LATENCY ANALYSIS OF LARGE VOLUME SATELLITE DATA TRANSMISSIONS

Weiguo Han[1,2], *Matthew Jochum*[2]

[1] Cooperative Programs for the Advancement of Earth System Science, University Corporation for Atmospheric Research, 3090 Center Green Drive, Boulder, CO 80301, USA
[2]Center for Satellite Applications and Research, National Oceanic and Atmospheric Administration 5830 University Research Court, College Park, MD 20740, USA

ABSTRACT

A wide array of time-sensitive satellite data is required in the research and development activities for natural hazard assessment, storms and weather prediction, hurricane tracking, disaster and emergency response, and so on. Identifying and analyzing the latencies of large volumes of real-time and near real-time satellite data is very useful and helpful for detecting transmission issues, managing IT resources, and configuring and optimizing data management systems. This paper introduces how to monitor and collect important timestamps of data transmissions, organize them in a NoSQL database, and explore data latency via a user-friendly dashboard. Taking Sentinel series satellite data as an example, data transmission issues are illustrated and investigated further. Latency analysis and explorations help data providers and managers improve data transmission and enhance data management.

Index Terms— Big Data, Satellite Data, Data Latency, Data Quality, NoSQL, MongoDB

1. INTRODUCTION

Acquiring and processing time-sensitive satellite data in a timely manner is one of the major concerns of researchers and decision makers involved in weather forecasting, severe weather warning, disaster and emergency response, environmental monitoring, etc [1]. Understanding and analyzing the latency of real-time and near real-time satellite data transmission is an important topic of data quality to identify possible issues, optimize data flow processes, and better connect data providers and end users.

The STAR (Center for Satellite Applications and Research of NOAA) Central Data Repository (SCDR) is a central repository to acquire, manipulate, and disseminate large volumes of near real-time satellite data to internal and external users [2]. SCDR collects over 200,000 (~2TB in size) satellite files continuously from multiple data providers via Internet or internal network every day. The SCDR storage system (in total, about 1.8PB) is a combination of various storage systems, including Dell, NetApp, and Nexentia. SCDR processes more than 100,000 data requests from users every day. In addition, more new satellite data is expected to be captured and ingested faster and more frequently.

A number of tools and utilities supporting SCDR have been built, including those for data retrieval, data integration, data profiling, and data exploration. Latency analysis helps address concerns on data transmission, manage hardware and software resources, configure the data management system, and optimize system performance [3]. These kinds of new and potential future requirements also present a great challenge to implement latency monitoring and analysis and other functions without burdening the current operational SCDR system. This paper describes how to develop new mechanism to satisfy these requirements.

2. METHODS

Obviously, the 3Vs of big data (Volume, Velocity, and Variety) should be dealt with in the SCDR [4]. An open source database management system (DBMS), namely PostgreSQL, serves as the central database system. Important information on satellite data like platform, instrument, observation begin/end time and spatial coverage, are ingested into the database for query. Like other traditional DBMSs, it lacks scalability and flexibility to manage more additional information associated with the original satellite data (i.e. timestamps of data transmission) without changing current data schema and burdening the operational system [5].

In the SCDR, important timestamps, including observation begin/end date/time, processing, uploading, downloading, and ingestion, are recorded in the log file. When some exceptional delay occurs, these timestamps are retrieved to find the root causes. For example, Figure-1 shows the download speed of Sentinel 1A data obtained from the Sentinel International Access Hub (IntHub, https://inthub.copernicus.eu) from 11/1/2016 to 12/5/2016 in a spreadsheet file. The unusual download speed experienced on 12/3/2016 – 12/5/2016 can be observed from the chart for further investigation. But the chart is created manually from the timestamps in the log file. It is not intuitive and interactive to explore durations of each phase of data transmission [6]. New collecting, reporting, and visualizing

