Comparing Architecture tactics for Big Data Analytics Frameworks : Apache Spark and Apache Flink

Palak Sharma[1]

Abstract— **Currently their exists many big data analytics frameworks that are suited for different machine learning operations on distributed systems [1].Making the most out of these frameworks is challenging because efficient executions strongly rely on complex parameter configurations and on an in-depth understanding of the underlying architectural choices [22]. Amongst these frameworks, Apache Spark [4] and Apache Flink [5] can be categorized as general purpose analytics frameworks. This paper aims to study the difference in the architecture of Apache Spark and Apache Flink, specifically in terms of these 4 architecture tactics : CheckPoint, Heartbeat, Polling and Scheduling. These tactics are choosen as both the frameworks are distributed systems, which have main characteristics of fault recovery, redundancy, communication and scheduling between the different components.**

I. INTRODUCTION

Apache spark emulates near real time processing by executing mini batches while Apache Flink supports real time analytics [3]. One of the design rationale for Spark is to accomplish iterative computations which makes it suitable for machine learning algorithms including classification, regression, and clustering [2]. As per Carbone et al., Flink provides support for ”streaming, batch, iterative, and interactive analytics” [6]. Although both framework share many common characteristics such as providing fault tolerance, they differ in their mechanism of how (steam or batch) and when (static or dynamic) the data is processed.

The remainder of this paper is organized as follows. Section II presents the motivation for this study.Section III described the basic terminologies used in the paper. Section IV gives methodology used for the study, while Section V and VI emphasizes on the architecture tactics present in the spark and Flink in detail. Section VII enlists the comparartive analysis between the two frameworks and VIII surveys the associated related works with this study. Finally, Section IX lists the gathered insights and concludes this study.

II. MOTIVATION

The solution to choosing a data analytics framework between Spark and Flink depends on the problem at hand. For example, if the business demands a fraud prevention strategy, then Flink is a better option while in case of fraud detection, spark may provide a better performance. The focus of this project is to identify what makes the two frameworks under study different at architecture level. The analyzed level of granularity is at the source code package and file level where the architecture tactics are implemented.Currently there are many resources that compare hadoop and spark [9,10], which are widely used for big data analytics, but the detailed research for Spark and Flink at the code level are limited.

[1]MS student in Software Engineering at Rochester Institute of Technology. Rochester, NY. 14623. `ps2671@rit.edu`

III. TERMINOLOGIES

- **Architecture-Tactics** - Bachman et al. defined architecture tactics as a means of satisfying a quality-attribute-response measure by manipulating some aspects of a quality attribute model through architectural design decisions [25].

- **Heartbeat** - The technique of communication between two components to ensure that the status of the task under progress. A failed heartbeat indicate that the other component has become inactive.

- **Check-Pointing** - *”Checkpointing is a technique to add fault tolerance into computing systems. It basically consists of saving a snapshot of the application’s state, so that it can restart from that point in case of failure”.* [26].

- **Polling** - One process communicates or interacts with another process to get an update on the status from the last poll time.

- **Scheduling** - Liang et. al, defined *”Job Scheduling is composed of at least two inter-dependent steps: the allocation of processes to workstations (space-sharing) and the scheduling of the processes over time (time-sharing), while there exist several optional complementary steps to further improve the performance.[27]*

- **Distributed-Systems** - A network of interconnected computing servers to perform one or more tasks to improve the performance, efficiency and throughput.

- **Map-Reduce** - As per the official Hadoop documentation ”A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system.[28]"

- **RDD** - "RDD stands for resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat" [11].

IV. METHODOLOGY

The methodology adopted for this research project is as follows :

- Creating the high level or ground truth architecture of the two frameworks under study by referring to their existing design and architecture documents.
- Based on the architecture tactic patterns proposed by Mirakhorli et.al [7], manually identifying the 4 architecture tactics (CheckPoint, Heartbeat, Polling and Scheduling) in source code of Apache Spark and Flink.
- Creating a comparison catalog by analyzing the differences in the implementation of the architecture tactics in both the frameworks.

