Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 190 (2021) 684–689

Procedia Computer Science

www.elsevier.com/locate/procedia

2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society

Intelligent container orchestration techniques for batch and micro-batch processing and data transfer.

Mikhail M. Rovnyagin[a]*, Vladislav A. Shipugin[a], Kirill A. Ovchinnikov[a], Sergei V. Durachenko[a]

[a] *National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation*

Abstract

In the modern world, a large number of systems require the use of scaling, the provision standard for which at present time is orchestration and containerization technologies. In particular, information systems often operate with a data stream with different intensity, which requires the use of dynamic orchestration. This paper will propose methods for improving orchestration technology to increase the performance of data processing systems.

The monograph proposes a method for optimizing the use of resources by predicting the load on the cluster. A method for efficient allocation of containers to physical nodes is described. A dynamic orchestration method based on system performance is also presented.

Peer-review under responsibility of the scientific committee of the 2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society

Keywords: Orchestration; Docker; Spark;

1. Introduction

In modern systems, orchestration technology is increasingly used for application instances management. This technology allows automating the process of management, scaling and deployment of applications.

Varied systems require dynamic orchestration, the standard approaches to which do not effectively solve the problem of scaling and placing services in a cluster. The criticality of the improvement of these methods can be seen

* Corresponding author. Tel.: +7-915-227-0086.
E-mail address: mmrovnyagin@mephi.ru

1877-0509
Peer-review under responsibility of the scientific committee of the 2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society
10.1016/j.procs.2021.06.079

in the example of a data processing system operating in the near real-time mode. In this article, various techniques for improving the scaling and placement approaches for orchestration are offered.

2. Related works

Currently, a large number of works [1,2,3] are devoted to the subjecting of containerization of software modules. Container orchestration is especially in demand concerning Machine Learning modules [4] and IoT applications [5]. Containerization of data processing systems is no exception [6,7]. Currently, many different scaling and orchestration improvements have been proposed. The most progressive methods of improving orchestration are associated with systems of speculative execution using machine learning [8,9]. This approach permits us to predict the need for scaling in advance and carry it out before reaching the peak load. For such tasks, the LSTM (Long Short-Term Memory) algorithm shows good enough efficiency [10,11]. Furthermore, the contribution to expanded resource allocation is important [12]. Expanded resource allocation permits intelligently to determine the plan for the placement of services following the peculiarities of their network interaction. Moreover, approaches with automatic orchestration based on the monitoring system and applied metrics deserve attention [13].

3. Predicting required nodes in a cluster

This section will propose a method for optimizing batch processing by predicting the load on the cluster. The essence of the method is in the premature determination of the number of cluster computational nodes required to process continuously incoming data (Figure 1). This number of nodes should be sufficient to exclude waiting for resources necessary for data processing and, at the same time, redundant, so as not to create downtime of computing resources.

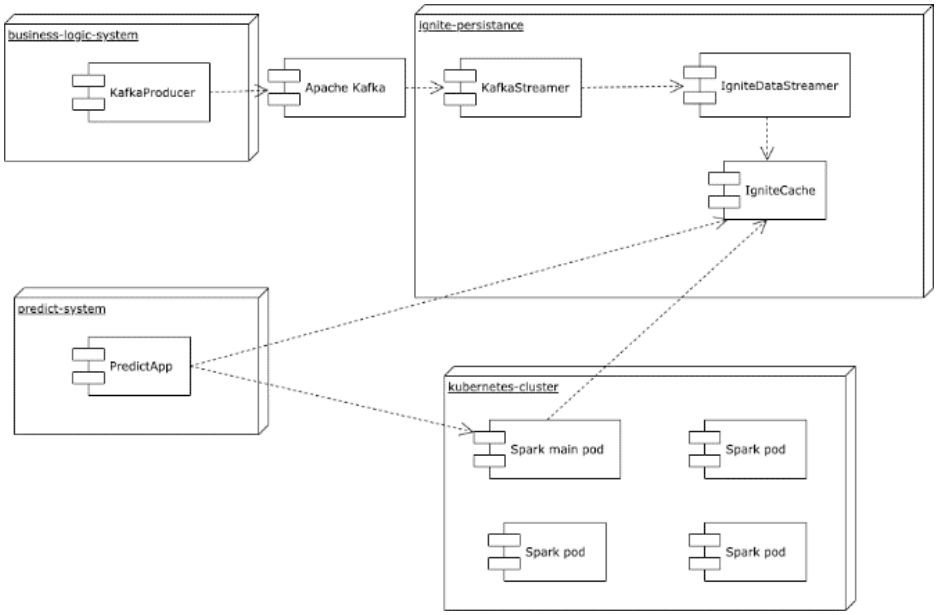

Fig. 1. the architecture of a batch processing system with load prediction.

