Моделирование и анализ информационных систем. Т. 26, № 2 (2019), с. 195–202
Modeling and Analysis of Information Systems. Vol. 26, No 2 (2019), pp. 195–202

Computer system organization

DOI: 10.18255/1818-1015-2019-2-195-202

UDC 004.415.538

Threshold Analysis of Request Degradation in the Computer Network

Kirnos V. P.

Received August 10, 2018
Revised May 13, 2019
Accepted May 15, 2019

Abstract. In this paper, the existing approaches to the assessment of computer networks performance are considered. The standard structure of a network of the application layer of the OSI model using the example of *SBIS3* application (product of Tensor Company) is treated.

Further, two approaches allowing to analyze degradations in a network are considered – on the basis of aggregated data and the operational analysis.

The degradation study of more than 60 000 request types between two versions of application which works on the basis of the computer network is the cornerstone of the first decision. Each type of requests is described by four based metrics, each metrics representing a time series. The input data are aggregated every 10 minutes before an analysis algorithm. Further, the threshold criteria based on mathematical expectation and dispersion within two adjacent versions of the software are used. Such an approach allows to significantly reduce time for the analysis of potential problems in case of updates within the computer network.

The second decision is based on not aggregated input data. It consists of detail information about all requests, there are data section of the computer network. A threshold criterion is based on durations in the selected queue. This analysis type allows to diagnose the errors with problem clients.

Keywords: threshold criteria, mean, metric, queue

For citation: Kirnos V. P., "Threshold Analysis of Request Degradation in the Computer Network", *Modeling and Analysis of Information Systems*, **26**:2 (2019), 195–202.

On the authors:
Vasilii P. Kirnos, orcid.org/0000-0002-6150-3352, QA performance engineer
Tensor Company
36 Uglichskaya str., Yaroslavl 150027, Russia, e-mail: vp.kirnos@tensor.ru

Introduction

The paper studies the problem of continuous delivery. It shows how we promote our software into production more and more effectively. Every version release candidate is treated as a real productive version. Our understanding of how the Web works has led us to develop better ways of having machine-to-machine communication. Virtualization platforms allowed us to provide and resize our machines at will, with infrastructure automation giving us a way to handle these machines at scale. The main solution is setting Service Level Agreement (SLA) for testing the system [2].

1. Infrastructure

In our company we divided all functionality in the service architecture. There are more than 700 services in our structure: 100 of them are services for users, 50 are for all cloud administration and 550 of other types. Services are not the smallest unit. In the world development practice you can find that the concept of microservices is more adaptive. To transform a services-oriented structure to a microservices-oriented architecture is not easy [1].

Every month we normally have a new code of all and new services in production with its own logic with machine to machine communication. Our top management does not like when the data center grows after the release iteration. That is the main issue for quality assurance. Every month we have to control the resources which are used by the code, that is rather complicated as there are more than 700 developers.

2. Logs in the System

All these services have a logging system, which is used by the developers to optimise their functionality. We use these logs to make issues in our troubleshooting system. But the data from the logs are so great, that nobody can easily analyze them string by string.

We decided that we would save the operation logs (Fig. 1) only for maximum 3 days, and if we want more, we need some aggregation. The aggregation is based on

