MID-TIER MODELS FOR BIG DATA

Jayesh Patel, Senior Member, IEEE

Abstract: With the rise of Big data, enterprises started accumulating significantly more data than they consume. Big Data lake made data consumption easier for all stakeholders, analysts, and developers. Variety, volume, and velocity of data and complexity of businesses added complexity in processing, organizing, and storing data to serve analytical solutions on a timely basis. It is often a big challenge for enterprises to cleanse, organize, classify, and store big data so that insights are accessible on time. Data consistency will also come into the picture when multiple data models define similar metrics. As numerous data sources are integrated into a single platform, stakeholders often analyze data from various subject areas. It leads to complex queries resulting in big joins and more processing power. Even with cheap storage and more processing power of Hadoop and big data technologies, modeling big data is a time-consuming and error-prone process. This paper addresses that challenge by introducing mid-tier models for big data. It discusses a novel data modeling-Mid-Tier models approach to organize and store big data in distributed storage. It outlines how it overcomes some of the challenges and showcases an example.

***Index Terms*—Enterprise Data Models, Mid-Tier Data Models, Big Data Lake, Dimensional Models, Big Joins, Hadoop, Spark**

I. Introduction

In this big data age, enterprise applications generate a large volume of structured, unstructured, and semi-structured data. These data are processed and transformed to generate insights which help to make strategic and tactical business decisions. Business users also use ad hoc analysis to identify patterns and predict the future. Enterprise data models store all useful insights from various data sources in a structured format. An enterprise big data lake contains several enterprise data models to facilitate data-driven decision support system for various stakeholders [2][10]. There are multiple challenges that business users face while querying an enormous volume of processed data from enterprise data models. On the other side, it is a very ambitious task to keep up with all changes and provide metrics on time.

Business users consume data from enterprise data models. Due to data volume, data consistency and security, raw data may not be available for them to access directly. Enterprise Data models are designed and built using raw data to serve their needs. Most times, they use data from multiple models for decision making. In a big data environment, these models are enormous. Joining large tables is a costly operation and result in higher response time. For instance, analyzing users' behavior on a social media platform involves joining millions of users with various activity facts to understand a full picture for a specified time period. Even with big data tools and technologies, this operation is slow and costly.

Additionally, when things go wrong, reprocessing or backfilling affected metrics takes even more time and resources. As most big data lakes on HDFS support insert and append operations, updating the existing model is not straightforward. When you add new metrics and want to backfill only those metrics, it requires rewriting files with all metrics. These existing methods are very inefficient and time-consuming. Due to cheap storage and more processing power, big data lake often contains multiple data models with redundant metrics and similar insights. This leads to data inconsistency, and time wastage on troubleshooting. There are several other issues with current data modeling techniques in big data platforms.

Clearly, enterprises need a better data modeling for big data platforms to overcome these challenges. This paper introduces the concept of mid-tier models which address these issues that enterprises face while using big data. Mid-tier models not only overcome these challenges but it also provides unified analytics from multiple business processes. Faster processing and easy maintenance are integral to these models.

The organization of this paper is as follows. Section II presents findings from relevant research in Big Data

[1]This paper is submitted to IEEE DataCom 2019 on 05/29/2019.
Jayesh Patel currently works as a Sr. Data Engineer for Rockstar Games, Carlsbad, CA 92008 USA and he is also a Senior IEEE member (e-mail: jayesh423@yahoo.com).

Modeling. Section III describes data modeling and challenges with them in a big data environment. Section IV explains mid-tier data models and provides details on how to design it.. Section V demonstrates experiments and results of mid-tier data models. Section VI concludes this paper with the relevant remarks and future scope.

II. Related Works

Data modeling, Enterprise data warehouse, and Big Data processing are not new as there have been new technologies and platforms offering an effective way of processing and modeling big data. Over the period of time, multiple research and methodologies have been proposed.

R. Kimball, one of the original architects of data warehousing [12], presented various use cases of big data analytics, challenges to model big data and outlined the necessary design elements to extend EDW to big data analytics. In [13], various techniques to integrate big data, and to process different types of data sources are summarized. This research further enumerated opportunities for better big data modeling techniques to address the challenge of deep analysis on big data.

