International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 2, March - April 2017 ISSN 2278-6856

Spark Streaming through Dynamic Batch Sizing

[1]Unnati Tandel, [2]Shyam Deshmukh, [3]Vatika Sharma

[1]GTU PG School, Ghandhinagar

[2]CDAC, Pune

[3]I-Verve company, Hadoop Developer, Ahmedabad

Abstract—*Handling big data means to handle huge data by dividing the data into batches and then data is being processed as per spark engine. Spark basically an extension of Hadoop MapReduce.This thesis focuses on how to provide dynamic batch sizing(Instead of fixed slots) as per user choice with increased performance.*
Keywords—Streaming; Spark; BatchSizing; Spark Streaming ; Dynamic Batch

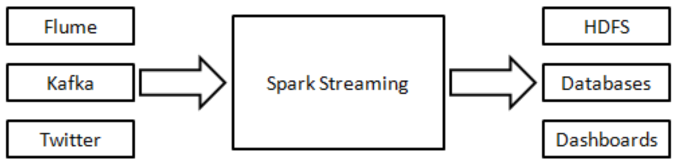
1.INTRODUCTION

There are without doubt many several approaches are available for the system to deal with the real-time data records before it stored into database for example foremost common open supply platforms for this apache spark. There are two elementary attributes of information stream process. First, every and each record within the system should have a timestamp that is done in statically. In this cases, Streaming were created at compile time. Second, every and each record is processed because it arrives. These second attributes guarantee a system that may react to the contents of each record, and may correlate across multiple records over time, even right down to time unit latency. In distinction, approaches like Spark Streaming method information streams in batches, wherever every batch contains a group of events that arrived over the batch amount (regardless of once the information were truly created). This can be fine for a few applications like easy counts and ETL into Hadoop, however the shortage of true record-by-record processes makes stream process and time-series analytics not possible.

In Existing Proposed System Recently projected frameworks have chosen to treat stream processing as fixed size batches of received Streaming Data. These types of framework set static batch size according to system workload characteristics which is done as Offline Profile Creation. In offline Creation, batch interval build based on cluster resources like memory, CPU, Workload characteristics, etc. If Any changes in the cluster resources (e.g., failed nodes, stragglers, new resources, etc.) or workload characteristics (e.g., changes in the number of aggregation keys, etc.) would change the actual behavior of the workload thus created profile will be useless. If unpredictable Incoming data is arrive into the system so there are some limitations (i) As Batch Interval is fixed, it becomes difficult to handle the unpredictable incoming data rate (ii) Increase in Latency. Distributed system is used some stream processing for achieving high throughput and low-latency. In existing stream data processing based on static data size. Data size based and fixed batch time based techniques have two challenges viz., profile modification in case of cluster resources and workload characteristics in case of unpredictable data rates as well types of workloads .These leads to increase latency and decrease throughput. A statically set batch size may either incur unnecessarily high latency under low load, or may not be enough to handle surges in data rates, causing the system to destabilize. Few existing dynamic batch sizing techniques are less prone to these two challenges as they are based on load shedding and offline learning. In load shedding, it loss the data which is not an option and aprior provisioning of resources for handling unpredictably high loads can be expensive. So making use of dynamic batch sizing we overcome these problems. This will improve the result of the throughput and the latency. In this work, we are proposing a control algorithm which will handle the above challenges and dynamically set the batch interval time. We will compare default stream processing with modified Spark stream processing framework. This will help us to improve the stream processing by implementing on it spark stream processing framework.

2.HOW SPARK STREAMING WORKS

Apache Spark is most important Open Source Cluster in Now Days as Computing Framework. Spark is used as an interface for Programming Cluster and gives the Result of Fault-tolerance and data Parallelism.
In Spark Streaming, there are three main events happen which are as following:



(i) Input Data are comes from many sources like Twitter Live
Data, Kafka and Flume.

(ii)Then, These Input Data are divided into some Batches.

(iii) These Batches of Data are processed by Spark Engine. It generates Final Output of Stream Data in Batches which are placed into HDFS, File System and Database.

Architecture of Spark

Spark is tops level project for Apache Software Foundation. It provides more number of programming language and spark support for the storage system.

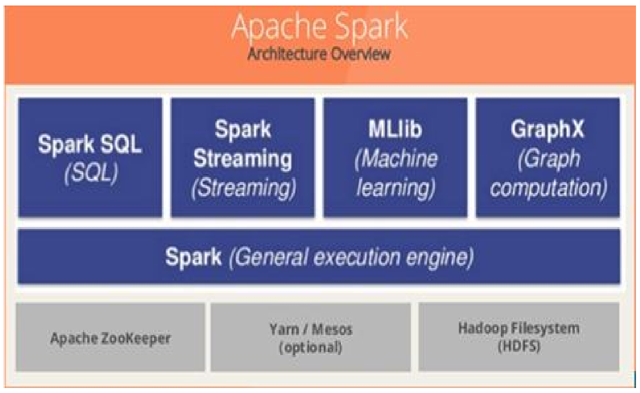


Fig: 2.1 Architecture of Spark

Spark Architecture has three main components:

[1] Data Storage

[2] Management Framework

[3] API

[1] Data Storage:

For Data Storage Purposes, Spark uses HDFS File System. It is work with Hadoop Data Source with Including HDFS, Cassandra, HBase, etc.