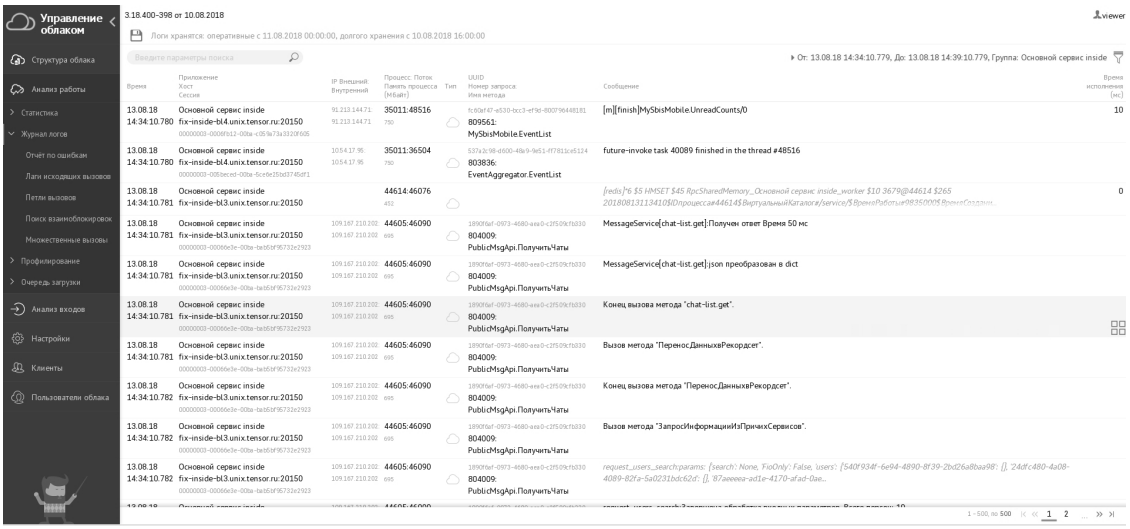

Fig 1. Operation logs

ClickHouse system (Fig. 2) developed by the Yandex team. It is a very fast column-oriented database for preparing the report. There are 6 stands in the company: two of them are only for developers, two for testing, one for debugging production database with a new middleware code and a pre-released stand.

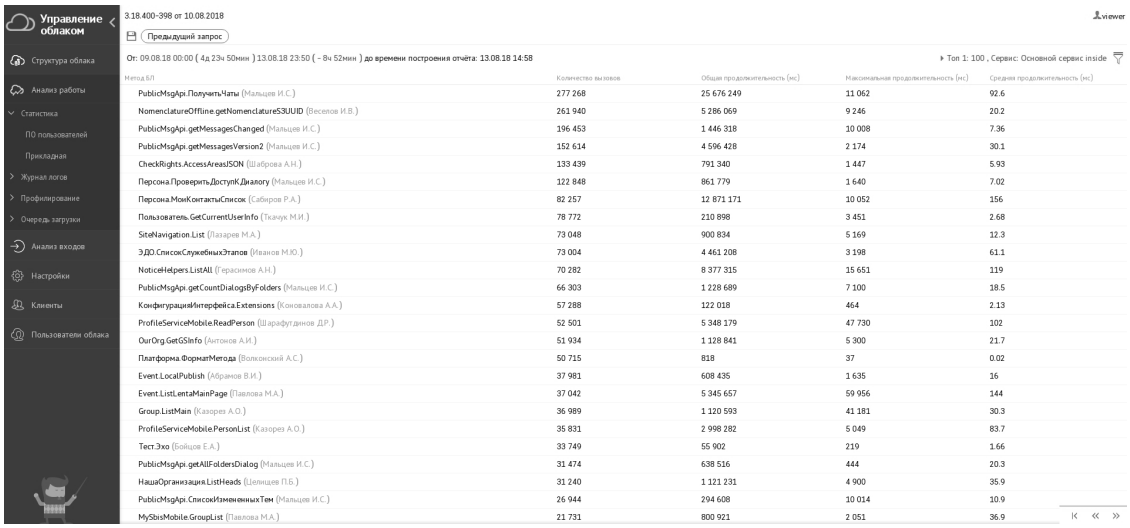

Fig 2. Aggregated logs

3. Aggregated Data Method of Analysis

The protocol of M2M communication in the cloud is json-rpc over http by default. Each service has its own method to communicate with the others. There are more than 60 000 methods of business logic in the cloud.

We are trying to compare methods performance between two major releases. There are almost more than 60 metrics for comparison. But for us the main metrics are Average time, Maximum time, Quantity of calls and Summarized time. All methods can be visualized and after that analyzed by engineers of the performance testing department, but it usually takes 4 seconds to prepare one figure and 2 seconds to analyze it in a good situation.

Preparing and analyzing 60 000 will take more than 83 hours of human time. That is why we try to automate such process.

3.1. Data Preparation

We send a post request to the special service in the cloud, which works with the ClickHouse database and can prepare the data in readable form for users.

In our case, it is the most difficult work because we use the functionality that was prepared for table visualization, not for high frequency calls. The bottleneck is Clickhouse: each request to the service should return to us a file in csv-format with 10 methods and 4 metrics with 10 minute detalization.

3.2. Data Analysis

Python3 is used to analyze data [3]. The guideline for the solution is based on an expected value (1) and dispersion (2). The expected value (3) must change not more than 50 percent from version to version, and dispersion (4) – not more than 100 percent. For the

test environment the dispersion is not indicative.

$$M(X) = \sum_{i=1}^{n} x_i/i, \tag{1}$$

$$\sigma^2(X) = \frac{\sum_{i=1}^{n} {x_i}^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}}{n}, \tag{2}$$

$$Mp(X_{v1}, X_{v2}) = \frac{\sum_{i=1}^{n_{v2}} x_i/i - \sum_{j=1}^{n_{v1}} x_j/j}{\sum_{j=1}^{n_{v1}} x_j/j} \cdot 100\%, \tag{3}$$

$$\sigma^2(X_{v1}, X_{v2}) = \left(\frac{\sum_{i=1}^{n_{v2}} {x_i}^2 - \frac{(\sum_{i=1}^{n_{v2}} x_i)^2}{n}}{\sum_{j=1}^{n_{v1}} {x_j}^2 - \frac{(\sum_{j=1}^{n_{v1}} x_j)^2}{n_{v1}} \frac{n_{v1}}{n_{v2}}} - 1 \right) \cdot 100\%, \tag{4}$$

After automatic analysis we have more than 3 000 suspicious methods metrics.

