A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications

Paul Le Noac'h
INSA Rennes
paul.lenoach@irisa.fr

Alexandru Costan
Inria / INSA Rennes
alexandru.costan@irisa.fr

Luc Bougé
ENS Rennes
luc.bouge@ens-rennes.fr

***Abstract*—Stream computing is becoming a more and more popular paradigm as it enables the real-time promise of data analytics. Apache Kafka is currently the most popular framework used to ingest the data streams into the processing platforms. However, how to tune Kafka and how much resources to allocate for it remains a challenge for most users, who now rely mainly on empirical approaches to determine the best parameter settings for their deployments. In this poster, we make a through evaluation of several configurations and performance metrics of Kafka in order to allow users avoid bottlenecks, reach its full potential and avoid bottlenecks and eventually leverage some good practice for efficient stream processing.**

***Index Terms*—Stream computing, Apache Kafka, Big Data**

I. Context and Problem Statement

Distributed stream computing is now an unavoidable processing paradigm. It allows to get results continuously and in real time from huge volumes of fresh data. Over the last years, the amount of data streams has not stopped to increase in many fields such as financial applications, network monitoring, sensor networks and web log mining [3]. Indeed, the Internet of Things accelerated the development of stream computing through the fast expansion of sensors deployed at geographically distributed locations [4], [5], [6]. Large Internet companies also shifted their workloads towards the stream model: for example, Facebook has to handle 10^6 events/s within a latency between 10 and 30s for advertisement purposes, which represents a data rate of 9GB/s at peak [3].

The typical stream processing piepline is illustrated in Figure 1 and consists of 4 phases:

- **Data Collection.** The first step is to collect the data in real-time, from a set of sensors for example.
- **Ingestion.** The purpose of this step is to distribute the data stream on several machines in order to process it in parallel. Relevant state-of-the-art systems: Apache Kafka [8].
- **Processing.** This phases consists in processing the stream of data, typically through a topology of distributed operators. Relevant systems: Apache Flink [9], Apache Spark [10], Apache Apex [11].
- **Storage.** The last step stores the results of the intermediate data in a persistent fashion for fault tolerance, archival as well as a posteriori processing purposes. Relevant systems: REDIS [12], Apache HDFS [13], Cassandra [7].

The focus of this poster is on the Ingestion phase. In particular, we are interested in evaluating the performance

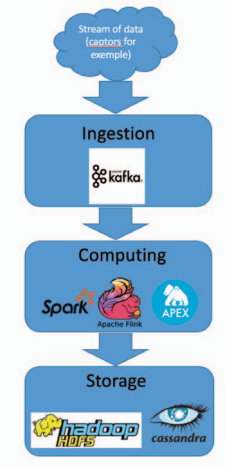

Fig. 1. The typical stream processing pipeline.

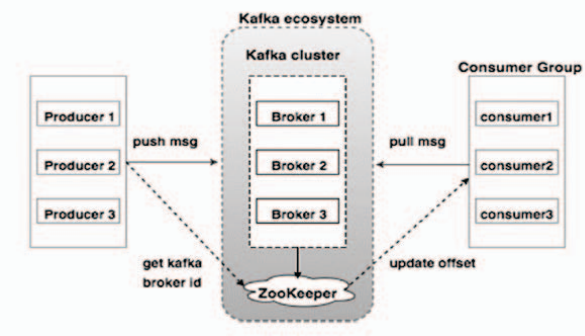

Fig. 2. Apache Kafka architecture. [14]

of Apache Kafka, currently the reference state-of-the-art for ingestion. As illustrated in Figure 2, the ingestion with Kafka relies on two sets of actors:

- **Producers** which provision the Apache Kafka brokers. The task of producers is to transfer the data from devices which produce the data to Apache Kafka nodes in order to distribute the data stream as messages.
- **Consumers** which have the task to pick up the data from the Kafka brokers to the processing nodes (e.g. to Spark or Flink).

