International Journal of Scientific Engineering and Applied Science (IJSEAS) – Volume-3, Issue-6, June 2017
ISSN: 2395-3470
www.ijseas.com

Real Time Big Data Analysis Using Apache Flink

[1]Gireesh Babu C N, [1]Anu Pokhrel, [1]Ashwini V, [2]Thungamani M,
[1]Department of Information Science and Engineering, BMSIT&M, Bengaluru
[2]University of Horticulture Sciences, GKVK, Bengaluru
gireeshbabu@bmsit.in, anupokhrel02@gmail.com, ashwinivsonu@gmail.com, thungamani_k@rediffmail.com

Abstract

Data type and the amount in the society is growing in a speedy fashion which is caused by emerging new services as cloud computing, internet of things and location-based services, the era of big data has arrived. As data has been fundamental resource, how to manage and utilize big data better has attracted much attention. Especially, with the development of internet of things(IOT), the processing of large amount of real-time data has become a great challenge in research and applications. Recently, cloud computing technology has attracted much attention with high-performance, but how to use cloud computing technology for large-scale real-time data processing has not been studied. This paper basically studies on the application known as SMART and all the components used in it. Moreover, it presents an overview on Apache Flink.

Keywords: *SMART, data-processing, Apache Spark, Apache Flink.*

I. INTRODUCTION

Big data[1] is a collection of large datasets that are so large or complex that traditional data processing application software is not sufficient to deal with them. Challenges include capture, storage, analysis, data curation, search, sharing, transfer, visualization, querying, updating and information privacy. There is doubt that the amount of data now available are indeed large, but that's not the most relevant characteristic of this new data ecosystem. Analysis of data sets can find new correlations to "spot business trends, prevent diseases, combat crime and so on. Scientists, business executives, practitioners of medicine, advertising and governments alike regularly have difficulties with large data-sets in areas including Internet search, finance, urban informatics, and business informatics. Scientists encounter various limitations including several environmental search, biology, and environmental research. MapReduce[2] (MR) is a programming framework adopted by many companies for Big Data processing, that executes "map, merge and reduce" data transformations. It addresses applications only based in batch model, normally in homogeneous environments such as large clusters in data centers.

Hadoop[4], a popular MR implementation, is considered an industrial standard to Big Data, but it does not provide services that can be composed and combined in multi-cloud or hybrid infrastructures to support different types of applications. Other transformations, such as event driven systems are hence necessary.

This work considers a large variety of data sources, ranging from wireless sensor nodes instrumenting open and indoor environments to large corporate databases, passing by social networks and broadcast media, where there is a clear need for standardization. Observing this large domain spectrum, this work proposes a modular framework for Big Data analysis called Small & Medium-sized Enterprise Data Analytic in Real Time (SMART) that aims to simplify the deployment of Big Data services by Small & Medium sized Enterprises (SMEs).

Apache Flink is an open source stream processing framework developed by the Apache Software Foundation. The core of Apache Flink is a distributed streaming dataflow engine written in Java and Scala. Flink executes arbitrary dataflow programs in a data parallel and pipelined manner. Parallel functions help creating more time for processing. Flink's pipelined runtime system enables the execution of bulk/batch and stream processing programs. Furthermore, Flink's runtime supports the execution of iterative algorithms respectively. Flink provides a high-throughput, low-latency streaming engine as well as support for event-time processing[7] and state management. Flink applications are fault-tolerant in the event of machine failure. Programs can be written in Java, Scala, Python, and SQL and are automatically compiled and optimized into dataflow programs that are executed in a cluster or cloud environment[6].

II. RELATED WORK

Previous work can be divided into the following topics: frameworks for Big Data analysis, and techniques for managing data and application deployment in hybrid infrastructure and across multiple clouds.

A. Frameworks for big data analysis

MR is a programming framework that abstracts the complexity of parallel applications by partitioning and scattering data sets across hundreds or thousands of machines, and by bringing computation and data closer together. *Map* and *Reduce* phases are handled by the programmer, whereas the *Shuffle* is performed while a task is being carried out. The data is serialized and distributed across machines that compose the Distributed File System (DFS). The application executions are represented as a Directed Acyclic Graph (DAG) under a batch processing model, which can provide high-latency response when applied to stream processing where data arrives constantly to be processed.