V. SPARK ARCHITECTURE

Similar to Hadoop, Spark makes use of Master-Slave architecture.The important component of Spark architecture are Driver (which interacts with the Master), Worker (which run the executors), the Cluster Manager (which is the master or single coordinator for the workers). Fig. 1. shows the high level architecture for Spark [11].

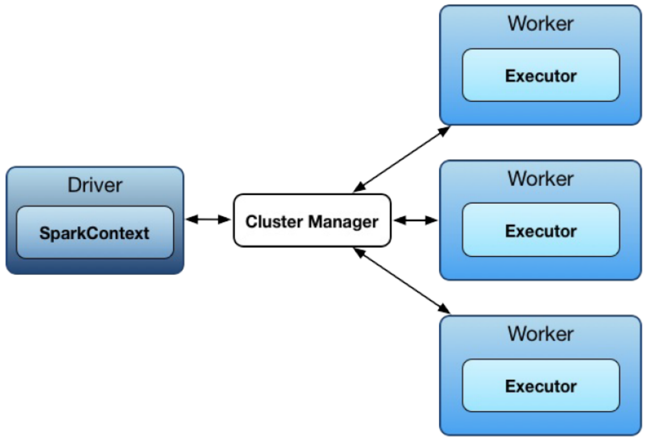

Fig. 1. High level architecture for Spark.

A. Identifying Architecture Tactics in Spark

As per the studied literature on spark, this document currently consists of these 4 architecture tactics in Spark: heartbeat, polling, check-pointing and scheduling. By studying the design documents and other literature available for Spark in greater detail, other tactics which are implemented in Spark could be identified. These remaining tactics will be covered in the coming weeks.

- **Heartbeat** - The executors sends metrics (like merge shuffle time, JVM garbage collection time) for active tasks to the driver every 10 seconds.Then a blocking Heartbeat message that holds the executor id, all accumulated updates per task, and BlockManagerId is sent to HeartbeatReceiver RPC endpoint. Fig 2 shows the hearbeat interaction between the Driver and Executor component.

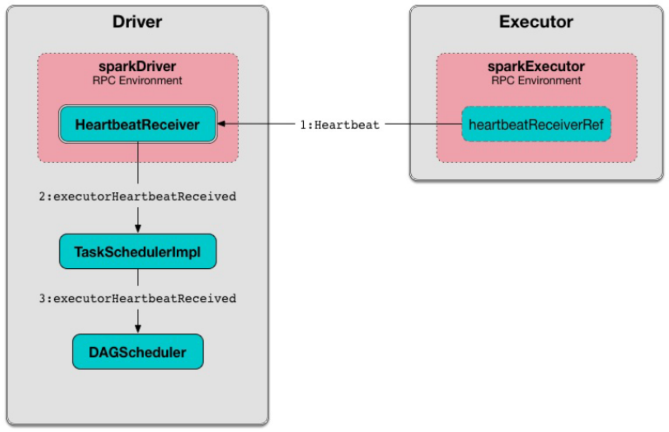

Fig. 2. Executor Heartbeat Mechanism.

- **Polling** - The "ConsoleProgressBar" shows the statistics regarding active tasks, completed tasks and total tasks on console for spark shell. It polls the status of active stages periodically, the progress bar is showed after the stage has ran at least 500ms [12]. If multiple stages run in the same time, the status of them will be combined together, showed in one line.
- **Scheduling** - JobScheduler class contains the implementation for scheduling jobs to be run on Spark. It uses the JobGenerator to generate the jobs and runs them using a thread pool.
- **Check-Pointing** - The driver saves the application-critical metadata to a fault-tolerant storage like HDFS to restore them later in the event of recovery from failure. The metadata includes configuration, code, and a list of queued but not yet completed batches [13].

