Research and Implementation of OLAP System Based on Cloud Computing Platform

Guangxin Zhu[a], Haiyang Chen[b], Mingjie Xu[c], Jun Yu[d], Yang Ling[e], Lin Wang[f] and Zhu Mei[g]

State Grid Electric Power Research Institute (SGEPRI) Nanjing, China

[a]zhuguangxin@sgepri.sgcc.com.cn, [b]chenhaiyang@sgepri.sgcc.com.cn, [c]xumingjie@sgepri.sgcc.com.cn, [d]yujun@sgepri.sgcc.com.cn, [e]842587420@qq.com, [f]wanglin18@sgepri.sgcc.com.cn, [g]meizhu2016@aliyun.com

Abstract. The advent of cloud computing addresses perfectly the problem of processing mass data. To further promote the development of our Geography information technology, cloud computing can be used to construct the system of the Geography environment information service application framework. It can also improve reusability and sharing of the information of Geography resources and extendibility of application system. In this paper, an OLAP system based on cloud computing platforms has been created referring to national welfare projects of ocean. The system provides its users multidimensional views, through which users can, from diverse angles and different levels, observe and study the data and comprehend them in depth.

1. Introduction

After years of construction, the State Gird has built a large number of professional geographical environment information about electric power databases, and built a number of information systems used for geographically diverse services. In order to further develop the effectiveness of the geographic information resource sharing platform that has been established in digital geography, build a geographic environment information integrated service application framework system, and build a low-cost test and operation environment that supports geographic environment information services, while improving the reusability and sharing of geographic resource information and the scalability of the application system, the development and application of cloud computing and cloud service technologies will provide end users with on-demand services through a standard model. The service has the advantages of saving investment, improving business support capability, improving operation and maintenance efficiency, reducing investment risk and decision risk, and green energy conservation.

2. System Overview

The overall framework of the OLAP System based on cloud computing platform implemented in this paper is shown in Figure 1. The system architecture is mainly divided into the application layer, OLAP engine layer [1] and storage layer. The application layer communicates with the OLAP engine layer using the HTTP protocol and the XML response. The OLAP engine layer is implemented by the open source OLAP engine Mondrian, and uses the open source MDX parser to parse the MDX statement into a HiveQL statement, while providing a JDBC interface to communicate with Hive.

1) Application layer: The application layer is implemented by JPivot. JPivot's tag library can render various tags of OLAP operations, enabling users to perform typical OLAP operations, such as drilling, down, rotating and slicing, and provide multi-dimensional visual display of query results, as well as common functions such as report conversion and printing.

2) OLAP engine layer: OLAP engine layer mainly involves two parts - metadata management Schema, OLAP engine Mondrian. The metadata management part is the mapping of the management database Hive to the multidimensional model, which is implemented by Schema file definition. Specifically, user can write a schema file to define the analysis topic, dimensions, metrics, calculation members, etc. of the multidimensional analysis dataset. The OLAP engine is responsible for building the cube from the schema file and getting the corresponding data from the database Hive so that the data is stored in multiple dimensions.

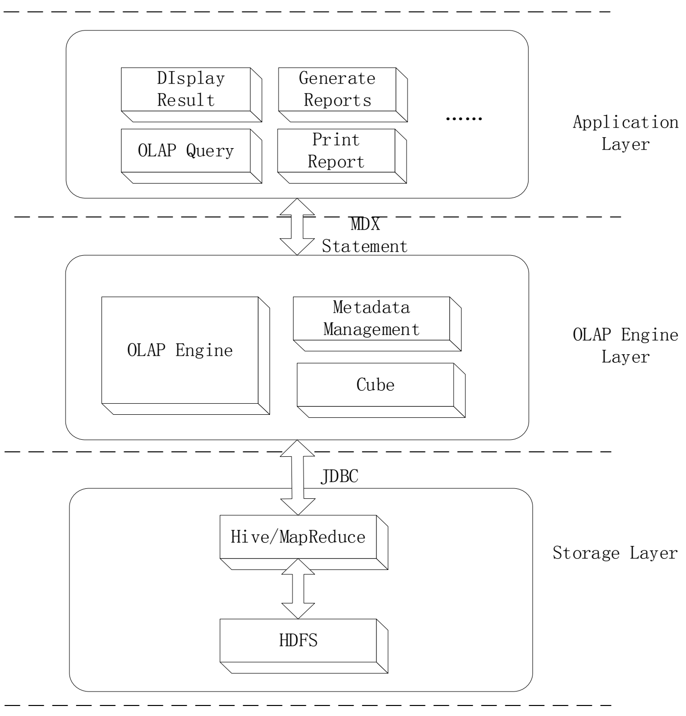

Figure 1. The overall structure of the online analytical processing system based on cloud computing platform

3) Storage layer: The storage layer runs in the Hadoop cluster. When the upper layer passes the aggregate load request or the member load request, the SQL generator HiveDialect generates the HiveQL statement of the column. The database Hive converts the HiveQL statement into a MapReduce job and submits it to the Hadoop platform computing tool for calculation. By calling Hive's JDBC interface, the system gets the data available for analysis and presentation and returns it to the OLAP engine layer to create a multidimensional model. All data of Hive is stored in the distributed file system HDFS.