There is a trouble connected with the unstability of the test stand. Every time our developers want to fix some bugs on the stands, this fix sometimes destroys the stand. We added an alerting database with a SLA trigger for stands. After getting the data from the logging system and the data of a SLA trigger, we processed the data by a logical AND.

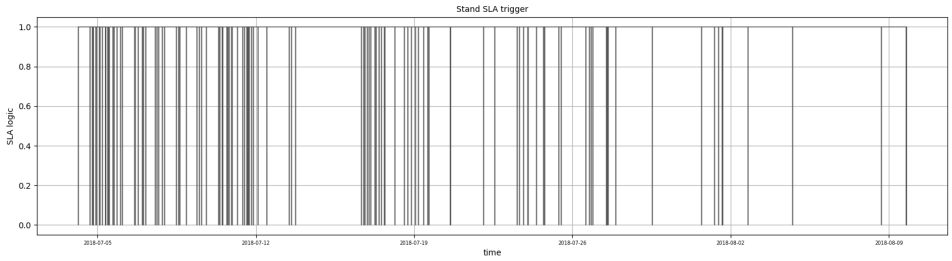

Fig 3. SLA trigger

3.3. Supervision Control

Not all metrics are needed for issue in the bugtracker system. We use supervision analysis over the automatic one. A quality assessment engineer checks all the suspicious results after the automatic script. There are also some troubles. At the production we have seasonal subseries, on the test stands they are not so seasonal. There are periodical peaks on the images. These peaks are the results of the system update. For better supervision control we divide all values into percentiles, and plot images of the metric fluctuation lower than 99th percentile (Fig. 4).

3.4. The Statistics Results

The table 1 shows the result by using the performance analysis.

In version1 we did not use the algorithm on the test stand, and there were a lot of performance bugs in the production. Version2 was the first attempt without the SLA database and the percentile division. Version3 is the current result.

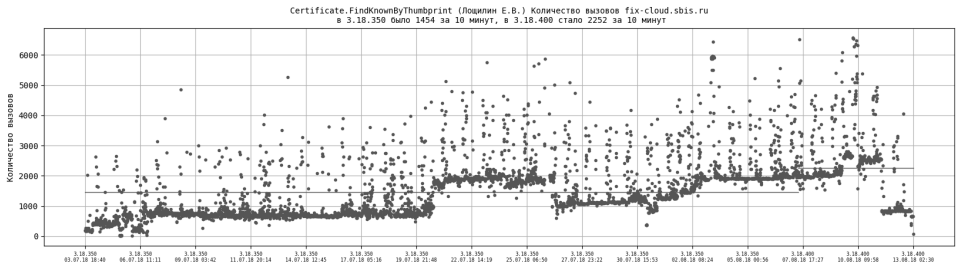

Fig 4. Degradation example

Table 1. Errors per version

version/stand	test *suspicious metrics / issues in bugtracker*	pre-release *suspicious metrics/ issues in bugtracker*	production *suspicious metrics / issues in bugtracker*
version1	-	-	4 218/18
version2	4 571 / 25	3 213 / 15	3 568 / 10
version3	5 764 / 34	3 319 / 10	3 211 / 2

4. Operations Log Analysis

This type of logs is not very convenient to analyze [4, 5]. The easiest way is to make some statistics with small step, less than 10 minutes. That is why we decided to prepare our statistics by the ClickHouse database based method.

4.1. The Queues

Our system is complicated, and for communication we used queues. The queues are infinite: we can add a lot of requests. There are three types of queues:

- **The main queue**.This queue is very important for customers. It is not permissible if a request is executed more than 2 seconds on production and 4 seconds on the test stands.
- **The service queue**. It is a queue that is not very important to customers. It was made for M2M communication.
- **The async queue**. This is an asynchronous queue. It is a part of the service queue, but it uses not all the processes in the service pool of the server's application (the service pool is light grey in Fig. 5 and the main pool is dark). This queue was made for long requests.

