NMSTREAM: A SCALABLE EVENT-DRIVEN ETL FRAMEWORK FOR PROCESSING HETEROGENEOUS STREAMING DATA

Fei Xiao [1], Chengming Li [1]*, Zheng Wu [1], Yinghao Wu [1]

[1] Chinese Academy of Surveying & Mapping, Beijing, China - xiaof22a@gmail.com, cmli@casm.ac.cn, wuzheng_gucas@163.com, wuyinghaohh@163.com

Commission IV, WG IV/6

KEY WORDS: Streaming data, Extract-Transform-Load, Apache Flume, Apache Cassandra

ABSTRACT:

ETL (Extraction-Transform-Load) tools, traditionally developed to operate offline on historical data for feeding Data-warehouses need to be enhanced to deal with continuously increased streaming data and be executed at network level during data streams acquisition. In this paper, a scalable and web-based ETL system called NMStream was presented. NMStream is based on event-driven architecture and designed for integrating distributed and heterogeneous streaming data by integrating the Apache Flume and Cassandra DB system, and the ETL processes were conducted through the Flume agent object. NMStream can be used for feeding traditional/real-time data-warehouses or data analytic tools in a stable and effective manner.

1. INTRODUCTION

The advancements of sensing technologies have dramatically improved the accuracy and spatiotemporal scope of the record. To better understand, protect and improve our living environment, a variety of sensors have been developed and deployed in many geoscience projects including disaster monitoring and assessment, climate change, ecosystem dynamics, and atmospheric pollution monitoring, which produce massive volumes of geospatial data, or big geoscience data (Li et al., 2015). The Sensor Web refers to the realization and development of a continuous, distributed, and cooperative data service system that aims at integrating and coordinating the multiple heterogeneous monitoring platforms to achieve complex tasks (Chen et al., 2013). The advancements of sensing technologies dramatically increase people's capabilities in acquiring geospatial data for building the Spatial Data Infrastructure (SDI) (Masser et al., 2005) and smart city (DE) (Craglia et al. 2012). Based on Yang et al. (2011), massive amounts of multi-dimensional data recording various physical phenomena are taken by the sensors across the globe, and these sensing data are collected rapidly with a daily increase rate of terabytes to petabytes. This increase is dramatically enhanced by novel crowd sourcing in situ ground-based sensor networks as well as the deployment of satellite systems which generates data with very high resolution (Zhao et al., 2012).

Several challenges should be faced for handling sensors and their data especially in emergency situations. First, sensors (both physical and social) are located in different networks and made available by different institutes and agencies. In this context, network configuration, sensor detection and discovery are difficult issues to be solved. Moreover, data produced by sensors are heterogeneous in structures (different types), in spatial and/or temporal granularities, in thematic. Therefore, there is the need of ETL jobs that can be applied on data streams for their reconciliation. There operations should be applied during data acquisition and bound with reactive capabilities in order to properly identify the relevant streams when abnormal events occur and undertake the proper actions. Finally, the specification and actuation of the ETL operations should be efficiently performed on-line and on fresh and timely data in order to properly handling big real-time data streams. All these technical requirements should be addressed in graphical, user-friendly environments supporting the user in the design and execution of the operations.

This paper is organized as follows: Section 1 introduces the background and objectives of this study. Section 2 describes some related work for data streaming ETL operations. The architecture and components of NMStream are documented in section 3. Section 4 introduces the data model and section 5 gives some demonstrations. The final conclusion ate given in section 6.

2. RELATED WORK

Many systems have been proposed for configuring programmable networks. Ahmed rt al. (2004) proposed a multidimensional structures or hypercubes to store and organized the streaming data with the goal of optimizing query response time. Tatbul (2008) developed a distributed stream processing engine that extends the first generation of data stream processing systems with advanced capabilities such as distributed operation, scalability and dynamic data and query modifications. However, all of these researches only focused on the data collection or query efficacy, but the ETL processes before feeding the data into databases. Qu et al (2017) proposed Hbelt system which tightly integrates the wide-column NoSQL database with a clustered & pipelined ETL engine. Mesiti et al (2016) developed a web-based ETL system called StreamLoader to integrate heterogeneous sensor data. Furthermore, a lot of open-source projects for streaming data collection and processing has been released in recent years, such as Yahoo S4 (Chauhan et al., 2012), Apache Spark Streaming and Apache Storm (Lqbal et al., 2015). However, all

* Corresponding author

of them are quite complex to configure, seldom provide web GUIs for designing and monitoring dataflows and are not integrated in a single tool, which limits their use in the management of emergency situations.

