A Comparative Analysis of the Performance of Implementing a Java Application Based on the Microservices Architecture, for Various AWS EC2 Instances

Damian Kubiak
Department of Microelectronics and Computer Science
Lodz University of Technology
Lodz, Poland

Wojciech Zabierowski
Department of Microelectronics and Computer Science
Lodz University of Technology
Lodz, Poland
wojciech.zabierowski@p.lodz.pl

2021 IEEE XVIIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) | 978-1-6654-2410-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/MEMSTECH53091.2021.9467912

***Abstract*—The IT industry is growing rapidly. Therefore, changes in the application deployment area take place very often. Developers need to adjust to the new solutions; however, the multitude of available tools may make it difficult to choose the best one. The article presents a summary of 3 AWS EC2 instances and their performance for various test cases running on a java, microservices based application. The purpose of the analysis was to indicate what should be followed when choosing the appropriate environment for the implemented application.**

Keywords—AWS, Cloud, Microservices, Performance measures

I. INTRODUCTION

THE multitude of cloud computing solutions is a huge privilege on the one hand, but also a difficulty, on the other. Proper choice can be the key to optimize performance against costs. There are many solutions available on the market, but the most popular ones are: AWS, Microsoft Azure and Google Cloud. The reason, why the first one was taken for this analysis, is its maturity and completeness, thanks to which, it is indicated as a leader in Gartner's magic quadrant analysis for the 10 years in a row already [1][2]. AWS is a very complex platform that currently has over 200 services (as of February 2021) [3]. Such a number offers a wide range of possibilities and a large field for analysis. However, the EC2 service can be considered as the focal point of the platform and this is where the research is focused as choosing the right instance can be crucial in terms of optimizing performance and costs.

Due to the growing popularity of computing clouds, the number of studies on them is also growing. However, the complexity of the cloud means that each of the studies adds its value. Kokkinos, Varvarigou, Kretsis, Soumplis and Varvarigos proposed an algorithm to optimize costs and utilization in the cloud, based on the set of data entered by the user, to provide solution with better performance for the same costs or lower costs for similar performance [4].

Villamizar et al. showed how important it is to use the right type of deployment to reduce infrastructure costs comparing the three types of deployments with each other. They obtained a difference of approximately 70% for the same application, just by using different approach [5].

Many other articles focus on comparing different cloud providers to compare their performance for a given functionality [6][7].

The mentioned publications are mainly focused on providing a solution or more general comparison, such as a set of different cloud providers. We wanted to indicate how important it is to select an adequate EC2 instance for a specific task, based on microservice Java application, instead of pointing to an appropriate solution, as each application behaves differently.

Applications used to collect and process data from sensor networks are more and more popular. Conversations with other teams show that the most common choice of a programming language or technology is knowing the language or technology in a given team. Based on our experience, we decided that the choice of technology may affect the performance or even the operation of the system based on a sensor network that the application is supposed to support.

The research we have done, and in principle their effects, are applicable to the software that collects data from the sensor network, and therefore they were carried out.

A more precise reference to a specific sensor is not important because it is really important to us what operations on the data from these sensors will be processed by the application.

Therefore, the effect of our research is applicable to all those who want to write an application that collects data from any sensor network, and based on their own knowledge of what kind of operations will prevail, they have to make decisions on the basis of the research results obtained by us. Exactly the same we did and we do with the implementation for various problems related to various issues with sensors. The choice of microservices technology is in line with the current trends and the increasing development of this technology.

II. METHODS

Application implemented for analysis consists of 4 custom services, all containerized with Docker using Dockerfile, and one additional container with Cassandra. It was implemented using Spring Boot (2.4.1), Java (11) and JAX-RS (2.1.1) as an API support. Only one of the microservices communicates with the external network, while the rest are part of internal system. For simplification services will be named accordingly: A, B, C, D – for custom ones and E for Cassandra. All of them communicate with each other and the flow is shown below.

Two functionalities were prepared for two different test scenarios. Both were designed to generate traffic between containers and the computational load. Furthermore, the last of the services, computes the Fibonacci sequence

978-1-6654-2410-3/21/$31.00 ©2021 European Union

implemented in recursive way, to figure out, how machine will behave for such a complex operation. To collect the results, tools built into AWS, which are CloudWatch and X-Ray, were used [8].