VI. FLINK ARCHITECTURE

Apache Flink is an open-source system for processing streaming and batch data. Flink is built on the philosophy that many classes of data processing applications, including real-time analytics, continuous data pipelines, historic

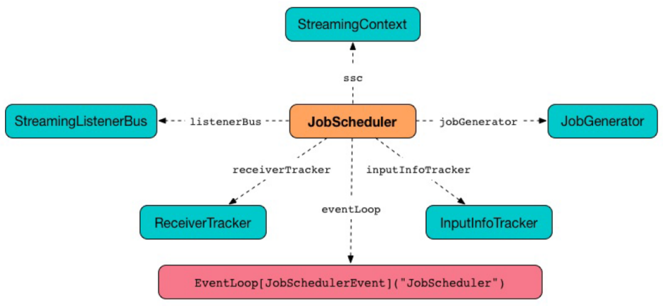

Fig. 3. Job Scheduler and dependencies.

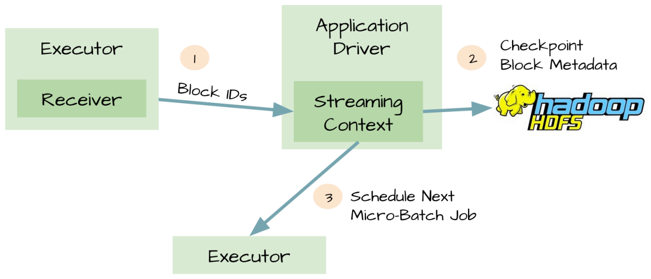

Fig. 4. Meta-data check-pointing in Spark.

data processing (batch), and iterative algorithms (machine learning, graph analysis) can be expressed and executed as pipelined fault-tolerant dataflows[14].

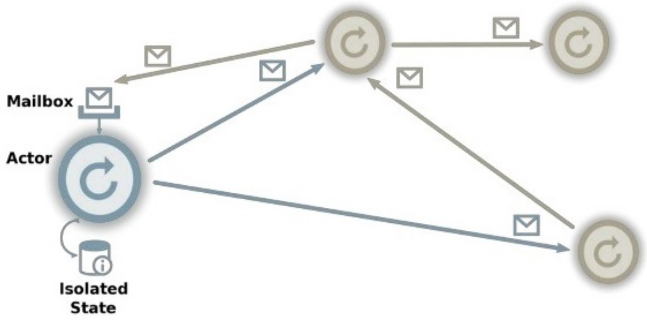

Fig. 5. Actor Model in Apache Flink.

Figure 5 describes the actor model implementation in Apache Flink [18]. Akka is a framework to develop concurrent, fault-tolerant and scalable applications. It is an implementation of the actor model and thus similar to Erlang's concurrency model. In the context of the actor model, all acting entities are considered independent actors. Actors communicate with other actors by sending asynchronous messages to each other.Each actor has a mailbox in which the received messages are stored. Furthermore, each actor maintains its own isolated state.An actor has a single processing thread which polls the actor's mailbox and processes the received messages successively. As a result of a processed message, the actor can change its internal state, send new messages or spawn new actors.An actor system is the container in which all actors live. It provides shared services such as scheduling, configuration and logging. The actor system also contains the thread pool from where all actor threads are recruited.

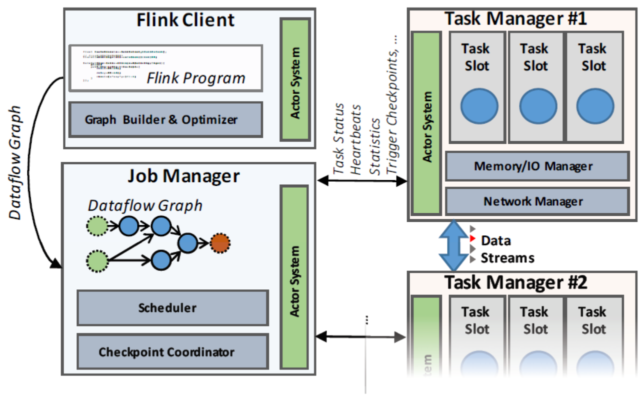

Fig. 6. Flink Process Architecture.