In this paper, we proposed NMStream, a highly scalable web-based ETL framework for heterogeneous streaming data collection and ETL operations during data stream acquisition stage on a programmable network. NMStream leverages different open-source software such as Quartz Scheduler for scheduling observation collection job, Apache Flume for transforming observing event, and Apache Cassandra for data storage and query. NMStream enables users to add new streaming data sources dynamically in the runtime environment without the need to stop or redeploy the whole system. Furthermore, NMStream provides a user-friend graphic interface for user to facilitate creating and editing ETL workflow in a drag-and-drop manner. In the remainder, we first briefly introduced the Apache Flume and Cassandra systems, and then we introduced the overall architecture of NMStream as well as its core components. At last, we illustrated the visual user interfaces as a demonstration and gave the conclusion and future work.

3. SYSTEM INTRODUCTION

NMStream leverages different open-source software such as Quartz Scheduler for scheduling observation collection job, Apache Flume for transforming observing event, and Apache Cassandra for data storage and query. The detailed introduction of these components were introduced below.

3.1 Apache Flume

Apache Flume was originally developed as a distributed and reliable service for productively gathering, aggregating, and moving various types of streaming data into the Hadoop Distributed File System (HDFS). It has a straightforward and adaptable engineering in light of streaming data streams; and is robust and fault tolerant with tunable dependability instruments for failover and recuperation. A Flume workflow can be constructed by linking one or more Agents, which is composed of Source, Sink, Channel and Interceptor. Figure 1 illustrates the structure of a Flume workflow consists of two agents. The messages flowing in Agent are called Event. Each Agent has one Source for receiving events and at least one Sink for processing events. Flume utilizes Channel based transactions to ensure reliable events delivery and Inceptors for filtering events.

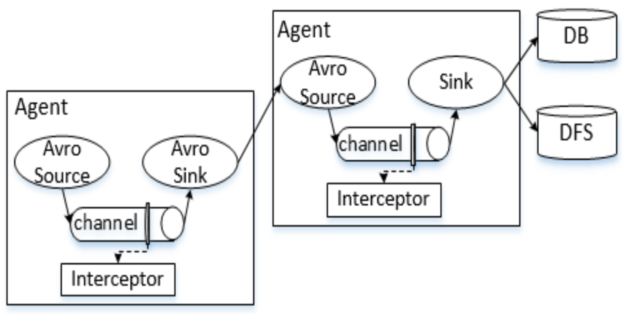


Figure 1. Structure of Flume workflow

3.2 Apache Cassandra

Cassandra is a fully distributed and highly scalable P2P-based NoSQL database developed within Facebook and open-sourced as Apache Incubator project on 2009. Cassandra DB is built on Amazon’s Dynamo and Google BigTable. Cassandra clusters can run on different commodity servers and even across multiple data centers. This properties gives it a linear horizontal scalability. In Cassandra, the nodes of the cluster are seen as parts of a ring where each node contains some chunks of data. The rows of data are partitioned based on their primary key. Cassandra uses a special primary key called a composite key to represent wide rows, also called partitions. The composite key is used to determine the nodes on which rows are stored and can itself consist of multiple columns. The clustering columns are used to control how data is sorted for storage within a partition.

3.3 NMStream architecture

Figure 2 shows the overall architecture of the NMStream system, which can be divided into three core components: Job Executor, Job Scheduler and Flume Server.

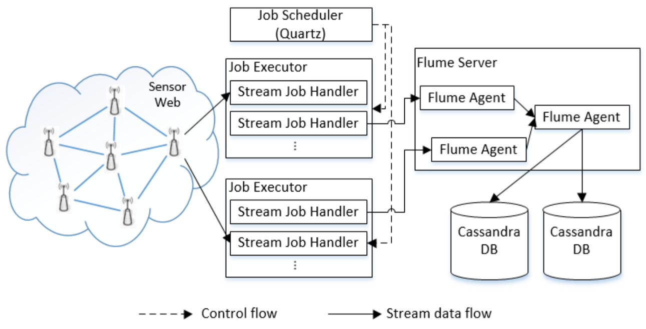


Figure 2.NMStream architecture

- Stream Job Handler and Job Executor

The Job Executor is an independent server for collecting data from sensor web environment. The processing unit of Executor is call Job Handler, and job handler corresponds to one type of streaming data source. During data collection lifestyle, the job handler first transforms the observing data into Flume event object, which was then delivered to a registered Flume Agent for processing. Furthermore, the job handler was embedded an Avro client, which is responsible for sending the event object to remote Avro sources through the networks. The Job Executor provides the container for job handlers to execute and monitor the executing status of them.

