Cloud Computing Architecture for High-volume Monitoring Processing

Mikhail M. Rovnyagin[1], Viktor V. Odintsev[2], Dmitrii Y. Fedin[3]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Moscow, Russian Federation
[1]m.rovnyagin.2015@ieee.org, [2]zakhams@gmail.com, [3]fediny@list.ru

Andrey V. Kuzmin
Penza State University
Penza, Russian Federation
flickerlight@inbox.ru

***Abstract* — Monitoring systems have historically been used to obtain, process and display metrics coming from applications or devices. Now providers are massively reoriented to SDN / NFV infrastructure. An increasing number of applications operate on a micro-service architecture. Thus, in addition to the natural evolutionary growth of the collected metrics number, there is an abrupt increase in connection with the change of technology. There is a need for correlation analysis, which does not allow scaling of old monitoring systems using fragmentation. This article proposes a linearly scalable architecture of the monitoring system. The proposed approach makes it possible to increase the reliability of processing metrics and to perform cross-domain correlation analysis with the use of computable metrics. The features of real-time control system construction based on the proposed architecture are also described.**

Keywords — high performance computing; cloud; monitoring; high-volume processing; scaling

INTRODUCTION

Computing clusters are actively used to handle large amounts of data. Tasks are divided into many servers in clusters and run in parallel [1]. Horizontal scaling is used to increase cluster performance (new computing servers are added to the cluster). But simultaneously with the increase in the number of servers, the probability of failures increases, as well as the complexity of monitoring its state. For the timely detection of problems and their possible prevention, monitoring systems are used. However, for monitoring systems, there is also a scaling problem. With the increase in the number of observed characteristics for the monitoring system, more computing power is required. Therefore, for clusters containing thousands of servers, there is a need for linearly scalable monitoring systems. A distributed monitoring system without a single processing server will have linear scalability, which will effectively control clusters with any number of machines.

This article proposes the architecture of a linearly scaled distributed monitoring system, equipped with a data analysis system.

I. COMPARISON OF MONITORING SYSTEMS

Most monitoring systems consists of the following components: monitoring server, database monitoring system, monitoring agents.

Important criteria for the monitoring system are: scalability, fault tolerance, portability, flexibility. For comparison, three popular monitoring systems are selected: Zabbix, Nagios and Zennos [2]. Comparative characteristics for the given criteria are presented in Tables 1, 2 and 3.

TABLE I. SCALABILITY

Tool	Comparative characteristics
Zabbix	Load distribution on multiple nodes using a proxy [3], as well as a database scaling. Performance is highly dependent on database performance.
Nagios	Load distribution into several nodes using Fusion [4], as well as master / worker separation using Nagios plugins.
Zenoss	The ability to distribute the load of various system components (collectors, hubs) on several nodes [5].

TABLE II. FAULT TOLERANCE

Tool	Comparative characteristics
Zabbix	It is possible to run several servers.
Nagios	
Zenoss	A management server can redistribute tasks between nodes. The operation of the nodes is not disrupted when the management server is temporarily unavailable.

TABLE III. PORTABILITY AND FLEXIBILITY

Tool	Comparative characteristics
Zabbix	It is possible to automatically search for monitoring objects on the host, work through SNMP, configuration via the Web-interface.
Nagios	The configuration is done through configuration files, monitoring is performed using various plugins.
Zenoss	Configuration and operation through various protocols (eg SNMP, SSH), the agent is not used.

Over the past decades, monitoring and analysis systems have evolved rapidly, taking into account the requirements of horizontal scalability and fault tolerance. Many modern monitoring systems were initially created as distributed fault-tolerant systems.

In the course of this study, papers [6] [7] were also studied. [6] presents a scalable real-time monitoring architecture built on the MapReduce model. This work also addresses the problems of the existing monitoring systems described above. [7] presents

the architecture of a scalable agentless monitoring system for small and medium-sized networks using SNMP.

