Orchestration of CPU and GPU consumers for high-performance streaming processing

Mikhail M. Rovnyagin[1], Aleksey D. Gukov[2], Kirill V. Timofeev[3], Alexander S. Hrapov[4], Roman A. Mitenkov[5]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation
[1]m.rovnyagin.2015@ieee.org, [2]alekseygukov.mk@gmail.com, [3]timofeev.log@yandex.ru, [4]arcanius@list.ru,
[5]r.mitenkov96@gmail.com

***Abstract*— In the modern world, there are many systems using streaming data processing. Often, these systems use CPU and GPU devices in their calculations. It should be noted that such systems can fail for various reasons. Therefore, to optimize throughput, system designers need to determine in advance how many CPUs and GPUs to configure the system with. In our article, we present a possible architecture of such a system and present what methods can be used to calculate the optimal number of CPUs and GPUs with optimal throughput and taking into account other factors, for example, the cost of devices and the failure rate of the environment.**

Keywords*— *Kafka, consumers, CPU, GPU, failure statistic

I. INTRODUCTION

Today, in the IT industry, streaming data processing systems are in demand more than ever, for example, they are used wherever there is a large flow of valuable information. They can be useful in the tasks of analyzing customer banking transactions, in tracking links that users of various sites click on to increase conversion from advertising, etc. Streaming processors are challenged with a wide variety of domain-specific tasks, but for all of them, resiliency, scalability, and high throughput are often key requirements. As you know, in enterprises in such systems, CPUs and GPUs are used to parallelize and accelerate computations. In our work, we propose a principle of optimal use of these devices in such systems based on containerization technology, from the point of view of Kafka[3] consumers.

II. RELATED WORKS

Currently, in streaming data processing systems, the most common technologies as a message broker are Apache Kafka, Rabbit MQ, NATS Streaming[10]. In our research we will use Apache Kafka because it has the lowest latency compared to its counterparts [5] [6]. It is also worth noting that for the isolation of Kafka consumers or other apps, it is necessary to use Docker [2][9][11], which allows the use of CPU and GPU devices [7][12].

III. ARCHITECTURE DESIGN

Based on the studies described above, this article proposes an experimental system architecture (Figure 1), with the following components:

1. Sender - service for generating input messages for processing.
2. IN_TOPIC – Kafka topic with messages for processing
3. CPU and GPU Kafka consumer containers - a Docker – containers [2], which are consumers of Kafka messages and perform calculations using the CPU or GPU.
4. Failure simulation module - service for simulating failures of Kafka – consumers
5. Output topics with a log of the operation of CPU and GPU containers before performing calculations
6. Output topics with a log of the operation of CPU and GPU containers after performing calculations
7. Deduplicator[4] - application, which determines the statistics of the execution for calculations on the CPU and GPU Kafka consumers.

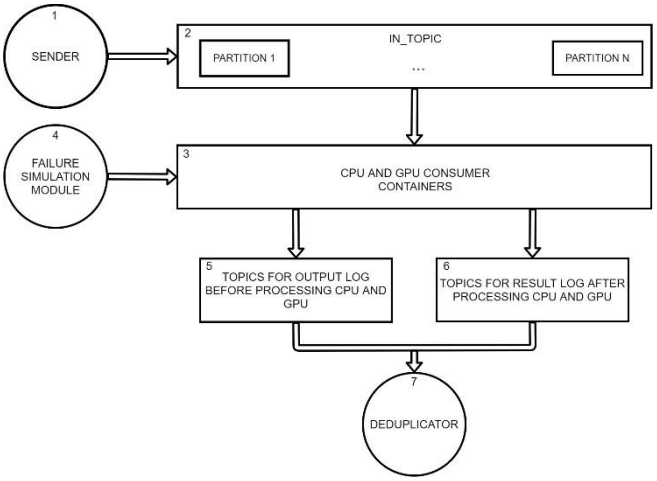

Fig. 1. Experimental system architecture

It is worth noting that CPU and GPU consumers, for greater approximation to real conditions, performs the task of ML analysis based on data coming from Kafka. In our case, these containers performed the function of recognizing character patterns from the MNIST dataset[1].

The general operating principle of the developed testing system can be described as follows. First, the sender starts, which writes predetermined number of messages to IN_TOPIC. The sender assigns a unique identifier to each message. Next, CPU and GPU consumers are launched. They cyclically perform the following actions:

978-0-7381-4275-3/21/$31.00 ©2021 IEEE

- Reading batch messages of a fixed size by consumers.
- Logging identifiers processed messages to the topic of preprocessing logs.
- Processing batch of messages by the investigated function.
- Logging the identifiers of processed messages to the post-processing log topic.
- Fixing the offset in IN_TOPIC.