Figure 6 shows the Flink process architecture. Its three main elements are : Flink Client, Job Manager and atleast one Task Manager. The client is responsible to convert the code to dataflow graph and provide to the JobManager. Further the JobManager task is to manage its distributed execution. Some of the tracked metrics includes state, progress and scheduling of operators and also coordinating the recovery and checkpoint activities to support fault-tolerance behavior. TaskManagers form the important component where the actual data processing happens. The operators are executed to generate streams or report to JobManager.

A. Identifying Architecture Tactics in Flink

The identified tactic implementation is quite recent in Flink. The component implementation for few tactics are still under progress [16,17].

- **Heartbeat** - Flink detects failed components by using Akka's DeathWatch mechanism [18]. DeathWatch allows actors to watch other actors even though they are not supervised by this actor or even living in a different actor system. Once a watched actor dies or is no longer reachable, a Terminated message is sent to the watching actor. Consequently, upon receiving such a message, the system can take steps against it. Internally, the DeathWatch is realized as heartbeat and a failure detector which, based on the heartbeat-interval, hearbeat-pause and failure threshold, estimates when an actor is likely to be dead. Default value for Heartbeat timeout is 10 seconds. Currently the implementation of heartbeat component is still in progress [16,17].

- **Polling** - The client makes use of Polling to track the status of its job in the JobManager.

The polling interval by default is 2 seconds.

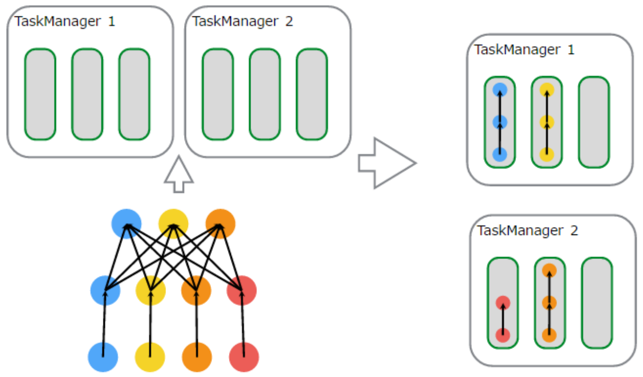

Fig. 7. Job Scheduling in Flink.

- **Scheduling** - As shown in Figure 7. the execution resources in Flink are defined through Task Slots [19]. Each TaskManager will have one or more task slots, each of which can run one pipeline of parallel tasks. A pipeline consists of multiple successive tasks. In figure 7, a pipeline consists of the sequence Source - Map - Reduce. On a cluster with 2 TaskManagers with 3 slots each, the program will be executed as described below. The scheduler is responsible for distributing the ready-to-run tasks among instances and slots. The scheduler supports two scheduling modes: Immediate scheduling - A request for a task slot immediately returns a task slot, if one is available, or throws a exception. Queued Scheduling - A request for a task slot is queued and returns a future that will be fulfilled as soon as a slot becomes available.

- **Check-Pointing** - The central part of Flinks fault tolerance mechanism is drawing consistent snapshots of the distributed data stream and operator state. These snapshots act as consistent checkpoints to which the system can fall back in case of a failure. Flinks mechanism for drawing these snapshots is described in Lightweight Asynchronous Snapshots for Distributed Dataflows [21]. barriers are injected into the data stream

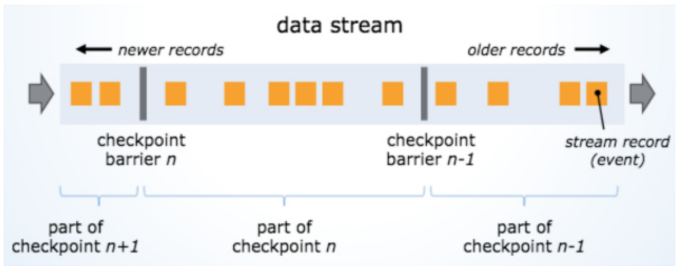

Fig. 8. Check-Pointing in Flink.

and flow with the records as part of the data stream. Barriers never overtake records, the flow strictly in line. Multiple barriers from different snapshots can be in the stream at the same time, which means that various snapshots may happen concurrently [22]. Recently, Spark has started to catch up on these memory issues with its Tungsten [23] project, highly inspired from the Flink model, for the explicit custom memory management aiming to eliminate the overhead of the JVM object model and garbage collection.