B. Hybrid infrastructure and multi-cloud

Organizations are increasingly relying on infrastructure from multiple providers as a means to increase tolerance to failures and avoid provider lock-in. When considering multiple clouds (*i.e.* hereafter also termed as multi-cloud), application deployment becomes complex as each individual cloud may have specific configuration parameters, and its resource availability and utilization can change dynamically. There is therefore a need for automatic configuration of complex cloud services at different abstraction levels. Users need means for efficiently mapping the computing requirements of their services to available resources. The lack of knowledge about the underlying infrastructure can lead to inefficient allocations where either allocated resources are not fully used or the Quality of Service (QoS) of applications is compromised due to allocating insufficient resources. As optimal allocation is difficult to achieve, an approximation strategy is generally acceptable. Enterprises and governments often organize their data across multiple cloud sites or availability zones in order to maintain resource proximity; create data stores with organizations that share common goals; and keep data replicas across regions for redundancy purposes. However, under certain scenarios data needs to be analyzed globally. When considering MR, one way of doing this is to aggregate data in a single data center, and another is to execute individual instances of MR jobs on each data set separately and then aggregate the results.
Flink follows a technique that applies data-stream processing as the model for batch processing, real- time analysis and continuous streams both in the programming model and in the execution engine. In combination with durable message queues that allow quasi-arbitrary replay of data streams (like Apache Kafka or Amazon Kinesis), stream processing programs make no distinction between processing the latest events in real-time, continuously aggregating data periodically in large windows, or processing terabytes of historical data. Instead, these different types of computations simply start their processing at different points in the durable stream, and maintain different forms of state during the computation. Through a flexible windowing mechanism, Flink programs can compute both early and approximate, as well as delayed and accurate, results in the same operation, obviating the need to combine different systems for the two use cases. Flink supports different notions of time (event-time, ingestion-time, processing-time) in order to give programmers high flexibility in defining how events should be correlated.

III. INFRASTRUCTURE MODEL

Different cloud infrastructures have their own configuration parameters, and the availability and performance of offered resources can change dynamically due to several factors, including the degree of over-commitment that a provider employs. In this context, solutions are needed for the automatic configuration of complex cloud services. Cloud infrastructure comprising heterogeneous hardware environments may need the specification of configuration parameters at several levels such as the operating systems, service containers and network capabilities. As users, who need to execute applications, may not know how to map their requirements to available resources, this lack of knowledge about the cloud provider infrastructure will lead either to overestimating or underestimating required capacity; both are equally bad as the former leads to waste of resources whereas the second sacrifices QoS.

Hybrid infrastructure, is a collection of many cloud providers with heterogeneous environments and configurations, which often needs something which can manage data inputs. The orchestrator must be decentralized in order to improve data distribution in the network. The infrastructure enables the use of highly heterogeneous machines. When considering the use of a public cloud to extend the capacity of a community cloud, or desktop grid, several scenarios and data strategies are possible. The extent to which a set of data-distribution strategies is applicable to a given scenario depends on how much bandwidth is available. If one considers MR, two distinct DFS implementations may be required to handle data distribution in two scenarios, namely low-bandwidth and high-bandwidth.

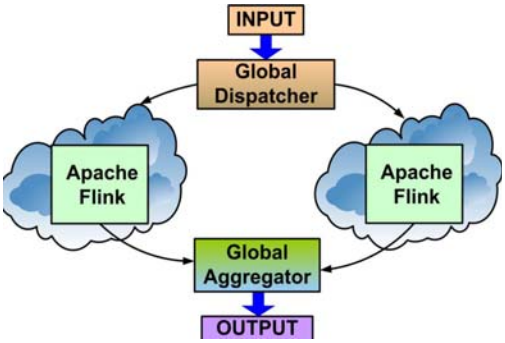


Fig 1. Smart Architecture

The work performed on MR for hybrid Fig 1 illustrates the solution proposed here to model a hybrid system which depicts a Global Dispatcher and Global Aggregator to be used on the infrastructure for services that use multiple data abstractions. The Global Dispatcher which is located outside the cloud has all the middleware functions for handling task assignment, and management of user-provided data. It is a centralized data storage system that manages policies for splitting data and distributing it in accordance with the needs of each system.. The Global Aggregator takes data output from both systems and merges them in order to obtain the final data set.