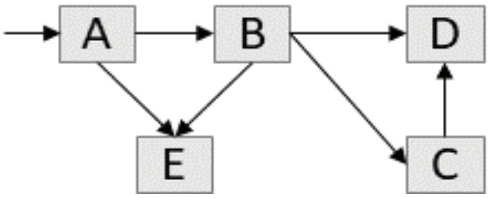

Fig. 1. Application flow

A. Test Rig

Each test was carried out for 3 AWS instances, they were successively: c5.large, c5ad.xlarge and m5zn.large. The machines were selected based on relatively similar costs, which were supposed to be small. Each stack was run under the same task definitions. Every container had an x ray in the form of "Side Car", and each xray-daemon container had allocated 32 CPU Units and 256 MiB soft memory limits. However, when it comes to custom services - CPU Units were not defined and soft memory limits were set at 500 MiB. Cassandra had assigned 268 CPU Units and 544 soft memory limits. Moreover, for Cassandra Task size had fixed values set to 800 MiB for Task memory and 300 units for task CPU.

B. First Scenario - Sequential Queries

First test case performed for every instance. Its main purpose was to test machine behavior for sequential processing. Tests were conducted for 1, 10 and 25 requests. To facilitate data collection, only one request was sent to the application. It contained the number of operations to be performed and then in application the appropriate number of inquiries was passed on in the loop, which ensured that the queries were run one after the other. In this scenario the only database operation was the write performed by services A and B, therefore, in addition to checking the run time, the behavior of writing data to Cassandra was also checked. D service was calculating 42nd word of the Fibonacci sequence to generate server load. The query flow is shown below.

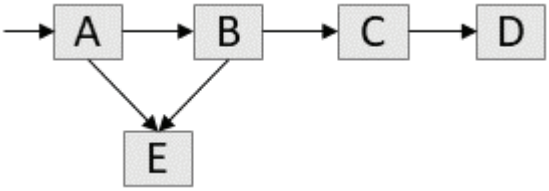

Fig. 2. First scenario flow

1.1.1 Single Query

The main assumption of the test performed for a single query was to check the behavior of the machine that was not warmed up. It therefore checked the usability of the instance for very rare queries. For example, private systems used for specific situations.

1) 10 Queries

The purpose of this scenario was to verify a more realistic system behavior. It was launched immediately after previous test, thanks to which the instance was already warmed up. The result of the test was the average time for each service.

2) 25 Queries

As before, here the tests were also run immediately after the previous ones. Its purpose was mainly to confirm the previous results and check instance stability. The result of the test was the average time for each service.

C. Second scenario - Parallel Queries

Second test case performed for every instance. Unlike the previous scenario, here queries were processed in parallel. Test were conducted for 1 and 10 requests. 25 requests were omitted because even 10 concurrent queries are rare in real cases and 10 were enough to compare the results and find the differences. The tests were run simultaneously thanks to the use of JMeter [9]. Services A and B communicated with the database to extract and delete data, thus making this scenario checking not only the run time of each containers, but also reading and deleting from Cassandra. D service was calculating 45th word of the Fibonacci sequence to generate server load and it was a more distant expression of the sequence to verify operation under even greater load than before. The query flow is shown below.

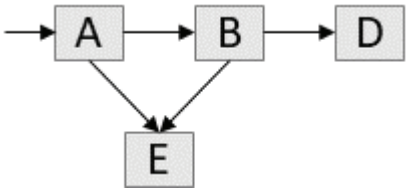

Fig. 3. Second scenario flow

1) Single Query

The main assumption of the test performed for a single query was to collect data that will be the reference for parallel queries and to check the performance of the machine for a computationally expensive task. It was run straight after previous scenarios, so the machine was already warmed up.

2) 10 Queries

Last scenario performed for every EC2 instance. It was run using JMeter [9], so that all queries were sent at the same time. It was also run for already warmed up machine. The purpose of this test was to verify how well the instance can handle multiple queries at once, thus imitating the behavior of a system with a high load.