II. System Architecture

To effectively monitor the status of monitoring objects, it is necessary to ensure rapid data transfer, reliable storage, and also equip the system with tools for processing and analyzing metrics.

To collect metrics, you can use both agents of existing monitoring systems, as well as the SNMP protocol used in many modern open and proprietary monitoring systems [8].

A. Storage Architecture

To store metrics and information about monitoring objects, the Apache Cassandra database [9] is used. This DBMS is selected for the following reasons:

- Cassandra is a fully distributed DBMS that will scale the cluster without degrading the performance of the monitoring system and the processing system of calculated metrics;
- Cassandra has a high speed of processing simple queries (for example, getting a range) due to the features of the internal structure.

To work with metrics, the database stores the following information: information about metric sources and raw metrics.

B. Queue for writing metrics

Creating a queue for recording metrics from remote sources to the database allows you to ensure the reliability of data transfer and storage, as well as load balancing on the database nodes. To implement such a queue, Apache Kafka is used.

To work with sources of raw metrics, a topic is registered in Kafka, and nodes, united in the consumer group in Kafka terminology, register to receive data from this topic [10], evenly process incoming metrics and record them in the database. The performance of such a system is provided by balancing the load on the nodes of the group, as well as using batch queries when writing to the database.

C. Metric calculation module

Work with calculated metrics is provided by several computational nodes interacting with each other via Apache ZooKeeper [11]. Each node processes previously recorded metrics in the database and computes values using formulas specified through a special REST interface and stored in the database. The most frequently used in monitoring systems are the following formulas:

$$x = a + b, \text{ triggered by } a \quad (1)$$

$$x = SUM(a, 100\ ms) \quad (2)$$

Simple formulas (1) are processed for each incoming metric and represent an arithmetic expression of arbitrary complexity. Formulas with a sliding window (2) perform an operation on several metrics through a given interval or for each metric. With this organization of the metric recording system, the delay of the input values is possible, which has a negative effect on the accuracy of the calculations, but the calculation values are formed almost immediately upon their arrival, providing a high response rate to the value [12].

D. Aggregation and Analysis Task Manager

To aggregate and analyze stored raw metrics, a system is used consisting of: a distributed task manager and individual modules that perform the necessary tasks of aggregation and correlations. The execution of the modules is controlled by the task manager.

The task manager has the following properties:

- Distributed work. There is no chief manager, all instances are equal;
- Managers monitor the active state of each other. In the event that one of the managers for too long does not show activity, the tasks he launches are redistributed among active managers as evenly as possible;
- Interaction between managers is carried out using a distributed database. At the beginning of the work, the manager creates a record about himself in the database. Further, during the operation, it periodically updates the time of the last activity. In the event that the manager did not update the time of the last activity during the specified period, it is considered to be non-working.

Programs that aggregate and analyze data are presented in the form of modules launched by the task manager. To start tasks, the corresponding records are added to the database. Further tasks are automatically allocated for execution between managers.

E. Aggregation module for raw metrics

For each metric, a task is created where the aggregation period and type are specified. The type can be: sum, maximum value, minimum value, average value. The results are saved in a separate table in the database.

F. Aggregated Metrics Analysis Modules

For the analysis, two types of modules are used: a module for detecting trends in metrics and a module for detecting correlations between metrics.

The tasks for the Trend Detection Module contain the following information: the metric to be analyzed, the action to be performed, the expected effect, the expected value, the time period. The module for detecting trends works as follows: for a given metric, the function of the dependence of the value of the metric on time is extrapolated. In the event that the function is expected to reach the set point for a given time, the specified action is performed.

The tasks for the correlation detection module contain the following information: the analyzed group of metrics, the value of the linear correlation coefficient module, performed when the correlation action is detected. The correlation detection module works as follows: for a given group of metrics, the modulus of the linear correlation coefficient is pairwise determined. If it exceeds the specified value, the specified action is performed.

Information about managers, information about tasks, and aggregated metrics are stored in the database.