Following Figure shows that the components of the spark architecture.

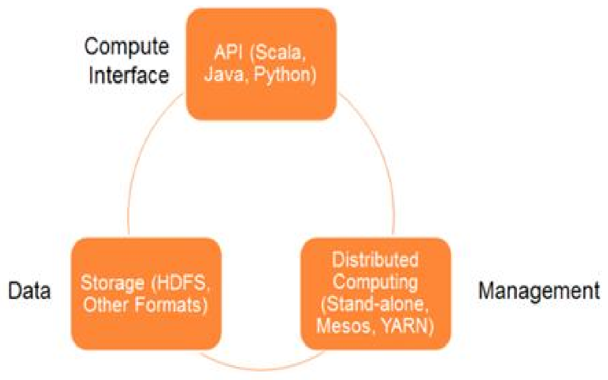

Compute Interface
API (Scala, Java, Python)
Data
Storage (HDFS, Other Formats)
Distributed Computing (Stand-alone, Mesos, YARN)
Management

Fig: 2.2 Components of Spark Architecture

[2] Resource Management:

Spark is deploying as impartial server and it can be run on Distributed Computing Framework like YARN or Mesos.

[3] API:

Spark based Applications create by Application Developers using a standard API interface. For many languages like Scala, Java and Python Spark provides the API for these all.

Spark Project is used main four libraries like Spark-Sql, Spark-MLlib, Spark-Streaming, and Spark-Graphx and also used Spark_core. For Making of new Appplication for spark Spark-core and one of them of four libraries are used for that application.

3.COMPARISION OF SPARK STREAMING WITH MULTIPLE APPROACH

There are unit several factors that have an effect on the performance of a stream process system - cluster size, similarity of operators, batch sizes, etc. Previous literature have studied numerous techniques to adapt to changes in operation conditions, either by elastically scaling the work or by discarding knowledge to shed load . However, in several sensible use cases (stock ticks, bank transactions, etc.), losing of data in these fields are unacceptable.

Traditional Approach (Static Spark Streaming)

In this type of Approach, Input Data are distributed on spark cluster. This data are accessing from that cluster and divided into fixed size of data and proceed Fixed batch time slots depends on previous measurement of system capacity. Then it generates the output. But if some time system is not robust due to the some reasons like server failure, more ingestion of data rates at that time system performance is decrease so it increases latency and takes more time for the result.
A statically set batch size may either incur unnecessarily high latency under low load, or may not be enough to handle surges in data rates, causing the system to destabilize.

Mordern approach (Dyanamic Spark Streaming)

In this model, Input data are takes from the scala database and then create spark cluster and these all the data are accessing from these cluster. But in this model have major advantage that it accessing not fixed size of data from the cluster it took dynamically data and then generate batch slot as dynamically .so this model will helpful for maintain the system stability due to the dynamically batch slot. And also decrease latency or increase more throughput.
So making use of dynamic batch sizing we overcome these problems. This will improve the result of the throughput and the latency.

4. Dynamic Batch Sizing

In this section, we tend to initial describe intimately our downside formulation. Then we tend to discuss why some initial solutions didn't reach the specified properties. Finally, we tend to discuss our algorithm rule which supported the dynamic batch sizing.

We want to find changes in the operation conditions and consequently increase the batch interval so that the system stability condition is maintained. Queuing delay can keep low, and therefore the system can stay stable at the upper rate, though with a better latency.

A. Introduction

Our goal is extremely similar – we have a tendency to would like to adapt the batch interval supported the steadiness of the streaming Data. Hence, at the primary look, one will devise an easy management rule that will increase the batch interval if the operational purpose is within the unstable zone and contrariwise.

Dynamically adapting the batch interval may allow the system to adapt in our desired manner. We use dynamically batching in spark streaming to adapt the batch size according to operating conditions. Following we describe why we choose dynamic batch sizing and overcome limitations of static batch interval.

1) Benefits over static batch interval:.
   a) Achieving minimum batch interval
   b) Ensure system stability
   c) Speed

2) Depending on the workload, there are some limitations in static batch sizing, which is as following:
   a) larger batches of data may allow the system to process data at higher rates.
   b) Data Size increased queue length also increase.
   c) Increase the Processing Time.

B. How to Achieve Dynamic Batch Sizing

Input Streaming Data are comes from the any input source such as files, Streaming Dataset, etc. these input data are proceed by the spark engine. At this, the streaming data are divides into some batches through default spark engine and these divided batches are slicing as dynamically according to the system workload capacity.

