EFFICIENT LOG MANAGEMENT USING OOZIE, PARQUET AND HIVE

Gopi Krishnan Nambiar
Salesforce, 50 Fremont Center, 50 Fremont St, San Francisco, CA, USA

ABSTRACT

The software industry produces a large volume of data in the form of application logs. This data is useful since it can help debug issues in production, provide many hidden insights and is a treasure trove for data scientists and researchers. The volume of the data, however causes multiple issues with querying, storage and maintenance over a period of time. In this paper, we share the architecture and the steps we implemented for efficiently storing the application logs based on the data requirements from our partners and also discuss various optimization techniques adopted.

KEYWORDS

Data Management, Oozie, MapReduce, Parquet, Scalability

1. INTRODUCTION

In this paper, we describe an efficient method of data management for large volumes of application logs that a software company generates, with techniques on how to enable efficient querying and automatic management of the data without compromising on the query time and storage efficiency.

To illustrate the scale of the system at Salesforce, we provide some statistics for the same: in the year 2016, we processed more than 3 PB of data, which contained important information that needed to be analyzed and queried by our internal customers and data scientists.

This paper is organized as follows: in section 2 we explore related works in this domain and compare those with our implementation. Section 3 describes the key technologies we used and why they were chosen, In Section 4, we discuss architecture of our workflow. We describe some of the key optimizations we implemented in our system in Section 5. We propose some improvements to the system in Section 6 and conclude in Section 7.

2. RELATED WORK

As part of the research for this implementation, we looked into sample implementations where log records were analyzed and made available for consumers. Mavridis and Karatza (2015) discuss how logs can be efficiently processed using Hadoop and Spark, but their focus is mainly on comparing the two technologies and benchmarking their performance. In our work, we offer a fully automated solution with architecture and optimizations that can be replicated and deployed at any software company for the purpose of efficient log management, reduction of query times and disk space usage optimization being few of the benefits offered by our methodology. Beitzel et al (2004) in their paper discuss the methods they used for the analysis of a categorized web application logs. The techniques they used involved examining the logs over hourly periods and narrowing down the patterns over the course of a day. They examined the popularity trends and other statistics at an hourly rate, whereas in our work we examine the entire dataset and allow the users the flexibility to query for any period of time and discover the patterns in the logs. We also focus on automation and optimization more than the pattern recognition and popularity of topics in the various log lines and allow users to query and discover the patterns rather than surfacing them upfront.

3. KEY TECHNOLOGIES

We chose Apache Oozie as a workflow management system because of the multiple capabilities it offered in terms of scalability, scheduling and ease of use. Oozie is an open source workflow engine that is specialized in executing workflow jobs with actions that execute MapReduce, Pig, Hive and Java jobs. A key benefit with Oozie was the flexibility to wait for the input data sets. In case the data for a particular date was delayed or not available, the Oozie job would not run and produce incorrect or incomplete output. Instead it would wait for the input data to be available and execute the workflow only if that condition is successful. This coupled with an internal retry logic and the ability to parameterize the execution of jobs based on time stamps were the main reasons for choosing Oozie. In their research paper Islam M. et al (2012) discuss that the key features of Oozie in detail. Oozie also provided the flexibility to configure and launch jobs for a long period of time (a year) that execute as and when the input dataset becomes available on a day to day basis.

Parquet is a columnar format that provides multiple advantages over raw files. Parquet provides us the benefit of compression and encoding which leads to disk space usage savings and also reads only the necessary columns while querying the data which provides memory savings. This format also provides schema evolution which is crucial since we may get log lines in the following releases where the schema has changed and we would be able to handle this change seamlessly. As mentioned in the research from Bisoyi et al (2017), it is a tough choice between Optimized Row Columnar format (ORC) and Parquet considering performance but in our case the storage optimizations offered by Parquet were more valuable than the read optimizations offered by ORC format.

Apache Hive is a data warehouse that facilitates reading, writing and managing large datasets residing in distributed storage and queried using SQL syntax. Hive also provided us with the flexibility of easily loading data from Parquet by means of external tables. The only requirement was that the location of the external table partition had to be specified. Our log data was originally stored in the ORC format then later switched to Parquet format considering space and query optimizations.

4. ARCHITECTURE OF THE JOB

The job consists of on an Oozie workflow that contains 4 different Oozie actions. First is the decisionNode, where we decide whether to clean up the existing data present at an output location. If the job is determined to be a catchup job (which aims at collecting late arriving data) then we do not delete the existing output folder. Otherwise, we assume that the folder contains old or invalid data and delete the folder and its data contents.

The next step consists of a MapReduce action. This task takes the raw log as input data and a set of log record types which are needed by downstream consumers for analysis and returns Parquet files in the compressed format (GZIP compression) as output. We use GZIP instead of Snappy as the space savings for GZIP format were in the range of 65% in comparison to Snappy.

The mapper has access to a list of log record types which is interpreted from a property file that consists of a list of key log record types requested by our consumers. The map task reads only those log record types and emits a key value pair with the hostname as the key and the logline as the value. The reducer has access to release schema files which contain a standard format for each release. This aids in creating a uniform schema and we also take extra precautions to collect extra fields that are added by different upstream users in the spirit of data completeness. The reducer builds a pig tuple, based on the already existing information it obtained from the schema information and stores the extra fields as well. The fields are stored in the form of a bytearray to maintain easy interoperability. We maintain an internal mapping from the Parquet schema to the Hive schema in the code. The Hive schema is created based on this internal field mapping and data is entered from each field in the Parquet schema. The final output of the map reduce job is a set of Parquet files compressed in GZIP format.

The third step is a Hive action. The data created in the map reduce action needs to be recognized by Hive as a partition and to enable this we run an alter table command to add the new partitions generated as part of the map reduce action.