III. Results

The results shown below confirm how important it is to choose the right machine for a given application to optimize costs and efficiency.

A. First Scenario - Single Query

1) c5.large machine

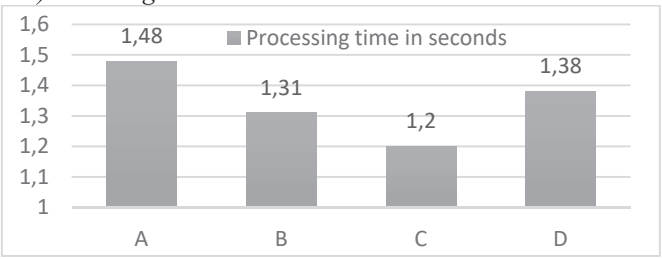

Fig. 4. Response time in seconds for single query in first scenario for c5.large machine

Fig. 4 shows how much impact it has on the results, when the machine is not warmed up. The total operation time was 5.37 seconds. However, potentially the most expensive operation located in service D took 1.38 seconds and its time

is similar to another services. Moreover, as it can be noticed when testing more queries sequentially, potentially simple operations took similar time as the most expensive one. Similar processing time for each service may suggest that for this configuration, the first launch takes a long time, regardless of the operation if performs for a given service.

2) c5ad.xlarge machine

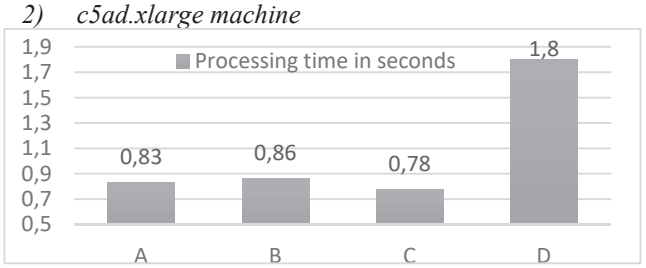

Fig. 5. Response time in seconds for single query in first scenario for c5ad.xlarge machine

Fig. 5 illustrates that the difference between services has increased. Despite the fact that the operation performed in service D took 0.42s more compared to c5.large instance, the total operation time decreased by 1.1s and took 4.27s.

3) m5zn.large machine

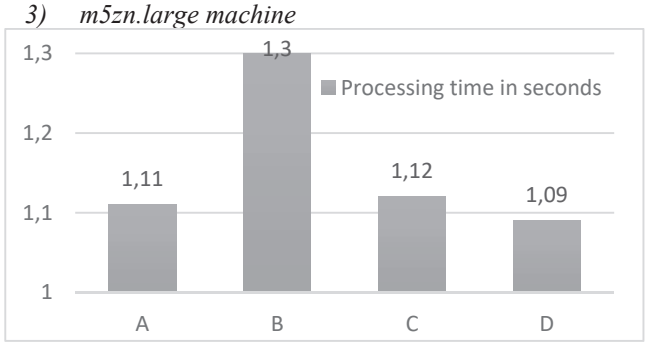

Fig. 6. Response time in seconds for single query in first scenario for m5zn.large machine

For m5zn.large machine total operation time was 4.62s. and was 0.35s worse than c5ad.xlarge one, even though the operation was processed in the fastest way in the service D.

4) Conclusions

For a single operation in first test scenario c5ad.xlarge machine turned out to be the best choice. This machine has the lowest performance processor of all 3 with 2.8GHz clock speed, compared to 3GHz for c5.large and 4.5GHz for m5zn.large. In this case, however, the dominant ones turned out to be having an SSD and a bigger number of virtual processor cores.

