlying engine, was thus developed and launched in 2016.

This paper examines the characteristics of Druid that make it suitable for time-series data processing, and introduces how they were adapted and improved by SK Telecom for Metatron. Chapters 2 and 3, which contain the introduction of Druid, were written with reference to the Druid website[1] and other official materials on Druid.[2-6]

2 Background of Druid development

Druid was originally designed to satisfy the following needs around ingesting and exploring large quantities of transactional events (log data):

First, the developers wanted to be able to rapidly and arbitrarily slice and dice data and drill into that data effectively without any restrictions, along with sub-second queries over any arbitrary combination of dimensions. These capabilities were needed to allow users of their data dashboard to arbitrarily and interactively explore and visualize event streams.

Second, the developers wanted to be able to ingest events and make them exportable almost immediately after their occurrence. This was crucial to enable users to collect and analyze data in real time for timely situational assessments, predictions, and business decisions. Popular open source data warehousing systems such as Hadoop were unable to provide the sub-second data ingestion latencies as required.

Finally, the developers wanted to ensure multitenancy and high availability for their solution services. Their systems needed to be constantly up and be able to withstand all sorts of potential failures without going down or taking any downtime. Downtime is costly and many businesses cannot afford to wait if a system is unavailable in the face of software upgrades or network failure.

3 Characteristics of open-source Druid

3.1 Data table components

Data tables in Druid (called “data sources”) are collections of timestamped events designed for OLAP queries. A data source is composed of three distinct types of columns (here we use an example dataset from online advertising).

Table 1: Time-series OLAP table example

Timestamp column	Dimension columns				Metric columns	
timestamp	publisher	advertiser	gender	country	click	price
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	USA	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	USA	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	USA	1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	1	1.53

Source: http://druid.io

- **Timestamp column:** Druid treats timestamp separately in a data source because all its queries center around the time axis. (If non-time series data is ingested in batch, all records are timestamped with the current time for use in Druid.)
- **Dimension columns:** Dimensions are string attributes of an event, and the columns most commonly used in filtering the data. Four dimensions are involved in the example dataset: publisher, advertiser, gender, and country. They each represent an axis of the data chosen to slice across.
- **Metric columns**: Metrics are columns used in aggregations and computations. In the example, the metrics are clicks and price. Metrics are usually numeric values, and computations include operations such as count, sum, and mean (Metatron has extended supported Druid data types).

3.2 Data ingestion

Druid supports real-time and batch ingestion. One major characteristic of Druid is real-time ingestion, which is enabled by real-time nodes (For details, see Section 3.7.1 Real-time nodes). Events ingested in real-time from a data stream get indexed in seconds to become queryable in the Druid cluster.

3.3 Data roll-up

The individual events in our example dataset are not very interesting because there may be trillions of

such events. However, summarizations of this type of data by time interval can yield many useful insights. Druid summarizes this raw data when ingesting it using an optional process called “roll-up.” Below is an example of roll-up:

Table 2: Data roll-up example

timestamp	domain	gender	clicked
2011-01-01T00:01:35Z	bieber.com	Female	1
2011-01-01T00:03:03Z	bieber.com	Female	0
2011-01-01T00:04:51Z	ultra.com	Male	1
2011-01-01T00:05:33Z	ultra.com	Male	1
2011-01-01T00:05:53Z	ultra.com	Female	0
2011-01-01T00:06:17Z	ultra.com	Female	1
2011-01-01T00:23:15Z	bieber.com	Female	0
2011-01-01T00:38:51Z	ultra.com	Male	1
2011-01-01T00:49:33Z	bieber.com	Female	1
2011-01-01T00:49:53Z	ultra.com	Female	0

➔

timestamp	domain	gender	clicked
2011-01-01T00:00:00Z	bieber.com	Female	1
2011-01-01T00:00:00Z	ultra.com	Female	2
2011-01-01T00:00:00Z	ultra.com	Male	3

Source: Interactive Exploratory Analytics with Druid | DataEngConf SF '17

The table on the left lists the domain click events that occurred from 00:00:00 to 01:00:00 on January 1, 2011. Since individual events recorded in seconds do not have much significance from the analyst’s perspective, the data was compiled at a granularity of one hour. This results in the more meaningful table on the right, which shows the number of clicks by gender for the same time period.

In practice, rolling up data can dramatically reduce the size of data that needs to be stored (up to a factor of 100), thereby saving on storage resources and enabling faster queries.

But, as data is rolled up, individual events can no longer be queried; the rollup granularity is the minimum granularity you will be able to explore data at and events are floored to this granularity. The unit of granularity can be set as desired by users. If necessary, the roll-up process may be disabled to ingest every individual event.

3.4 Data sharding

A data source is a collection of timestamped events and partitioned into a set of shards. A shard is called a segment in Druid and each segment is typically 5–10 million rows. Druid partitions its data sources into well-defined time intervals, typically an hour or a day, and may further partition on values from other columns to achieve the desired segment size.

The example below shows a data table segmented by hour:

Segment `sampleData_2011-01-01T01:00:00:00Z_2011-01-01T02:00:00:00Z_v1_0`:

```
2011-01-01T01:00:00Z  ultratrimfast.com  google.com  Male  USA  1800  25  15.70
2011-01-01T01:00:00Z  bieberfever.com    google.com  Male  USA  2912  42  29.18
```

Segment `sampleData_2011-01-01T02:00:00:00Z_2011-01-01T03:00:00:00Z_v1_0`:

```
2011-01-01T02:00:00Z  ultratrimfast.com  google.com  Male  UK  1953  17  17.31
2011-01-01T02:00:00Z  bieberfever.com    google.com  Male  UK  3194  170  34.01
```

This segmentation by time can be achieved because every single event in a data source is timestamped.

Segments represent the fundamental storage unit in Druid and replication and distribution are done at a segment level. They are designed to be immutable, which means that once a segment is created, it cannot be edited. This ensures no contention between reads and writes. Druid segments are just designed to be read very fast.

In addition, this data segmentation is key to parallel processing in Druid's distributed environment: As one CPU can scan one segment at a time, data partitioned into multiple segments can be scanned by multiple CPUs simultaneously in parallel, thereby ensuring fast query returns and stable load balancing.

3.5 Data storage format and indexing

The way Druid stores data contributes to its data structures highly optimized for analytic queries. This section uses the Druid table below as an example:

Table 3: Druid table example for edits that have occurred on Wikipedia

Timestamp	Page	Username	Gender	City	Characters Added	Characters Removed
2011-01-01T01:00:00Z	Justin Bieber	Boxer	Male	San Francisco	1800	25
2011-01-01T01:00:00Z	Justin Bieber	Reach	Male	Waterloo	2912	42
2011-01-01T02:00:00Z	Ke$ha	Helz	Male	Calgary	1953	17
2011-01-01T02:00:00Z	Ke$ha	Xeno	Male	Taiyuan	3194	170

Source: Druid: A Real-time Analytical Data Store

3.5.1 Columnar storage and indexing

Druid is a column store, which means each individual column is stored separately. Given that Druid is best used for aggregating event streams, column storage allows for more efficient CPU usage as only the columns pertaining to a query are actually loaded and scanned in that query. In a row oriented data store, all columns associated with a row must be scanned as part of an aggregation. The additional scan time can introduce significant performance degradations.

Different columns can employ different compression methods and have different indices associated with them to reduce the cost of storing a column in memory and on disk. In the example above, the page, user, gender, and city columns only contain strings. Storing strings directly is unnecessarily costly; instead, they can be mapped into unique integer identifiers. For example:

Justin Bieber -> 0

Ke$ha -> 1

This mapping allows the page column to be represented as an integer array where the array indices correspond to the rows of the original dataset. For the page column, we can represent the unique pages as follows:

[0, 0, 1, 1]

Thus, strings are replaced by fixed-length integers in storage, which are much easier to compress. Druid indexes data on a per-shard (segment) level.

3.5.2 Indices for filtering data

Druid creates additional lookup indices that facilitate filtering on string columns. Let us consider the above example table again. A query might be: “How many Wikipedia edits were done by users in San Francisco who are also male?” This example query involves two dimensions: City (San Francisco) and Gender (Male). For each dimension, a binary array is created where the array indices represent whether or not their corresponding rows match the query filter, as shown below:

San Francisco (City) -> rows [1] -> [1][0][0][0]

Male (Gender) -> rows [1, 2, 3, 4] -> [1][1][1][1]

And the query filter performs the AND operation between the two arrays:

[1][0][0][0] AND [1][1][1][1] = [1][0][0][0]

As a result, only row 1 is subject to scanning, which retrieves only the filtered rows and eliminates unnecessary workload. And these binary arrays are very easy to compress as well.

This lookup can be used for the OR operation too. If a query filters on San Francisco or Calgary, array indices will be for each dimension value:

San Francisco (City) -> rows [1] -> [1][0][0][0]

Calgary (City) -> rows [3] -> [0][0][1][0]

And then the OR operation is performed on the two arrays:

[1][0][0][0] OR [0][0][1][0] = [1][0][1][0]

Thus the query scans rows 1 and 3 only.

This approach of performing Boolean operations on large bitmap sets is commonly used in search engines.