A. Gandomi, and M. Haider [14] reviewed big data analytical techniques for structured and unstructured data. This research stated the needs of major innovations in analytical modeling techniques for massive volumes of less trustworthy big data. Analysis on Big Data analytical methods [15] asserted a critical challenge for outdated data processing and modeling space for big data. Old methods of data modeling no longer apply toward big data processing. There is a need for new methods to manage big data for getting maximum value. It grouped big data modeling methods in three groups- descriptive models, predictive models and prescriptive models based on the review of over 100 research papers.

A. Sebaa, F. Chick, A.Nouicer, and A. Tari [16] stated that traditional modeling techniques and tools don’t scale up due to processing complexities and limitations of underlying hardware and infrastructure. Authors further reviewed big data warehousing in Hadoop and different use cases with their limitations.

A hybrid approach based on data filtering and processing [17] is proposed to build an effective dimensional big data model. Proposed method is described by three points: data mart oriented, filtering based, and hybrid ETL/MapReduce implemented. As data processing with MapReduce is slower, there is a need for something better.

Recently, M. Golfarelli and S. Rizzi [18] summarized more than 20 years of research on data warehousing applications and modeling. Different architectures and methodologies to process big data are summarized.

III. Data Modeling

A. *Data Models*

At any time, enterprises generate more data than they consume. Due to volume, variety, and structure, it is difficult for end users to use data as is. Raw data need processing and transformation to fit in a business context. After applying fundamental business rules, standardize key metrics are generated to track the performance of business processes. This process of managing a vast amount of data is known as data modeling [2][3]. Data models represent the intricacies of the business process in the enterprise, and they are widely used medium to communicate with stakeholders effectively. Hence, enterprise data models should be designed wisely.

For instance, books in a library are correctly classified and arranged on shelves in the proper order to assure easy access to each book. Analogously, for a large volume of big data, we need a suitable method or system to sort and store data. Readers may not read all the books in the library, but they refer to the books of their interest. Likewise, business users will only use metrics and data that help them in making wise decisions.

B. *Dimensional Models*

Dimensional model, introduced by Ralph Kimball, is a unique data model optimized for data analytics [1]. It comprises of facts from business processes and contextual information about the business process known as dimensions. One of the fundamental concepts of this model is to define the most granular attributes in the business process. It represents the leaf level facilitating summarized and detailed view of the same data. The granularity of a dimension model is a combination of dimensions. Star model and snowflake model are widely used dimensional models in data warehouses and big data lakes.

An idea behind the dimensional model is to denormalize at a certain level to avoid joins at a later stage. It works perfectly fine for a specific business process to have all required tables denormalized. However, problems arise when you want to report metrics on different business processes or to analyze different business processes at the same time. Business users end up joining multiple fact tables to complete the desired analysis. It is not recommended to join numerous fact tables in an Enterprise data warehouse or data lake [1][4][5]. Additionally, dimensional models works really well with slicing and dicing of data. However, big data usage is not limited to slicing and dicing. With distributed and democratic usage patterns, dimensional models fall short to serve analytical needs on time.

There are several other types of modeling techniques but they fall short in several ways for big data platforms. This paper will not go in detail for those modeling techniques.

IV. Mid-Tier Models

A. *Designing Mid-Tier Models*

Mid-tier model is inspired from the dimensional model, and it implements a functionality which is shared across multiple business processes and reporting needs. Mid-tier model is a pre-computed collection of attributes and metrics on big data with predefined granularity. It shifts the business focus from process-centric to data-centric. The goal is to keep all metrics with the same granularity from multiple processes in a single model so that reporting and analysts queries do not need to repeat the same expensive scans and join on raw data and other models. If the granularity of dimensions changes, it will be a different model.

In case of dimensional models, there are multiple fact tables, each depicting specific process or reporting needs. However, the needs with big data shift towards exploratory analysis and more democratic usage. Fig. 1 shows an example of a mid-tier model built using data from multiple processes and sources. To simplify the example, three sources are considered for medical practice use case. This mid-tier model is designed to build provider level summary with metrics from claim processing, patient visits and appointments. Grain for this model is provider_id and date_key. Metrics from multiple processes with this grain is stored in this model.

This mid-tier model can be built by using raw data sources instead of using dimensional facts. Dimensional facts are shown in this example to simplify the explanation.