4.2. The Main Metrics in Pools

There are three main metrics which we analyze in the queue:

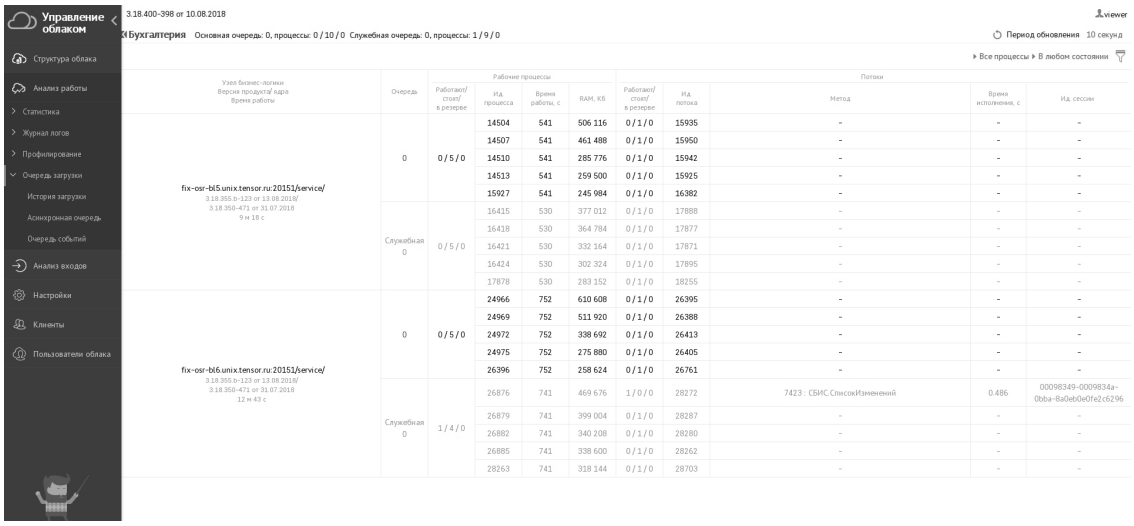

Fig 5. Example of workers pools

- **Waiting in the queue**.This time request is waiting until the worker in the pool is free for execution (Fig. 6).
- **The queue length**. This metrics shows us how many requests are in the queue (Fig. 7).
- **Execution time**. The main metrics of the request.
- **All time in the cloud**. This time is the sum of the previous two. The customer can normally see this time in the browser when the browser is building the page.

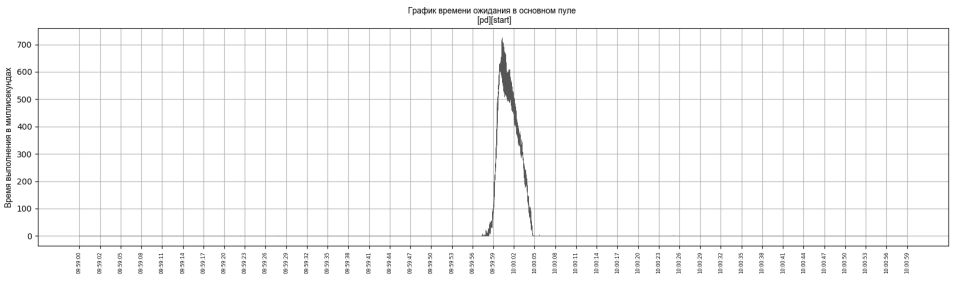

Fig 6. Example of workers pools

All those metrics help us to make a decision what type of request has become the queue reason.

4.3. Queue Analysis

The level for starting queue analysis is 250 ms on the production and 4 000 ms on the test environment.

To begin our analysis, firstly we build the top on the summarize execution time (Fig. 8). After that we use correlation between each method in the top and the waiting “func-

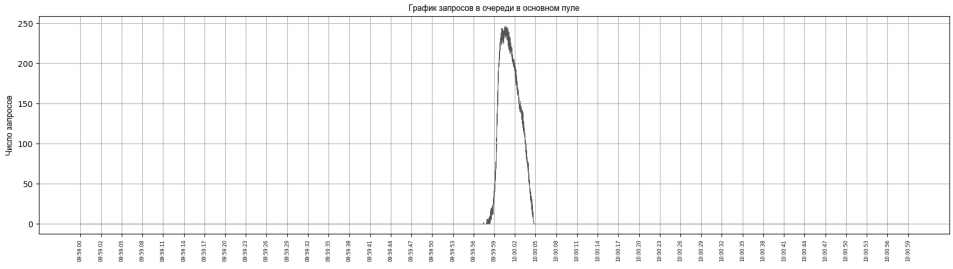

Fig 7. Example of workers pools

tion" (Fig. 6). In 95% the trouble type of requests is in the top-5, but sometimes these types of request are rather fast and they can be out of the top.