3. Design and Implementation of System

3.1. Design and Implementation of the Storage Layer

The storage layer of the system is composed of the data warehouse [2-6] Hive of the cloud platform. The OLAP engine accesses the Hive data source by calling the JDBC interface to obtain the data to be analyzed. Because the SQL statement generated by the OLAP engine Mondrian is not compatible with Hive, the system implements a dialect design for Hive, which implements Mondrian's reading of data in the Hive data source by adding the HiveDialect class, which provides a data foundation for the next step in OLAP analysis.

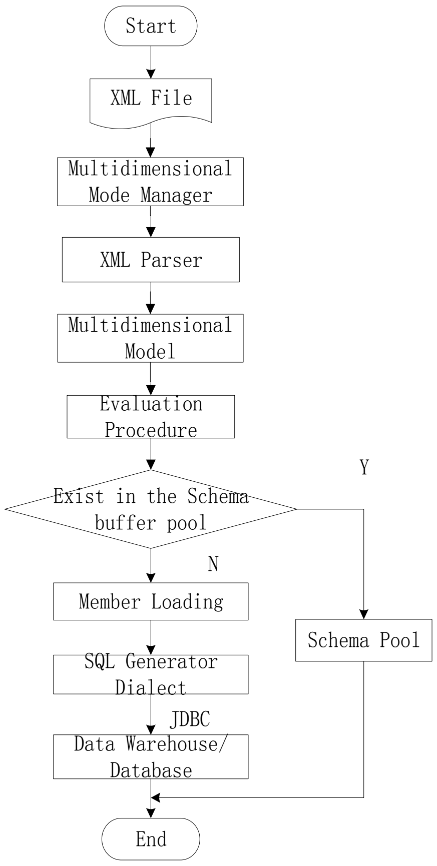

Figure 2. Process of multidimensional model mapping

Mondrian is an OLAP engine written in Java that executes queries through the MDX language, reads data from a data warehouse or database, and presents the results of the query in a multidimensional form through the Java API.Mondrian's storage tier is at the bottom of its structure, which is implemented by the database system. Mondrian stores the data used in the analysis in the database, defines the multidimensional data model through the schema file, and uses the JDBC interface to implement the mapping between the physical data in the database and the multidimensional data model. Figure 2 shows the specific flow chart of the mapping. First, the multidimensional schema manager loads the multidimensional schema file through the XML parser to generate a multidimensional model. The dimension manager then initializes the evaluation function and loads the specific dimensions in the generated multidimensional model. If the member is already in the multidimensional data pool, load it directly. If not, the system will map with the columns in the database to obtain the members of the dimension. Through the SQL generator Dialect class, JDBC will be used to load the columns in the database into the multidimensional model.

It can be seen that Mondrian does not really store data, but only stores the mapping between physical data and multidimensional data models. Mondrian uses ROLAP storage, and the data uses a star schema or a snowflake schema in the cube. Usually, the star schema is used. There is only one fact table in this structure, so the relationship between the dimension table and the fact table is through the foreign key. The storage mode of the data table in the corresponding database should also be a primary table associated with other tables through foreign keys.

Mondrian runs on most databases with a JDBC interface, and it provides developers with a standardized interface called Dialect to support operations on new databases [7-12]. So we can achieve Mondrian compatibility with Hive by writing a new dialect HiveDialect. The structure diagram of the Hive-based storage layer is shown in Figure 3. When creating a cube to load a dimension member or requesting a unit to perform an aggregation operation, the system needs to pass the datasource parameter to the SQL generator, the HiveDialect class converts the query statement into a HiveQL statement, and reads the data in the Hive through the JDBC interface. And return to the dimension manager or the aggregation manager to complete the member loading or aggregation loading process. Since Mondrian provides a common dialect of database based on JDBC connection and management metadata, we only need to redevelop this set of common dialects and convert it into HiveQL dialect to meet Mondrian's support for Hive.