The data for processing can accumulate, both at different rates and different volumes, which complicates the forecasting task. Data processing is performed periodically, at specified intervals. At the right time, the cluster is launched, and then the data is processed, and the result is saved for further analysis.

The problem with standard batch processing of data is that it requires a certain amount of time to start a cluster, and during this time the amount of data can increase non-linearly. Thus, the calculated amount of resources required

for processing will not be enough for efficient work. Speculative load prediction will allow you to start the cluster in advance and take into account future data to be processed, which will eliminate waiting for the release of resources necessary for computation and, thereby, speed up these calculations.

Forecasting can be implemented using linear regression algorithms, but the accuracy of such forecasting is not enough for complex systems. Using machine learning algorithms such as LSTM is better at predicting tasks.

The data stream continuously sends data through the Apache Kafka streams API to the Apache Ignite Cache. At some time, the PredictApp prediction system starts, predicts the planned amount of data and transmits the result to the data processing system. The data processing system launches a compute cluster with a predicted number of Apache Spark nodes. The cluster performs data processing and collects statistics for further analysis.

4. Intelligent container allocation

In this section will be proposed a method for ensuring the locality of the interaction of containerized systems by determining the most efficient allocation of containers across physical nodes of the cluster. The idea behind the method is to reduce network interaction between cluster nodes by placing the most frequently interacting services into one physical node.

The solution to the problem is reduced to the problem of clustering a graph model into K clusters, where K is the number of nodes. You can use your own tracing solution or any available tracing solution to get a graph describing the interaction of services.

Consider a graph $G = <V, E>$, where V is the set of vertices (containers), and E is the set of edges (the weight of an edge characterizes the frequency of transfers and the size of messages). Let C be the set of nonempty subsets of the set V. Thus, a partition into clusters of a set V of vertices of a graph G is a mapping $\varphi: V \to C$, for which condition (1) is satisfied.

$$\mathbb{E}(\varphi) = \{C_j\}_{j=1,k} \subset C, \forall\, i,j\ (1 \le i,j \le k): C_i \cap C_j = \emptyset, V = \cup_{i=1}^{k} C_i \tag{1}$$

Elements of the set of values $\mathbb{E}(\varphi)$ of the mapping φ will be called clusters.

Let some estimate of the partition $Q(\varphi)$ be given, then the problem of clustering the set of vertices V of the graph G can be written in the form of the formula (2).

$$\varphi_{max} = arg\ max\ Q(\varphi) \tag{2}$$

To satisfy the locality requirement, the partition estimation function must encourage the minimization of inter-cluster connections. Thus, as an estimate for Q, we will consider modularity [14], which will be calculated using the Newman-Girvan formula (3).

$$Q = \frac{1}{2W}\sum_{ij}(A_{ij} - \frac{d_i d_j}{2W})\delta(P_i, P_j) \tag{3}$$

W is the sum of the weights of all edges, A_{ij} is the weight of the edge between nodes i and j, P_i is the subset to which the vertex i belongs, δ is the delta function.

When using the greedy clustering algorithm, you must also take into account the resource capacity of the physical cluster nodes. That is, it is necessary to prohibit any permutations that lead to a situation when the sum of the resource capacities of the vertices (containers) exceeds the resource capacity of the physical cluster node. This condition is expressed in the form of the formula (4).

$$\sum_{i}^{n}\left(u_{i_k}\right) < N_k, \tag{4}$$

where n is the number of vertices in a cluster, k is a resource (CPU, RAM, Persistent Volumes), N is a physical node, u_i is a cluster node.

Also, the initial partition φ_0, must not violate conditions (4) for all clusters.

Thus, by redistributing containers among nodes using the described mechanism, the throughput of the entire system can be increased.

5. Micro-batch data processing systems orchestration

This section will offer options for optimizing containerized data processing by cluster scaling depending on the system load and using micro-batch data stream processing.

Apache Spark is one of the most favoured data processing tools recently. Apache Spark provides two standard interfaces for streaming data processing: Spark Streaming and Structured Streaming. Below, three approaches to data flow processing will be compared in terms of performance and throughput: processing using Spark Streaming, Structured Streaming and the proposed micro-batch processing approach. In all approaches, the consuming and producing of data is performed as a stream using Apache Kafka.

The first approach of data processing implements streaming processing using Spark Streaming. Spark Streaming provides a high-level abstraction called discretized stream, which represents a continuous stream of data. Discretized stream is defined by a continuous series of RDDs (resilient distributed dataset), which is Spark's abstraction of an immutable, distributed dataset. Each RDD in a discretized stream contains data from a certain interval.