The purpose of the Kafka cluster is to manage the interaction

978-1-5386-2715-0/17/$31.00 ©2017 IEEE

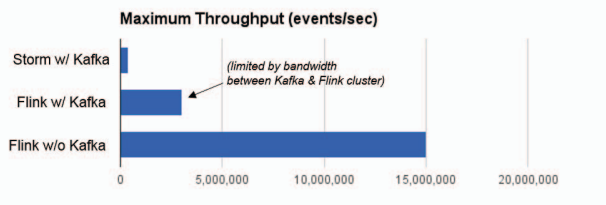

Fig. 3. Performances of Apache Flink while using Apache Kafka [2]

between producers, brokers and consumers by synchronizing the nodes, managing the message waiting queues and the notifications.

The main contribution of this poster is a study on **how the ingestion performance can impact the overall stream processing**. As shown in Figure 3, the processing performance (i.e. Apache Flink) is highly penalized when used with Kafka as opposed to a standalone deployments. This suggests that the ingestion phase can become a bottleneck in the overal streaming pipeline. While the authors of the cited evaluation mainly blame the bandwidth for this limitation, we argue that this is not the only factor contributing to the performance degradation but others have to be taken into account (e.g. parameters tuning, number of nodes used etc.). This is precisely the focus of this poster: **identifying the impact of different parameter settings on Kafka's overall performance**.

II. Evaluation Methodology and Results

In a first phase we are interested in isolating the performance of each Kafka component. This is why we have split the experiments in 2 parts: evaluating the performance of producers running alone and evaluating the performance of the consumers running alone. We leave for future work the evaluation of the two combined. For each experiment we have varied a set of parameters and measured a set of performance metrics, as follows.

- Parameters :
  - message size
  - batch size
  - Acquirement strategy
  - threads for sending the data on the network and for disk I/O
  - message replication
  - hardware used
- Performance Metrics :
  - Throughput (MB/s, items/s)
  - Latency
  - CPU usage
  - Disk usage
  - Memory usage
  - Network usage

We have deployed Kafka on Grid5000 [1] (a large-scale and versatile test-bed) using up to 32 nodes. We choose homogeneous servers in order to have a fair comparison when increasing the node number.

Our first experiments test *the impact of the replication, size of messages, number of nodes and batch size* on the consumers and producers performances. We noticed that the variation of the batch size (Figure 4 and 5) shows that there is a range of batches with a better performance. Also, when varying the number of nodes we witness in some scenarios a sudden performance drop, which we suspect is due to the internal Kafka synchronizations as well as the underlying network. We are currently investigating more thoroughly on the reasons of this behavior in order to determine the optimized batch range.

III. Conclusions

In this work, we emphasized how to detect the bottlenecks of of Apache Kafka and how to fine tune its deployment. This will help stream processing developers in optimizing their Big Data processing architecture for a given use-case.

As the mechanisms of ingestion are better understood, we choose to continue this work by further evaluating the reference processing frameworks (Apache Spark and Flink) with several configurations as for Apache Kafka.

Aknowledgments

This work is supported by the ANR OverFlow project (ANR-15-CE25-0003).

References

[1] "Grid5000. www.grid5000.fr.
[2] https://data-artisans.com/blog/extending-the-yahoo-streaming-benchmark
[3] Lu, Ruirui, Wu, Gang, Xie, Bin and Hu, Jingtong. "Stream Bench: Towards Benchmarking Modern Distributed Stream Computing Frameworks.." Paper presented at the meeting of the UCC, 2014.
[4] Milan ermk, Daniel Tovark, Martin Latovika and Pavel eleda, A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems, Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP
[5] Shukla A., Simmhan Y. (2017) Benchmarking Distributed Stream Processing Platforms for IoT Applications. In: Nambiar R., Poess M. (eds) Performance Evaluation and Benchmarking. Traditional - Big Data - Internet of Things. TPCTC 2016. Lecture Notes in Computer Science, vol 10080. Springer, Cham
[6] Thilina Buddhika and Shrideep Pallickara, NEPTUNE: Real Time Stream Processing for Internet of Things and Sensing Environments, Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium, 2016
[7] http://cassandra.apache.org
[8] https://kafka.apache.org
[9] https://flink.apache.org
[10] https://spark.apache.org
[11] https://apex.apache.org
[12] https://redis.io
[13] https://hadoop.apache.org
[14] http://www.tutorialspoint.com/apache_kafka/