These dynamically slicing of streaming data are adapting by the one control algorithm. CPU scheduler picks the process from the queue. Using of this algorithm set the batch interval time and set the timer to interrupt after time slice and dispatches it.

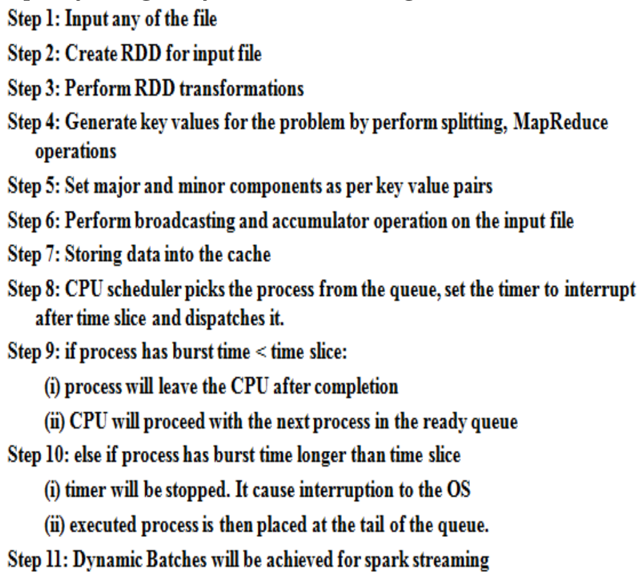
This control algorithm is useful to check the burst time of the system and dynamically set it according to the system workload capacity so the system will become stable and gives the result as fast. It set the burst time according to the time slice which is check following conditions and set the time slice.

(i) Completion time is less then time slice, process will leave the CPU after completion and CPU will proceed with next process in the ready queue.

(ii) Completion time is larger than time slice, timer will be stopped and caused interruption to the OS and executed process is placed tail of the queue.

C. Algorithm

This algorithm is used for achieving the dynamic batch sizing on arriving spark streaming data. Streaming data are slicing through dynamically at run time of the system. Algorithm is useful to slicing data according to the system capacity using of dynamic batch sizing.



```
Step 1: Input any of the file
Step 2: Create RDD for input file
Step 3: Perform RDD transformations
Step 4: Generate key values for the problem by perform splitting, MapReduce
    operations
Step 5: Set major and minor components as per key value pairs
Step 6: Perform broadcasting and accumulator operation on the input file
Step 7: Storing data into the cache
Step 8: CPU scheduler picks the process from the queue, set the timer to interrupt
    after time slice and dispatches it.
Step 9: if process has burst time < time slice:
      (i) process will leave the CPU after completion
      (ii) CPU will proceed with the next process in the ready queue
Step 10: else if process has burst time longer than time slice
      (i) timer will be stopped. It cause interruption to the OS
      (ii) executed process is then placed at the tail of the queue.
Step 11: Dynamic Batches will be achieved for spark streaming
```

5. Conclusion

In this proposed work, we are presenting control module for dynamically adapting the batch interval in batch stream processing system such as spark streaming.

In this work, we would like to show that control algorithm improve response time, throughput and complexity by comparing default spark streaming with the proposed one.

6. Future Work

In this section, we will improve spark streaming performance on the basis of their Response time, Latency and complexity.

References

[1] Javier Cervino, Evangelia Kalyvianaki, Joaqu ın Salvachua and Peter Pietzuch, “Adaptive Provisioning of Stream Processing Systems in the Cloud”, 28th International Conference on Data Engineering Workshops, 2012.

[2] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu and Zheng Zhang,” TimeStream: Reliable Stream Computation in the Cloud”, ACM, 2013.

[3] Xing Wu and Yan Liu,” Optimization of Load Adaptive Distributed Stream Processing Services”, IEEE International Conference on Services Computing, 2014.

[4] Andre Martin, Tiaraju Smaneoto, Tobias Dietze, Andrey Brito and Christof Fetzer, “User-Constraint and Self-Adaptive Fault Tolerance for Event Stream Processing Systems”, 45 Annual IEEE/IFIP International Conference on Dependable Systems and Network, 2015.

[5] Xinyi Liao, Zhiwei Gao, Weixing Ji, Yizhuo Wang- 2015,“An Enforcement of Real time Scheduling in Spark Streaming”, Sixth International Green and Sustainable Computing Conference, 2015.

[6] Lisa Amini, Navendu Jain, Anshul Sehgal, Jeremy Silber, Olivier Verscheure,” Adaptive Control of Extreme-scale Stream Processing Systems”, 26th IEEE International Conference on Distributed Computing Systems, 2006.

[7] Quan Zhang, Yang Song, Ramani R. Routray, Weisong Shi “Adaptive Block and Batch Sizing for Batched Stream Processing System”, IEEE International Conference on Autonomic Computing, 2016.