3.6 Query languages

Druid’s native query language is JSON over HTTP. Druid queries include:

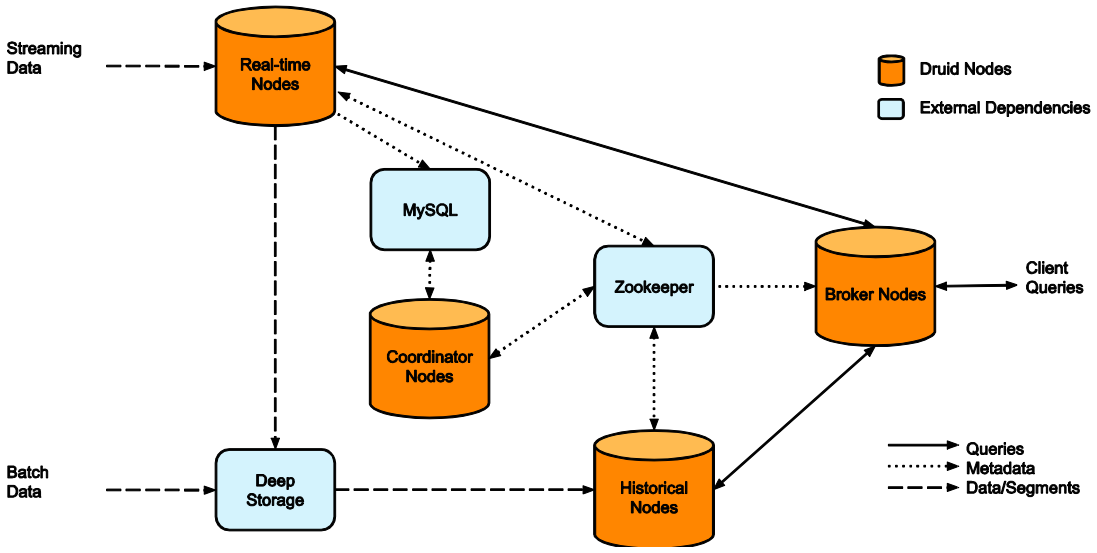
- Group By
- Time-series roll-ups
- Arbitrary Boolean filters
- Sum, Min, Max, Avg and other aggregation functions
- Dimensional search

In addition to these, query libraries in numerous languages, including SQL, are developed and shared.

3.7 Basic cluster architecture

A Druid cluster consists of different types of nodes and each node type is designed to perform a specific set of things:

Figure 1: An overview of a Druid cluster and the flow of data through the cluster

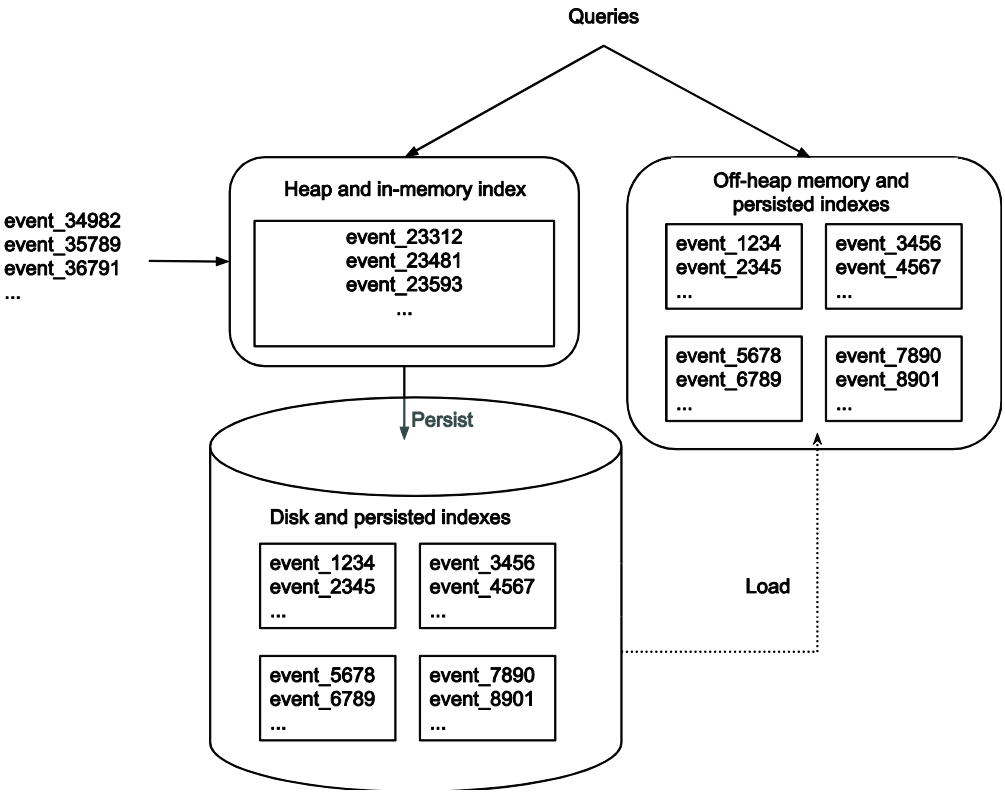


Source: Druid: A Real-time Analytical Data Store

3.7.1 Real-time nodes

Real-time nodes function to ingest and query event streams. The nodes are only concerned with events for some small time range and periodically hand them off to the deep storage in the following steps:

Figure 2: Data processing in real-time nodes



Source: Druid: A Real-time Analytical Data Store

1) Incoming events are indexed in memory and immediately become available for querying.

2) The in-memory data is regularly persisted to disk and converted into an immutable, columnar storage format.

3) The persisted data is loaded into off-heap memory to be still queryable.

4) On a periodic basis, the persisted indexes are merged together to form a “segment” of data and then get handed off to deep storage.

In this way, all events ingested into real-time nodes, regardless before or after persisted, are present in memory (either on- or off-heap) and thus can be queried (queries hit both the in-memory and persisted indexes). This functionality of real-time nodes enables Druid to conduct real-time data ingestion meaning that events can be queried almost as soon as they occur. In addition, there is no data loss during these steps.

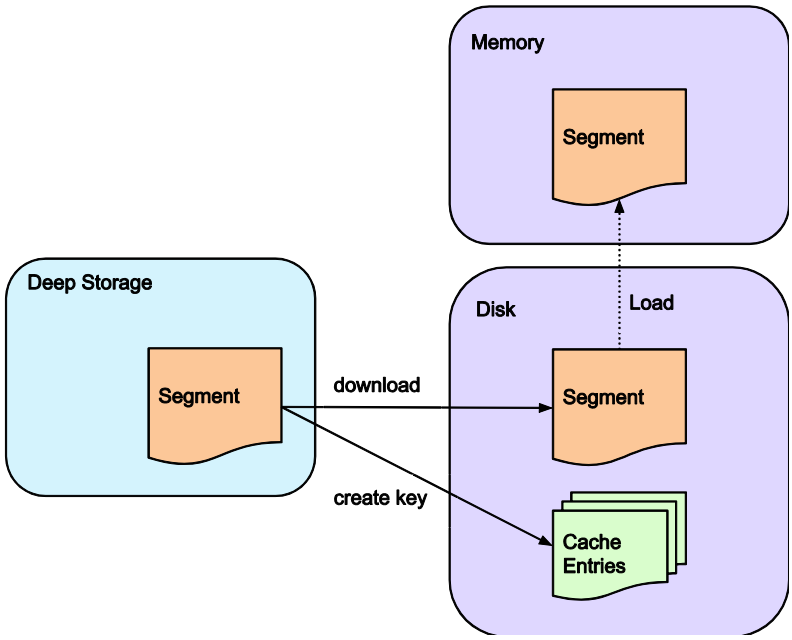
Real-time nodes announce their online state and the data they serve in Zookeeper (an external dependency for the Druid cluster; see Section 3.7.5) for the purpose of coordination with the rest of the Druid cluster.

3.7.2 Historical nodes

Historical nodes function to load and serve the immutable blocks of data (segments) created by real-time nodes. These nodes download immutable segments locally from the deep storage and serve queries over those segments (e.g., data aggregation/filtering). The nodes are operationally simple based on a shared-nothing architecture; they have no single point of contention and simply load, drop, and serve segments as instructed by Zookeeper.

A historical node’s process of serving a query is as follows:

Figure 3: Historical nodes download data from deep storage and load it in memory



Source: Druid: A Real-time Analytical Data Store

Once a query is received, the historical node first checks a local cache that maintains information about what segments already exist on the node. If information about a segment in question is not present in the cache, the node will proceed to download the segment from deep storage. On the completion of the processing, the segment is announced in Zookeeper to become queryable and the node performs the requested query on the segment.

Historical nodes can support read consistency because they only deal with immutable data. Immutable data blocks also enable a simple parallelization model: historical nodes can concurrently scan and aggregate immutable blocks without blocking.

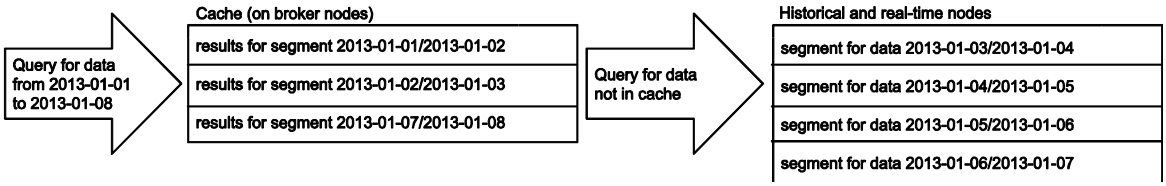
Similar to real-time nodes, historical nodes announce their online state and the data they are serving in Zookeeper.

3.7.3 Broker nodes

Broker nodes understand the metadata published in Zookeeper about what segments are queryable and where those segments are located. Broker nodes route incoming queries such that the queries hit the right historical or real-time nodes. Broker nodes also merge partial results from historical and real-time nodes before returning a final consolidated result to the caller.