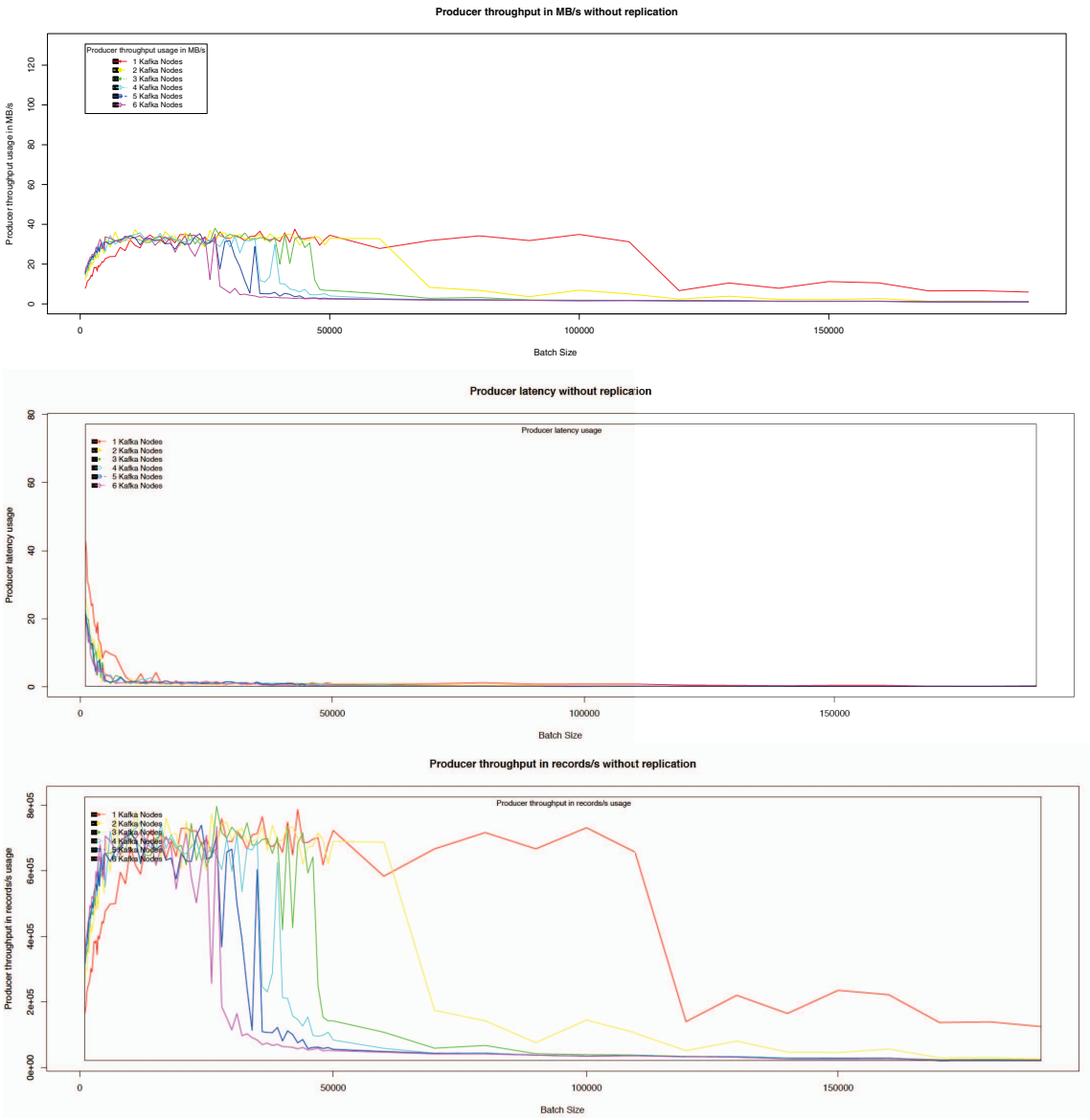

Fig. 4. Producer performances when modifying batch size for several number of nodes and a message size of 50B