The second approach implements streaming processing using Spark Structured Streaming. The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended. This leads to a new stream processing model that is very similar to a batch data processing model.

The third approach to data processing implements micro-batch data processing. In this case, the data consumes as a stream and is stored in an unbounded table. Next, a scheduled job starts processing, which receives data from an unbounded table using a sliding window and aggregates it. Data processing is performed by the Spark RDD batch processing interface.

The data processing approaches described above were compared in terms of performance and throughput in a series of experiments. The graph of the data processing latency versus the sliding window size is demonstrated in Figure 2.

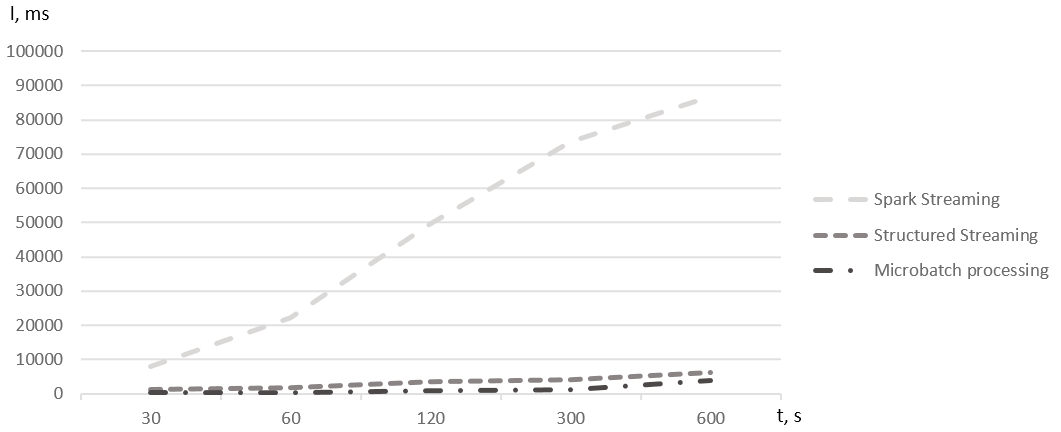

Fig. 2. Data processing performance

Figure 3 illustrates a graph of the throughput (messages per second) of each approach versus the sliding window size.

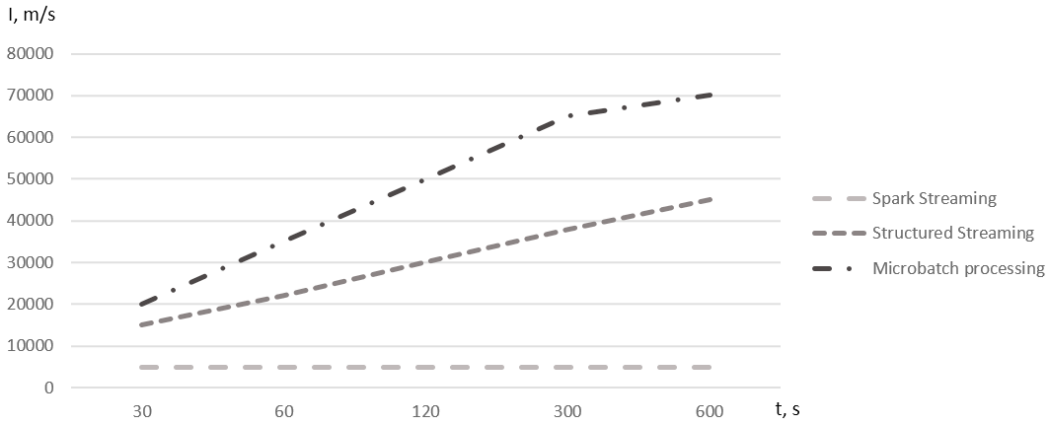

Fig. 3. Data processing throughput.

Experimental results demonstrate that micro-batching processing is superior in performance and has significantly higher throughput compared to standard streaming methods implemented in Apache Spark.

In streaming data processing, data can arrive at a different intensity, which will lead to an increase in processing time under high load or suboptimal resource use under low loads. Using metric-based container scaling solves this problem [15]. There are various approaches to this solution, including using machine learning [13]. Below an automated orchestration approach based on queuing theory is proposed.

The number of processors required in a streaming data processing system can be calculated using queuing theory. The main parameter in these calculations will be the intensity of the input flow - frequency of messages occurrence or the average number of messages received per unit of time. In our case, we will assume that the intensity of the input flow is a constant value in the time interval. Furthermore, to calculate the number of handlers, it is necessary to know the parameter intensity of the service flow - the average number of messages that one node in the cluster can serve per unit of time and the probability of a node failure. The optimal number of handlers in the cluster can be calculated with the next formula:

$$n = \frac{\lambda}{\mu}(1 - p_r), \tag{5}$$

where n is the number of handlers, λ is the intensity of the input flow, μ is the intensity of the service flow, p_r is the probability of a node failure.