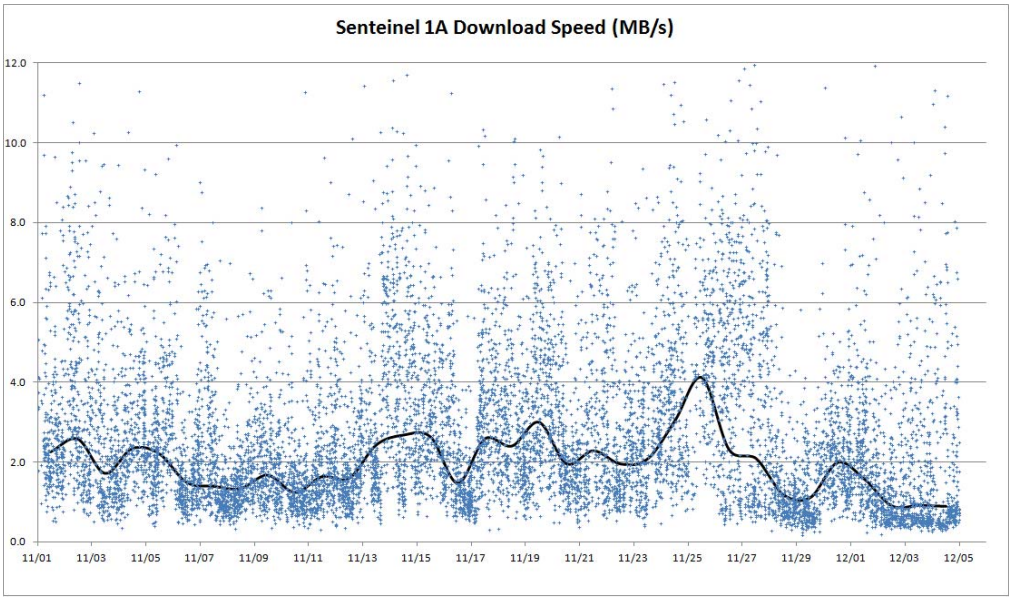

Figure 1 Sentinel 1A data download during 11/1/2016 – 12/5/2016

mechanisms of data latency are required.

An additional database system is necessary to collect latency and other attribute information of large volumes of satellite data, without changes to the current system, and to figure out the time length of each transmission phase effortlessly. The Open source NoSQL database, MongoDB (www.mongodb.org), is widely adopted to store and analyze the log information within cloud-based environments in the big data era because of its features of dynamic schema, easy aggregation, and fast data processing [7-11]. MongoDB not only offers built-in utilities to support data migration, and storage and an interactive JavaScript Interface (mongo shell) to query and update data as well as manage database, but also provides a set of functional APIs for developing programs in various programming languages. It is selected to manage the logged timestamps and other data information in our implementation [12].

In addition, a user-friendly Web portal is required to visualize and characterize the latency and other satellite data quality information interactively. So the phases of satellite data transmission, including observation begin and end, generating satellite data products from observations, uploading the file to a server, offering the file link information, downloading the file to the local host, and ingesting the file to SCDR, can be easily figured out from the stored timestamps. The latencies and downloading speed can be observed intuitively by band, day of week, hour of day, or other attributes, in the form of a chart or table, and the anomalies in data transmission can be easily detected and reported through the user interface of the portal. We leverage Ext JS (http://www.sencha.com/products/extjs/), a Javascript framework proving different types of user interface components for web application development, to create the browser client which consumes and displays latency information obtained from the MongoDB database [6].

3. RESULTS

Taking Sentinel 1A and 1B data as an example, data transmission latencies are organized, analyzed and illustrated in the prototype system utilizing the methods described in Section 2.

Sentinel 1A and 1B data for the21 target ocean areas are queried and downloaded from IntHub everyday, which offers a direct and stable Sentinel satellite data link for international partners.

The timestamps (observation beginning/ending, file creation, and file upload) can be obtained from the metadata file of the downloaded file. And the timestamp of download is recorded once the downloading is finished. These timestamps and other information (name, size and download duration) are organized in the JSON format as below and stored in the log file:


```
{
  "name": "S1A_IW_GRDH_1SSV_20161231T195759_20161231T195828_014628
_017C84_9231.zip",
  "obsbegin": "2016-12-31T19:57:59.034",
  "obsend": "2016-12-31T19:58:28.047",
  "creation": "2016-12-31T22:52:01.111",
  "update": "2016-12-31T22:52:30.808",
  "download": "2017-01-01T00:09:59",
  "size": 611725208,
  "duration": 198
}
```


So these logged records can be imported directly as the *documents* to the specified *database* and *collection* in the MongoDB using *mongoimport* utility.

