Modeling and Simulating Apache Spark Streaming Applications

Johannes Kroß
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
kross@fortiss.org

Helmut Krcmar
Technische Universität München
Boltzmannstr. 3
85748 Garching, Germany
krcmar@in.tum.de

Abstract

Stream processing systems are used to analyze big data streams with low latency. The performance in terms of response time and throughput is crucial to ensure all arriving data are processed in time. This depends on various factors such as the complexity of used algorithms and configurations of such distributed systems and applications. To ensure a desired system behavior, performance evaluations should be conducted to determine the throughput and required resources in advance. In this paper, we present an approach to predict the response time of Apache Spark Streaming applications by modeling and simulating them. In a preliminary controlled experiment, our simulation results suggest accurate prediction values for an upscaling scenario.

1 Introduction

Big data systems enable organizations to store and analyze data with high volume, velocity, and variety [4]. Corresponding processing techniques can be categorized into batch and stream processing [5]. Stream processing systems receive broad concentration nowadays as algorithms, transformations, and windowing mechanisms for streaming data constantly become more sophisticated and libraries for machine learning are available. They are mainly applied to process data and provide results in real time [5]. Therefore, the performance of such systems is particularly significant, for instance, to ensure high throughput for different workload scenarios and prevent queueing up of input stream data. However, planning the requirements of such applications and systems is complicated since environments and conditions to evaluate the performance for different scenarios, system configurations, and realistic workloads are usually not met as in productive environments [3, 8].

We propose a modeling and simulation approach to predict the response time of stream processing applications i.e., Apache Spark Streaming. Therefore, we use the Palladio component model (PCM) [1] and an extension for big data systems that we have presented in previous work [7]. The extension is open source[1] and includes a distributed call action to model parallel and distributed external calls in a service effect specification (SEFF) and a cluster resource specification to model a cluster of master and worker nodes and distribute resource demands to worker nodes.

[1] https://git.fortiss.org/pmwt/bd.pcm.extension

2 Related Work

Regarding modeling, simulation, or analytical solving the performance of big data systems, most of prior research focuses on Apache Hadoop and its MapReduce paradigm and, therefore, on batch processing. There is one approach by Wang and Khan [9], that focuses on predicting the response time of Apache Spark applications, however, only batch applications. Regarding stream processing, there is one patent by Ginis and Strom [2] that describes a method on predicting the performance of publish-subscribe middleware messaging systems using queueing theory that, however, does not take resource demands for CPU, memory, or hard disk drives into account.

3 Modeling and Simulation Approach

There are several known stream processing systems available such as Apache Samza, Apache Storm, Apache Spark Streaming and Apache Flink [6]. We focus on Apache Spark Streaming[2] in this paper as it is one of the sophisticated technologies with a large community and supported by known benchmarks. It comprises a micro-batch model, in contrast to other technologies that use an operator-based model [6].

A Spark Streaming application is constructed as follows[3]: it receives incoming data from streaming sources using a discretized stream called *DStream*. This data is fetched in the form of a micro-batch *job* that is iteratively executed in stream intervals. A *DStream* is represented by several resilient distributed datasets (*RDDs*). Afterwards, transformations such as *map* or *reduce* operations can be applied on a *DStream*. Spark builds a distributed acyclic graph (DAG) based on these related operations and splits them into *stages* of *tasks*. The number of parallel *tasks*

[2] http://spark.apache.org/streaming/

[3] https://spark.apache.org/docs/1.6.0/streaming-programming-guide.html

is limited by the number of partitions of an *RDD*. Furthermore, transformations with narrow dependencies are consolidated in one *stage*, for instance, *map* and *filter* operations that do not require to shuffle data. *Stages* are executed sequentially and finally make up one *job*.

In order to model a Spark Streaming application, we specify one *job* executor component with a SEFF that involves a loop to start several asynchronous forked behaviours as displayed in Figure 1.