Broker nodes use a cache for resource efficiency as follows:

Figure 4: Caching in broker nodes



Source: Druid: A Real-time Analytical Data Store

Once a broker node receives a query involving a number of segments, it checks for segments already existing in the cache. For any segments absent in the cache, the broker node will forward the query to the correct historical and real-time nodes. Once historical nodes return their results, the broker will cache these results on a per-segment basis for future use. Real-time data is never cached and hence requests for real-time data will always be forwarded to real-time nodes. Since real-time data is perpetually changing, caching the results is unreliable.

3.7.4 Coordinator nodes

Coordinator nodes are primarily in charge of data management and distribution on historical nodes. The coordinator nodes determine which historical nodes perform queries on which segments and tell them to load new data, drop outdated data, replicate data, and move data to load balance. This enables fast, efficient, and stable data processing in a distributed group of historical nodes.

As with all Druid nodes, coordinator nodes maintain a Zookeeper connection for current cluster information. Coordinator nodes also maintain a connection to a MySQL database that contains additional operational parameters and configurations, including a rule table that governs how segments are created, destroyed, and replicated in the cluster.

Coordinator nodes undergo a leader-election process that determines a single node that runs the coordinator functionality. The remaining coordinator nodes act as redundant backups.

External dependencies

Druid has a couple of external dependencies for cluster operations.

- **Zookeeper:** Druid relies on Zookeeper for intra-cluster communication.
- **Metadata storage:** Druid relies on a metadata storage to store metadata about segments and configuration. MySQL and PostgreSQL are popular metadata stores for production.
- **Deep storage:** Deep storage acts as a permanent backup of segments. Services that create segments upload segments to deep storage and historical nodes download segments from deep storage. S3 and HDFS are popular deep storages.

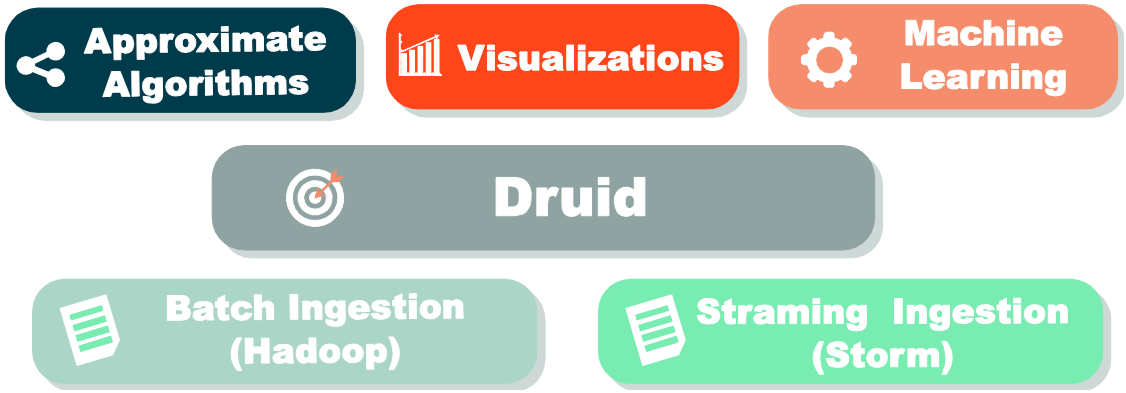
3.7.5 High availability

Druid is designed to have no single point of failure. The different node types operate fairly independent of each other and there is minimal interaction among them. Hence, intra-cluster communication failures have minimal impact on data availability. To run a highly available Druid cluster, you should have at least two nodes of every node type running.

3.7.6 Architecture extensibility

Druid features a modular, extensible platform that allows various external modules to be added to its basic architecture. An example of how Druid's architecture can be extended with modules is shown below:

Figure 5: Example of Druid's extended architecture



Source: MetaMarkets - Introduction to Druid by Fangjin Yang

Metatron, an end-to-end business intelligence solution to be introduced in this paper, was also built by adding various modules to the Druid engine.

4 Druid performance assessments

With Druid being a data store that supports real-time data exploration, its quantitative assessments are focused on two key aspects:

- Query latency
- Ingestion latency

This is because the key to achieving “real-time” performance is to minimize the time spent on query processing and ingestion. A number of organizations and individuals, including the developers of Druid, have established benchmarks for Druid performance assessment based on the two key aspects, and shared how Druid compares to other database management systems.

4.1 Self-assessment by Druid developers

Druid: A Real-time Analytical Data Store[2] was published by the developers in 2014. Chapter 6. Performance contains details of Druid assessment, with a particular focus on query and ingestion latencies. The benchmarks of Druid performance are briefly introduced in the following sections.

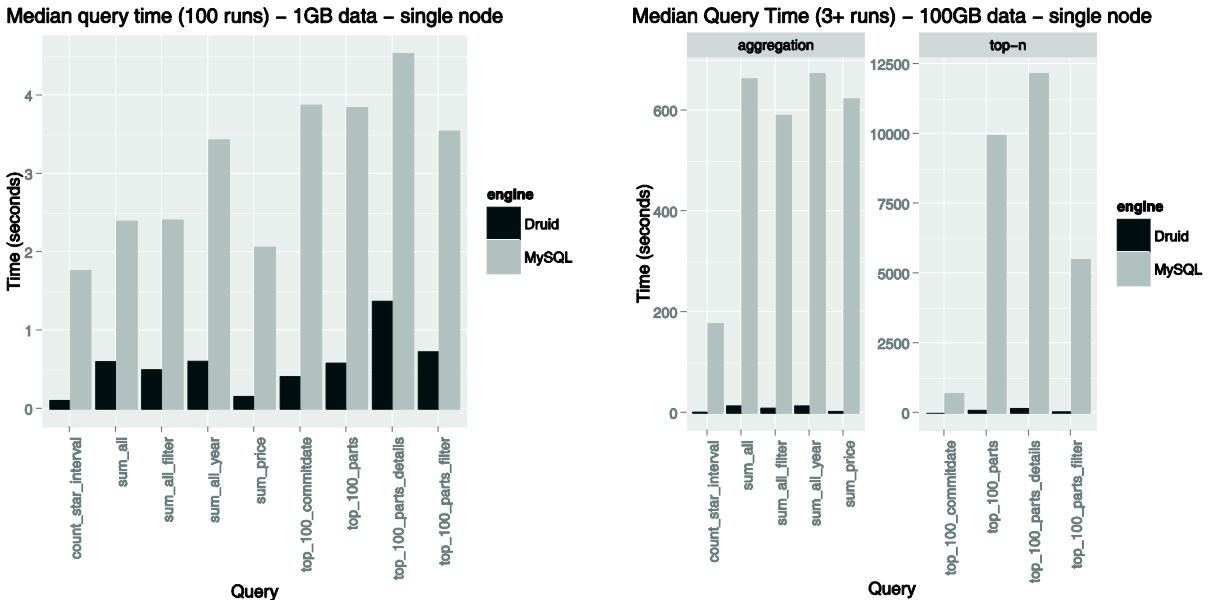
4.1.1 Query latency

Regarding Druid’s query latency, the paper discusses two performance assessments—one was conducted on eight data sources that had been most queried at Metamarkets and the other was on TPC-H datasets. In this section, we review the latter assessment. The latencies from querying on TPC-H datasets were measured by comparing with MySQL, and the cluster environment was as follows:

- Druid historical nodes: Amazon EC2 m3.2xlarge instance types (Intel® Xeon® E5-2680 v2 @ 2.80GHz)
- Druid broker nodes: c3.2xlarge instances (Intel® Xeon® E5-2670 v2 @ 2.50GHz)
- MySQL Amazon RDS instance (The same m3.2xlarge instance type as Druid)

The figure below shows the query latencies resulting from Druid and MySQL when tested on the 1GB and 100GB TPC-H datasets:

Figure 6: Druid & MySQL benchmarks – 1GB and 100GB TPC-H data

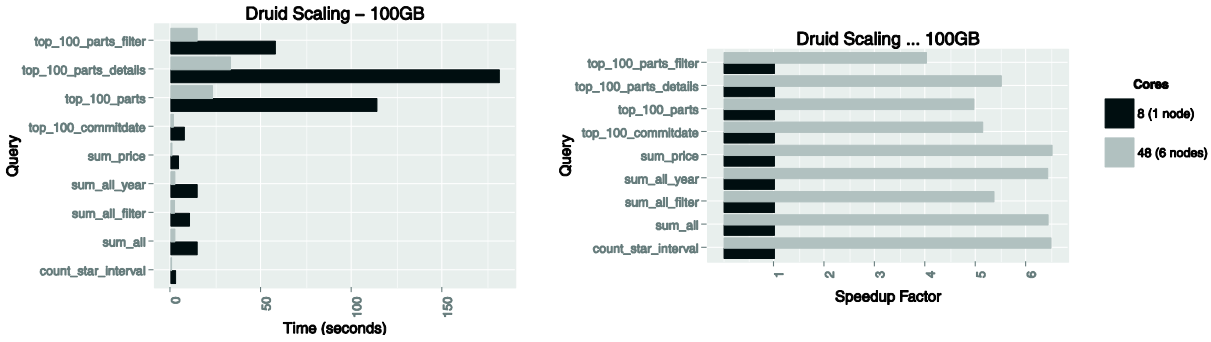


Source: Druid: A Real-time Analytical Data Store

By showcasing these results, the paper suggests that Druid is capable of extremely faster query returns compared to legacy relational database systems.

