An effective resource management approach in a FaaS environment

Andreas Christoforou, Andreas S. Andreou
Cyprus University of Technology
{andreas.christoforou,andreas.andreou}@cut.ac.cy

Abstract

Serverless computing introduces a new Cloud service which consist an increasingly popular architecture for building distributed applications. This paper investigates and proposes a new resource management approach in a FaaS platform, based on intelligent techniques. A number of experiments applied through an indicative framework consisting of a client application and a Lambda function. Three Genetic Algorithms were employed to deliver optimal solutions in a multi-objective environment.

1 Introduction

Serverless computing is a relatively new processing paradigm or model that emerged through the continuous and vast development of the Cloud. Serverless computing provides a service in which developers can write and deploy code without provisioning or managing servers or containers. The adoption of this paradigm has great impact on several software engineering aspects such as development process, pricing model and Quality of Sercice (QoS) assurance.

Despite the various benefits and advantages of serverless computing, like zero server management, no up-front provisioning, high availability, auto-scalability and pay only for the resources used, there are also some weaknesses that should also be taken into account: As currently offered from providers, it is not suitable for long term tasks because of the limited time a service can run; additionally, there is increasing complexity of the underlying architecture, which is intensified by the lack of appropriate operational tools.

The main representative of this new service architecture is Function as a Service (FaaS) or event-based programming [1], where a function may triggered through an api call or by an event. Since Amazon introduced Lambda serverless platform in late 2014 [2], many other cloud providers adopted and currently support and offer this architecture. AWS Lambda [3], IBM Cloud Functions [4], Google Cloud Functions [5] and Microsoft Azure Functions [6] are the major serverless providers. This new cloud paradigm is becoming increasingly popular and is gaining great attention from the software industry and research community. A number of new technical challenges and open problems [7] have emerged and a number of questions have been set about the importance and the future of serverless computing.

Resource management support in such a FaaS environment is essential for the software development process itself, which is directed towards satisfying the SLA and providing QoS assurance. The identification of the optimum scenario for resource allocation to serve adequately a specific workload is a tedious, computationally complex and time-consuming process since multiple objectives need to be satisfied.

This research work is motivated by the following two Research Questions (RQ)

- RQ1: Is it possible to implement easy to use and efficient resource management algorithms in a FaaS platform?
- RQ2: How intelligent techniques can deliver efficient resource management to developers in a FaaS environment with the minimum possible cost and time?

The Amazon Lambda platform is considered a complete platform as it offers the most features, while presents the greatest market share; for these reasons it was selected to constitute our experimental environment.

The rest of this paper is organized as follows: Section 2 provides a brief overview of relevant literature focusing more on resource management approaches. Section 3 introduces the proposed approach which addresses

the two research questions. The experimental process is described in section 4, while section 5 discuses the results obtained. Finally, section 6 concludes the paper and outlines future research steps.

2 Literature Review

A short literature review has been carried out that is not limited to resource management approaches, since very few relevant research works were identified.

Two research works refer to efficient resource management: In [8] a solution using a well-known resource allocation strategy for a Lambda platform was presented based on the model predictive controller (MPC). This solution designs a resource allocation policy through the understanding of the run-time attributes of the workload. The authors in [9] performed a dedicated test to identify if cost optimization is feasible when utilizing increased resources for lowering processing times.

Evaluation of main FaaS providers was performed in [10] and relevant results were presented in terms of throughput, network bandwidth, a file I/O and performance computed according to concurrent invocations. In [11], the authors report results from a comprehensive investigation on the performance of microservices hosted by a serveless platform. The investigation dealt with implications of infrastructure elasticity, load balancing, provisioning variation, infrastructure retention, and memory reservations. A micro-benchmark introduced in [12] was used to evaluate the performance and cost model of popular FaaS providers.

An open-source FaaS tool called Snafu was introduced in [13] which is employed for managing, executing and testing functions across provider-specific interfaces. Finally the work of Hong et al. [14] describes six serverless design patterns that can be used to build serverless applications and services.

3 Multi-Objective Optimization Approach

Our general aim is to allocate a sufficient amount of resources in a FaaS environment that should be able to serve a specific workload. By utilizing an exhaustive algorithm we first aim to identify the optimal solutions for both objectives, cost and performance. Exhaustive approaches have great demands on resources and, subsequently, on cost, and thus they cannot be the answer to the first research question. In this research work an exhaustive algorithm will be applied on a low demand environment and a small scale workload with the results obtained being considered as the reference data for the proposed intelligent approaches, the latter being able to reach to solutions faster. Our proposition is the employment of a multi-objective genetic algorithms (MOGAs) [15] as our optimization method that will generate near-optimal solutions and their performance will be compared with the reference optimal solutions extracted by the exhaustive algorithm.