The final step is a validation action. Each job needs to be validated against the source on its correctness. The jobs may run into multiple issues, may lose data or contain corrupt records and we do not want our

customers to have their jobs running on incomplete or incorrect data. The validation job serves this purpose. It validates the correctness of the produced data with the source and sends an email alert for notifying the team in case the validation does not pass. Figure 1 below illustrates all the actions in the form of a detailed architecture diagram

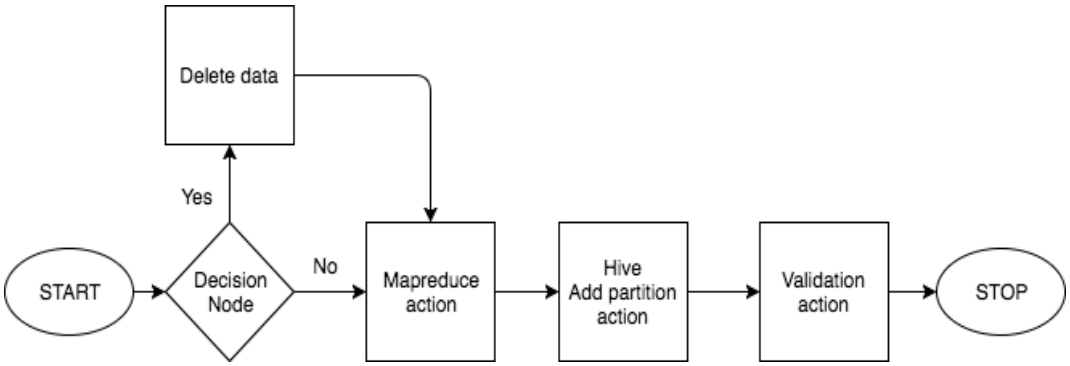

Figure 1. Architecture Diagram

5. OPTIMIZATIONS TO MAPREDUCE TASK

In the initial stages, the map reduce job used to take significant amount of time to complete for each datacenter, about 20 hours. This was problematic since it would cause a significant lag and impede efficient log processing for the organization. Hence, we conducted multiple experiments to optimize the job completion times. For MapReduce jobs, a key suspect for long running jobs are inefficiently distributed mapper keys and is termed as a data skew, which is explained by Kwon et al (2013). An important investigation in this case would be to modify the map keys so that they are evenly distributed and your reducers receive an even distribution of keys to process. This phenomenon can be detected by observing completion times for the reducers. If you observe some reducers completing in few minutes whereas others take multiple hours to complete, then it is a clear indication of data skewness. Once we modified the logic to achieve an even distribution of keys, the job run times were reduced to 2 hours.

Another important optimization is to tweak the memory allocated to mappers and reducers. Examining the counters and physical memory utilized by the Hadoop jobs can give valuable insights into the amount of memory needed. The parameters that you should look at are mapreduce.map.memory.mb, which is the upper limit that Hadoop allocates to its mappers and similarly mapreduce.reduce.memory.mb for the reducer. If you have simple mapper jobs and complex reducer jobs, the memory assignment should take this into account. Other properties that help in job optimizations are mapreduce.map.java.opts, and mapreduce.reduce.java.opts. These properties control the amount of maximum heap memory allocated to the Java process running inside the mapper and reducer respectively. We also tweaked the yarn.scheduler.minimum-allocation-mb, which controls the minimum amounts of RAM increments that the Resource Manager can allocate to containers so that the increments did not need to be in terms of the default 1024 MB and granted us more flexibility.

6. FUTURE WORK

The current data management technique is susceptible to schema changes as the current Parquet schema we use is fixed and a change in the log file format (addition of new log fields) would cause the system to ignore the updates. Though Parquet has the schema evolution feature available, due to limitations of our experiment and time constraints, this has not yet been incorporated. This is a beneficial and critical feature for this project to be future proof and compatible with later versions of log files. Currently, the job is constrained based on the schema and fields that are predefined, but if we build and utilize capability to incorporate a

dynamic schema, then it would be easy to include new fields and consume metadata for logs and hence improve the runtime further. As per Mehmood et al in (2016), exploring alternate options like Impala in place of Hive also look promising due to the benefits it offers with the Parquet file format.

7. CONCLUSION

The data management technique illustrated in this paper has been beneficial to Salesforce for optimizing disk space consumption and improving query processing times while keeping the data available for partners and customers to query efficiently.

We have discussed the job architecture, optimizations and storage format for data management and also suggested areas in which the data management model could potentially improve.

REFERENCES

Beitzel, S. M., Jensen, E. C., Chowdhury, A., Grossman, D., & Frieder, O. (2004, July). Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 321-328). ACM.

Bisoyi, S.S., Mishra, P. and Mishra, S.N., 2017. Relational Query Optimization Technique using Space Efficient File Formats of Hadoop for the Big Data Warehouse System. *Indian Journal of Science and Technology*, *8*(1).

Islam, M., Huang, A. K., Battisha, M., Chiang, M., Srinivasan, S., Peters, C., ... & Abdelnur, A. (2012, May). Oozie: towards a scalable workflow management system for hadoop. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies* (p. 4). ACM.

Kwon, Y., Ren, K., Balazinska, M., Howe, B., & Rolia, J. (2013). Managing Skew in Hadoop. *IEEE Data Eng. Bull.*, *36*(1), 24-33.

Mavridis, I., & Karatza, E. (2015). Log file analysis in cloud with Apache Hadoop and Apache Spark

Mehmood, A., Iqbal, M., Khaleeq, M., & Khaliq, Y. (2016, August). Performance analysis of shared-nothing SQL-on-Hadoop frameworks based on columnar database systems. In *Innovative Computing Technology (INTECH), 2016 Sixth International Conference on* (pp. 128-133). IEEE.