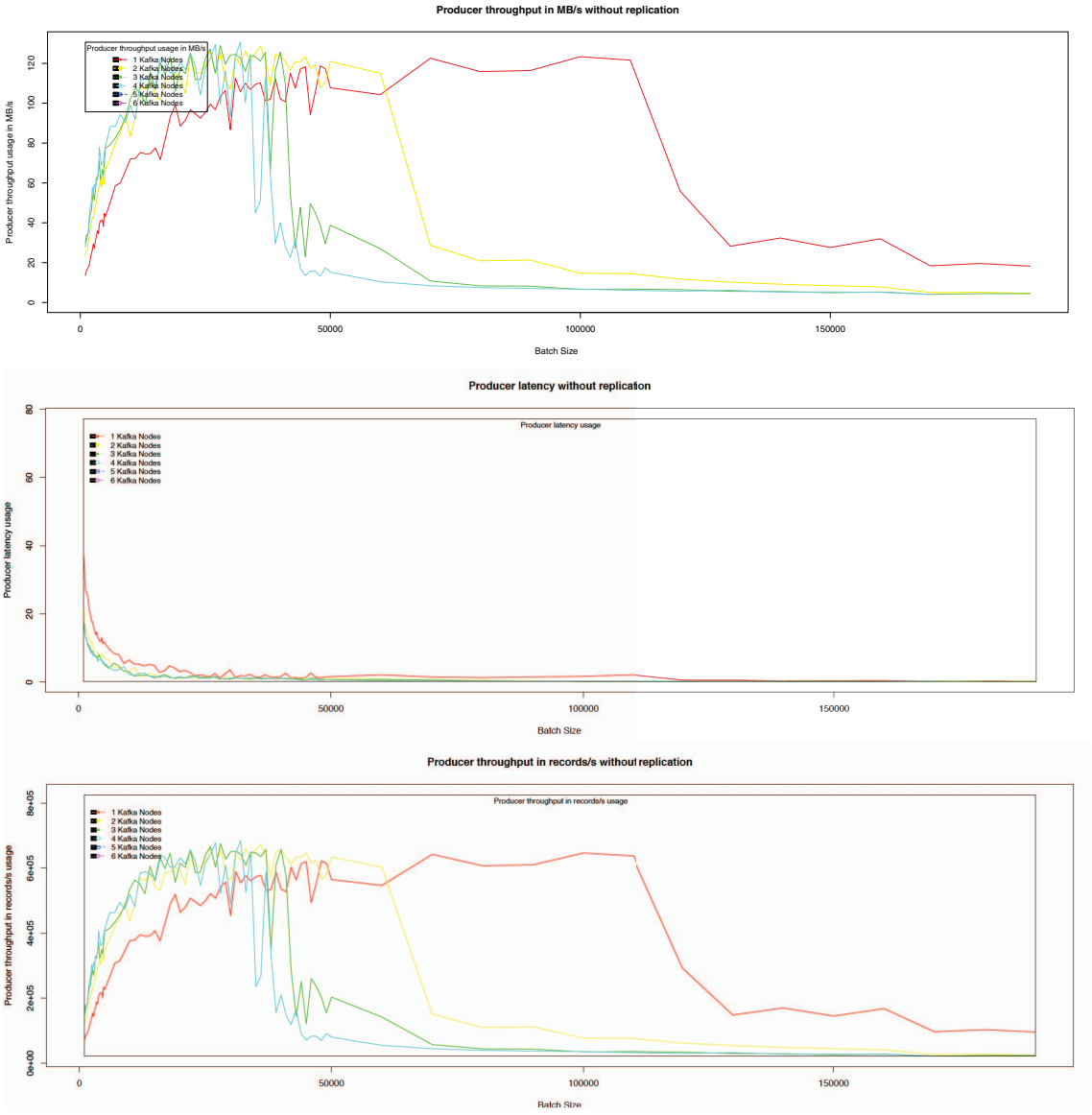

Fig. 5. Producer performances when modifying batch size for several number of nodes and a message size of 200B