Genetic algorithms are a type of evolutionary algorithm, which are widely used to solve search-based optimization problems by simulating the theory of natural evolution on a population of individuals (candidate solutions). Problems like the one this study is dealing with, require the optimization of multiple criteria at the same time. In such a case multi-objective genetic algorithms can be adopted, where the goal is to find the best solution by otimizing a set of objective functions. In case of conflicting or competing objectives, a multi-objective genetic algorithm normally delivers a set of optimal solutions instead of a single one. This set of optimal solutions is called Pareto optimal set (or Pareto front) and contains those solutions that are not dominated by any other solution yielded during evolution. Each optimal solution constitutes a specific balance between the objectives under optimization, where any improvement in one of them leads to worsening the other (conflicting targets, e.g. cost vs time in our case). Therefore, a decision maker is provided with the set of optimal solutions and is therefore supported to take decisions as to which values of the decision variables are most suited based on the targets and the requirements of his/her application.

In the context of our application, the candidate solutions, or alternatively the decision variables, consist of two of the available configuration options offered by AWS Lambda platform, namely memory allocation and number of maximum concurrency functions, as well as a third variable, the batch size, that represents the number of inputs that each individual function is required to process. Our approach is graphically represented in Figure 1 where the three decision variables in conjunction with the characteristics of the workload define the level of Cost and Duration.

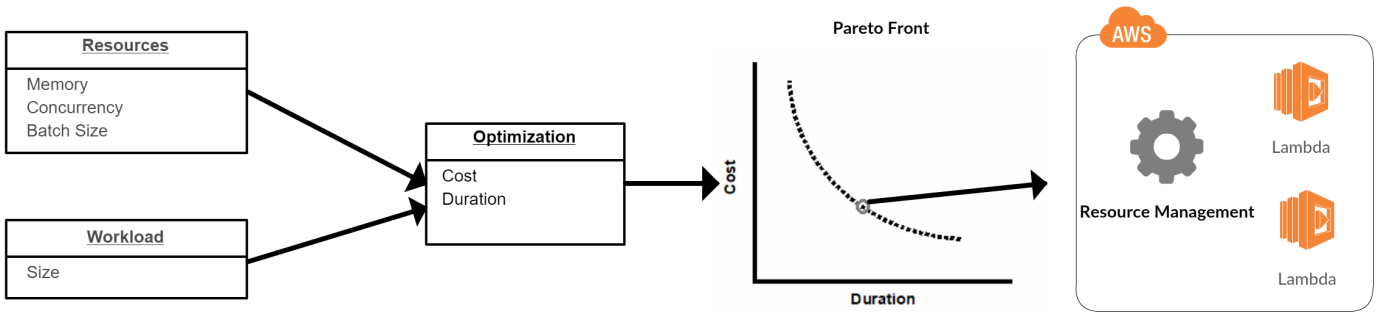

Figure 1: Proposed Multi-objective Optimization Approach

4 Experimental Process

4.1 Experimental Environment

An application that is used to perform the experimental process has been implemented by utilizing services offered by Amazon AWS platform. More specifically, this application is based on the idea introduced in Amazon Big Data Blog [1] where in a map-reduce style it counts the words in files stored in an S3 [2] bucket. This application streams the total number of words each function has calculated, returns it back to the user in real-time and demonstrates how a Lambda function can efficiently process large amounts of data in short time and provide immediate results to users.

In our experimentation study our application was implemented as follows: On the client side, besides the main function that triggers the whole process, a cascade function was also implemented which is responsible to sense the workload size and accordingly distribute the data in a synchronous way over a number of lambda functions. This function is also responsible to collect and aggregate the results taken from the lambda functions responses and when all functions are completed it returns the final result to the user. On the AWS platform a lambda function was created that counts the words of a given batch of files which are stored in a AWS S3 bucket. Data stored in the S3 bucket represents the workload that will be processed with specific features. Both the client and lambda sides were implemented in Python 3.7 and the integration with the AWS services, S3 and Lambda was achieved through the Boto3 [3], the AWS SDK for Python. The synchronous invocations of lambda functions is executed and controlled by utilizing Python's multi-threading module. The integrated experimental environment is depicted in Figure 2.