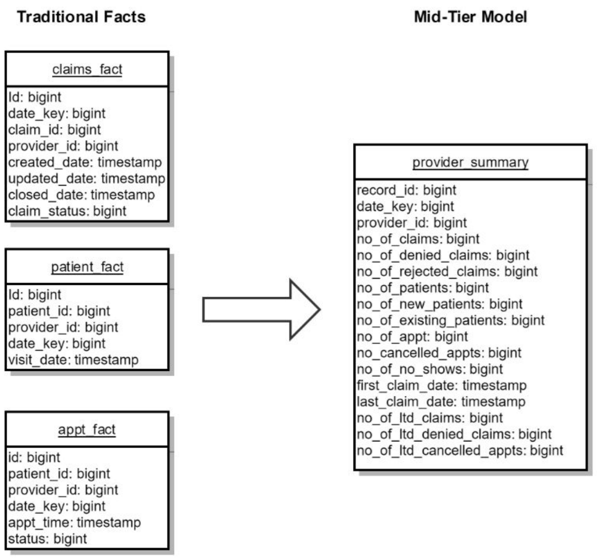


Fig 1. Mid-Tier model showing metrics from multiple processes

Mid-tier model is also a data structure technique based on divide and conquer principle to aggregate and store data [11]. It is expensive to join one month of insurance claims data with millions of users and providers. Instead, processing these transactional sources by day or hour will perform much faster and economical. Mid-tier model is based on that fact that processing smaller batches of data will be quick and efficient. Even for a large volume of daily data, this process can be broken down further on either time batches or other batches to speed up processing [7].

With big data, volume can affect data model processing. It may be a time consuming operation when you join multiple transactional sources daily. Mid-Tier model suggest using smaller batches to break processing down. Fig. 2 shows an example of a batched mid-tier model. Batchid can be derived from a timestamp value or any other value identifying the batch of records. It uses claims transactions and patient appointment data. For each batch, it calculates the number of claims and appointments by providers. It also stores a date key to identify transaction date. It requires a control mechanism to keep track of completed batches to avoid data loss.

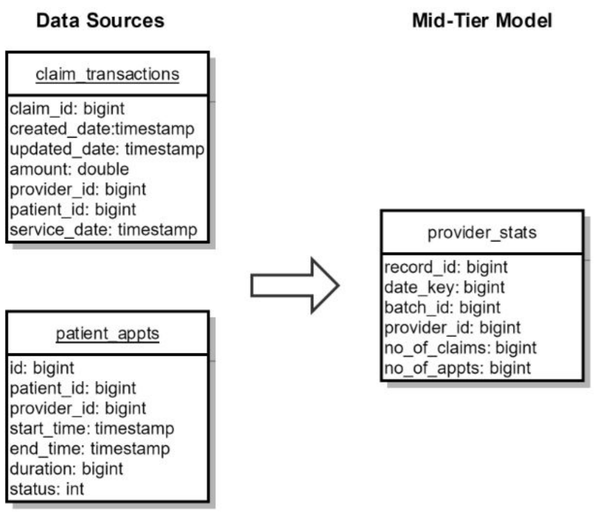


Fig 2. Mid-Tier model showing batch processing

Mid-tier model also proposes using complex types to join multiple transactions into one table to avoid big joins. For the same granularity, multiple transactions with different attributes can be stored in a consolidated mid-tier model. For example, claim_ transactions and patient_appts can be merged in a transactional mid-tier if user needs transactional level details. Fig. 3 shows a graphical representation of a transactional mid-tier model. As transactions at provider level are already joined, no further joins are required.

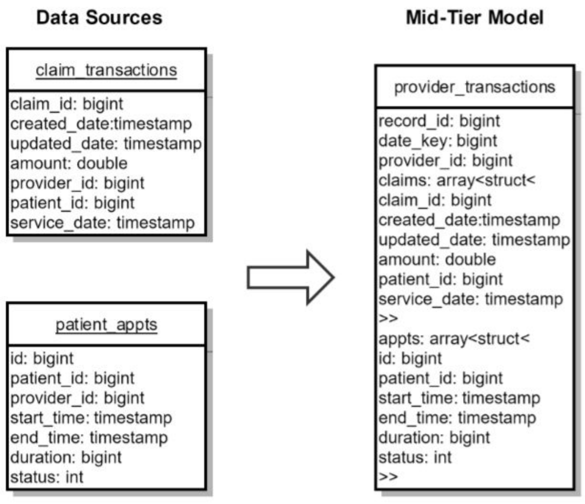


Fig 3. Transactional Mid-Tier

For big data models stored in Hadoop, the biggest question is for storage. As storage on HDFS is immutable, only insert and append operations are permissible with HDFS. For mid-tier model, it is recommended to use mutable storage for models requiring full table scans. Mutable storage such as Kudu and Cassandra supports upserts and update operations. Updates are essential to backfill specific metrics or to add new metrics at a later stage. Update operation will make maintenance easier and it is well supported in the iterative and fast development cycle. For example, after a week of model going live in production, an issue is reported in business logic, and there is a need to fix it and backfill affected metrics. It will be much easier to refresh the impacted metrics using new business logic than regenerating entire model again. Apache Spark, Kudu, and other big data technologies make it possible for a big data platform [6][8][9].