The scheme of the system architecture is shown in Fig. 1.

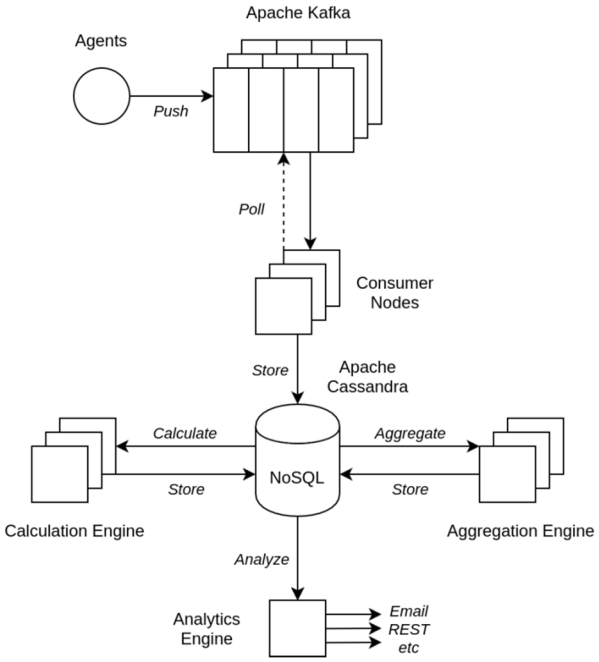

Fig. 1. System architecture

III. Description of the Test Stand

The test stand is made in the form of an isolated environment from a set of Docker containers located on virtual machines [13] and managed by Docker Compose. The following types of containers are used:

- Kafka Broker - Apache Kafka node;
- Producer - provides generation of test metrics;
- Consumer - provides recording of test metrics in the database;
- Cassandra - database node Apache Cassandra;
- Calculation Node - the node for aggregating and calculating modules;
- ZooKeeper - provides synchronization of Kafka Broker-nodes and Calculation Engine-nodes;
- REST - a web application for managing formulas;
- Logstash - a server for collecting logs from various nodes;
- Elasticsearch - log storage and processing server that works in conjunction with Logstash.

The number of nodes of different types is a configurable parameter in the configuration file. A special configuration script creates a docker-compose.yml file that describes the architecture of the test stand that you can use in Docker Compose.

Controlling the generation of test data and interaction with the REST interface is also done through the use of scripts.

The generated configuration files are distributed among the virtual machines from the point of view of optimal loading of each machine. The test stand is shown in Fig. 2.

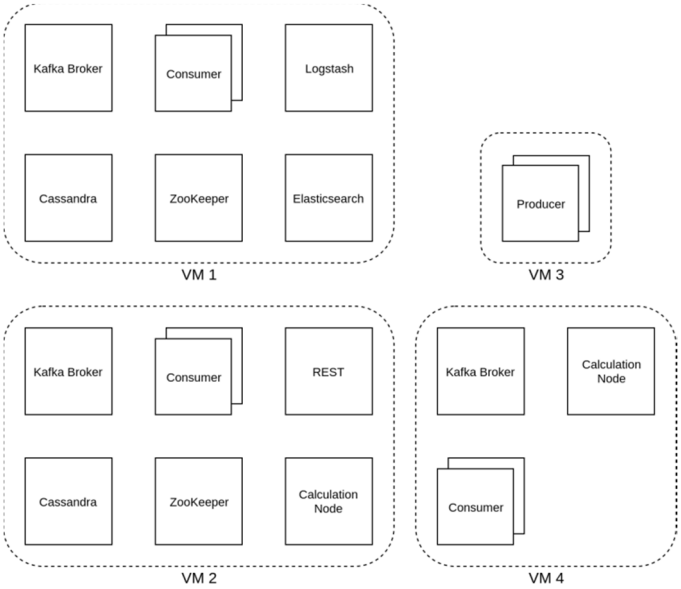

Fig. 2. Scheme of the test stand

After the system was started, 4 simple formulas were created, making the summation of the two types of metrics. The Kafka Producer nodes were configured to generate test metrics of 8 types.