Simultaneously with the launching of the CPU and GPU consumers, the container failure simulation module is launched, which, once every *failurePeriod* seconds, restarts the CPU and GPU containers of the consumers.

After the CPU and GPU consumers finish processing all the messages from IN_TOPIC, their work, as well as the work of the container failure simulation module, ends. The execution time of CPU and GPU of consumers is recorded. And then, it starts deduplicator who reads all the messages from the topics of the preprocessing and post-processing logs, and makes statistics for the CPU and GPU consumers about the number of messages processed by them, and the number of duplicates.

IV. Analysis of failure statistics

The container failure simulation module allows generating customer failures with a specified period. Since fixing the offset of CPU and GPU consumers in IN_TOPIC occurs only after logging and processing the batch, a failure that occurred in the middle of this process before fixing the offset will lead to the fact that consumers after restarting will re-process the messages of this batch. Thus, duplicates appear, the number of which directly affects to the operating time and throughput of consumers.

In general, we can say that the shorter the failure period (the higher the failure rate) and the larger the batch size, the lower the bandwidth of the CPU and GPU consumers. However, this dependence can be expressed in different ways and its nature is not necessarily the same for all combinations of the studied input parameters. This implies the need for additional analysis of the statistics obtained, determining the type of dependence and finding the appropriate coefficients. Knowing the dependence of the throughput of the function under study on a certain combination of input parameters, it can be used to calculate the proportions of the CPU and GPU launch of consumers to ensure the target throughput in an industrial environment.

Fixing the batch size in a series of experiments and studying the dependence of the throughput on the failure period (1):

$$throughput = f(failurePeriod), \quad (1)$$

a number of measurements are made. Since the analytical value of the function $f(failurePeriod)$ is unknown, there is a challenge to find a formula that would describe the functional relationship. This problem was solved using numerical methods. Let us present the empirical formula (2):

$$\widetilde{throughput} = \tilde{f}(failurePeriod), \quad (2)$$

the values of which at $failurePeriod = failurePeriod_i$ would differ slightly from the experimental data. The sought function (2) must belong to a sufficiently narrow class of functions (for example, linear, logarithmic, etc.) in order to make the problem more definite. The task comes down to determining the type of empirical formula and finding its coefficients.

The statistics collected by the testing system is presented as a set of graphs of the throughput versus the failure period for a fixed batch size. For each such graph, it is necessary to determine the type of empirical formula and find the values of the corresponding coefficients. The search for an empirical formula was considered on the example of experimental data obtained in the study of the MNIST function, as described in the architecture section. The resulting graphs of dependencies for CPU and GPU consumers are shown in the figure 2.

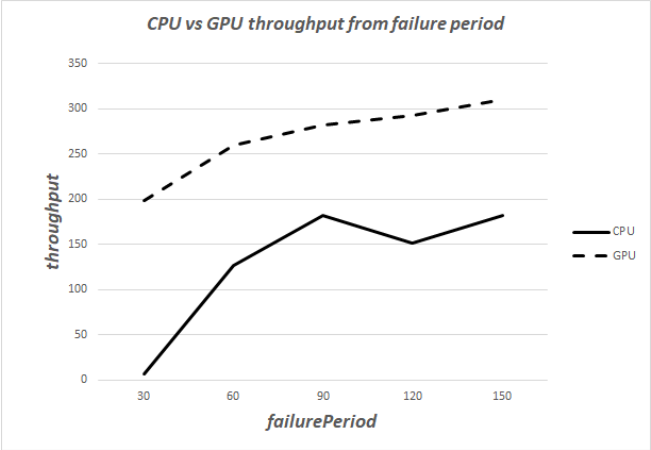

Fig. 2. Graphs of throughput versus failure period. (Experimental data)

The statistics collected by the testing system is presented as a set of graphs

The amount of experimental data in our example was relatively small, but according to the trend of the graph (Figure 2), we can make an acceptable approximation and assume that the functional dependence has a linear form and there is reason to believe the presence of a linear dependence. In this case, when searching for an empirical formula, the problem is reduced to finding the coefficients of a linear function of the form (3):

$$y = ax + b, \quad (3)$$

Where y – is a throughput and x – is a failurePeriod. After choosing the type of empirical formula, the problem is reduced to finding unknown coefficients.