The Druid paper also presents how faster query returns are achieved when multiple nodes are joined together in a cluster. When tested on the TPC-H 100 GB dataset, the performance difference between a single node (8 cores) and six-node cluster (48 cores) was as follows:

Figure 7: Druid scaling benchmarks – 100GB TPC-H data



Source: Druid: A Real-time Analytical Data Store

It was observed that not all types of queries achieve linear scaling, but the simpler aggregation queries do, ensuring a speed increment almost proportional to the number of the cores (SK Telecom have made improvements to achieve much more linear scalability).

4.1.2 Ingestion latency

The paper also assessed Druid's data ingestion latency on a production ingestion setup consisting of:

- 6 nodes, totalling 360GB of RAM and 96 cores (12 x Intel®Xeon®E5-2670).

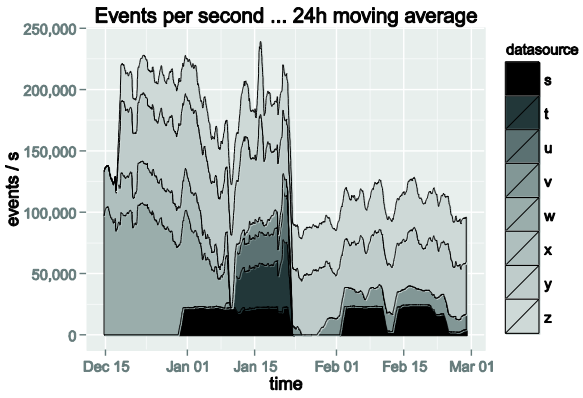
A total of eight production data sources were selected for this assessment. The characteristics of each data source and their ingestion results are shown below. Note that in this setup, several other data

sources were being ingested and many other Druid related ingestion tasks were running concurrently on the machines.

Figure 8: Characteristics of the datasets ingested into Druid and their ingestion latencies

Data Source	Dimensions	Metrics	Peak events/s
s	7	2	28334.60
t	10	7	68808.70
u	5	1	49933.93
v	30	10	22240.45
w	35	14	135763.17
x	28	6	46525.85
y	33	24	162462.41
z	33	24	95747.74

Ingestion characteristics of various data sources



Combined cluster ingestion rates

Source: Druid: A Real-time Analytical Data Store

Druid's data ingestion latency is heavily dependent on the complexity of the dataset being ingested, but the latency measurements present here are sufficient to demonstrate that Druid well addresses the stated problems of interactivity.

4.2 Druid performance assessment by SK Telecom

SK Telecom also measured the query and ingestion latencies of Druid as detailed below:

4.2.1 Query latency test

The conditions of query latency measurement were as follows:

- Data: TPC-H 100G dataset (900 million rows)
- Pre-aggregation granularity: day
- Server: r3.4xlarge nodes, (2.5GHz * 16, 122G, 320G SSD) * 6
- No. of historical nodes: 6
- No. of broker nodes: 1

The query times for five queries of the TPC-H 100G dataset were as follows (the query times in Hive were also measured as a reference):

Figure 9: Druid and MySQL benchmarks – 100GB TPC-H data

Source: SK Telecom T-DE WIKI Metatron Project

* The reasons why the Hive benchmark performed poorly include that some processes were performed through Thrift and the dataset wasn't partitioned.

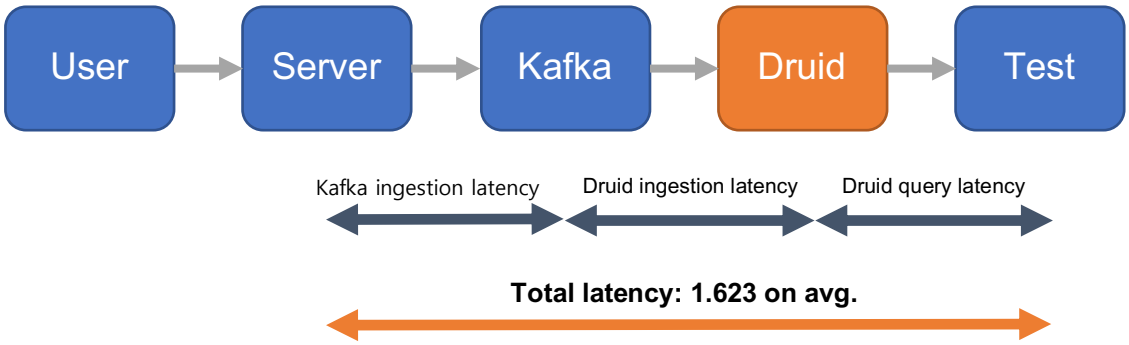
4.2.2 Ingestion latency test

The conditions of ingestion latency measurement were as follows:

- Ingestion data size: 30 million rows/day, 10 columns
- Memory: 512 GB
- CPU: Intel (R) Xeon (R) Gold 5120 CPU @ 2.20 GHz (56 cores)
- No. of historical nodes: 100
- No. of broker nodes: 2
- Jobs performed by three out of ten middle-manager nodes
- Ingestion tool: Apache Kafka

Data ingestion was performed 100 times under the conditions specified above, and the average ingestion latency was **1.623439 seconds**. As illustrated below, ingestion latency was computed as the sum of Kafka ingestion latency, Druid ingestion latency, and Druid query latency.

Figure 10: Architecture and latency measurement criteria for ingestion latency test



Source: SK Telecom T-DE WIKI Data기술원 metatron Project

4.3 Druid assessments by third parties

4.3.1 Druid assessment by Outlyer

In the Outlyer blog, twenty open source time-series database systems were assessed in a post[9] titled Top 10 Time Series Databases and published on August 26, 2016. The author Steven Acreman ranked Druid in the 8th place, and his set of criteria was as follows:

Table 4: A summary of Druid assessment by Outlyer

Items	Druid performance
Write performance - single node	25k metrics/sec Source: https://groups.google.com/forum/#!searchin/druid-user/benchmark%7Csort:relevance/druid-user/90BMCxz22Ko/73D8HidLCgAJ
Write performance - 5-node cluster	100k metrics / sec (calculated)
Query performance	Moderate
Maturity	Stable
Pro's	Good data model and cool set of analytics features. Mostly designed for fast queries over large batch loaded datasets which it's great at.
Con's	Painful to operate, not very fast write throughput. Real time ingestion is tricky to setup.

4.3.2 Druid assessment by DB-Engines

DB-Engines,[10] an online website, publishes a list of database management systems ranked by their current popularity each month. To measure the popularity of a system, it uses the following parameters:

- Number of mentions of the system on websites: It is measured as the number of results in queries of the search engines Google, Bing and Yandex.
- General interest in the system: For this measurement, the frequency of searches in Google Trends is used.

- Frequency of technical discussions about the system: The ranking list uses the number of related questions and the number of interested users on the well-known IT-related Q&A sites Stack Overflow and DBA Stack Exchange.
- Number of job offers, in which the system is mentioned: The ranking list uses the number of offers on the leading job search engines Indeed and Simply Hired.
- Number of profiles in professional networks, in which the system is mentioned: The ranking list uses the internationally most popular professional networks LinkedIn and Upwork.
- Relevance in social networks. The ranking list counts the number of Twitter tweets, in which the system is mentioned.

As of July 2018, Druid ranked 118th out of a total of 343 systems, and 7th out of 25 time-series database systems.

4.4 Comparison with Apache Spark

Comparing Druid with Apache Spark is meaningful because both technologies are emerging as next-generation solutions for large-scale analytics and their different advantages make them very complementary when combined together. SK Telecom's Metatron makes use of this combination: Druid as the data storage/processing engine and Spark as an advanced analytics module.

This section briefly introduces a report comparing the performance of Druid and Spark[7-8] published by Harish Butani, the founder of Sparkline Data Inc. Prior to the performance comparison, the report states that the two solutions are in complementary relations, rather than competitors.

4.4.1 Apache Spark characteristics

Apache Spark is an open-source cluster computing framework providing rich APIs in Java, Scala, Python, and R. Spark's programming model is used to build analytical solutions that combine SQL, machine learning, and graph processing. Spark supports powerful functions to process large-scale and/or complex data manipulation workflows, but it isn't necessarily optimized for interactive queries.