B. First scenario - 10 Queries

1) c5.large machine

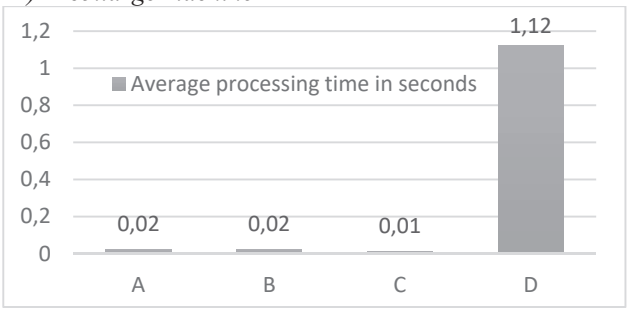

Fig. 7. Average response time in seconds for 10 queries for c5.large machine

Fig. 7 proves that the results were influenced by the fact that the machine was not warmed up. For 10 operations, average response time took 1.17s, which means approximately 459% performance in relation to not warmed up one. In addition, in this case, the vast majority of the time is spent on processing a costly operation in service D, the efficiency of which has not increased as much as in the case of other services.

2) c5ad.xlarge machine

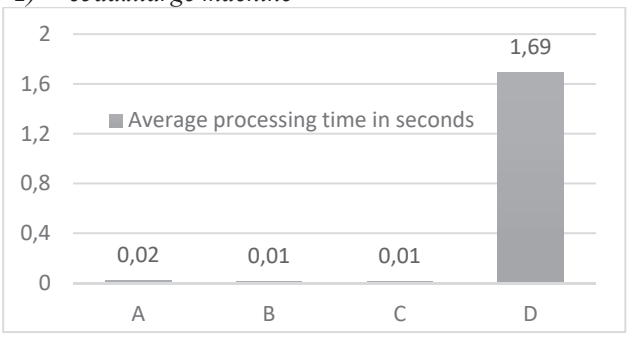

Fig. 8. Average response time in seconds for 10 queries for c5ad.xlarge machine

Graph for 10 queries for c5ad.xlarge machine also points to a significant increase in performance for the warmed-up machine. The average execution time was 1.73s and that means approximately a 247% performance in relation to not warmed up one. It can be seen, however, that due to the weakest processor in the given set and the fact, that service D is the main factor in generating processing time, this instance turns out to be a worse choice for such a case, than c5.large.

3) m5zn.large machine

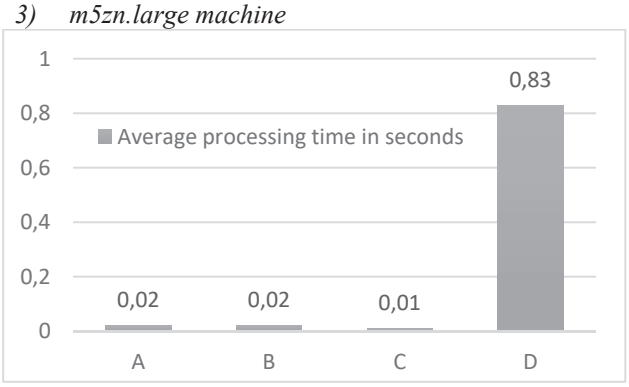

Fig. 9. Average response time in seconds for 10 queries for m5zn.large machine

Fig. 9 indicates that the main factor affecting performance for a given case is the efficient processor. Operations in services A, B and C were performed quickly for each instance and the main impact on the result is the operation requiring an efficient processor located in the D service, which is confirmed by the above results. In addition, this instance noted the highest increase in efficiency in relation to the not warmed up machine, which amounted to approximately 526% in relation to previous result.

4) Conclusions

The test for 10 sequentially performed operations showed the importance of efficient processor for computationally complex tasks. For such a case the profits from having a fast SSD and more virtual cores did not turn out to be so important, thus in this case c5.large and m5zn.large machines performed better. Second of them is approximately 33% more efficient

than the first one and approximately 97% more efficient comparing to c5ad.xlarge. The choice of the instance for this case should be made between c5.large and c5ad.xlarge. Of course, the second one is more efficient, but the first one, depending on the region, is about twice cheaper, so it seems to be the best price-quality ratio. However, if you care only about performance, then m5zn.large is the best choice.

C. First Scenario - 25 queries

The results for 25 sequentially run queries only confirm the results of 10 queries and they are approximately the same. Thus, the efficiency gains compared to the rest of the instances remained approximately the same also. This means that a larger number of queries does not increase the efficiency of machines and that they work stably.