There are several use cases where mid-tier model with mutable storage provides best results. One such use case is processing life-to-date metrics or life-to-date (ltd) values. As life-to-date calculation requires full table scan, it is a widely known problem with big data. Usually source tables or models are scanned fully and then aggregated to calculate ltd metrics. It becomes a serious issue when ltd metrics are calculated from multiple tables and models. That is not a viable solution when dealing with big data. Mid-tier model handles that issue really well using mutable storage on hdfs. Fig. 1 shows ltd metris in a mid-tier model. no_of_ltd_claims shows total number of claims created since the beginning. Mid-tier model calculates it by using last known no_of_ltd_claims. It adds no_of_claims for the most recent day and last known no_of_ltd_claims to generate no_of_ltd_claims for the most recent day. Mutable storage allows breaking that processing down to calculate daily metric first and then use it to calculate ltd value.

Mid-tier model can be processed in multiple iterations based on metrics and facts. First iteration should handle all attributes and metrics that are trivial to source from a raw data. For example, metric “no_of_submitted_claims” is simply the value of “count(claim_id)” from the source table. The first iteration can be further broken down by source so that multiple sources can be updated independently. For instance, metrics from claims data be loaded in one iteration and metrics from appointment data be loaded in other iteration. Subsequent iteration handle derivative attributes and metrics. For example, "ltd_submitted_claims" is the cumulative sum of "no_of_submitted_claims", which was calculated in the first iteration.

Additionally, iterations can be used to process one source at a time. For example, metrics from claims transactions can be calculated and stored in first iteration. In the second iteration, metrics from appointments data can be calculated and so on.

While designing mid-tier model, the number of iterations should be smaller for practical reasons. Based on the research and testing with Spark and Kudu, 5 or less iterations will be optimal. More iterations may lead to performance issues.

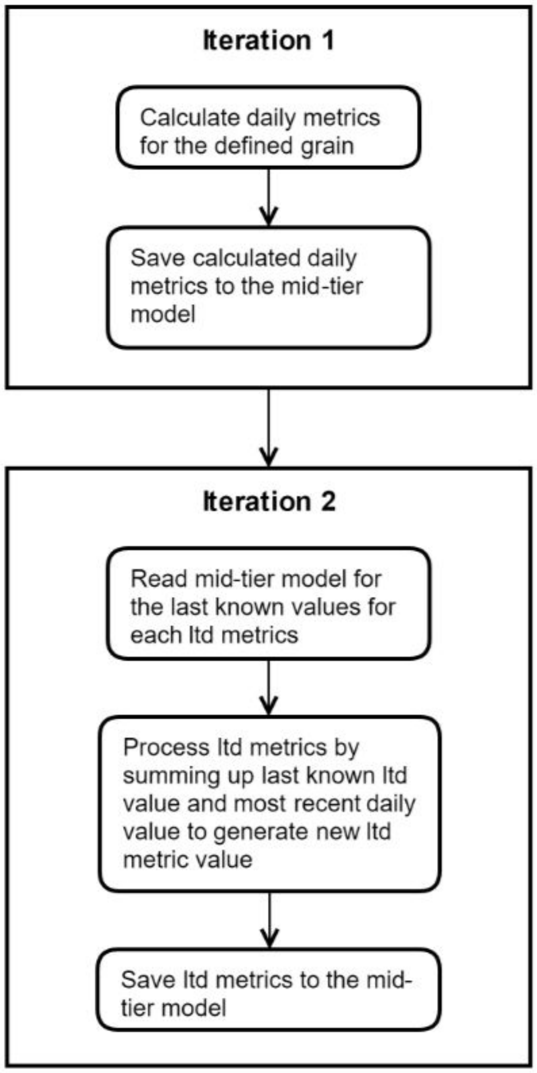

Iteration 1
Calculate daily metrics for the defined grain
Save calculated daily metrics to the mid-tier model
Iteration 2
Read mid-tier model for the last known values for each ltd metrics
Process ltd metrics by summing up last known ltd value and most recent daily value to generate new ltd metric value
Save ltd metrics to the mid-tier model