Let's find the unknown coefficients using the least squares method. Following the method of least squares, the following notation is introduced. Let n be the number of points of experimental data, m - the number of unknown coefficients of the empirical formula. The empirical formula (4) is represented as:

$$y = \tilde{f}(x; a_1, a_2, \dots, a_m), \quad (4)$$

and the concept of deviation of the empirical formula from the initial data is introduced as (5):

$$\varepsilon_i = \tilde{f}(x_i; a_1, a_2, \dots, a_m) - y_i, \ i = 1,2, \dots, n, \quad (5)$$

According to the least squares method [8], the best coefficients $a_1, a_2, \dots, a_m$ are those for which the sum of the squares of the deviations (6) is minimal:

$$S(a_1, a_2, \dots, a_m) = \sum_{i=1}^{n} (\tilde{f}(x_i; a_1, a_2, \dots, a_m) - y_i)^2, \quad (6)$$

Then, the problem is reduced to finding the minimum of a function of several variables. Let's compose the system of equations (7):

$$\frac{\partial S}{\partial a_1} = 0, \frac{\partial S}{\partial a_2} = 0, \dots, \frac{\partial S}{\partial a_m} = 0, \quad (7)$$

If this system has a unique solution, then it will be the desired one. The system of equations for linear dependence according to the least squares method has the following form (8):

$$\begin{cases} a \sum_{i=1}^{n} x_i^2 + b \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ a \sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i \end{cases}, \quad (8)$$

Since the values of x_i and y_i are known, we calculate the values of the sums and substitute them into the system of equations. Solving the system of equations by one of the known methods, for example, by the Cramer method, we find the values of the coefficients *a* and *b*, at which the empirical linear function will approximate the experimental points in the best way in comparison with any other linear function.

As a result of the search for the coefficients of the empirical linear function by the least squares method for the experimental values of the points of the graph above, the following functional dependencies for the CPU and GPU Kafka consumers can be obtained (9):

$$\widetilde{throughput}_{CPU} = a_{CPU} failurePeriod + b_{CPU} \quad (9)$$

$$\widetilde{throughput}_{GPU} = a_{GPU} failurePeriod + b_{GPU}$$

The graphs of these functional dependencies, with the calculated values of the coefficients for us experiment, are presented in the graph (Figure 3).

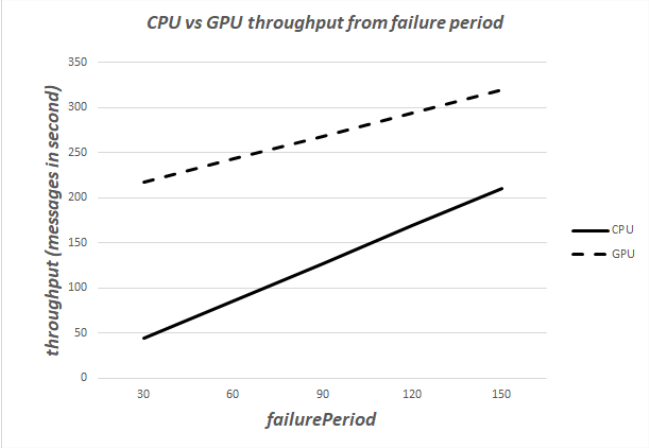

Fig. 3. The graph of the empirical dependence of the throughput dependence on the failure period.

V. Optimal proportions of CPU and GPU performers

Now that we know the approximate dependence of consumer throughput on failure frequency, we can estimate how many CPU and GPU performers need to have to achieve a given throughput at the lowest cost of equipment rental costs.

Suppose we have the following input parameters:

- $requiredThroughput$ - desired message processing throughput
- $failureFrequency$ - failure frequency per minute, which affects consumers Kafka
- $costCPU$ – the cost renting a single machine without the GPU, which can be run on the CPU computation only
- $costGPU$ - the cost renting a single machine without the CPU, which can be run on the GPU computation only

In the previous paragraph, we derived functions (9). These values can be used to calculate (10):

$$minNumCpuOnly = \frac{requiredThroughput}{\widetilde{throughput}_{CPU}} \quad (10)$$

$$minNumGpuOnly = \frac{requiredThroughput}{\widetilde{throughput}_{GPU}},$$

Where $minNumCpuOnly$ and $minNumGpuOnly$ are the minimum number of CPU and GPU consumers, allowing to achieve the *requiredThroughput* when using only the CPU or only the GPU Kafka consumers, respectively.