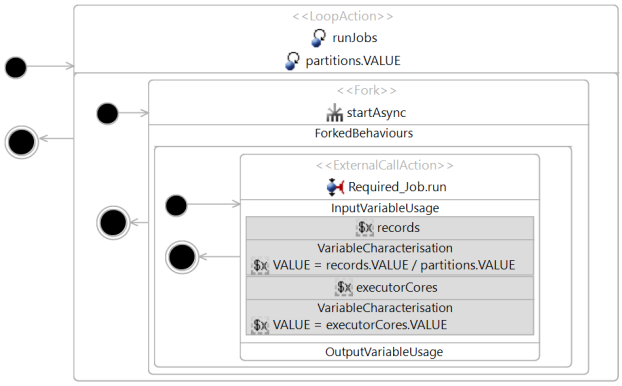

Figure 1: SEFF of job executor component

The loop length and, therefore, the number of executed behaviors depends on the value of the parameter *partitions* that is used to describe the number of topic partitions. In the forked behavior, we call the SEFF of the *stage* executor component with the parameters *records* and *executorCores*. The former parameter describes the number of records for each partition, the latter the number of cores that is configured when starting a Spark application with the equivalent parameter *spark.executor.cores*. The SEFF is illustrated in Figure 2.

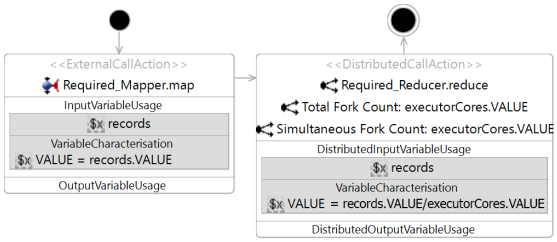

Figure 2: SEFF of stage executor component

For each *stage*, we model an external call or distributed call [7], respectively, with the number of *records* as input parameter. Our sample application involves two stages *map* and *reduce*. The first SEFF *map* is invoked once since there is one *DStream* for each partition. The second SEFF is invoked with a distributed call of which the parallelism depends on the *executorCores* value. The *map* and *reduce* SEFFs involve three consecutive internal actions each with one parametric resource demands to specify the scheduler delay, serialization time, and computing time.

In order to model the hardware environment, we specify a resource container with a cluster resource specification [7] for each node. For the master node, we model one parent resource container that includes a round robin action scheduling policy and a master resource role. Dependent on the number of worker nodes, we specify several nested resource containers with a worker resource role. In the usage model, we invoke the *job* executor component with its three input parameters, model a closed workload with one user, and specify the think time according to the stream interval.

4 Controlled Experiment

In our controlled experiment, we use the HiBench benchmark suite[4] of which we use the *distinct count* application. It involves two *stages map* and *reduce*. Therefore, data are streamed to a so-called *topic* in an Apache Kafka[5] cluster, a distributed publish-subscribe messaging system. The application is connected to that *topic* and applies a direct stream to query data from Apache Kafka using *DStreams*. Thereby, the level of parallel streams is defined by the number of partitions of one *topic* which, consequently, equals the number of *map stages*.

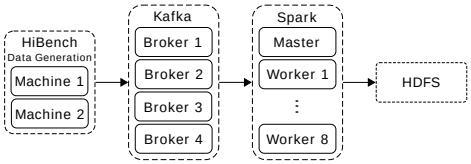

Figure 3: Testbed setup

Our experimental setup is shown in Figure 3. For the workload, we setup 2 virtual machines (VMs) that we use to generate data and a cluster consisting of 4 VMs for Apache Kafka brokers. For the application, we setup 1 VM for the master node, and 8 VMs for the worker nodes where the benchmark application will be executed. We use Apache YARN (2.7) as cluster manager and Apache Spark (1.6) as processing framework. We specified one Spark executor per worker node with 6 cores and 24 gigabytes memory.

We conducted four scenarios with a stream interval of 10 seconds as listed in Table 1.

Table 1: Conducted experiments

Scenario	**Workload** (events/second)	**Kafka broker**	**Topic partitions**	**Spark worker**
2 nodes	~ 450,000	1	2	2
4 nodes	~ 450,000	2	4	4
6 nodes	~ 450,000	3	6	6
8 nodes	~ 450,000	4	8	8

Based on the 2 nodes scenario, we measured the delay and CPU resource demands for all *tasks* using the Spark monitoring API, adjusted the demands in dependence of the number of records, and included them into our repository model as listed in Table 2. We used this repository model for all upscaling scenarios. We adapted the number of workers in the resource

[4] https://github.com/intel-hadoop/HiBench
[5] http://kafka.apache.org/

environment model and the partition parameter in the usage model for each scenario.