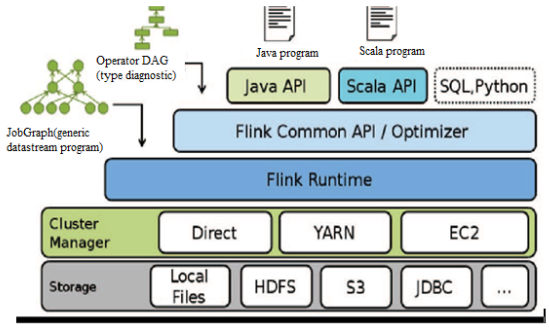


Fig 2. Apache Flink component stack

Apache Flink, which is the base infrastructure of the SMART framework as shown in Fig 2. Its flexible pipeline enables several map-reduce and extended functions like Map, MapPartition, Reduce, Aggregate, Join and Iterative. It can be used in order to allow this cloud extension. The setting will be transparent to users because a middleware in a top level abstracts the complexity away from the users.The different layers of the stack build on top of each otherand raise the abstraction level of the program representations they accept:

- The API layer implements multiple APIs that create operator DAGs for their programs. Each API needs to provide utilities like serializers, comparators which describes the interaction between its data types and the runtime. All programming APIs are translated to an intermediate program representation that is compiled and optimized via a cost-based optimizer.
- The Flink Common API and Optimizer layer takes programs in the form of operator DAGs. The operators are specific (*e.g.*, Map, Join, Filter, Reduce, FlatMap, MapPartition, ReduceGroup, Aggregate, Union, Cross, etc) and the data is in non-uniform type. The concrete types and their interaction with the runtime are specified by the higher layers.
- The Flink Runtime layer receives a program in the form of a JobGraph. A JobGraph is a generic parallel data flow with arbitrary tasks that consume and produce data streams. The runtime is designed to perform very well both in settings with abundant memory and where memory is scarce.

SMART approach take advantage of cloud, multi-cloud and hybrid infrastructures to provide support for SME service operation.The heterogeneous resources, in this scale, impose challenges to the data management and synchronizations, task distributions, result aggregations and failure tolerance mechanisms. The strategy to avoid the input data aggregation in a single data center for Big Data analysis promotes less data movement and reduces bandwidth needs. The new architecture improves SME competitiveness, because it allows them to choose the best resources with lowest prices. Many of the Flink features listed below–state management, handling of out-of-order data, flexible[3] windowing–are essential for computing accurate results on unbounded datasets and are enabled by Flink's streaming execution model.

Apache Flink follows a technique that embraces data-streaming and batch streaming as shown in Fig 3, as the unifying model for real-time analysis, continuous streams, and batch processing both in the programming model and in the execution engine. In combination with durable message queues that allow quasi-arbitrary replay of data streams (like Apache Kafka or Amazon Kinesis), stream processing programs make no distinction between processing the latest events in real-time, continuously aggregating data periodically in large windows, or processing terabytes of historical data. Instead, these different types of computations simply start their processing at different points in the durable stream, and maintain different forms of state during the computation. Flink programs can compute both early and approximate through a highly flexible windowing mechanism, as well as delayed and accurate, results in the same operation, ignoring the need to combine different systems for the two use cases. Flink supports different notions of time (event-time, ingestion-time, processing-time) in order to give

programmers high flexibility in defining how events should be correlated.