- Job Scheduler

The responsibility of Job Scheduler is to control the execution of job handler. The Quartz scheduler toolkit, which is a job scheduling library that can be integrated into a wide variety of Java applications, was applied to schedule the execution of handlers. There were two components in the Job Scheduler: a scheduler and a graphical control panel. The scheduler is used to manage the execution of job handler based on the execution parameters configured by users. In order to start a new data collection job, the administrator need to set the name of job handler class, the required parameters and a Cron-Expression which is string that is actually consists of seven sub-expressions, that describe individual details of the schedule such as ‘every 5 minutes between 9:00 am and 10:00 am on every Monday’. The graphical control panel provides a user-friend operation UI for users to configure and manage the job handlers as well as monitor the real-time execution status.

- Flume Agent and Flume Server

The Flume agent is responsible for managing the ETL processes. Figure 3 illustrates the overall structure of a Flume agent used by NMStream. As shown in Figure 3, the Job Handler delivers the event object to Agent source (AvroSource) through an embedded RpcClient. As introduced in section 3.1, the Flume agent consists of four components: Source, Sink, Channel and Interceptors. Each agent can be bind with one or more interceptors which are used to transform or filter event, and these interceptors can be connected to form a data flow. Upon receipt of the event objects, the Avro Source first feeds them into channel and then these events will be filtered and transformed by the interceptor(s) and finally reach to the SDESink. The interceptors can chained together to form an data flow chain. The SDESink, which is used to write data into database through SDE (Spatial Data Engine).

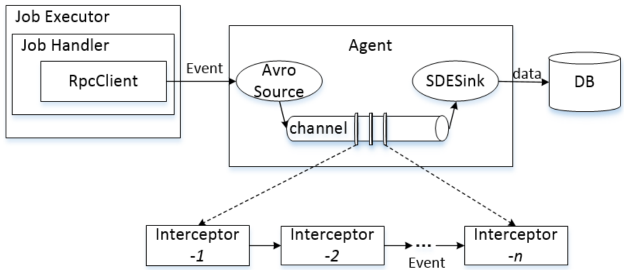


Figure 3.The structure of NMStream Agent

What should be emphasized here is that all of the Agent components can be added, removed and configured dynamically without the need to stop or restart the NMStream system. This feature is very useful especially for an unstable and dynamic sensor web environment. For example, if the observation parameters of a sensor was changed, the administrator only need to adjust the corresponding ETL Agent in runtime without to affect the other ETL components.

The whole system was highly distributed. All of these three components can be deployed on different machines and interact with each through message-based systems based on TCP/IP protocol. This mechanism has two advantages. First, it is fault-tolerant because the communication between the components is asynchronized and each node's failure won't affect the others. Second, it is dynamically scalable. The system user can add, change or remove every service components based on their requirement without the need to stop the whole system.

4. NMSTREAM STREAMING DATA MODEL

Figure 4 illustrates the logic data model of NMStream for organizing and representing sensor web data. The streaming data sources are represented by Catalog object in NMStream. Each catalog owns a list of station objects, which represents a sensor equipment. The stations are uniquely identified by the 'stationid' attribute. The observation was represented by the Event object, which is a time-series data updated with custom time interval.

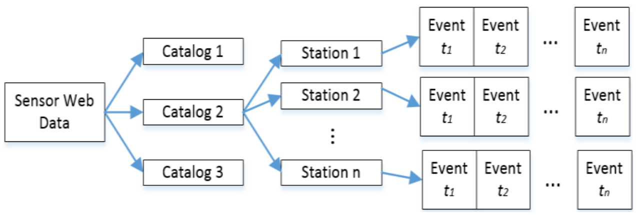


Figure 4: NMStream architecture

Figure 5 illustrates the physical data model for storing streaming data in Cassandra database. As shown in Figure 5, there are two basic tables (catalog and station) and a series of event tables: the 'catalog' table contains records which represent one type of observing data, i.e. air quality, water quality, weather, while the 'station' table stores the detailed information of a sensor. As introduced in Figure 4, the catalog and station has a one-to-many relationship. The 'event' table in Figure is an abstract table which can be defined by user based on the catalog, each catalog item corresponds to one event table, and the items information of catalog table has the detailed information of event data columns.