IV. Performance Evaluation

Kafka Producer nodes can be configured to generate metrics at a specified interval, which can be adjusted by the parameter and the addition of new nodes.

To evaluate the performance of the system on these virtual machines, a test metric generator was generated, generating 10,000 metrics at different intervals, after which the average delay in processing the metric was calculated. In this experiment, the virtual machine VM 4 was not involved. The resulting histogram for different metric values per second (MPS) is shown in Fig. 3.

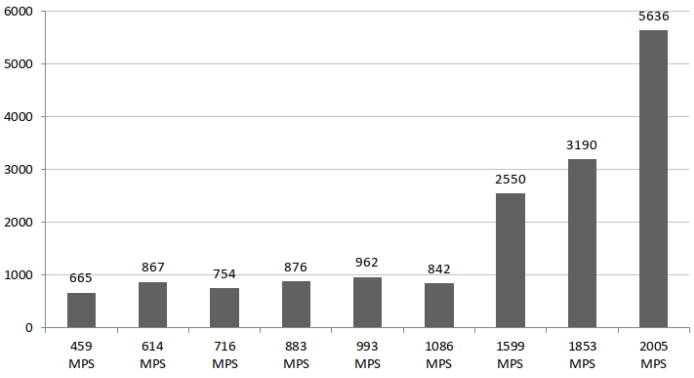

Fig. 3. Histogram of delay in milliseconds versus the number of metrics per second

The delay indicates the speed of the system response to the incoming metric. From this graph it can be seen that the system copes with the flow of metrics in 1086 MPS, but at 1599 MPS the reaction delay greatly increases with time.

A comparative graph of the server load for these MPS values is shown in Fig. 4. The graph shows the first 100,000 metrics from the start of the test metric generator.

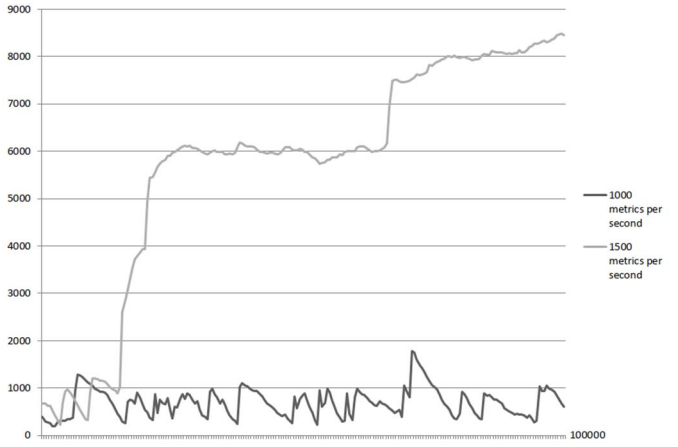

Fig. 4. Graph of delay versus time of data load start in milliseconds with different values of MPS

This graph demonstrates a constant low delay at a generator speed of 1000 MPS and a time-increasing value at a generator speed of 1500 MPS (overload of the current system configuration [14]).

With the use of another virtual machine, the system allows processing metrics at a speed of 1500 MPS. The server load schedule for VM 4 is shown in Fig. 5.

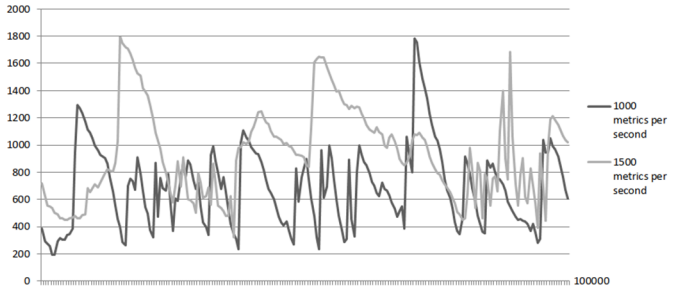

Fig. 5. A plot of the delay time of the load data in milliseconds for various values of MPS when enabled virtual machine VM 4

This graph shows that the system is well scalable horizontally.