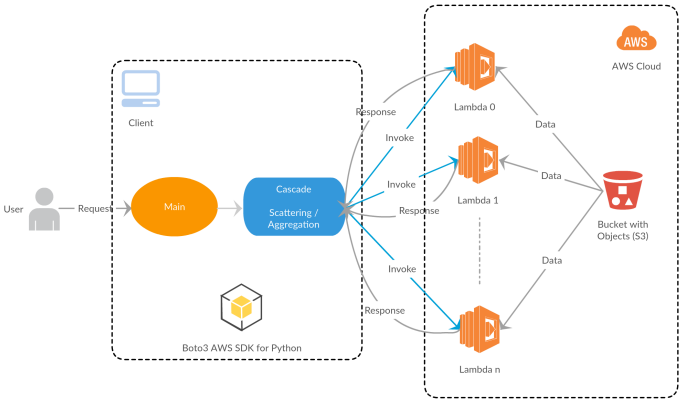

Figure 2: Experimental Environment

4.2 Exhaustive Algorithm

As mentioned before, our multi-objective optimization approach was adjusted and configured based on the AWS Lambda platform and taking into account the available options offered. The two objectives are cost and performance. The cost objective is the minimization of the total cost required for the completion of the process of the input workload and is calculated using the formulas and rules as these are given by Amazon [4]. The calculation of the cost depends on the number of lambda functions executions, the total duration of all executed functions and the allocated memory. The performance objective is also the minimization of the total duration needed for the workload process completion and is calculated as the time from the moment the

[1] https://aws.amazon.com/blogs/big-data/building-scalable-and-responsive-big-data-interfaces-with-aws-lambda/
[2] https://aws.amazon.com/s3/
[3] https://aws.amazon.com/sdk-for-python/
[4] https://aws.amazon.com/lambda/pricing/

user sends the start request until the application delivers back to the user the total count of words. The set of decision variables consists of the memory allocation size, the number of maximum concurrent functions and the batch size. Memory allocation denotes the amount of memory you want to allocate for your lambda function. The values memory can get, ranged from 128MB to 3008 MB with 64MB increment step. The concurrent execution limit can be set from 1 to 1000. Finally, the batch size represents the number of files that each function will process and in our experiments its values are relative to the percentage to the workload size and fall into the following set: [1, 2, 5, 10, 20, 25, 50, 100]. The S3 bucket which is used for the workload, contains 100 text files with each file containing 638 words. The exhaustive algorithm that was used calculated and delivered all possible candidate solutions. To reduce execution time and cost we discarded solutions for concurrency limit over 100 since the size of the workload is 100 and it does not make sense to take these solutions into account. The number of Possible Solutions (PS) is calculated using equation 1 and is calculated to be equal to 36800.

$$|PS| = N \times M \times K \quad (1)$$

where, N is the number of the possible values of the concurrency limit and is equal to 100, M denotes the total number of values for memory and is equal to 46 and K is the number of values the workload batch size can get and is equal to 8.

4.3 Multi-objective Genetic Algorithms

We selected three well-known and widespread MOGAs to assess their ability to solve the problem in hand and compare their results: The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [15], the Non-dominated Sorting Genetic Algorithm III (NSGA-III) [16] and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [17]. The selection of these specific algorithms was made after performing some preliminary experimentation which indicated that these three present a consistently good performance.

The relative configuration of the required parameters, as well as the overall implementation of the algorithms, were performed using Platypus [5], a Python-based multi-objective optimization algorithms library. The aim is to minimize a vector consisting of the two objective functions, Cost and Duration, for values that belong to the PS set (see equation 2).

$$minimize f(x) = (f_{duration}(x), f_{cost}(x)), x \in PS \quad (2)$$

Since all three decision variables are real-valued, we accordingly use best practices for setting the MOGAs configuration. For crossover and mutation operators, the Simulated Binary Crossover (SBX) and Polynomial Mutation (PM) were selected respectively. The same parameters and settings were used for all executions. As one can easily discern from Figure 3, all algorithms yielded very similar solutions which are almost identical to the reference optimal. A more detailed analysis of the results follows in the next section.

5 Results and Discussion

The execution of the exhaustive algorithm on the experimental application delivered a complete list of PS. The extracted values constitute the aggregation of five different executions in order to minimize or even eliminate possible variations between the results under exactly the same configuration conditions. As described above, the results from the exhaustive algorithm were considered as the reference data and were used for the assessment of the three MOGAs employed. Each MOGA was run 100 times for different values of fitness evaluations (FE) ranging from 500 to 4500 with increment step 500. The Pareto optimal front that emerged from the reference optimal solutions in contrast to the Pareto near-optimal solutions yielded by each MOGA for 1000 fitness evaluations, is depicted in Figure 3. By observing the Pareto fronts, one can easily conclude that all three MOGAs approached the optimal solutions to a high degree; in fact, in some cases the dominant MOGA solutions are exactly the same as those of the reference set. At this point we will utilize some metrics aiming to assess further and compare the performance [18] of the three MOGAs we employed on our approach.