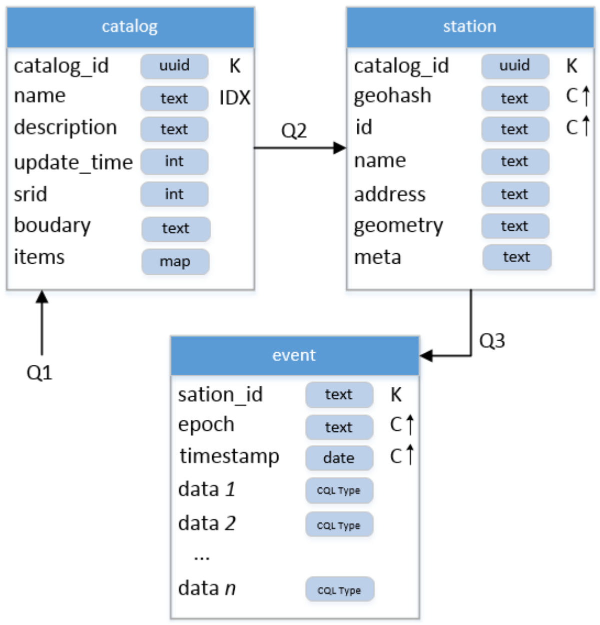


Figure 5: The physical data model for storing streaming data in Cassandra database. K means partition key, C ↑ mean clustering column sorted in the ASC mode, IDX means secondary index.

The query process of streaming data can be divided into three sub procedures, which were marked in Figure 5 as Q1-3. First, user should build the query Q1 to get the catalog information of required streaming data source. One Q1 example for querying the 'airquality' data source is shown below:

*SELECT * FROM catalog WHERE name='airquality' allow filtering;*

Once the user get the detailed information of a catalog, then Q2 query can be built to retrieve all stations belong to this catalog through the 'catalog_id' column. If the user want to get stations in a defined boundary, the 'geohash' value should be set:

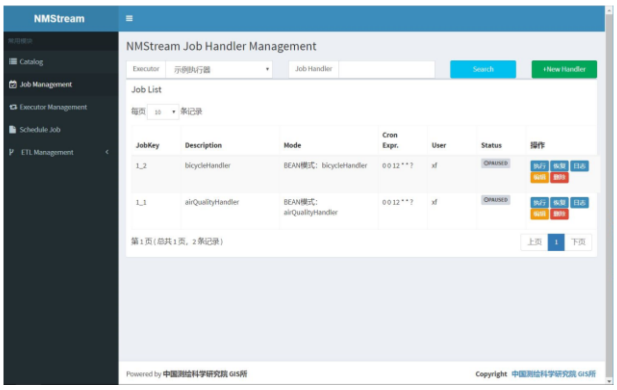
*SELECT * FROM station where catalog_id='...' and geohash> and geohash< allow filtering.*

At last, user can build Q3 query using the 'station_id' and 'date' parameters to get required observing data. The example illustrates how to get observation of station '001' during the period of 2018-04-28 08:00:00 and 2018-04-28 12:00:00:

*SELECT * FROM even WHERE station_id='001' and epoch='20180405' and time>='2018-04-28 08:00:00' and time <=' 2018-04-28 12:00:00';*

5. SYSTEM DEMONSTRATION

By using a visual interface in Figure 6, users can drag-and-drop data sources and apply the proposed operations on streams, i.e. adding new job handler base on requirement in the Job Management panel and checking the job’s current status. As shown in figure 6(a), there are two job handlers with status running. Figure 6(b) illustrates the Agent editor panel, where system administrator can edit the ETL procedure on the fly in a drag-and drop manner. The ETL procedure in figure contains one Avro Source for data receipt, one Sink for writing data into database, and three chained interceptors for transforming and filtering event.