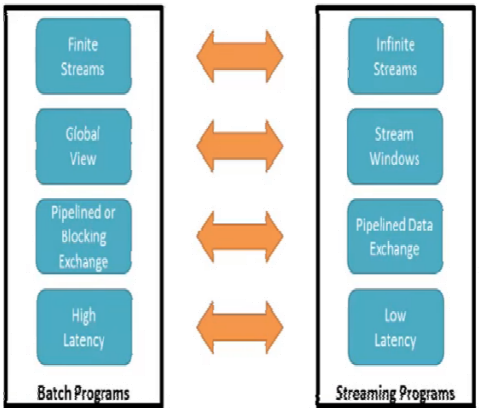


Fig 3. Batch vs Stream Program

Flink has idea that there is a need for dedicated batch processing (dealing with static data sets). Complex queries over static data are still a good match for a batch processing abstraction. Moreover, batch processing is still needed for legacy implementations of streaming use cases, and for analysis applications where no efficient algorithms are yet known that perform this kind of processing on streaming data. Batch programs are special cases of streaming programs, where the stream is finite, and the order and time of records does not matter (all records implicitly belong to one all-encompassing window). However, to support batch use cases with competitive ease and performance, Flink has a specialized API for processing static data sets, uses specialized data structures and algorithms for the batch versions of operators like join or grouping, and uses dedicated scheduling strategies. The result is that Flink presents itself as a full-fledged and efficient batch processor on top of a streaming runtime, including libraries for graph analysis and machine learning. Flink is a top-level project of the Apache Software Foundation that is developed and supported by a large and lively community, and is used in production in several companies.

The core of Flink is the distributed dataflow engine, which executes dataflow programs.There are two core APIs in Flink: the DataSet API for processing finite data sets (often referred to as batch processing), and the DataStream API for processing potentially unbounded data streams (often referred to as stream processing). Flink's core runtime engine can be seen as a streaming dataflow engine, and both the DataSet and DataStream APIs create runtime programs executable by the engine. On top of the core APIs, Flink bundles domain-specific libraries and APIs that generate DataSet and DataStream API programs, currently, FlinkML for machine learning, Gelly for graph processing and Table for SQL-like operations.

IV RESULTS AND DISCUSSIONS

A. Batch Processing:

For batch processing we used Terasort benchmark which was initially developed as a benchmark for evaluating Apache Hadoop Map reduce jobs. This benchmark has been used by many performance validations and competitions, hence considered as one of the most popular applications to benchmark batch processing applications. It was enhanced further to benchmark Apache Flink and Apache Spark by modifying the Map Reduce functionality. His experimental analysis provided a great foundation to start our analysis. We could reuse his experimental code developed for Terasort as it was already analyzed and accepted as a fair analysis to compare the performance by the open source communities.

B. Stream Processing[5]:

For stream processing we researched two main stream processing benchmarks. They were Intel HiBench Streaming benchmark and Yahoo Stream benchmark, which were recently developed to cater the requirement of comparing stream data processing. We evaluated Yahoo Stream benchmark as it includes a more realistic demo of a sample application which simulates an advertising campaign. But We Karamalized HiBench to make it reproducible via Karamel.

C. Word-count in java:

Flink program programs look like regular programs that transform collections of data. Each program consists of the same basic parts:

- Obtain an execution environment,
- Load/create the initial data,
- Specify transformations on this data,
- Specify where to put the results of your computations,
- Trigger the program execution

Depending on the type of data sources, i.e. bounded or unbounded sources, you would either write a batch program or a streaming program where the DataSet API is used for batch and the DataStream API is used for streaming. This guide will introduce the basic concepts that are common to both APIs but please see our Streaming Guide and Batch Guide for concrete information about writing programs with each API.

Program:-

```
ExecutionEnvironment env =
  ExecutionEnvironment.getExecutionEnvironment();

DataSet<String> text = readTextFile (input);

DataSet<Tuple2<String, Integer>> counts= text
  .map (l -> l.split("\\W+"))
  .flatMap ((String[] tokens,
             Collector<Tuple2<String, Integer>> out) -> {
    Arrays.stream(tokens)
    .filter(t -> t.length() > 0)
    .forEach(t -> out.collect(new Tuple2<>(t, 1)));
  })
  .groupBy(0)
  .sum(1);

env.execute("Word Count Example");
```