Table 2: Parametric resource demands

Map SEFF	
scheduler delay	10
deserialization	0.000078549 ∗ records.VALUE
computing	$Norm$(0.003320415 ∗ records.VALUE, 0.0001553647 ∗ records.VALUE)
Reduce SEFF	
scheduler delay	10
deserialization	0.000013227 ∗ records.VALUE
computing	0.000023370 ∗ records.VALUE

A boxplot of the measured response time (MRT) and the simulated response time (SRT) is illustrated in Figure 4. For the 2 nodes scenario, the mean MRT is 7.88 seconds and the mean SRT is 7.94 seconds, which gives a relative reponse time prediction error (RTPE) of 0.67%. In the 4 nodes scenario, the values deviate more with a RTPE of 21.14%. Our analysis of the measurements suggests that the processing time for each task did not behave as linear as in the other scenarios. In the 6 nodes and 8 nodes scenarios, the RTPE result in 3.41% and 2.26%.

Our models, simulation and measurements results, and analysis script are publicly available online [10].

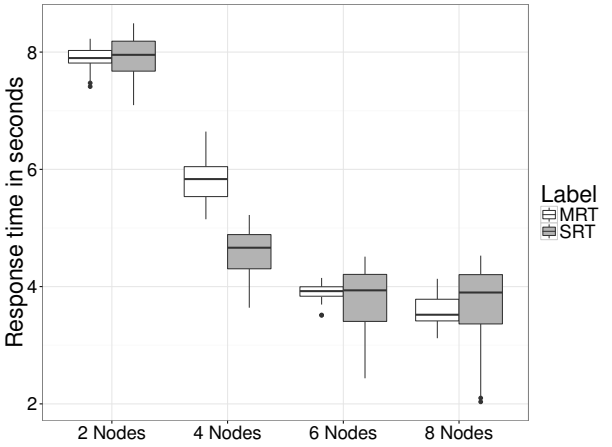

Figure 4: Measured and simulated response times

5 Conclusion and Future Work

In this paper, we proposed a modeling approach for stream processing systems using the example of Apache Spark Streaming. In a small controlled experiments, we simulated an upscaling scenario in which we increased the cluster size. Our predicted response times approach the measured ones closely.

At the moment, our extension for the simulation framework is for PCM 3.4.1 and we only consider delay and CPU demands. Therefore, we plan to incorporate our extension in the up to date PCM version and to additionally evaluate resources such as memory and network. Furthermore, we plan to extend our approach for operator-based processing frameworks such as Apache Flink and Apache Storm. Our long-term goal is to automatically derive performance models based on monitoring data, e.g., provided by APIs of processing frameworks.

References

[1] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio component model for model-driven performance prediction". In: *The Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[2] Roman Ginis and Rober E. Strom. *Method for predicting performance of distributed stream processing systems*. US Patent 7,818,417. Oct. 2010.

[3] Andreas Brunnert et al. "Performance management work". English. In: *Business & Information Systems Engineering* 6.3 (2014), pp. 177–179.

[4] Michael Schermann et al. "Big Data - an interdisciplinary opportunity for information systems research". In: *Business & Information Systems Engineering* 6.5 (2014), pp. 261–266.

[5] Ibrahim A.T. Hashem et al. "The rise of "big data" on cloud computing: Review and open research issues". In: *Information Systems* 47 (2015), pp. 98–115.

[6] Guenter Hesse and Martin Lorenz. "Conceptual Survey on Data Stream Processing Systems". In: *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. Dec. 2015, pp. 797–802.

[7] Johannes Kroß, Andreas Brunnert, and Helmut Krcmar. "Modeling Big Data Systems by Extending the Palladio Component Model". In: *Softwaretechnik-Trends* 35.3 (Nov. 2015).

[8] Johannes Kroß et al. "Stream Processing on Demand for Lambda Architectures". English. In: *Computer Performance Engineering*. Ed. by Marta Beltrán, William Knottenbelt, and Jeremy Bradley. Vol. 9272. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 243–257.

[9] Kewen Wang and Mohammad M.H.K. Khan. "Performance Prediction for Apache Spark Platform". In: *Proceedings of the 17th International Conference on High Performance Computing and Communications (HPCC)*. New York, NY: IEEE, Aug. 2015, pp. 166–173.

[10] Johannes Kroß and Helmut Krcmar. *Models, Simulations, Measurements, and Analysis for Modeling and Simulating Apache Spark Streaming Applications*. Available: http://dx.doi.org/10.5281/zenodo.61243. Aug. 2016.