5.1 MOGAs performance through quality indicators

The hypervolume (HV) [19] and the inverted generational distance (IGD) [20] quality indicators were selected to assist in comparing the three MOGAs with

[5] https://platypus.readthedocs.io/en/latest/index.html

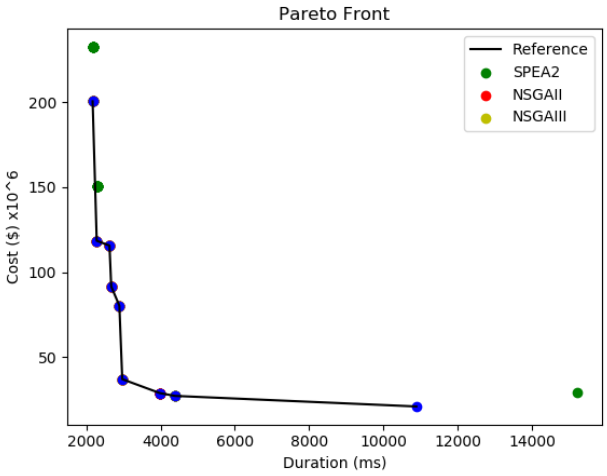

Figure 3: Pareto front for 1000 fitness evaluations (FE)

respect to performance and scalability, given the ability of the aforementioned metrics to assess both convergence and diversity (uniformity and spread) of the algorithms. Specifically, the HV indicator assesses the volume covered by the non-dominated solutions of a Pareto front in the objective space. Therefore, the larger the volume covered by the solutions generated in a run, the higher the HV value, which indicates a better performance. The IGD indicator assesses how far the elements of the true Pareto front (reference data in our case) are from the non-dominated points of an approximation Pareto front. Therefore, the greater the extent of the true Pareto front that is covered by the non dominated points generated by a run in the objective space, the lower the IGD value, which denotes a better performance. Each algorithm was run 100 times for each number of FE and the median values of the HV and IGD were calculated for each algorithm. These values are presented in Tables 1 and 2 respectively.

The differences observed in the indicators values between the compared algorithms are too small, and this most probably is the result of the small complexity of the workload used; however, in cases of application workloads with increasingly greater complexity and/or scale, these differences will become more profound. As regards the HV indicator, SPEA2 presents the best performance, that is, the highest hypervolume value, and this is consistent along all fitness evaluation numbers used. Second best for this indicator is the NSGA-II. It is important to observe that in the case SPEA2 in the 1000 fitness evaluations rub the HV value stabilizes to a constant value from the second measurement on-

Table 1: Hypervolume(HV) values

FE	**HV** ($x10^{-6}$)		
	NSGA-II	**NSGA-III**	**SPEA2**
500	999667.063	999658.576	999755.043
1000	999755.043	999656.438	999960.329
1500	999960.329	999655.593	999960.329
2000	999960.329	999654.920	999960.329
2500	999960.329	999655.384	999960.329
3000	999960.329	999655.777	999960.329
3500	999960.329	999655.230	999960.329
4000	999960.329	999655.687	999960.329
4500	999960.329	999657.071	999960.329

Table 2: Inverted Generational Distance(IGD) values

FE	**IGD** ($x10^{-6}$)		
	NSGAII	**NSGAIII**	**SPEA2**
500	60727.255	61344.367	57375.237
1000	60250.148	57791.945	57792.130
1500	57792.130	57792.189	57792.130
2000	57792.130	57787.907	57792.130
2500	57792.130	57792.125	57792.130
3000	57792.130	57791.964	57792.130
3500	57792.130	57790.733	57792.130
4000	57792.130	57792.097	57792.130
4500	57792.130	57792.179	57792.130

Table 3: Pairwise comparison for HV indicator

	NSGA-II	**NSGA-III**	**SPEA2**
NSGA-II		**0.074**	**0.18**
NSGA-III			0.005
SPEA2			

Table 4: Pairwise comparison for IGD indicator

	NSGA-II	**NSGA-III**	**SPEA2**
NSGA-II		**0.241**	**0.18**
NSGA-III			**0.285**
SPEA2			

wards. The same behaviour is also observed for NSGA-II but from the third measurement onwards. On the contrary, NSGA-III presents more fluctuations in its measurements.