Thus, the required bandwidth can be achieved using only *minNumCpuOnly* CPU consumers or only *minNumGpuOnly* GPU consumers. This is a valid solution, but it does not take into account the possible differences in resource costs for renting a unit of a node of the corresponding type. Thus, to find the optimal solution in terms of resource costs, it is necessary to find the minimum function (11):

$$totalCost = costCPU * numCPU + costGPU * numGPU \quad (11)$$

where $totalCost$ is the total cost of the resulting solution, $numCPU$ and $numGPU$ are the unknown numbers of CPU and GPU consumers, respectively, to be found. Moreover, the resulting solution must satisfy (12):

$$estimatedThroughput \geq requiredThroughput \quad (12)$$

where,

$$estimatedThroughput = \widetilde{throughput}_{CPU} * numCPU + \widetilde{throughput}_{GPU} * numGPU \quad (13)$$

- the expected throughput when using the solution must be greater than or equal to the required.

Since *failureFrequency* is the frequency of failure per minute, *failurePeriod* will be (14):

$$failurePeriod = \frac{60}{failureFrequency} \quad (14)$$

Substituting *failurePeriod* and function coefficients into empirical formulas (9) we obtain approximate values of the consumer CPU and GPU throughput for a given value of the failure period (15,16):

$$\widetilde{throughput}_{CPU} = a_{CPU} * \frac{60}{failureFrequency} + b_{CPU} \quad (15)$$

$$\widetilde{throughput}_{GPU} = a_{GPU} * \frac{60}{failureFrequency} + b_{GPU} \quad (16)$$

And further, substituting the found values into the formulas (10) find *minNumCpuOnly* and *minNumGpuOnly* rounding up the values obtained as a result of division. In order to find the unknown *numCPU* and *numGPU* in formula (13), we need to enumerate (17):

$$numCombinations = minNumCpuOnly * minNumGpuOnly \quad (17)$$

combinations CPU and GPU proportions of consumers. For each possible combination of CPU and GPU proportions, the condition (12) should be checked. And among all the combinations that satisfy it, there is one for which the value (13) is minimal. As a result, a combination of *numCPU* and *numGPU* is found, which for the given input parameters allows you to achieve the *requiredThroughput* with minimal cost (*totalCost*).

Thus, the developed method for calculating the proportions of CPU and GPU Kafka consumers allows to calculate the number of CPU and GPU consumers with minimal costs, with the required throughput and failureFrequency. This approach can be useful in streaming systems where component failure thresholds are predefined and budget constraints exist.

REFERENCES

[1] Tensorflow MNIST handwritten digits example: [Online] / https://www.tensorflow.org/overview

[2] Docker: Enterprise Container Platform: [Online] / https://www.docker.com

[3] T. Shapira, Kafka: the Definitive Guide. O'Reilly Media, Inc, 2017, ISBN: 978-1491936153

[4] Mikhail M. Rovnyagin, Valentin K. Kozlov, Roman A. Mitenkov, Alexey D. Gukov, Anton A. Yakovlev, "Caching and Storage Optimizations for Big Data Streaming Systems", 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, 2020,doi: 10.1109/EIConRus49466.2020.9039502

[5] Philippe Dobbelaere and Kyumars Sheykh Esmaili. 2017. Industry Paper: Kafka versus RabbitMQ. In Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017, 12 pages. doi: 10.1145/3093742.309390

[6] Ranjith G Hegde, Nagaraja G S "Low Latency Message Brokers", International Research Journal of Engineering and Technology (IRJET) Volume: 07 Issue: 05 | May 2020, pp 2731-2738

[7] Nandinbaatar Tsog, Mikael Sjödin, Fredrik Bruhn, "Using Docker in Process Level Isolation for Heterogeneous Computing on GPU Accelerated On-Board Data Processing Systems" URN: urn:nbn:se:mdh:diva-45939

[8] Demidovich B.P., Maron I.A., Shuvalova E.Z. Numerical methods of analysis. Approximation of functions, differential and integral equations Moscow.: science, 1967 –pp. 79-101

[9] Han Wu, Zhihao Shang, Guang Peng, Katinka Wolter, "A Reactive Batching Strategy of Apache Kafka for Reliable Stream Processing in Real-time", 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 12-15 Oct. 2020, doi: 10.1109/ISSRE5003.2020.00028

[10] NATS Streaming: NATS Docs: [Online] / https://docs.nats.io

[11] Mikhail M. Rovnyagin; Kirill V. Timofeev; Aleksandr A. Elenkin; Vladislav A. Shipugin, "Cloud Computing Architecture for High-volume ML-based Solutions", 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, 2019, doi: 10.1109/EIConRus.2019.8656765

[12] D. A. Shulga, A. A. Kapustin, A. A. Kozlov, A. A. Kozyrev and M. M. Rovnyagin, "The scheduling based on machine learning for heterogeneous CPU/GPU systems," 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), St. Petersburg, 2016, pp. 345-348