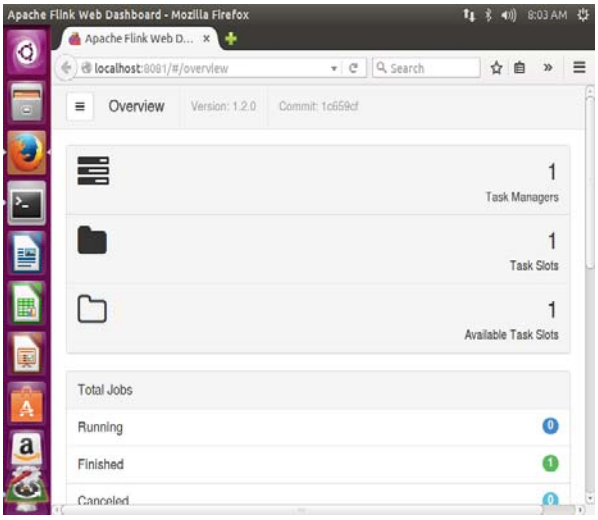


Fig 6. Apache Flink Web Dashboard

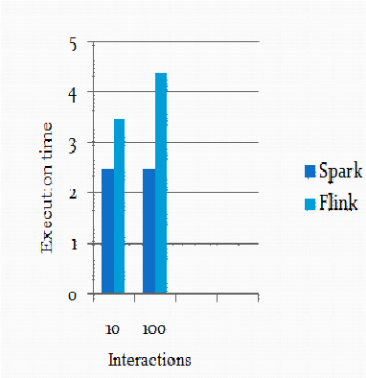
Apache Flink Dashboard basically depicts the overview of the running programs in the system in Fig 6. As shown in figure, we can see the no of running, finished and cancelled jobs in the system. We get an overview of the cluster resources and running jobs. Likewise we can also see the task slots, task managers and available task slots.

In order to face the emerging challenges in cloud-based Big Data processing, this work presented a framework consisting of composable data-analysis services that can be combined to address needs of specific applications. Focusing on applications for small and medium-sized organizations, the framework offers a flexible and lightweight approach that allows these organizations to take advantage of Big Data analysis in the cloud without incurring in the maintenance of heavy cloud infrastructures. Another important aspect to be highlighted is that of handling heterogeneous data sources, which makes the proposal applicable to a great number of companies and organizations running business in very different domains.

Preliminary results show good scalability of the SMART proposal, and the profile execution does not change with workload or host number. In streaming systems, the performance is workload sensitive which indicates a need for more detailed evaluation. The SMART implementation achieves better performance than Spark for CPU-intensive applications, and a workload increase does not have important impacts on the system performance. In large scale, the SMART simulation has a similar performance for large workloads in data-intensive applications.

V CONCLUSION

Focusing on applications for small and medium-sized organizations, the framework offers a flexible and lightweight approach that allows these organizations to take advantage of Big Data analysis in the cloud without incurring in the maintenance of heavy cloud infrastructures. Another important aspect to be highlighted is that of handling heterogeneous data sources, which makes the proposal applicable to a great number of companies and organizations running business in very different domains. Preliminary results show good scalability of the SMART proposal, and the profile execution does not change with workload or host number. In streaming systems, the performance is workload sensitive which indicates a need for more detailed evaluation. The SMART implementation achieves better performance than Spark for CPU-intensive applications, and a workload increase does not have important impacts on the system performance. In large scale, the SMART simulation has a similar performance for large workloads in data-intensive applications.



Also, we can totally say that Apache Flink works quite better than Apache spark in terms of their execution time an also the rate of performance. Below shows a small figure depicting the graph between the two.

REFERENCES

[1] M. Stonebraker *et al.*, "Intel "Big Data" Science and Technology Center Vision and Execution Plan," *SIGMOD Rec.*, vol. 4data2, no. 1, pp. 44–49, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2481528.2481537

[2] J. C. S. Anjos *et al.*, "BIGhybrid – A Toolkit for Simulating MapReduce in Hybrid Infrastructures," in *Computer Architecture and High Performance Computing Workshop (SBACPADW), 2014 International Symposium on*, Oct 2014, pp.132–137.

[3] J. Dean and S. Ghemawat, "MapReduce - A Flexible Data Processing Tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[4] T. White, *Hadoop - The Definitive Guide*, 3rd ed. O'Reilly Media, Inc., 2012, vol. 1.

[5] M. Rychly *et al.*, "Scheduling Decisions in Stream Processing on Heterogeneous Clusters," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2014 Eighth International Conference on*, July 2014, pp. 614–619.

[6] D.-H. Le *et al.*, "SALSA: A Framework for Dynamic Configuration of Cloud Services," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 146–153.

[7] Pooja K.S *et al.,"* Complex Event Processing In Smart Homes" in International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015 ISSN: 2395-3470.