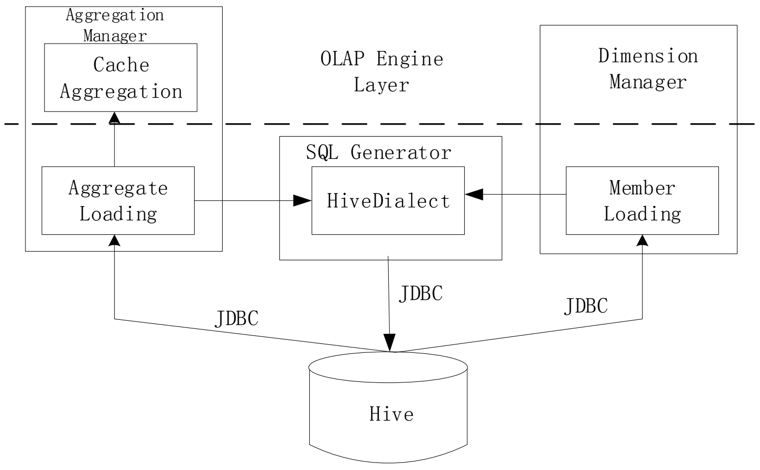

Figure 3. Structure of the Hive-based storage layer

3.2. Design and Implementation of OLAP Engine Layer

The OLAP engine layer of this system is implemented by the open source project Mondrian [13-15], which is mainly composed of Schema manager, session manager, aggregation manager and dimension manager. Through the parsing of the multidimensional model XML file, the mapping of the columns in the Hive to the corresponding members in the multidimensional model is completed, and the cube is created. Then through the parsing of the MDX query statement, the data is processed accordingly and the result set is returned to the upper application.

The OLAP engine layer is mainly implemented by Mondrian. The main architecture of the OLAP engine layer is shown in Figure 4. The OLAP engine layer is mainly composed of a Schema manager, a session manager, an aggregation manager, and a dimension manager.

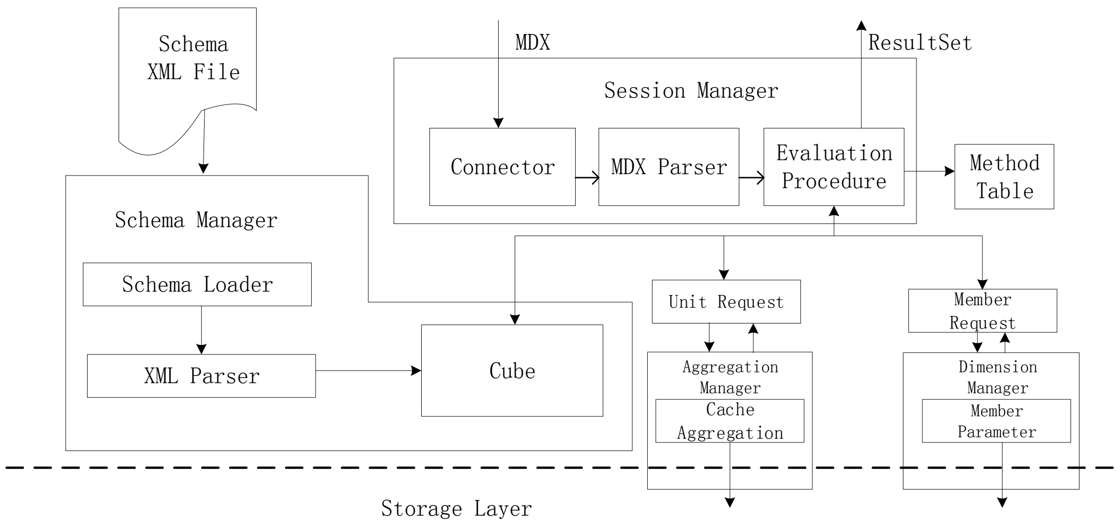

Figure 4. Architecture of OLAP engine layer

The session manager mainly accepts MDX queries, parses MDX queries, and returns query results to the application layer [16]. A query first calculates the coordinate axes and then calculates the value of each cell. In terms of efficiency, the query calculation is a collection of unit data obtained from the aggregation manager in batches. The Schema Manager is mainly related to initialization, completes the construction of the cache pool and the generation of the multidimensional model. The multidimensional model schema file Schema defines the analysis topic, dimensions, metrics, calculation members, etc. of the multidimensional analysis data set. The cube is then built according to the schema file schema, and the data obtained from the storage layer is stored in a multidimensional form. The aggregation manager implements the aggregation mechanism, mainly the management of the OLAP cache, which belongs to the performance optimization part. The dimension manager mainly implements the mapping of dimensions in the multidimensional model and columns in the Hive database. The loading capabilities of the Dimension Manager and Aggregation Manager are implemented in the storage tier.