Fig 4. Iterative processing in Mid-Tier Model

B. Benefits of Using Mid-Tier Models

Mid-tier model helps solving several problems with big data models. Here are some of the common issues and how mid-tier model resolves them:

1) Big Joins: It is expensive to join multiple fact tables or models in big data platforms due to data volume and complexity of business rules. It leads to big joins. With mid-tier model, big joins are broken into smaller ones with more frequency and parallel computing. Additionally, metrics and data for the same granularity will not be repeated in different models. As a result, data is consolidated into fewer models. For instance, to calculate life to date (LTD) values, you don't need to scan the entire table or join multiple tables to calculate them. Processing is broken down to update LTD metrics on a daily or more frequent basis in the mid-tier model. Hence, they are readily available to use from mid-tier models.
2) Backfilling: Backfilling is one of the pressing issues for enterprises. It may be required for various reasons such as operational issues, late arriving data, hardware failures, etc. Either you will reprocess the entire model or re-run it for affected sources. With mid-tier, it is possible to re-run model only for affected data sources as it supports upserts in the model. It is less time-consuming and has less overhead.
3) Maintenance: As things change over a period of time, the model needs to be updated accordingly. There may be a requirement to either to add new metrics in the existing model or to update business rules for existing metrics. Mid-tier model in Kudu offers easy ways to add and update columns in tables [6][8]. You can run custom scripts to update affected metrics one time and it is done.
4) Ease of use: If all metrics you want for your machine learning pipeline or for your analysis are stored in fewer models, remaining steps become easier. Responsive queries and faster responses will make mid-tier models the first choice to use for various purposes. Additionally, it speeds up the development of downstream models by reducing the RAM/disk/CPU resources required to build the expensive models.

V. Example

Cloudera QuickStart VM was used to set up the environment for experimentation and testing. VMWare Image was used with 16GB of memory. Data from CSV file was loaded to HDFS in a columnar Parquet format. Apache Spark, Impala, Hive, and Kudu were used for experimentation.

To validate the mid-tier model and run some experiments, Iowa liquor sales data was used. The Iowa Department of Commerce necessitates that every store that sells alcohol in bottled form for off-the-premises consumption must hold a class 'E' license. Hence, all alcoholic sales made by stores registered with the Iowa Department of Commerce are logged in the departmental system. This data set is published as open data by the State of Iowa. Iowa liquor sales data contains information on the name, kind, price, quantity, and location of sales transactions of individual containers or packages of containers of alcoholic beverages from January 1, 2012 to October 31, 2017. This denormalized dataset has over 16 million rows with 3.3 GB in total size.

The mid-tier model was built using this dataset to illustrate some of the design methodology proposed in this paper. Comparison with traditional EDW modeling is also discussed. Mid-tier model to analyze various metrics by date, store and item is prepared. A similar model is also prepared with dimensional modeling methods. The first use case tested was daily and life-to-date metrics.

Dimensional model for daily and LTD metrics is designed as two models- daily_sales_fact and ltd_sales_fact. Due to immutable file system, processing complexity, easy maintenance and extensibility, ltd metrics are stored in a separate model- ltd-sales_fact. Both models are partitioned by date and stored in immutable parquet format on HDFS. Fig. 5 shows the schema of these two models. Daily process should add a partition in both models calculating respective metrics. daily_sales_fact is populated by aggregating daily transactions. ltd_sales_fact is calculated by using daily_sales_fact and prior ltd metrics. The downside in this approach is a costly join when using daily and ltd metrics for the analysis.