TABLE I. INSTANCE COMPARISON FOR SEQUENTIAL MULTIPLE OPERATIONS

	First test scenario		
	c5.large	***c5ad.xlarge***	***m5zn.large***
10 operations time	11.7s	17.3s	8.78s
approx. time gain over m5zn.large	33%	97%	-
25 operations time	29.1s	42.9s	21.8s
approx. time gain over m5zn.large	33%	97%	-

D. Second Scenario - Single Query

1) c5.large machine

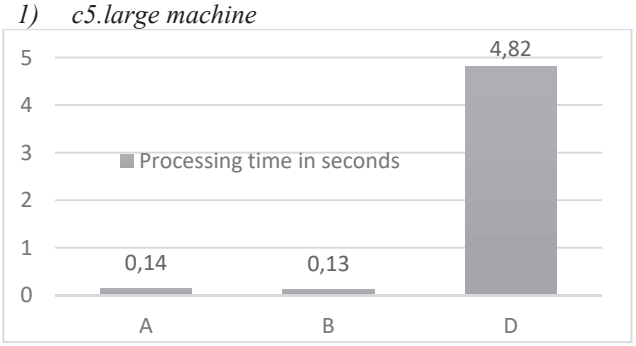

Fig. 10. Average response time in seconds for single query in second scenario for c5.large machine

Scenario was run just after previous tests, so machine was already warmed up. Processing on service D takes even longer than before, due to the calculation of the further expression of the Fibonacci sequence. The whole operation took 5.09s.

2) c5ad.xlarge machine

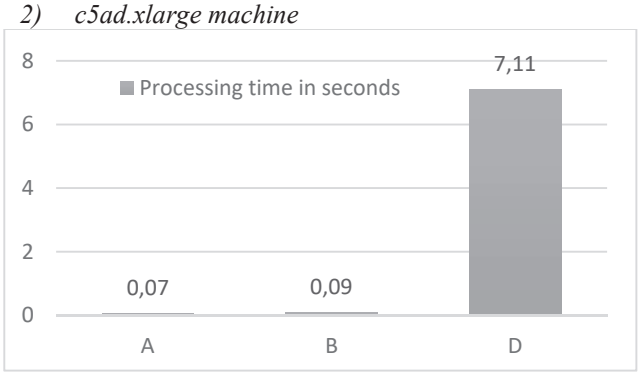

Fig. 11. Average response time in seconds for single query in second scenario for c5ad.xlarge machine

Fig. 11 shows that absolute difference in processing time between services increased. Total processing time in this case was 7,27s. This confirms that the least efficient processor in c5ad.xlarge is not good for computationally complex tasks and that processor should be the main selection factor for such cases.

3) m5zn.large machine

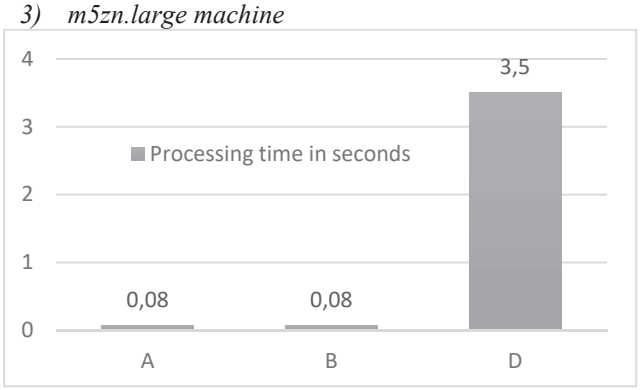

Fig. 12. Average response time in seconds for single query in second scenario for c5ad.xlarge machine

M5zn.large instance again proved to be the most efficient for this kind of task. With a total processing time of 3,66s. Thus, being approximately 39% more efficient than c5.large and approximately 99% more efficient than c5ad.xlarge.

4) Conclusions

Similarly, as with processing multiple queries sequentially in first scenario, so is also for a single query in this test scenario m5zn.large and c5.large instances turned out to be the best choice for such case. As the machines were already warmed up, the obtained results were very similar to those for 10 and 25 sequentially run queries in previous test. C5.large machine is the best option regarding to price-quality ratio, while m5zn.large is the best choice when it comes to performance.