4.4.2 Dataset, queries, performance results

For the benchmark, the 10G TPC-H dataset was used. The 10G star schema was converted into a flattened (denormalized) transaction dataset and reorganized to be queryable in Druid and Spark. The sizes of the resulting datasets were:

- TPCH Flat TSV: 46.80GB
- Druid Index in HDFS: 17.04GB
- TPCH Flat Parquet: 11.38GB
- TPCH Flat Parquet Partition by Month: 11.56GB

And then, a number of queries were chosen to test the performance differences in various aspects as shown below:

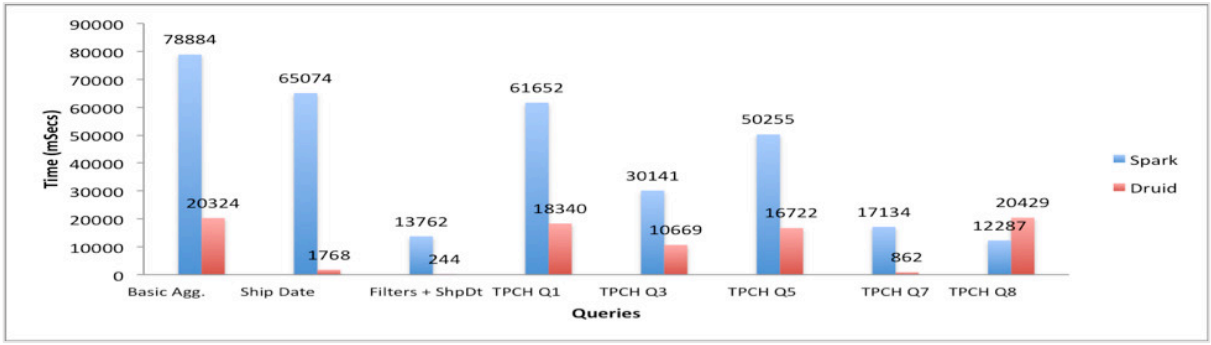
Table 5: Queries used for query latency comparison between Druid and Apache Spark

Query	Interval	Filters	Group By	Aggregations
Basic Aggregation.	None	None	ReturnFlag LineStatus	Count(*) Sum(exdPrice) Avg(avlQty)
Ship Date Range	1995-12/1997-09	None	ReturnFlag LineStatus	Count(*)
SubQry Nation, pType ShpDt Range	1995-12/1997-09	P_Type S_Nation + C_Nation	S_Nation	Count(*) Sum(exdPrice) Max(sCost) Avg(avlQty) Count(Distinct oKey)
TPCH Q1	None	None	ReturnFlag LineStatus	Count(*) Sum(exdPrice) Max(sCost) Avg(avlQty) Count(Distinct oKey)
TPCH Q3	1995-03-15-	O_Date MktSegment	OKey ODate ShipPri	Sum(exdPrice)
TPCH Q5	None	O_Date Region	S_Nation	Sum(exdPrice)
TPCH Q7	None	S_Nation + C_Nation	S_Nation C_Nation ShipDate.Year	Sum(exdPrice)
TPCH Q8	None	Region Type O_Date	ODate.Year	Sum(exdPrice)

Source: Combining Druid and Spark: Interactive and Flexible Analytics at Scale

The test results are as follows:

Figure 11: Query latency test results for Druid and Apache Spark



Source: Combining Druid and Spark: Interactive and Flexible Analytics at Scale

- The Filters + Ship Date query provides the greatest performance gain (over 50 times over Spark) when Druid is used. This is not surprising as this query is a typical slice-and-dice query tailor-made for Druid. Along the same lines, TPCH Q7 shows a significant performance boost when running on Druid: milliseconds on Druid vs. 10s of seconds on Spark.
- For TPCH Q3, Q5, and Q8 there is an improvement, but not to the same level as Q7. This is because the OrderDate predicate is translated to a JavaScript filter in Druid, which is significantly slower than a native Java filter.

- The Basic Aggregation and TPCH Q1 queries definitely show improvement. The Count-Distinct operation is translated to a cardinality aggregator in Druid, which is an approximate count. This is definitely an advantage for Druid, especially for large cardinality dimensions.

These results can vary with testing conditions, but one thing is clear: Queries that have time partitioning or dimensional predicates (like those commonly found in OLAP workflows) are significantly faster in Druid.

4.4.3 Implications

The testing results showcase that combining the analytic capabilities with Spark and the OLAP and low latency capabilities of Druid can create great synergy. Druid ingests, explores, filters, and aggregates data efficiently and interactively, while the rich programming APIs of Spark enable in-depth analytics. By leveraging these different capabilities, we can build a more powerful, flexible, and extremely low latency analytics solution.

5 Metatron powered by the Druid engine

5.1 Metatron development background and Druid integration

5.1.1 Metatron as a big data analytics solution

As a telecommunications service provider with the most number of subscribers in South Korea, SK Telecom has exerted significant efforts to establish a stable network environment through by using the mass amounts of network data logs generated by its users.

Due to the limitations of existing IT infrastructure in mass data processing, SK Telecom needed a big-data warehousing system (Apache Hadoop) and a big-data analytics solution compatible with the system. The company built its own Hadoop infrastructure to store mass amounts of data at low cost, but faced the following limitations:

First, network data generated by the countless users could not be analyzed in real time. Although it was possible to store and process big data, visualizations could be implemented only with a sampled subset of data in the same way as on legacy systems.

Second, having different solutions and different managers support each stage of data analytics, such as ETL, DW, and BI, not only involved significant time and costs, but also resulted in poor data accessibility. An end-to-end solution was needed to analyze all stages at once in a simple and quick manner.

5.1.2 Why the Druid engine

Druid was the optimal engine for the Metatron solution because it fulfilled the aforementioned needs

with the features below:

First, Druid collects mass amounts of data in real time and indexes them into a queryable format, ensuring very fast data aggregations (a few seconds at the slowest) based on distributed processing.

Second, Druid's OLAP time-series data format enables analysts to perform data exploration, filtering, and visualization as desired. Such free and flexible data exploration is essential for users to intuitively select the required data and determine correlations between different dimensions on it.

Third, Druid's extensible architecture allows modules to be easily added. Built on this architecture, Metatron is an end-to-end solution that embraces all layers of data collection, storage, processing, analysis, and visualization.

5.1.3 Druid engine integration

The Druid engine was integrated in Metatron as follows:

First, with Druid as the basic engine for processing/analytics, the GUI was designed to support users in different professional domains and big-data analysts in data-related tasks such as data preparation, analytics, and visualization, as well as the sharing of results.

Second, IT administrators can manage/monitor data sources in Druid, and they can establish data preparation rules if data sources of higher quality are required.

5.2 Druid functions reinforced in Metatron

The open-source Druid, despite its strengths in data collection and processing, had to be improved for Metatron to properly function as an end-to-end solution. This section examines the limitations of the open-source Druid and the functions reinforced in Metatron.

5.2.1 Limitations of the open-source Druid

The open-source Druid has the following limitations:

- Since Druid does not yet have full support for joins, Metatron uses another SQL engine for data preparation.
- Druid supports only a subset of SQL queries.
- For a data lake, a traditional SQL engine is more appropriate.
- Druid cannot append to or update already indexed segments, except for in some unusual cases.
- Nulls are not allowed.
- Filtering is not supported for metric columns.
- Linear scalability is not ensured. Increasing the number of servers doesn't improve the performance as much.
- Only a few data types are supported and it is difficult to add a new one.
- The management and monitoring tools are not powerful enough.

Druid functions reinforced in Metatron

The following functions of Druid were strengthened in Metatron:

Query functionality improvements

- Improved the functionality of the GroupBy query type.
- Slightly improved the functionality of other types of queries.

Features added

- Virtual columns (map, expression. etc.)
- New metric types (double, string, array, etc.)
- New expression functions
- Druid query results can be stored on the HDFS or exported into a file.
- Queries for meta information and statistics
- New aggregate functions (variance, correlation, etc.)
- (Limited) Window functions (lead, lar, running aggregations, etc.)
- (Limited) Joins
- (Limited) Sub-queries
- Temporary data sources
- Complex queries (data source summarization, correlation between data sources, k-means, etc.)
- Custom columns grouping
- Geographic information system (GIS) supported
- Columnar histograms
- Bit-slice indexing

Index structure improvements

- Histograms for filtering on metrics
- Lucene format supported for text filtering

Connectability with other systems

- Hive storage handler
- Ingestion into Hive tables (based on connection with the Hive metastore)
- Ingestion into the ORC format
- RDBMS data ingestion via based on JDBC
- (Limited) SQL support backported

Miscellaneous improvements

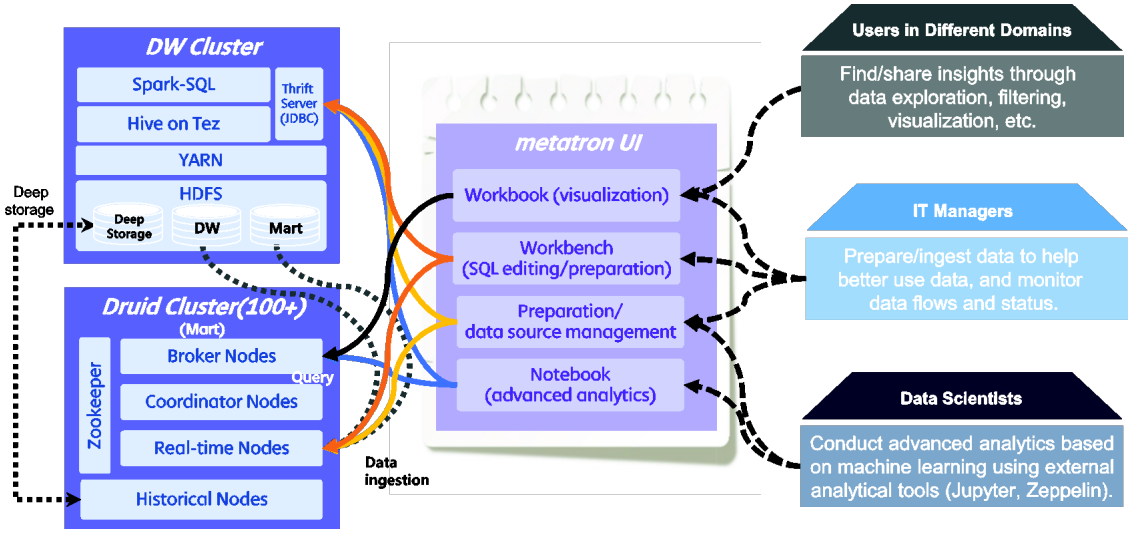
- Bug fixes (+50) and minor improvements

5.3 Metatron architecture

5.3.1 Basic architecture

The figure below shows the basic architecture of Metatron.