These timestamps are imported as the type of *String* by default. To calculate the differences between two

timestamps, they are converted into the type of *Date* easily in the mongo shell as follows:

```
db.sentinel1.find().forEach(function(doc) {
    doc.obsbegin = new Date(doc.obsbegin);
    doc.obsend = new Date(doc.obsend);
    doc.creation = new Date(doc.creation);
    doc.update = new Date(doc.update);
    doc.download = new Date(doc.download);
    db.sentinel1.save(doc);
})

db.sentinel1.aggregate([
    {
        $match: {
            "download": {
                $gte: new Date( '4/1/2017' ),
                $lt: new Date( '4/21/2017' )
            }
        }
    },
    {
        "$project": {
            "formattedDate": {
                "$dateToString": {
                    "format": "%Y-%m-%d",
                    "date": "$download"
                }
            },
            "dwnspd": {
                $divide: [ "$size", "$duration" ]
            },
            "time1": {
                $subtract: [ "$creation", "$obsend" ]
            },
            "time2": {
                $subtract: [ "$update", "$creation" ]
            },
            "time3": {
                $subtract: [ "$download", "$update" ]
            }
        }
    },
    {
        "$group": {
            "_id": "$formattedDate",
            "avgdownload": { "$avg": "$dwnspd" },
            "avgtime1": { "$avg": "$time1" },
            "avgtime2": { "$avg": "$time2" },
            "avgtime3": { "$avg": "$time3" }
        }
    },
   {
        $sort : { _id : -1}
    }
])
```

The aggregation function of MongoDB generates the aggregation values from multiple *documents*. This function provides similar operators (like *$project*, *$match*, *$group*, *$sort*) to the ones (like *select*, *where*, *groupby*, *orderby*) of common database SQL statement. So, the latency of each phase (processing, uploading, and downloading) as well as download speed by date or hour of day can be easily figured out using the aggregation operators (like *$divide*, *$subtract*, *$avg*). The mongo shell script above generates the daily average latencies and download speed sorted by date ascending for the Sentinel 1 data obtained during the period of April 1 - 21, 2017.

To query and display the latencies interactively in a browser client, a J2EE Web application is built using Java servlet, which interacts with back-end database based on MongoDB Java driver, and ExtJS for browser client. A *ServletContextListener* is configured to initialize and close database connection. In the servlet, the MongoDB Java functions corresponding to those aggregation operators are utilized to interact with database and output the results in the JSON format for displaying in form of chart and table in the browser client.

In the browser client, the ExtJS components of grid and stacked bar chart are utilized to visualize the results returned by the servlet. From the table below, it can be seen that the download speed was nearly doubled and the download time was obviously reduced since ESA upgraded the firewall of IntHub on March 31, 2017.

Date	Download Speed	Processing	Uploading	Downloading	Total
2017-03-28	2 MB/s	15911 s	13 s	7628 s	23552 s
2017-03-29	1.9 MB/s	14423 s	11 s	6617 s	21051 s
2017-03-30	2.6 MB/s	31678 s	13 s	5470 s	37161 s
2017-03-31	3.6 MB/s	14041 s	11 s	7871 s	21923 s
2017-04-01	6.3 MB/s	14359 s	12 s	1471 s	15842 s
2017-04-02	7.2 MB/s	13053 s	12 s	1314 s	14379 s
2017-04-03	6.9 MB/s	12878 s	14 s	1484 s	14376 s
2017-04-04	6.9 MB/s	12472 s	15 s	1514 s	14001 s

The stacked bar chart shows a clear picture of the whole latency by date or hour of day and its segments represent latency of each phase (processing, uploading and downloading). The segments can be shown or hidden according to needs. As seen in Figure 2, the uploading latency can be ignored in comparison with other twos; the downloading latency is largely unchanged from the firewall upgrading; and a maintenance activity of IntHub conducted on April 19, 2017 caused the longer latency of processing in the forthcoming days.

The proposed methods are evaluated and developed in the prototype system. In the future, we will automate the whole process from logging timestamps to explore the latencies. Furthermore, other satellite data quality functions, like data completeness checking, are also expected to be implemented based on this database and integrated into this Web portal.

4. CONCLUSIONS

Latency analysis provides data providers, data managers, and data users, clear and actionable insight on how to improve data transmission of near real-time satellite data, enhance its acquisition and management, and overcome performance and management issues. This kind of analysis also contributes to modernize the infrastructure of satellite data management. An additional NoSQL database offers flexible data models for better architecture and new analytics functions.