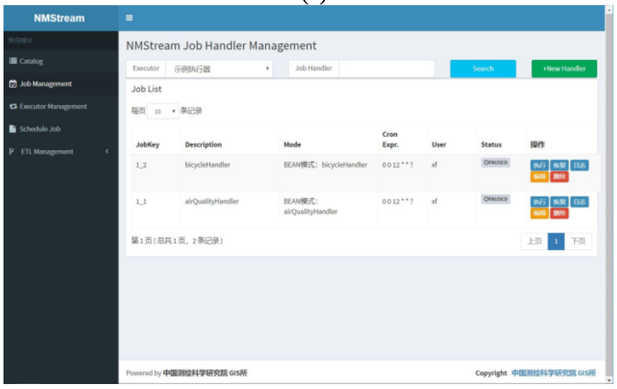


NMStream Job Handler Management

JobKey	Description	Mode	Cron Expr.	User	Status	操作
1_2	bicycleHandler	BEAN模式：bicycleHandler	0 0 12 * * ?	xf	PAUSED	执行 恢复 日志 编辑 删除
1_1	airQualityHandler	BEAN模式：airQualityHandler	0 0 12 * * ?	xf	PAUSED	执行 恢复 日志 编辑 删除

第1页(总共1页，2条记录)

(a)



NMStream Job Handler Management

JobKey	Description	Mode	Cron Expr.	User	Status	操作
1_2	bicycleHandler	BEAN模式：bicycleHandler	0 0 12 * * ?	xf	PAUSED	执行 恢复 日志 编辑 删除
1_1	airQualityHandler	BEAN模式：airQualityHandler	0 0 12 * * ?	xf	PAUSED	执行 恢复 日志 编辑 删除

第1页(总共1页，2条记录)

(b)

Figure 6: User interface of NMSream

The demonstration will thus prove the flexibility of the developed system in the specification of data flow to be executed at network level and actuated on the fly. All the ETL operations that have been considered can be applied on-line on fresh data arriving from sensors of different types. Different interceptors have been included in the dataflow in order to perform the ETL workflow.

6. CONCLUSION

In this paper, a scalable event-driven ETL system called NMStream was presented for integrating distributed and heterogeneous streaming data sources by leveraging the Apache Flume and Cassandra DB technologies. The NMStream consists of three main components: Job Scheduler, Job Executor and Flume Server. The Job Scheduler is developed based on Quartz scheduling toolkit and the Flume agent model was applied in this study for ETL operations. Furthermore, a scalable streaming data model was designed for effective management of distributed streaming data in NoSQL databases. Finally, a user-friend visual interactive environment was designed to facilitate the interaction between system and administrator. NMStream can be used for feeding traditional/real-time data-warehouses or data analytic tools in a stable and effective manner.

ACKNOWLEDGEMENTS

This study was funded by the Basic Research Funding of Chinese Academy of Surveying and Mapping (7771804).

REFERENCES

Dubayah, R.O., Swatantran, A., Huang, W., Duncanson, L., Tang, H.,Johnson, K.,Dunne, J.O., and Hurtt, G.C., 2017. CMS: LiDAR-derived Biomass, Canopy Height and Cover, Sonoma County, California, 2013. ORNL DAAC, Oak Ridge, Tennessee, USA https://doi.org/10.3334/ORNLDAAC/1523.

Gago-Silva, A., 2016. GRASS GIS in Grid Environment. Figshare https://doi.org/10.6084/m9.figshare.3188950.

GRASS Development Team, 2017. Geographic Resources Analysis Support System (GRASS) Software, Open Source Geospatial Foundation http://grass.osgeo.org (20 September 2017).

GRASS Development Team, 2015. Geographic Resources Analysis Support System (GRASS) Software, Version 6.4. Open Source Geospatial Foundation http://grass.osgeo.org (1 June 2017).

Lennert, M. and GRASS Development Team, 2017. Addon i.segment.stats. Geographic Resources Analysis Support System (GRASS) Software, Version 7.2, Open Source Geospatial Foundation https://grass.osgeo.org/grass7/manuals/addons/i.segment.stats.html (1 June 2017).

Maas, A., Rottensteiner, F., Heipke, C., 2017. Classification under label noise using outdated maps. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. IV-1/W1, pp. 215-222, doi.org/10.5194/isprs-annals-IV-1-W1-215-2017.

Smith, J., 1987a. Close range photogrammetry for analyzing distressed trees. *Photogrammetria*, 42(1), pp. 47-56.

Smith, J., 1987b. Economic printing of color orthophotos. Report KRL-01234, Kennedy Research Laboratories, Arlington, VA, USA.

Smith, J., 1989. *Space Data from Earth Sciences*. Elsevier, Amsterdam, pp. 321-332.

Smith, J., 2000. Remote sensing to predict volcano outbursts. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. XXVII-B1, pp. 456-469.