E. Second Scenario - Multiple Queries

The implementation was not adapted to this type of task, but its assumption was to prepare the same conditions for all machines and compare the results, to point, what is the main factor affecting performance of parallel processing.

1) c5.large machine

TABLE II. RESPONSE TIMES OF PARALLEL QUERIES PROCESSED BY C5.LARGE MACHINE

AGE	METHOD	RESPONSE	RESPONSE TIME
54.3 sec	POST	204	40.4 sec
54.3 sec	POST	204	40.5 sec
54.3 sec	POST	204	38.9 sec
54.3 sec	POST	204	41.1 sec
54.3 sec	POST	204	37.8 sec
54.3 sec	POST	204	40.8 sec
54.3 sec	POST	204	40.4 sec
54.3 sec	POST	204	39.5 sec
54.3 sec	POST	204	39.7 sec
54.3 sec	POST	204	40.0 sec

Table II illustrates the difference between single query and multiples ones. In the case of single processing the whole operation took 5.09s, while for parallel processing response time is in the range from 37.8s to 41.1s. Average response

time for this instance is 39,91s That shows that c5.large is not a good choice for such case.

2) c5ad.xlarge machine

Results for the c5ad.xlarge instance, presented in table III, show, how important is the number of virtual cores for simultaneous processing. The results in this case are in the range from 14.1s to 21s, and the average response time took 18,29s, being the same result by approximately 118% better than the c5.large instance.

TABLE III. RESPONSE TIMES OF PARALLEL QUERIES PROCESSED BY C5AD.XLARGE MACHINE

AGE	METHOD	RESPONSE	RESPONSE TIME
2.0 min	POST	204	17.1 sec
2.0 min	POST	204	19.2 sec
2.0 min	POST	204	18.5 sec
2.0 min	POST	204	16.9 sec
2.0 min	POST	204	21.0 sec
2.0 min	POST	204	14.1 sec
2.0 min	POST	204	18.8 sec
2.0 min	POST	204	18.0 sec
2.0 min	POST	204	18.8 sec
2.0 mm	POST	204	20.5 sec

3) m5zn.large machine

TABLE IV. RESPONSE TIMES OF PARALLEL QUERIES PROCESSED BY M5ZN.LARGE MACHINE

AGE	METHOD	RESPONSE	RESPONSE TIME
2.2 min	POST	204	29.0 sec
2.2 min	POST	204	30.5 sec
2.2 min	POST	204	30.7 sec
2.2 min	POST	204	30.4 sec
2.2 min	POST	204	30.1 sec
2.2 min	POST	204	29.3 sec
2.2 min	POST	204	29.8 sec
2.2 min	POST	204	23.7 sec
2.2 min	POST	204	30.3 sec
2.2 min	POST	204	30.5 sec

Based on the previous results, it could be concluded that also in this case the results would be worse than for c5ad.xlarge instance. Number of virtual cores for m5zn.large machine is the same as for c5.large and even the most efficient processor could not cover these shortcomings. The results in this case are in the range from 23.7s to 30.7s, and the average response time took 29.43s, thus making c5ad.xlarge instance by approximately 61% better.

4) Conclusions

For this scenario, c5ad.xlarge instance turned out to be by far the best choice. The most important factor to consider in parallel processing turned out to be the number of virtual cores, which improves the simultaneous operation of many tasks. Despite the fact that the other instances had a more efficient processor, the profit from one was much smaller than the profit from multi-core operations.

IV. DISCUSSION

TABLE V. SUMMARY OF THE PERFORMANCE OF ANALYZED INSTANCES

	Summary of the performance of the analyzed instances		
	c5.large	***c5ad.xlarge***	***m5zn.large***
First scenario - single	5,37s	4,27s	4,62s
Approximate time increment in relation to c5ad.xlarge	26%	-	8%
First scenario - 10	11,7s	17,3s	8,78s
Approximate time increment in relation to m5zn.large	33%	97%	-
First scenario – 25	29,1s	42,9s	21,8s
Approximate time increment in relation to m5zn.large	33%	97%	-
Second scenario - single	5,09s	7,27s	3,66s
Approximate time increment in relation to m5zn.large	39%	99%	-
Second scenario - 10	39,91s	18,29s	29,43s
Approximate time increment in relation to c5ad.xlarge	118%	-	61 %
Average time to complete all queries	32,2s	18,2s	23,8s
Approximate time increment in relation to c5ad.xlarge	77%	-	31%