VII. COMPARISION CATALOG

- **Memory Management** - In Flink, most of the operators are implemented so that they can survive with very little memory (spilling to disk when necessary). Although Spark can serialize data to disk, it requires that (significant) parts of the data to be on the JVMs heap for several operations; if the size of the heap is not sufficient, the job dies.

- **Task Execution** - Flink often executes successive tasks concurrently: For Streaming programs, that happens in any case, but also for batch programs, it happens frequently [19].

- **Iteration** - In Spark, iterations are implemented as regular for-loops. Therefore, each iteration operates on the result of the previous iteration which is held in memory. While in Flink, iterations are executed as cyclic data flows. Basically, data is flowing in cycles around the operators within an iteration. Since operators are just scheduled once, they can maintain a state over all iterations [22].

- **Pipeline** - The pipelined execution brings important benefits to Flink, compared to the staged one in Spark. There are several issues related to the pipeline fault tolerance, but Flink is currently working in this direction [24].

- **Optimization** - Flink has an automatic build in Optimization while in Spark, optimization is manual as adapted to the dataset under consideration.

- **Configuration** - Parameter configuration proves tedious in Spark, with various mandatory settings related to the management of the RDDs (e.g. partitioning, persistence). Flink requires less configuration for the memory thresholds, parallelism and network buffers, and none for its serialization (as it handles its own type extraction and data representation) [22].

- **Resources** - A major current concern is that Flink is still under development and hence remains unexplored to the extend of Spark. Currently there are many in progress architecture tactics [16,17] and pending design documents [20] but Spark has a wide user base along with extensive documentation.

VIII. RELATED WORKS

Spangenberg et al., performed an experiment based study for comparing the performance of Spark and Flink based on four different types of machine learning algorithms [1]. They conclude that Flink perform better for iterative computations while Spark performs better for batch processing. Their results are based on the comparison between the execution time of the four algorithms on both the frameworks. It would be interesting to look into the architecture level details of these two frameworks to identify the main reasons which led to these execution time differences.

Ovidiu-Cristian et al., compared the performance between spark and flink based on the architecture choices and other configuration parameters which are essential for proper execution of algorithms on the datasets. They conclude that how the architecture components and the configuration parameters relate to the performance of the two frameworks.

Both the studies [1,22] are similar in the context of the algorithms used to compare the performance between the two frameworks.

IX. CONCLUSIONS

While Spark is popular and has higher adoption rate, Flink is growing fast and is more technologically advanced. As per the architecture comparison catalog derived in section VI between the two, Flink is meant for real time analytics while Spark's core purpose is batch processing. Clearly, both have there own advantages, and in order to choose a framework between the two, a trade-off needs to be made based on the business requirements. The challenge remains how to merge the differences in order to utilize the advantages of the two.

REFERENCES

[1] Spangenberg, Norman, Martin Roth, and Bogdan Franczyk. ”Evaluating new approaches of big data analytics frameworks.” International Conference on Business Information Systems. Springer International Publishing, 2015.

[2] Meng, Xiangrui, et al. ”Mllib: Machine learning in apache spark.” JMLR 17.34 (2016): 1-7.

[3] Mayo, KDnuggets. By Matthew. ”KDnuggets.” Analytics Big Data Data Mining and Data Science. N.p., n.d. Web. 04 Nov. 2016.

[4] Apache Spark - Lightning-Fast Cluster Computing. N.p., n.d. Web. 04 Nov. 2016.

[5] Apache Flink: Scalable Batch and Stream Data Processing. N.p., n.d. Web. 04 Nov. 2016.