3.3. Design and Implementation of Application Layer

The application layer of the system provides users with a good user image interface, and users can perform typical OLAP operations, printing and conversion into EXCEL forms and other common functions. The user graphical interface of the system is a web page, including html tags and jsp tags. Using the custom JSP tag library JPivot, the JSP tag library is used to provide related buttons and data presentation for performing OLAP operations, including image presentation and table presentation, and connection to the underlying data model, which supports JDBC access to the data source.

The user sends an MDX request from the browser. After receiving the request, the application layer server calls the relevant program to pass to the OLAP engine layer and the storage layer, and executes the query in the relevant data source. After the query is completed, the query result set is returned to the application layer, organized and rendered according to the OLAP model of the application layer JPivot, and finally output to the WEB page. The interaction between the application layer and the lower layer is mainly achieved through the custom tag library JPivot's MondrianQuery tag. The main task of the tag is to let Mondrian execute the specified query. Connect Mondrian to the specified database by getting parameters such as JDBC, URL, Schema, and user-defined MDX queries. The MondrianQuery tag is implemented by the MondrianOLAPModelTag class, which instantiates the MondrianModel class, along with all extensions, and places the extension as an internal property in the MondrianModel and then in the proxy (OlapModelProxy). The instantiation of MondrianModel can get the metadata needed for MDX query; and each extension is an operation for the user to access the page, such as reels, slices and dicing.

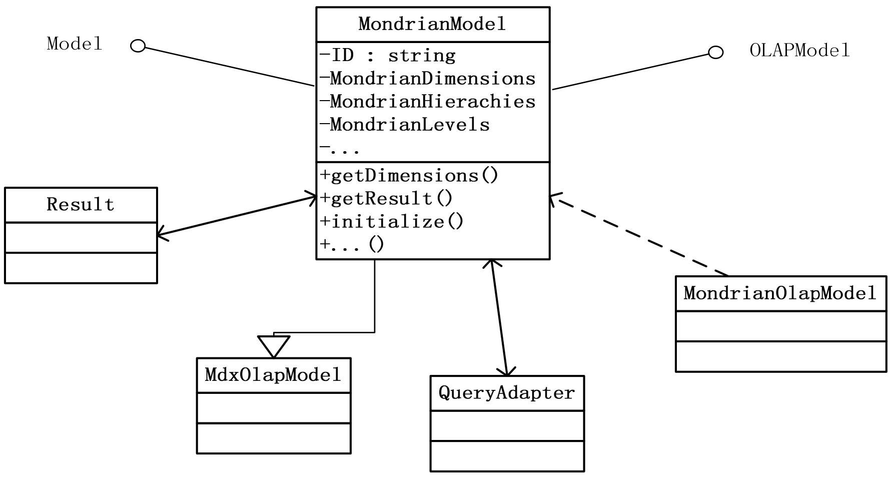

Figure 5. Related Class diagram of MondrianModel

The MondrianModel class is the key class that implements the interaction between JPivot and Mondrian. Figure 5 shows the class diagram associated with MondrianModel, which includes a Mondrian connection, query adapter, and query results. The query adapter adapts the Mondrian query object to the JPivot component, which parses the MDX statement into the components that Jpivot wants to expose and manipulate.

The results of the application layer visualization display mainly through the JPivot tag library to render the query result set, generate OLAP analysis charts, and can provide users with typical OLAP navigation, perform drill-down, slice and dicing operations.

Figure 6 shows the execution flow of the results visualization.First, the application layer sends an MDX request to the OLAP engine layer. After the OLAP engine layer executes the MDX query, the query result set is returned to the application layer. The result set is then organized according to the OLAP model of JPivot and placed in the session. Process the Table tag, construct the report control, associate the object saved in the session with the report control and the RequestFilter in the previous step, and then put it into the session. The labels of controls such as Toolbar and Navigator are handled in the same way. Process each RenderTag, get the navigation object from the session, and generate the xml. Use xsl to convert the Navigator's xml format to html format and output it to the interface via Response, completing the rendering process. Then generate the corresponding UI control, output to the html client and display the data. At this point, an http request ends. The user can perform multiple operations, and the system ensures the continuity of multiple operations between the user and the system through the session mechanism, and also ensures that multiple users can simultaneously access the system without interference.