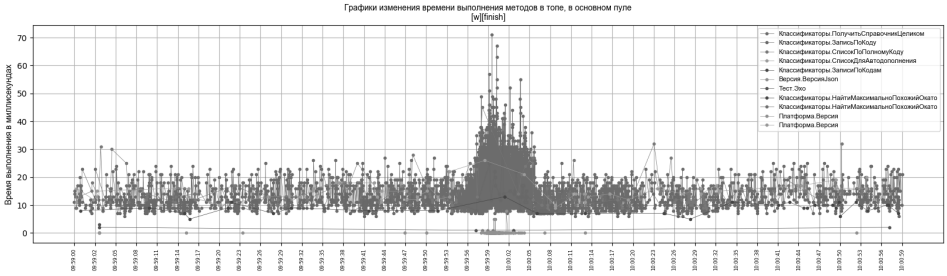

Fig 8. Top-10 of the summarize execution time

The statistics by graphics for some QA engineers in our company is not the easiest way to describe an error to bugtracker, that is why we prefer to add statistics in the table for the correct decision. Some troubles in the system are connected with the actions of some users. We use statistics by ip-address to detect such users and to predict user's scenario of the bug case.

5. Conclusion and Future Ideas

Here are two solutions that we use to detect troubles in our company. There are still a lot of issues need to be solved in future:

- **The statistics analyzer**. The current aggregated solution gives a lot of suspicious metrics of methods back to the supervisor. It is not very convenient to write a bug. We need some experiments with other statistic criteria to make better automatic decisions.
- **The queue analyzer**. The developers are not satisfied, because we could not give them the correct user case to reproduce the bug with the queue on the service.

References

[1] Newman S., *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, New York, 2015.

[2] Wieder P., Butler J. M., Theilmann W., Yahyapour R., *Service Level Agreements for Cloud Computing*, Springer Publishing Company, Incorporated, 2011.

[3] Irdis I., *Python Data Analysis*, Packt Publishing, 2014.

[4] Anand M., "Cloud monitor: Monitoring applications in cloud.", *IEEE Cloud Computing for Emerging Markets, CCEM 2012 - Proceedings*, 2012, 58–61.

[5] Khazaei H., Fokaefs M., Zareian S., Beigi-Mohammadi N., Ramprasad B., Shtern M., Gaikwad P., and Litoiu M., "How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey", *Big Data and Information Analytics (BDIA)*, **(2):1** (2016).

Кирнос В. П., "Пороговый анализ деградации запросов внутри вычислительной сети", *Моделирование и анализ информационных систем*, **26**:2 (2019), 195–202.

DOI: 10.18255/1818-1015-2019-2-195-202

Аннотация. В данной работе рассматриваются существующие подходы к оценке производительности вычислительных сетей. Представлена типовая структура сети прикладного уровня модели OSI на примере приложения *СБИС3* (продукт Компании Тензор). Далее рассмотрены два подхода, позволяющие анализировать деградации внутри сети на основе агрегированных данных и оперативного анализа.

В основе первого решения лежит исследование деградации более 60 000 типов запросов между двумя соседними версиями приложения, которое работает на базе вычислительной сети. Каждый тип запросов описывается четырьмя основными метриками, каждая метрика представляет собой временной ряд. На вход алгоритму анализа поступают агрегированные данные выборками по 10 минут. Далее используются пороговые критерии, основанные на математическом ожидании и дисперсии в рамках двух соседних версий программного обеспечения. Такой подход позволяет существенно сократить время для анализа потенциальных проблем при обновлениях в вычислительной сети.

Информация о каждом запросе на том или ином участке вычислительной сети служит входными данными для второго решения. В качестве порогового критерия выбирается продолжительность ожидания в очереди. Этот тип анализа позволяет диагностировать дефекты, которые становятся причиной δ-образных очередей продолжительностью от нескольких секунд до 10 минут. Подобные дефекты практически не диагностируются в рамках первого подхода.

Ключевые слова: пороговый критерий, матожидание, метрика, очередь

Об авторах:
Кирнос Василий Павлович, orcid.org/0000-0002-6150-3352, программист-тестировщик
ООО Компания "Тензор"
ул. Угличская, 36, г. Ярославль, 150027 Россия, e-mail: vp.kirnos@tensor.ru