TABLE VI. SUMMARY OF THE COSTS OF ANALYZED INSTANCES

	Summary of the costs of the analyzed instances for eu-central-1 region (Frankfurt)		
	c5.large	***c5ad.xlarge***	***m5zn.large***
On demand installation	$74,38	$149,57	$148,04
Approximate cost increase in relations to c5.large	-	101%	99%
One year reservation	$48,10	$95,55	$94,16
Approximate cost increase in relations to c5.large	-	99%	96%
3 years reservation	$34,23	$66,35	$66,28
Approximate cost increase in relations to c5.large	-	94%	94%

For sequential processing, instances with an efficient processor seem to be the best. M5zn.large instance, with a 4.5GHz processor was on average 97% more efficient than c5ad.xlarge with 2.8GHz processor, while having similar maintenance costs. On the other hand, it was only 33% more efficient than twice as cheap – c5.large. The right approach when choosing machine for such case seems to be finding ones with processors with the highest computing power, compare their costs and based on such data, make a choice.

This is different for parallel processing, as in this case the number of virtual cores seems to be the most important. The summary presented in table V shows it perfectly. C5ad.xlarge instance, for single operation for second test scenario was almost twice less efficient than m5zn.large. However, when it was required to process many queries at the same time, having

4 virtual cores available made the machine suddenly 61% more efficient than m5zn.large and 118% more efficient than c5.large, as those instances had only 2 virtual cores available. Bearing in mind the high absolute processing time for this test, c5ad.xlarge seems to be the only sensible choice for such case from this list.

The most difficult choice seems to be for applications with mixed types of queries. For the tested case, the best choice turned out to be c5ad.xlarge with an average processing time of 18.2s, which makes it by approximately 31% faster than m5zn.large and by approximately 77% faster than c5.large. However, it must be remembered that the main factor influencing this result was parallel processing. Therefore, for each case, a good solution would be to carry out performance tests and, on their basis, select the appropriate instance.

The obtained results show how important it is to choose the right instance for proper task. It is impossible to indicate the best machine from the presented ones, because each of them works best in different area. As has been shown, the right choice allows to significantly improve the performance of application or largely reduce costs, while maintaining similar performance. The analysis also showed that each case should be considered separately, to optimize the performance of the application.

The obtained results will help to answer doubts in the use of microservice technology for processing data from sensor networks. Choosing the right configuration and services thanks to these results does not have to be a coincidence but a deliberate action to improve the efficiency of the effects of the sensor network itself and of the application, and thus of the complete system.

REFERENCES

[1] “Top cloud providers in 2021: AWS, Microsoft Azure, and Google Cloud, hybrid, Saas players” https://www.zdnet.com/article/cloud-computing-in-the-real-world-the-challenges-and-opportunities-of-multicloud/ (accessed Feb. 20, 2021r).

[2] https://aws.amazon.com/blogs/aws/aws-named-as-a-cloud-leader-for-the-10th-consecutive-year-in-gartners-infrastructure-platform-services-magic-quadrant/ (accessed Feb. 20, 2021r).

[3] https://aws.amazon.com/what-is-aws/ (accessed Feb. 21, 2021r).

[4] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis and E. A. Varvarigos, "Cost and Utilization Optimization of Amazon EC2 Instances," 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, 2013, pp. 518-525.

[5] M. Villamizar et al., "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures," 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, 2016, pp. 179-182.

[6] M. Kang, D. Kang, J. P. Walters and S. P. Crago, "A Comparison of System Performance on a Private OpenStack Cloud and Amazon EC2," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 310-317.

[7] C. Kotas, T. Naughton and N. Imam, "A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing," 2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2018, pp. 1-4.

[8] https://docs.aws.amazon.com/xray/latest/devguide/xray-services-cloudwatch.html (accessed Feb. 21, 2021r).

[9] https://jmeter.apache.org/ (accessed Mar. 07, 2021r).