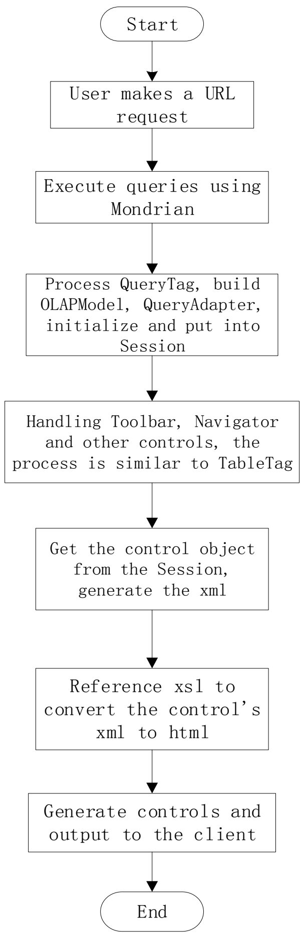
Start
User makes a URL request
Execute queries using Mondrian
Process QueryTag, build OLAPModel, QueryAdapter, initialize and put into Session
Handling Toolbar, Navigator and other controls, the process is similar to TableTag
Get the control object from the Session, generate the xml
Reference xsl to convert the control's xml to html
Generate controls and output to the client
End

Figure 6. Process of Results Visualization

4. Conclusion

This paper implements an OLAP system based on cloud platform technology, and carries out secondary development of the OLAP engine used in the system, and has made some improvements. According to the characteristics of MapReduce programming model, the OSRed-based parallel algorithm based on MapReduce is proposed and some research and experiments are done. This paper starts from the research background of the project and explains the source of the project. Then, according to the project requirements, an analysis of the existing OLAP system is made, pointing out that the existing system is worthy of reference and deficiencies, and finally the research work and research purposes of this paper are determined.

Acknowledgments

This work was financially supported by the State Grid Corporation of Science and Technology (WBS number: 521104170019).

References

[1] Nadim W.Alkharouf, D.Curtis Jamison, Benjamin F. Mathhews. Online Analytical Processing (OLAP): A fast and effective data mining tool for gene expression databases [J], Journal of Biomedicine and Biotechnology, 2005, 2005 (2): 181-188.
[2] Golfarelli M, Rizzi S. Data warehouse design: Modern principles and methodologies [M], McGraw-Hill, 2009, 87-88.
[3] Inmon W H. Building the data warehouse [M], USA: John wiley & sons, 2005, 102-104.

[4] Bialecki A, Cafarella M, Cutting D, et al. Hadoop: a framework for running applications on large clusters built of commodity hardware [J], Wiki at http://lucene. apache. org/hadoop, 2005, 11.
[5] Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework [J], Proceedings of the VLDB Endowment, 2009, 2 (2): 1626-1629.
[6] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [A], In the Proceedings of the 6th Symposium on Operating System Design and Implementation [C], 2004, 137-150.
[7] Chu C, Kim S K, Lin Y A, et al. Map-reduce for machine learning on multicore [J], Advances in neural information processing systems, 2007, 19: 281.
[8] Chen k, Zheng WM. Cloud computing: System instances and current research [J], Journal of Software, 2009, 2009: 1337-1348.
[9] Gates A F, Natkovich O, Chopra S, et al. Building a high-level dataflow system on top of Map-Reduce: the Pig experience [J], Proceedings of the VLDB Endowment, 2009, 2 (2): 1414-1425.
[10] Thusoo A, Shao Z, Anthony S, et al. Data warehousing and analytics infrastructure at facebook [A], Proceedings of the 2010 ACM SIGMOD International Conference on Management of data [C], 2010, 1013-1020.
[11] ParkB K, Han H, Song IY. XML-OLAP: A multidimensional analysis framework for XML warehouses [M], Springer Berlin Heidelberg: Data Warehousing and Knowledge Discovery, 2005, 32-42.
[12] Thusoo A, Sarma J S, Jain N, et al. Hive-a petabyte scale data warehouse using hadoop [A], Data Engineering (ICDE), 2010 IEEE 26th International Conference on [C], 2010, 996-1005.
[13] Vieira M, Madeira H. A dependability benchmark for OLTP application environments [A], Proceedings of the 29th international conference on Very large data bases-Volume 29 [C], 2003, 742-753.
[14] Thomsen E. OLAP solutions: building multidimensional information systems [M]. Wiley, 2002, 210-213.
[15] Eric Thomsen, George Spofford. Microsoft OLAP Solutions [M], USA:Wiley Computer Publishing, 1999, 67-69.
[16] Whitehorn M, Zare R, Pazusmansky M. Problem solving for MDX for SQL Server 2005 [J], 2007 (14):16-17.