A study of the state of the system and the contents of the logs allows you to determine that the message queue and the compute nodes make the greatest load if there are a large number of processed formulas. To increase performance, you need to include additional nodes Kafka Broker, Consumer, and Calculation Node, if you want to process more formulas.

V. Conclusions

The proposed architecture of monitoring systems allows to collect metrics from thousands of servers of cloud infrastructures and perform their further analysis. Distributed architecture allows to effectively control clusters with any number of machines, as well as provides fault tolerance of the monitoring system.

In the course of the study, it was determined how well the system copes with the long-term load, and also experimentally verified the property of linear scalability of the architecture proposed in the article.

References

[1] M. M. Rovnyagin and A. A. Kuznetsov, "Application of hybrid computing technologies for high-performance distributed NFV systems," 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, 2017, pp. 540-543.

[2] J. Hernantes, G. Gallardo, and N. Serrano, "It infrastructure-monitoring tools," Software, IEEE, vol. 32, no. 4, pp. 90–92, Jul. 2015.

[3] "Zabbix Documentation: Distrubuted monitoring, proxies" [Online]. Available at: https://www.zabbix.com/documentation/3.2/manual/distributed_monitoring/proxies (accessed 15 December 2017).

[4] "Monitoring Architecture Solutions For Managed Service Providers" [Online]. Available at: https://assets.nagios.com/downloads/general/docs/Monitoring_Architecture_Solutions_For_MSPs.pdf (accessed 15 December 2017).

[5] "Zenoss Service Dynamics 5.0: Architecture Overview" [Online]. Available at: https://www.zenoss.com/sites/default/files/zenoss-service-dynamics-architecture-overview-wp.pdf (accessed 15 December 2017).

[6] M. Andreolini, M. Colajanni, and M. Pietri, "A scalable architecture for real-time monitoring of large information systems," 2012 Second Symposium on Network Cloud Computing and Applications, IEEE, pp. 143-150, Dec. 2012.

[7] M. Brattstrom and P. Morreale, "Scalable Agentless Cloud Network Monitoring," 2017 4th International Conference on Cyber Security and Cloud Computing, IEEE, pp. 171-176, Jul. 2017.

[8] J. Swarna, C. Senthil Raja and Dr.K.S. Ravichandran, "Cloud monitoring based on SNMP," Journal of Theoretical and Applied Information Technology, vol. 40, no. 2, pp. 188-193, Jun. 2012.

[9] A. Lakshman and P. Malik, "Cassandra - A Decentralized Structured Storage System", SIGOPS Operating System Review, vol. 44, no. 2, 2010.

[10] Z. Wang, W. Dai, F. Wang, H. Deng, S. Wei, X. Zhang and B. Liang, "Kafka and Its Using in High-throughput and Reliable Message Distribution," 8th International Conference on Intelligent Networks and Intelligent Systems, IEEE, pp. 117-120, Nov. 2015.

[11] L.B. Goel and R. Majumdar, "Handling Mutual Exclusion in a Distributed Application through Zookeeper," 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), pp. 457-460, Jul. 2015.

[12] A. Pras, J. Schoenwaelder, M. Burgess, O. Festor, G. Martinez Perez, R. Stadler, and B. Stiller, "Key research challenges in network management," IEEE Communications Magazine, vol. 45, no. 10, p. 105, Oct. 2007.

[13] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," 2017 International Systems Engineering Symposium, IEEE, Oct. 2017

[14] I. S. Kamenskikh, D. M. Sinelnikov, D. S. Kalintsev, A. A. Kozlov, M. M. Rovnyagin and D. A. Shulga, "Software development framework for a distributed storage and GPGPU data processing infrastructure," 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), St. Petersburg, 2016, pp. 216-219.