Figure 12: Basic Metatron architecture



As shown in the figure, the basic architecture deploys the Druid cluster described in Section 3.7. The functions accessible in the Metatron GUI include data ingestion and storage, query input and return, visualization, and connection to various advanced analytical tools. The interactions between these extended modules and node groups in the Druid cluster are as follows:

- **Real-time nodes:** Receives real-time inputs of various raw data. Capable of ingesting not only data from the HDFS and Thrift server in the DW cluster, but also data from external databases or files. Data can be ingested as received, or after preparation with the workbench or data pre-processing module embedded in Metatron.
- **Historical nodes:** Downloads data from deep storage in the DW cluster.
- **Broker nodes:** Receives queries from the Metatron UI.

The Metatron UI includes:

[Preparation modules for data ingestion]

- **Data preparation:** Pre-processes raw data to provide high-quality data for ingestion into Druid. (See Section 5.3.2.)
- **Workbench:** Supports data analysis/preparation in SQL prior to Druid ingestion.

[Data management/analytics/visualization modules for data ingested in Druid]

- **Data source management:** Manages and monitors data sources stored in Druid. (See Section 5.3.2)
- **Workbook:** Provides insights by exploring, filtering, and visualizing Druid data sources. (See Section 5.3.3)

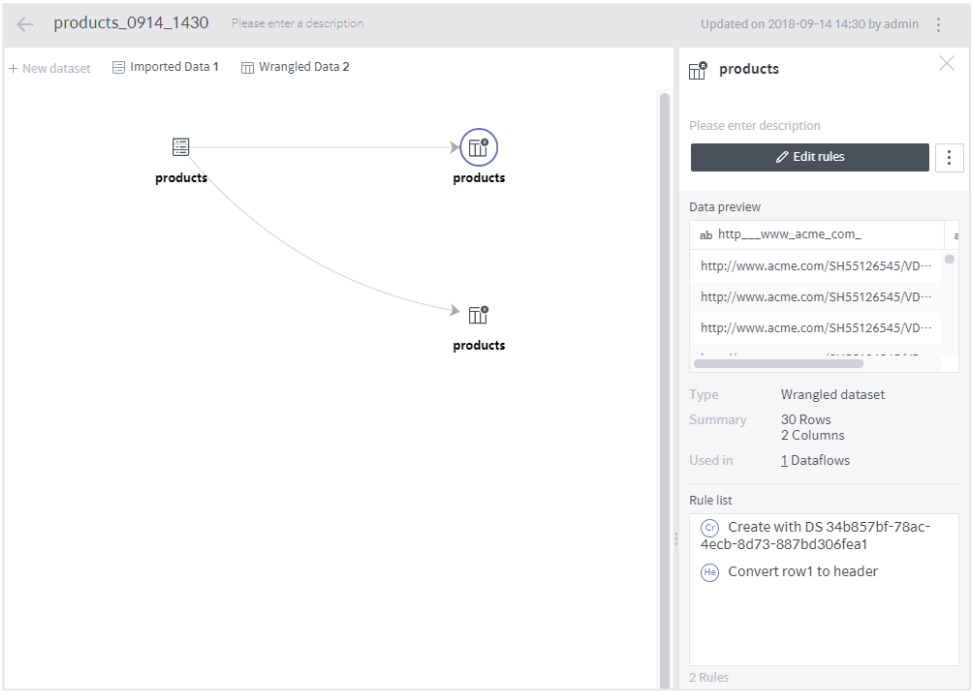
- **Notebook:** Retrieves Druid data via an external analytical tool (Jupyter or Zeppelin) and facilitates advanced analytics based on machine learning. (See Section 5.3.4.)

5.3.2 Druid data ingestion and management

Preparation for ingestion

Preparation helps to optimize data for analytics and visualization before it is ingested into the Druid engine. Users can conveniently set various preparation rules for each data stream, such that incoming data is automatically refined according to the rules.

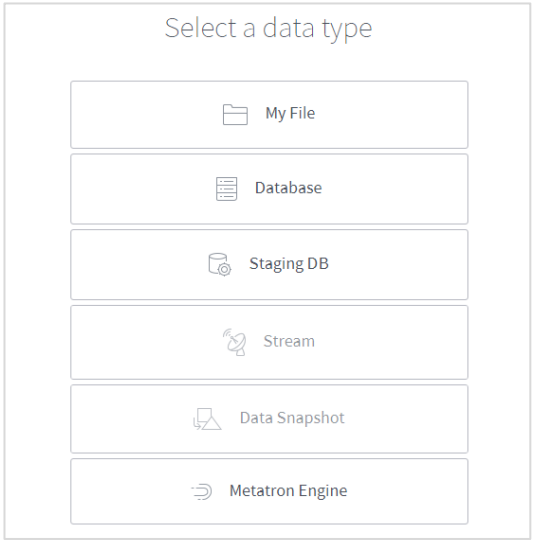
Figure 13: Data preparation in Metatron



Data ingestion into the Druid engine

The Druid engine generates data sources by ingesting batch data of various types and real-time event streams. Data sources are optimized for data analytics and visualization, and users can load them in a workbook or notebook.

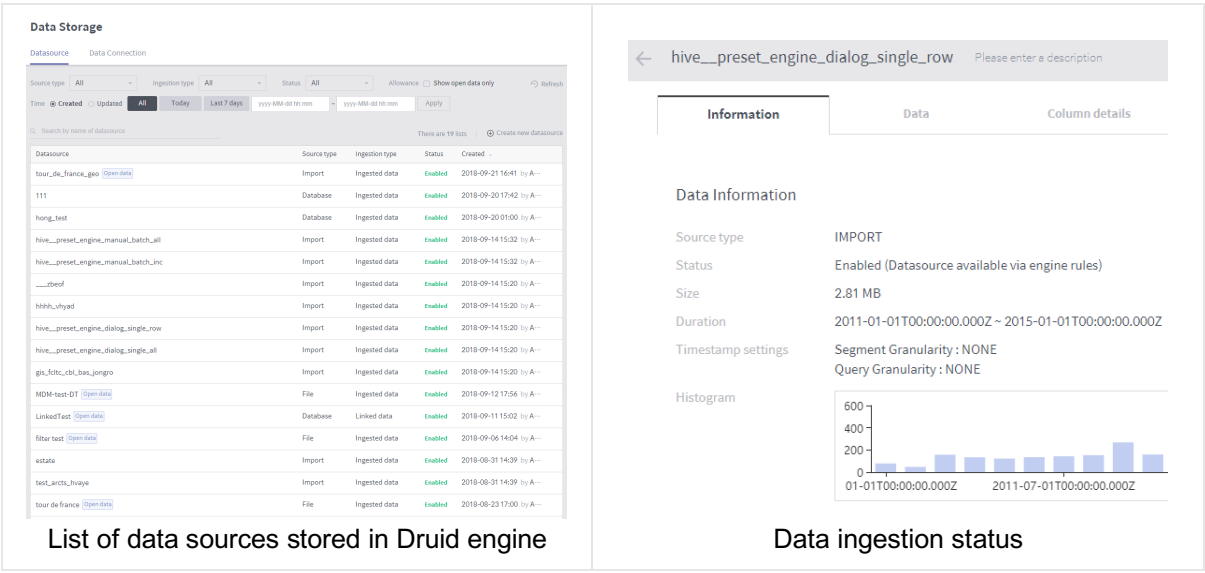
Figure 14: Selection of data type for ingestion in Metatron



View and management of data source information

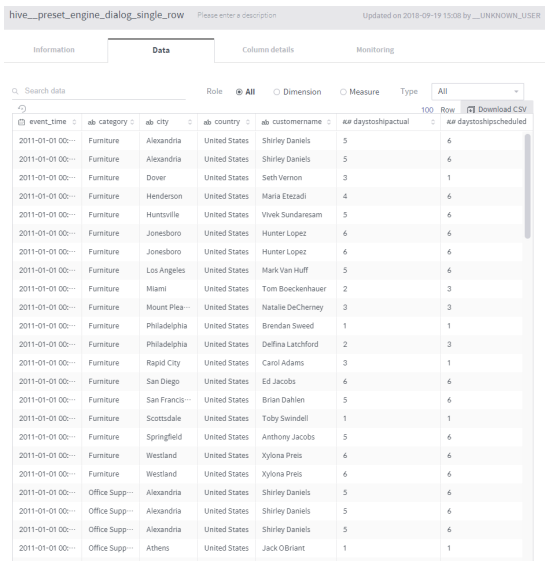
A data source's details and properties can be viewed, and data status monitoring is supported for IT managers.