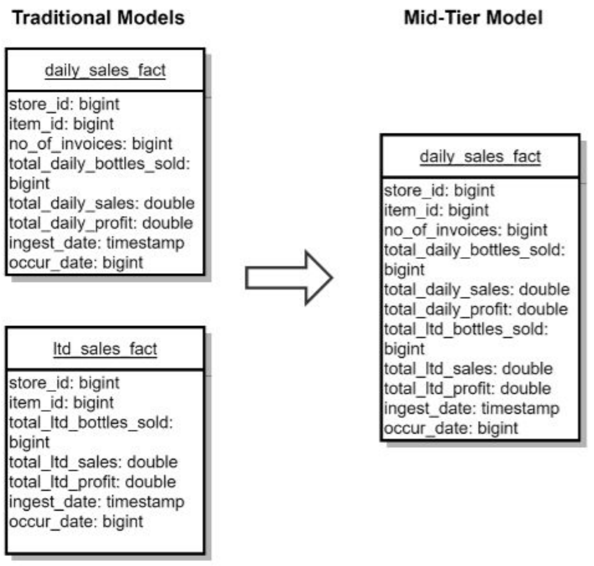
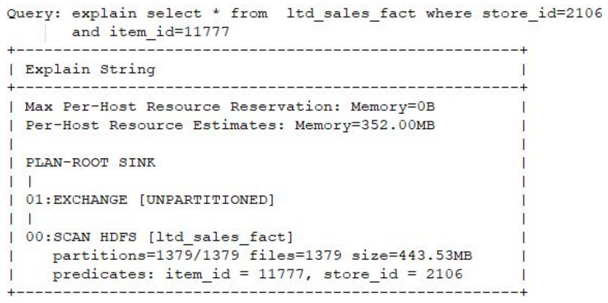


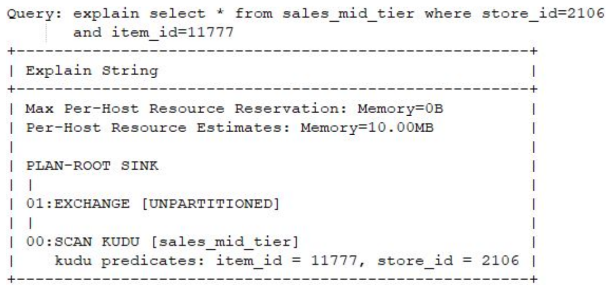
Fig 5. Traditional vs Mid-Tier Models

Mid-Tier model is proposed combining daily and ltd metrics in the same model using mutable storage- Kudu. As Kudu model has primary keys on store_id and item_id, queries were faster than dimensional model. Sample queries with mid-tier models were completed 5 times faster than running them on joining dimensional models. To present the memory consumption, query plans are shown for both cases. Fig. 6 shows query plan for running a simple query on LTD dimensional model. Fig. 7 shows the query plan for the same query using mid-tier model in Kudu. Fig.8 shows a plan for a query with joins using dimensional models. As mid-tier model doesn't require joins in this case, query plan remains the same. Memory requirement using dimensional models is higher than using mid tier model in Kudu.



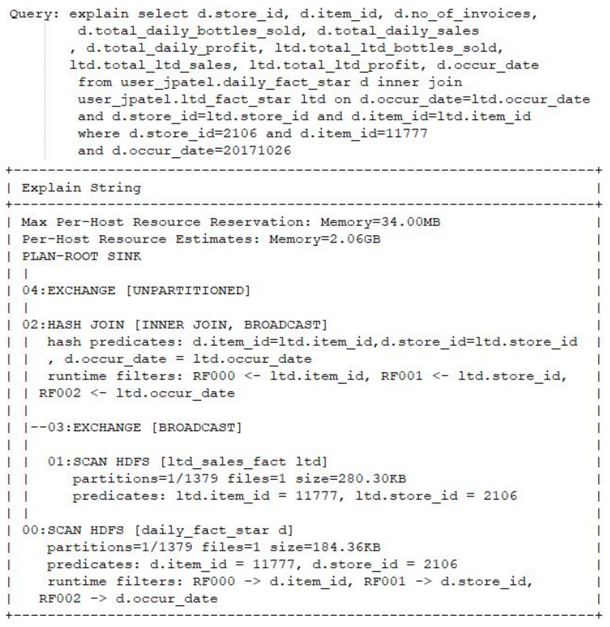
```
Query: explain select * from  ltd_sales_fact where store_id=2106
     and item_id=11777
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=0B              |
| Per-Host Resource Estimates: Memory=352.00MB              |
|                                                           |
| PLAN-ROOT SINK                                            |
| |                                                         |
| 01:EXCHANGE [UNPARTITIONED]                               |
| |                                                         |
| 00:SCAN HDFS [ltd_sales_fact]                             |
|    partitions=1379/1379 files=1379 size=443.53MB          |
|    predicates: item_id = 11777, store_id = 2106           |
+-----------------------------------------------------------+
```

Fig 6. Sample Query Plan for Dimensional Model



```
Query: explain select * from sales_mid_tier where store_id=2106
     and item_id=11777
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=0B              |
| Per-Host Resource Estimates: Memory=10.00MB               |
|                                                           |
| PLAN-ROOT SINK                                            |
| |                                                         |
| 01:EXCHANGE [UNPARTITIONED]                               |
| |                                                         |
| 00:SCAN KUDU [sales_mid_tier]                             |
|    kudu predicates: item_id = 11777, store_id = 2106      |
+-----------------------------------------------------------+
```