The value of the service flow intensity is a value inversely proportional to the service time or, in our case, to the value of the data processing latency. Then formula (5) takes the form:

$$n = \lambda l(1 - p_r), \tag{6}$$

where l is the processing latency.

Containers scaling with the calculation of the required containers number according to formula (6) will solve the problem of non-optimal use of resources with an input flow of variable intensity.

6. Conclusions

Thus, this article introduces various container orchestration and placement techniques that can improve overall system throughput and performance. These approaches can be applied in systems with the need for dynamic orchestration depending on metrics. In the future, additional research is planned to compare the performance of the proposed mechanisms with standard ones.

References

[1] M. M. Rovnyagin, K. V. Timofeev, A. A. Elenkin and V. A. Shipugin, "Cloud Computing Architecture for High-volume ML-based Solutions," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 2019, pp. 315-318, doi: 10.1109/EIConRus.2019.8656765.

[2] M. M. Rovnyagin and A. S. Hrapov, "Presentation of the PaaS System State for Planning Containers Deployment Based on ML-Algorithms," 2020 Moscow Workshop on Electronic and Networking Technologies (MWENT), Moscow, Russia, 2020, pp. 1-5, doi: 10.1109/MWENT47943.2020.9067488.

[3] M. M. Rovnyagin, A. S. Hrapov, A. V. Guminskaia and A. P. Orlov, "ML-based Heterogeneous Container Orchestration Architecture," 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 2020, pp. 477-481, doi: 10.1109/EIConRus49466.2020.9039033.

[4] R. R. Karn, P. Kudva and I. A. M. Elfadel, "Dynamic Autoselection and Autotuning of Machine Learning Models for Cloud Network Analytics," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 5, pp. 1052-1064, 1 May 2019, doi: 10.1109/TPDS.2018.2876844.

[5] S. Kikuchi et al., "Orchestration of IoT Device and Business Workflow Engine on Cloud," 2018 3rd Cloudification of the Internet of Things (CIoT), Paris, France, 2018, pp. 1-2, doi: 10.1109/CIOT.2018.8627118.

[6] V. Munerman and D. Munerman, "Realization of Distributed Data Processing on the Basis of Container Technology," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 2019, pp. 1740-1744, doi: 10.1109/EIConRus.2019.8656766.

[7] Y. Fu et al., "Progress-based Container Scheduling for Short-lived Applications in a Kubernetes Cluster," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 278-287, doi: 10.1109/BigData47090.2019.9006427.

[8] B. Xie, G. Sun and G. Ma, "Prediction-based autoscaling for container-based PaaS system," 2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS), Kota Kinabalu, 2017, pp. 19-24, doi: 10.1109/I2CACIS.2017.8239026.

[9] P. Yazdanian and S. Sharifian, "Cloud Workload Prediction Using ConvNet And Stacked LSTM," 2018 4th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS), Tehran, Iran, 2018, pp. 83-87, doi: 10.1109/ICSPIS.2018.8700546.

[10] Y. Yang and L. Chen, "Design of Kubernetes Scheduling Strategy Based on LSTM and Grey Model," 2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Dalian, China, 2019, pp. 701-707, doi: 10.1109/ISKE47853.2019.9170419.

[11] N. Marie-Magdelaine and T. Ahmed, "Proactive Autoscaling for Cloud-Native Applications using Machine Learning," GLOBECOM 2020 - 2020 IEEE Global Communications Conference, Taipei, Taiwan, 2020, pp. 1-7, doi: 10.1109/GLOBECOM42002.2020.9322147.

[12] R. Kandan, M. F. Khalid, B. I. Ismail and O. H. Hoe, "Advanced resource allocation and Service level monitoring for container orchestration platform," 2019 IEEE International Conference on Sensors and Nanotechnology, Penang, Malaysia, 2019, pp. 1-4, doi: 10.1109/SENSORSNANO44414.2019.8940058.

[13] R. Chowdhury, C. Talhi, H. Ould-Slimane and A. Mourad, "A Framework for Automated Monitoring and Orchestration of Cloud-Native applications," 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 2020, pp. 1-6, doi: 10.1109/ISNCC49221.2020.9297238.

[14] Fortunato S. Community detection in graphs. Physics reports, 2010, vol. 486. N. 3-5, pp. 75-174. https://doi.org/10.1016/j.physrep.2009.11.002

[15] E. Casalicchio and V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, AZ, 2017, pp. 207-214, doi: 10.1109/FAS-W.2017.149.