Figure 15: Druid data source management in Metatron

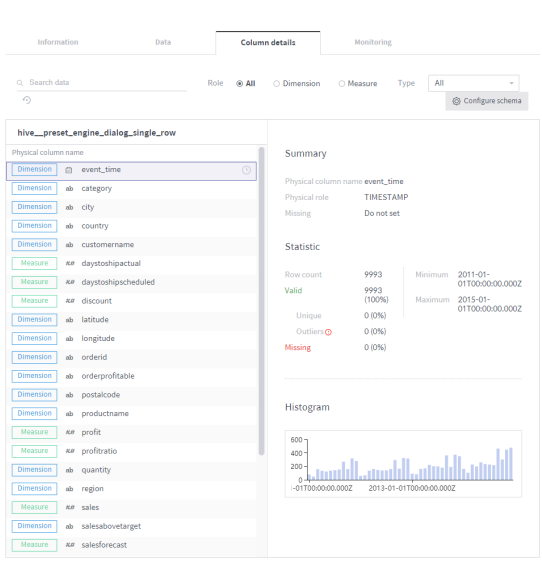


List of data sources stored in Druid engine

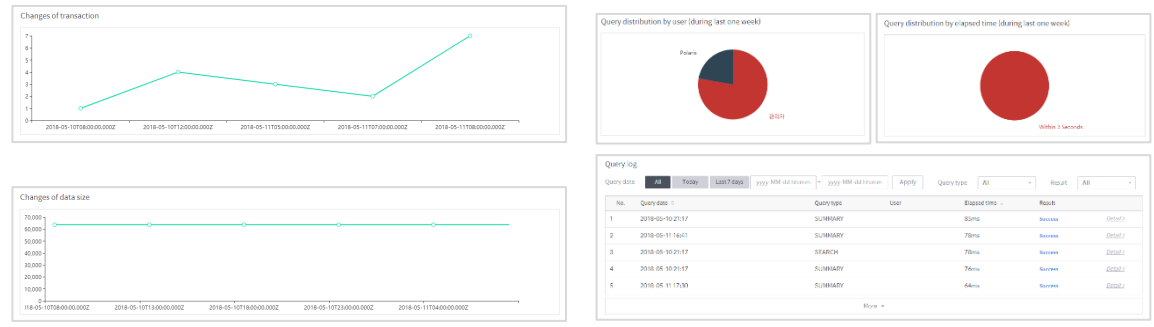
Data ingestion status



Data source details and search



Viewing and modification of metadata



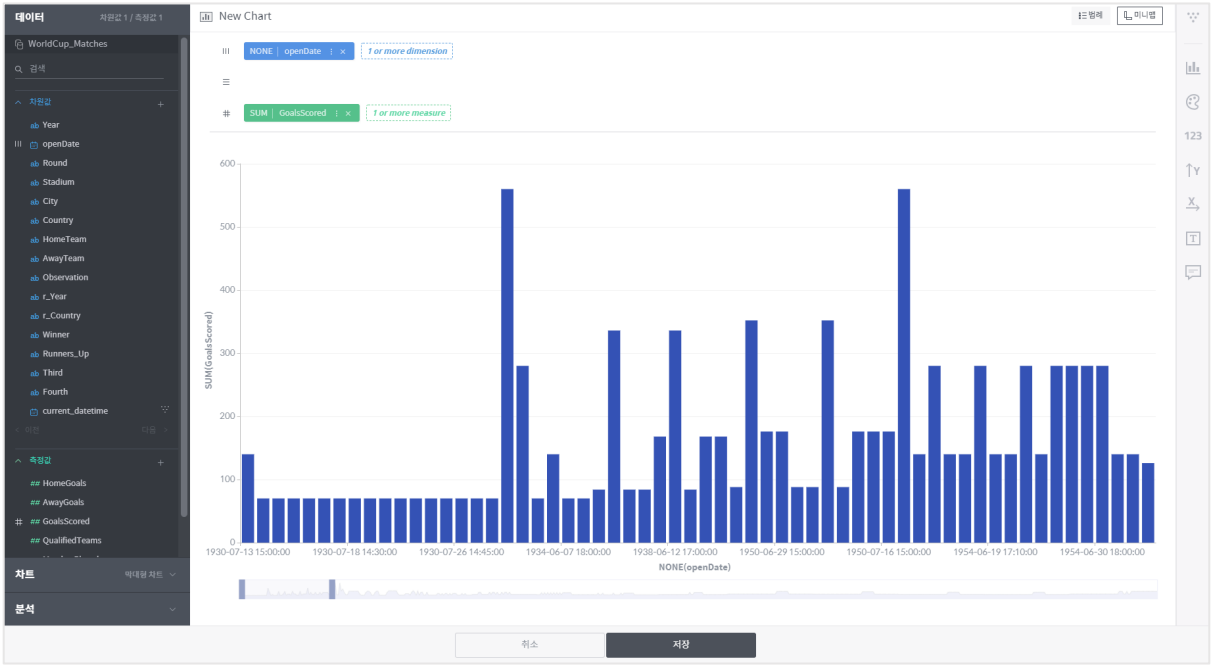
Monitoring of log containing details on data source use

5.3.3 Workbook: Data visualization and PPT UX-based dashboards

Chart creation

The Metatron UI allows users to create charts easily by dragging and dropping columns of data sources to pivot them. Depending on what data columns are selected, suitable chart types are recommended.

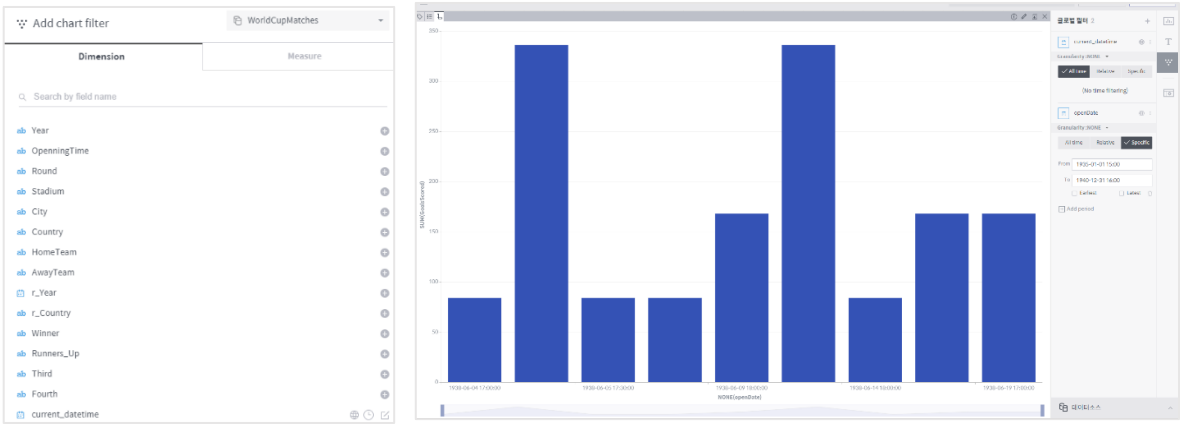
Figure 16: Chart creation in Metatron



Filtering

Data values in the column of choice can be filtered to show a specific range in the chart. Metatron supports filtering of metrics, which is not possible in the existing open-source Druid. The figure below shows the chart with a time filter (years 1935–1940) applied.

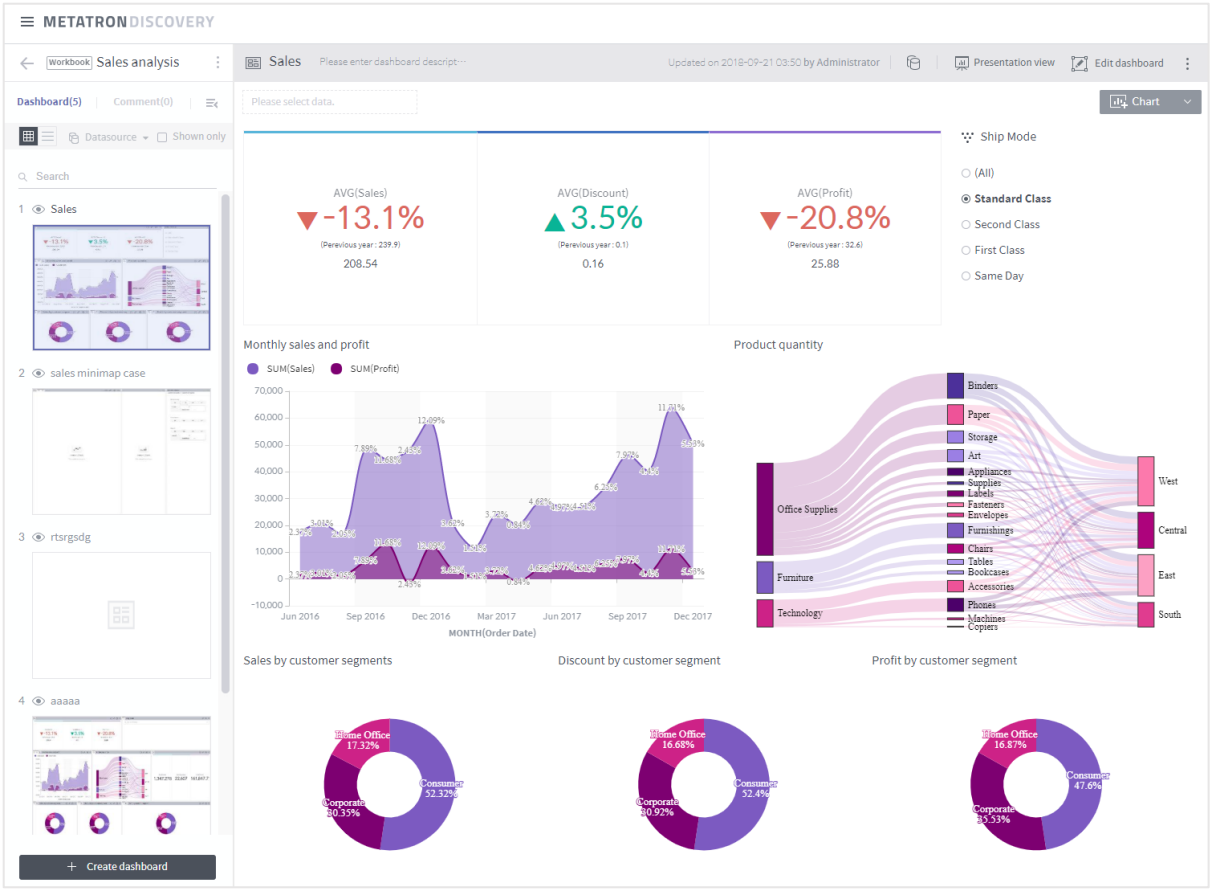
Figure 17: Result of chart filtering in Metatron



PPT UX-based dashboard

Charts are displayed in a user-friendly PPT UX, and can be integrated in various reports and presentations.