Fig 7. Sample Query Plan for Mid-Tier Model



```
Query: explain select d.store_id, d.item_id, d.no_of_invoices,
        d.total_daily_bottles_sold, d.total_daily_sales
       , d.total_daily_profit, ltd.total_ltd_bottles_sold,
       ltd.total_ltd_sales, ltd.total_ltd_profit, d.occur_date
        from user_jpatel.daily_fact_star d inner join
        user_jpatel.ltd_fact_star ltd on d.occur_date=ltd.occur_date
        and d.store_id=ltd.store_id and d.item_id=ltd.item_id
        where d.store_id=2106 and d.item_id=11777
        and d.occur_date=20171026
+-----------------------------------------------------------------------+
| Explain String                                                        |
+-----------------------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=34.00MB                     |
| Per-Host Resource Estimates: Memory=2.06GB                            |
| PLAN-ROOT SINK                                                        |
| |                                                                     |
| 04:EXCHANGE [UNPARTITIONED]                                           |
| |                                                                     |
| 02:HASH JOIN [INNER JOIN, BROADCAST]                                  |
| |  hash predicates: d.item_id=ltd.item_id,d.store_id=ltd.store_id     |
| |  , d.occur_date = ltd.occur_date                                    |
| |  runtime filters: RF000 <- ltd.item_id, RF001 <- ltd.store_id,      |
| | RF002 <- ltd.occur_date                                             |
| |                                                                     |
| |--03:EXCHANGE [BROADCAST]                                            |
| |                                                                     |
| |  01:SCAN HDFS [ltd_sales_fact ltd]                                  |
| |     partitions=1/1379 files=1 size=280.30KB                         |
| |     predicates: ltd.item_id = 11777, ltd.store_id = 2106            |
| |                                                                     |
| 00:SCAN HDFS [daily_fact_star d]                                      |
|    partitions=1/1379 files=1 size=184.36KB                            |
|    predicates: d.item_id = 11777, d.store_id = 2106                   |
|    runtime filters: RF000 -> d.item_id, RF001 -> d.store_id,          |
|   RF002 -> d.occur_date                                               |
+-----------------------------------------------------------------------+
```

Fig 8. Sample Plan for a Query with Joins using Dimensional Models

The second use case experimented was iterative processing with the mid-tier model. With traditional facts, daily and ltd metrics must be processed in the same execution due to immutable storage. This leads to larger memory footprint and causes issues with large volume of data. On the other hand, mid-tier model can be processed in iterations- first iteration loads daily metrics and second iteration load ltd metrics. As processing is broken down, data load is faster. As data used in experiments was already denormalized with lower volume, there was not much difference in performance. However, in the real world when data has been aggregated from multiple sources, mid-tier models performs better. Additionally, a use case was tried for a maintenance and extension. A new metric-total_profit_last_30_days to store total profit on an item in a store in the last 30 days is required to be added and to be loaded for historical data. If it is considered to be added in traditional model-ltd_sales_fact, it requires reprocessing data since 2012 and rewriting parquet files due to immutable format. It is a larger operation even in this experiment. However, it is much easier to write an impala query to update metric in mid-tier model- daily_sales_fact. Similarly, traditional models needs reprocessing data in case of issues. If someone has reported an issue with source data and there is a need to reprocess. It is much costly to reprocess using traditional processing than doing it in mid-tier model. Mid-tier model can also be refreshed for just affected metrics. It doesn't need to reprocess all metrics.

VI. Future Work

Mid-tier model is breaking processing and joins in smaller batches which will resolve few issues. It also proposes mutable storage for HDFS and that is one of the evolving space in big data platform. Partitioning of data is a major factor in achieving good performance. As mid-tier models are intended to use further, query response time should be low. Partitioning really depends on the nature of data and source applications. Partitions on transaction date should be enough for most use cases. Hash and Range partitions should be considered as needed. However, recommendations for better partitioning scheme is left for future research.

VII. Conclusion

Organizing big data is not an easy task. This paper proposed an overview of mid-tier data models which can help alleviate certain issues with current data modeling techniques. It discussed approach to build a mid-tier model and how it can solve some of the challenges in big data environments. Mid-tier model is an effective way to organize facts in a big data environment. With evolving tools and processes, there will be further enhancements in how data is stored in big data platforms to enhance decision support systems.