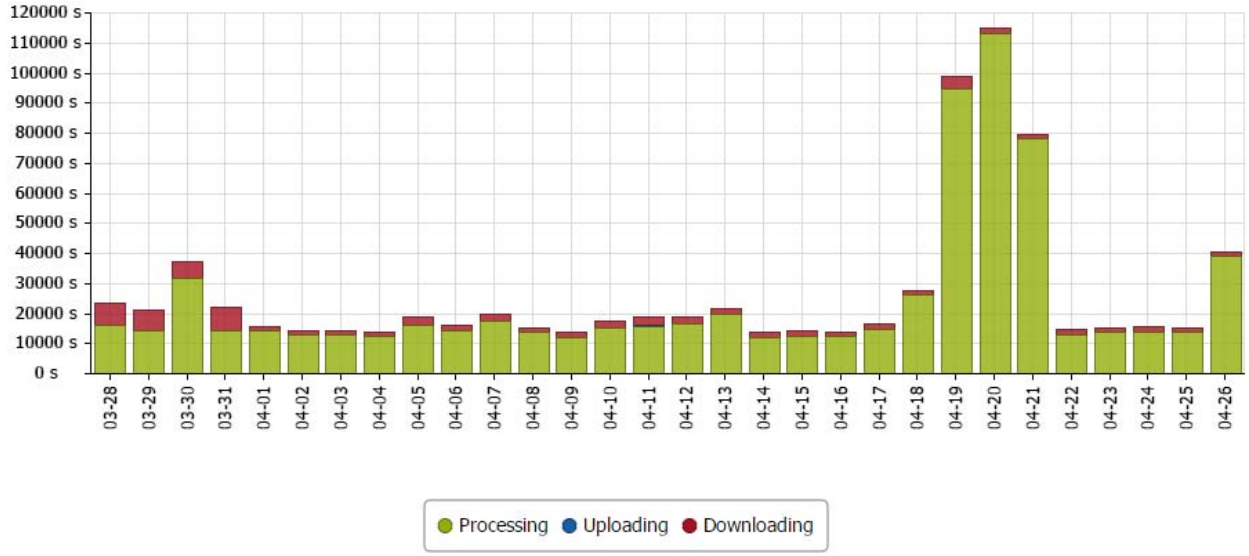

Figure 2 Sentinel 1A data latencies during 3/28/2017 – 4/26/2016

5. ACKNOWLEDGEMENTS

The manuscript contents are solely the opinions of the authors and do not constitute a statement of policy, decision, or position on behalf of NOAA or the U.S. government.

6. REFERENCES

[1] E. Berndt, A. Molthan, W. W. Vaughan, and K. Fuell (2017), Transforming satellite data into weather forecasts, Eos, 98, doi:10.1029/2017EO064449. Published on 05 January 2017.

[2] W. Han, and J. Brust, "Central satellite data repository supporting research and development", AGU Fall Meeting, 2015.

[3] A. Bouakaz, P. Fradet, and A. Girault, "Symbolic computation of the latency for dataflow graphs", Integrating Dataflow, Embedded computing and Architecture (IDEA'2016), 2016.

[4] W. Han and M. Jochum, "Near real-time satellite data quality monitoring and control," 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Beijing, 2016, pp. 206-209.

[5] A. Shi-Nash, and D. R. Hardoon, "Data analytics and predictive analytics in the era of big data", Internet of Things and Data Analytics Handbook, 2016, pp.329.

[6] W. Han, L. Di, G. Yu, Y. Shao, and L. Kang, "Investigating metrics of geospatial web services: The case of a CEOS federated catalog service for earth observation data", Computers & Geosciences, 92, 1-8, 2016.

[7] V.M. Hirota, R. Santos, J. Raddick, and A. Thakar, "Mining the SDSS SkyServer SQL queries log", SPIE Defense+ Security, International Society for Optics and Photonics, 2016, pp. 98510S-98510S.

[8] C.B. Jiang, I. Liu, Y.N. Chung, and J.S. Li, "Novel intrusion prediction mechanism based on honeypot log similarity", International Journal of Network Management, 26:156-175, 2016.

[9] I. Mavridis, H. Karatza, "Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark", Journal of Systems and Software, 125:133-151, 2017.

[10] B. Oonhawat and N. Nupairoj, "Hotspot management strategy for real-time log data in MongoDB," 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, 2017, pp. 221-227.

[11] S. Agarwal, and K. S. Rajan. "Performance analysis of MongoDB versus PostGIS/PostgreSQL databases for line intersection and point containment spatial queries", Spatial Information Research, 24(6): 671-677, 2016.

[12] K. Chodorow, "MongoDB: the definitive guide", O'Reilly Media, Inc., 2013.