[6] Carbone, Paris, et al. ”Apache flink: Stream and batch processing in a single engine.” Data Engineering (2015): 28.

[7] Mirakhorli, Mehdi, and Jane Cleland-Huang. ”Detecting, tracing, and monitoring architectural tactics in code.” IEEE Transactions on Software Engineering 42.3 (2016): 205-220.

[8] ”Understand Static Code Analysis Tool.” N.p., n.d. Web. 04 Nov. 2016.

[9] Gu, Lei, and Huan Li. ”Memory or time: Performance evaluation for iterative operation on hadoop and spark.” High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013.

[10] Lin, Xiuqin, Peng Wang, and Bin Wu. ”Log analysis in cloud computing environment with Hadoop and Spark.” Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on. IEEE, 2013.

[11] Laskowski, Jacek. ”Overview of Apache Spark.” Overview of Apache Spark Mastering Apache Spark 2.0. N.p., n.d. Web. 18 Nov. 2016.

[12] ”Class ConsoleProgressBar” N.p., n.d. Web. 18 Nov. 2016. Retrieved from https://spark.apache.org/docs/1.3.1/api/java /org/apache/spark/ui/ConsoleProgressBar.html

[13] ”Recent Evolution of Zero Data Loss Guarantee in Spark Streaming With Kafka” N.p., n.d. Web. 18 Nov. 2016. Retrieved from http://getindata.com/blog/post/recent-evolution-of-zero-data-loss-guarantee-in-spark-streaming-with-kafka/.

[14] Carbone, Paris, et al. ”Apache flink: Stream and batch processing in a single engine.” Data Engineering (2015): 28.

[15] ”[FLINK-4478] Implement heartbeat logic - ASF JIRA.” [FLINK-4478] Implement heartbeat logic - ASF JIRA. N.p., n.d. Web. 15 Dec. 2016.

[16] ”[FLINK-4354] Implement TaskManager side of heartbeat from ResourceManager - ASF JIRA. N.p., n.d. Web. 15 Dec. 2016.

[17] ”[FLINK-4364] Implement TaskManager side of heartbeat from JobManager - ASF JIRA. N.p., n.d. Web. 15 Dec. 2016.

[18] ”Akka and Actors” Retrieved from https://cwiki.apache.org/confluence/display/FLINK/Akka+and+Actors. N.p., n.d. Web. 15 Dec. 2016.

[19] ”Jobs and Scheduling” Retrieved from https://ci.apache.org/projects/flink/flink-docs-release-1.0/internals/job scheduling.html. N.p., n.d. Web. 15 Dec. 2016.

[20] Retrieved from https://cwiki.apache.org/ confluence/display/FLINK/Design +Documents. N.p., n.d. Web. 15 Dec. 2016.

[21] Carbone, Paris, et al. ”Lightweight asynchronous snapshots for distributed dataflows.” arXiv preprint arXiv:1506.08603 (2015).

[22] Marcu, Ovidiu-Cristian, et al. ”Spark versus flink: Understanding performance in big data analytics frameworks.” Cluster Computing (CLUSTER), 2016 IEEE International Conference on. IEEE, 2016.

[23] Project Tungsten. https://databricks.com /blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html.

[24] Flink-2250. https://issues.apache.org/jira/ browse/FLINK-2250.

[25] F. Bachmann, L. Bass, and M. Klein, Deriving architectural tactics: A step toward methodical architectural design, Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2003-TR-004, 2003.

[26] Retrieved from https://en.wikipedia.org/wiki/ Application-checkpointing. N.p., n.d. Web. 15 Dec. 2016.

[27] Liang, Dongning, Pei-Jung Ho, and Bao Liu. ”Scheduling in Distributed Systems.” Department of Computer Science and Engineering University of California, San Diego (2000).

[28] Retrieved from https://hadoop.apache.org/docs/ r1.2.1/ mapred-tutorial.html. N.p., n.d. Web. 15 Dec. 2016.