References

[1] R. Kimball, M. Ross, "The Data Warehouse Toolkit" in The definitive Guide to Dimensional Modelling in Depth, Indianapolis, IN:Wiley Publishing.

[2] B. Inmon, "The Single Version of The Truth", Business Intelligence Network (Powell Media LLC), 2004, [online] Available: http://www.b-eye-network.com/view/282.

[3] V. Jovanovic, D. Subotic and S. Mrdalj, "Data modeling styles in data warehousing," 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2014, pp. 1458-1463.

[4] W. Inmon, D. Strauss, G. Neushloss, DW 2.0-the architecture for the next generation of data warehousing, Morgan Kaufmann, 2008.

[5] I. Bojičić, Z. Marjanović, N. Turajlić, M. Petrović, M. Vučković and V. Jovanović, "A comparative analysis of data warehouse data models," 2016 6th International Conference on Computers Communications and Control (ICCCC), Oradea, 2016, pp. 151-159.

[6] T. Lipcon, D. Alves, D. Burkert, J.-D. Cryans, A. Dembo, M. Percy, S. Rus, D. Wang, M. Bertozzi, C. P. McCabe, and A. Wang. Kudu: Storage for fast analytics on fast data. Technical report, Cloudera, Inc., 2015.

[7] Alexander Böhm , Jens Dittrich , Niloy Mukherjee , Ippokratis Pandis , Rajkumar Sen, Operational analytics data management systems, Proceedings of the VLDB Endowment, v.9 n.13, p.1601-1604, September 2016.

[8] Apache Kudu. https://kudu.apache.org/.

[9] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave,X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin et al., "Apache Spark: a unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 56–65, 2016.

[10] H. Fang, "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem," 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), Shenyang, 2015, pp. 820-824.

[11] W. Zhang, X. Wang, J. Yan and H. Zha, "A Divide-and-Conquer Approach for Large-Scale Multi-label Learning," 2017 IEEE Third International Conference on Multimedia Big Data (BigMM), Laguna Hills, CA, 2017, pp. 398-401.

[12] R. Kimball. 2012. The evolving role of the enterprise data warehouse in the era of big data analytics. White paper, Kimball Group, pages 1–31.

[13] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou. 2013. "Big Data Challenge: A Data Management Perspective." Frontiers of Computer Science 7 (2): 157–164.

[14] A. Gandomi, and M. Haider. 2015. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35(2), 137-144. http://dx.doi.org/10.1016/j.ijinfomgt.2014.10.007

[15] U. Sivarajah, M. M. Kamal, Z. Irani, & V. Weerakkody. 2017. Critical analysis of Big Data challenges and analytical methods. Journal of Business Research, 70 (2017), pp. 263-286. https://doi.org/10.1016/j.jbusres.2016.08.001

[16] A. Sebaa, F. Chick, A. Nouicer, and A. Tari. 2017. Research in Big Data Warehousing using Hadoop. Journal of Information Systems Engineering & Management, 2(2), 10. doi: 10.20897/jisem.201710

[17] M. E. Houari, M. Rhanoui, and B. E. Asri. 2017. Hybrid big data warehouse for on-demand decision needs. 2017 International Conference on Electrical and Information Technologies (ICEIT), 1-6.

[18] M. Golfarelli, and S. Rizzi. 2018. From Star Schemas to Big Data: 20 $$+$$ Years of Data Warehouse Research, Cham:Springer, pp. 93-107.

[19] Cloudera QuickStart VM. https://www.cloudera.com/documentation/enterprise/5-13-x/topics/quickstart_vm_administrative_information.html

[20] Data.iowa.gov. (2019). [online] Available at: https://data.iowa.gov/Sales-Distribution/Iowa-Liquor-Sales/m3tr-qhgy

Jayesh Patel became a member of IEEE in 2019 and a Senior Member in 2019. Born in 1983 in western part of India, he completed his Bachelors of Engineering from Nirma University, Ahmedabad, Gujarat, India in 2005 and Masters of Business Administration from San Diego State University, San Diego, California, US in 2009. His major was information systems.

He currently work as SR DATA ENGINEER for Rockstar Games in Carlsbad, CA. He architects and develops scalable data-driven decision-making processes on Big Data Platform for Rockstar Games. He has successfully built machine learning pipelines and architected big data analytics solutions for more than 12 years. He is passionate about researching data integration and information management.