Figure 18: Metatron dashboard



5.3.4 Notebook: Advanced analytics using external modules

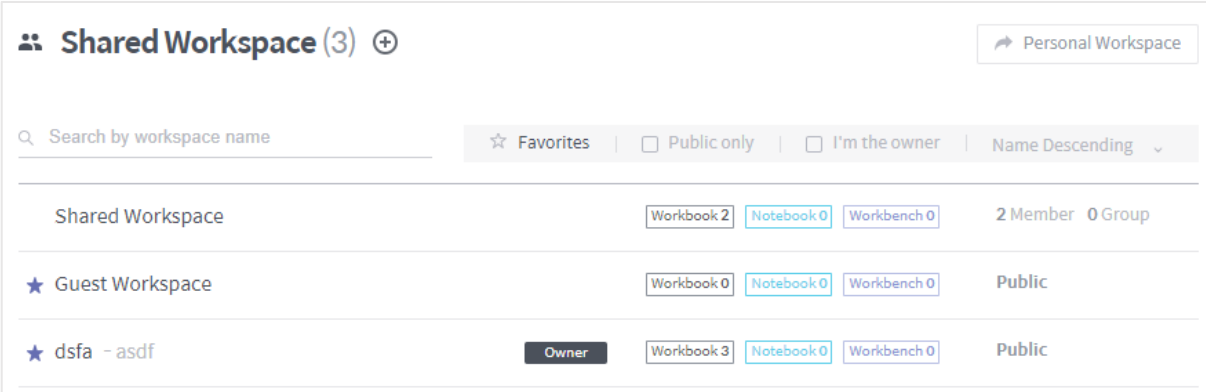
External modules can be used to perform advanced analytics on data sources stored in Druid or results obtained from queries based on machine learning. The compatible external analytics tools (and languages) are Jupyter (R, Python) and Zepplin (Spark).

Figure 19: Coding with an external analytics tool in Metatron

5.3.5 Sharing functionality

Metatron facilitates collaboration by enabling members of your organization to share the results of data ingestion, analytics, and visualization with one another.

Figure 20: Shared workspace list in Metatron



5.4 Applications

Using the sub-second multidimensional column-based feature of the Druid engine, Metatron supports various analytical functions that were not available in legacy systems. This section introduces two key applications.

5.4.1 Improved quality of mobile telecommunications services

SK Telecom successfully improved its service competitiveness through the implementation of Metatron.

With the proliferation of smartphones and establishment of high-speed wireless data communications infrastructure in recent years, there has been a noticeable difference in the usage pattern of mobile telecommunications services, namely, an exponential increase in data communications. Subscribers prioritized fast and stable data communications, and this became a critical factor in determining customer satisfaction.

However, SK Telecom faced many constraints in assessing customer experience, especially considering the unexpected change in usage patterns.

Various datasets related to subscribers' use of telecommunications services were being stored separately in more than 30 silo systems, and the daily amount collected was far too large (50 billion rows per day, 50TB) for analysis. This issue could be resolved by establishing a Hadoop cluster to unify the data.

Data, even if unified, cannot be utilized in the raw state as it is associated with too many indicators related to service quality. As such, SK Telecom integrated the various indicators under the Customer Experience Index (CEI). This index measures the level of inconvenience experienced by subscribers in using mobile devices on a scale of 0–100.

To calculate the CEI scores, more than 100 workflows are executed through various ETL tools (Spark, Hive, MapReduce). The data stores (ODS, DW, and DM) each perform different roles. The calculated CEI scores are sent to regional offices every two years to be compared with the actual experiences of

users, and corrected if necessary.

When this simplified CEI data accumulates, however, it becomes too vast to be analyzed in Excel or other legacy tools. This necessitates a BI solution that processes the mass amounts of data and facilitates visualization/sharing of the results.

Against this backdrop, the Druid-based Metatron was developed. It is not only capable of rapidly analyzing and visualizing CEI data, but also provides a convenient GUI for non-expert users to perform related tasks according to their needs and share results across departments. Below are the results of analyzing CEI in Metatron by device type, region, and date.

Figure 21: CEI-related charts generated in Metatron

Source: Big Data: The best way to truly understand customers in Telco

The implementation of the Metatron system has significantly enhanced customer satisfaction. The company was able to respond effectively to complaints through a multidimensional assessment of user experience. The customer satisfaction rating, which fell in the early days of LTE communications, has steadily increased since rebounding in late 2013.

5.4.2 Tracking of cause of product defects

Data extracted from thousands of equipment in the semiconductor and LCD industries are essential for defect tracking and yield improvement. However, legacy systems are unable to store or analyze all data records because of their massive size and varying data types by processing facility.

Metatron, on the other hand, stores all generated data in their original forms regardless of data type, pre-processes each data stream to fit the purpose of analytics, and ingests pre-processed data into Druid data sources. Operations staff can visualize and analyze these data sources from various angles on their own, thereby easily tracking the causes of defects. Moreover, the immediate return of results for analysis, aggregation, and visualization queries significantly enhances working efficiency.

6 When alternatives to Druid may be considered

Druid may not in all circumstances be the perfect solution for big data analytics. Alternatives to Druid may be considered if an organization does not require real-time analytics on mass data, interactive calculations for aggregation queries, or an effective redundant system for high availability. Some cases in which alternatives may be more useful are given below:

- The amount of data can easily be handled by RDBMS.
- Individual events have more significance than analyzing them.
- Data updates and deletes are constantly needed.
- It is enough to ingest data in batch.
- Downtime does not significantly matter.

7 Conclusion

This white paper briefly introduced Druid, a distributed column-oriented data store for real-time analytics, and Metatron, a solution built on the Druid engine.

Through its unique data storage method and distributed cluster, Druid enables real-time ingestion of mass data and sub-second query processing. It supports flexible, stable system operations with its extensible architecture and high fault-tolerance.

Metatron was developed such that any queries made to Druid can be entered and returned through the GUI. It is an end-to-end solution that improves upon Druid while offering the same advantages, and caters to the increasing need for big data analytics. Metatron allows users across industries to find and share insights on data in their respective professional domains, and helps to reduce expenses on data-related tasks that used to be incompletely handled at much higher costs in legacy systems.

8 References

[1] Druid website. http://druid.io, July 2018.

[2] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli. (2014). *Druid: a real-time analytical data store*. Retrieved from http://druid.io/docs/0.12.1/design/index.html

[3] Hakka Labs. (2014, Jan 7). *MetaMarkets - Introduction to Druid by Fangjin Yang* [Video file]. Retrieved from https://www.youtube.com/watch?v=hgmxVPx4vVw&t=1852s

[4] Hakka Labs. (2017, Jul 21). *Interactive Exploratory Analytics with Druid | DataEngConf SF '17* [Video file]. Retrieved from https://www.youtube.com/watch?v=rbQaCazQ0gI&t=1395s

[5] Metamarkets. (2012, Dec 10). *Beyond Hadoop: Fast Ad-Hoc Queries on Big Data* [Video file]. Retrieved from https://www.youtube.com/watch?v=eCbXoGSyHbg

[6] Strange Loop. (2016, Sep 17). *"Druid: Powering Interactive Data Applications at Scale" by Fangjin Yang*. Retrieved from https://www.youtube.com/watch?v=vbH8E0nH2Nw

[7] Harish Butani. (2018, Sep 18). *Combining Druid and Spark: Interactive and Flexible Analytics at Scale*. Retrieved from https://www.linkedin.com/pulse/combining-druid-spark-interactive-flexible-analytics-scale-butani

[8] Harish Butani. (2015, Aug 28). *TPCH Benchmark*. Retrieved from https://github.com/SparklineData/spark-druid-olap/blob/master/docs/benchmark/BenchMarkDetails.pdf
[9] Steven Acreman. (2016, Aug 26). *Top 10 Time Series Databases*. Retrieved from https://blog.outlyer.com/top10-open-source-time-series-databases.
[10] DB-Engines website. https://db-engines.com, July 2018.
[11] metatron website. https://metatron.app, July 2018.
[12] SK Telecom. (2016. Apr 9). *Metatron Architecture Overview*.
[13] SK Telecom. Metatron 제품 소개서.
[14] SK Telecom. (2018, Jan 25). Metatron 2.0 사용자가이드.
[15] SK Telecom. (2018, May 16). Metatron 2.0 Discovery User Manual.
[16] SK Telecom. Functions added in SKT Druid (2017, November).
[17] 황성필. 빅데이터 디스커버리 솔루션 개발 및 사례. Tech planet 2016, Seoul. Retrieved from http://readme.skplanet.com/wp-content/uploads/%ED%8A%B8%EB%9E%991-1.%EB%B9%85%EB%8D%B0%EC%9D%B4%ED%84%B0-%EB%94%94%EC%8A%A4%EC%BB%A4%EB%B2%84%EB%A6%AC-%EC%86%94%EB%A3%A8%EC%85%98-%EA%B0%9C%EB%B0%9C-%EB%B0%8F-%EC%82%AC%EB%A1%80.pdf
[18] Kyungtaak Noh. Big Data: The best way to truly understand customers in Telco. Strata Data Conference 2017, Singapore.