In the case of the IGD indicator, things appear more complicated since none of the algorithms seems to prevail. A notable point is that for the lowest measure of the fitness evaluations numbers, SPEA2 clearly outperforms the others. The values for NSGA-II and SPEA2 starting from the third measurement onwards are fully identical, while NSGA-III shows ups and downs and in three cases seems better than the other two.

The Wilcoxon signed-rank test was applied on both quality indicators to detect whether or not a statistically significant difference exist among the three algorithms. To handle the family-wise error rate accumulated, *p-values* were adjusted using a post-hoc Holm procedure. The *p-values* resulting from the pairwise comparison for HV and IGD indicators are shown in Tables 3 and 4 respectively where pairs of algorithms with a statistically significant difference *($p < 0.05$)* are shown in boldface. According to the pairwise comparisons, no significant difference is observed between NSGA-II and NSGA-III, NSGA-II and SPEA2, and NSGA-III and SPEA2 in either indicator except only for the pair NSGA-III and SPEA2 in the case of HV.

6 Conclusions and Future Work

This research work performed a preliminary investigation to assess whether heuristic approaches for multi-objective optimization, like the specific MOGAs used, are able to solve the problem of finding a set of near-optimal solutions that support developers in a FaaS environment to select an efficient resource allocation scheme with respect to cost and time. In our case this was verified through the experimental process followed; therefore, the question raised now, and will be the source of future research work, is whether the support obtained by the MOGAs is worth investigating further in terms of real-time response, "execution on the fly as workloads come in" and the level to which this support can be generalized to cover also workloads of unknown characteristics.

Acknowledgments

The paper is part of the outcomes of the Twinning project Dossier Cloud. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 692251.

References

[1] Fox GC, Ishakian V, Muthusamy V, Slominski A. Status of serverless computing and function-as-a-service (faas) in industry and research. arXiv preprint arXiv:170808028. 2017;.

[2] Adzic G, Chatley R. Serverless computing: economic and architectural impact. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM; 2017. p. 884–889.

[3] AWS Lambda;. Accessed: 2018-09-30. `https://aws.amazon.com/lambda/`.

[4] IBM Cloud Functions;. Accessed: 2018-09-30. `https://www.ibm.com/cloud/functions`.

[5] Google Cloud Functions;. Accessed: 2018-09-30. `https://cloud.google.com/functions/`.

[6] Microsoft Azure Functions;. Accessed: 2018-09-30. `https://azure.microsoft.com/en-us/services/functions/`.

[7] Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, et al. Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing. Springer; 2017. p. 1–20.

[8] HoseinyFarahabady M, Taheri J, Tari Z, Zomaya AY. A dynamic resource controller for a lambda architecture. In: Parallel Processing (ICPP), 2017 46th International Conference on. IEEE; 2017. p. 332–341.

[9] Hoang A. Analysis of microservices and serverless architecture for mobile application enablement (PhD Dissertation). California State University, Northridge; 2017.

[10] Lee H, Satyam K, Fox G. Evaluation of production serverless computing environments. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE; 2018. p. 442–450.

[11] Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S. Serverless computing: An investigation of factors influencing microservice performance. In: Cloud Engineering (IC2E), 2018 IEEE International Conference on. IEEE; 2018. p. 159–169.

[12] Back T, Andrikopoulos V. Using a Microbenchmark to Compare Function as a Service Solutions. In: European Conference on Service-Oriented and Cloud Computing. Springer; 2018. p. 146–160.

[13] Spillner J. Snafu: Function-as-a-service (faas) runtime design and implementation. arXiv preprint arXiv:170307562. 2017;.

[14] Hong S, Srivastava A, Shambrook W, Dumitras T. Go serverless: securing cloud via serverless design patterns. In: 10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18). USENIX; 2018. .

[15] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation. 2002;6(2):182–197.

[16] Deb K, Jain H. An evolutionary many-objective optimization algorithm using reference-point-based non dominated sorting approach, part I: Solving problems with box constraints. IEEE Trans Evolutionary Computation. 2014;18(4):577–601.

[17] Zitzler E, Laumanns M, Thiele L. SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-report. 2001;103.

[18] Riquelme N, Von Lücken C, Baran B. Performance metrics in multi-objective optimization. In: Computing Conference (CLEI), 2015 Latin American. IEEE; 2015. p. 1–11.

[19] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE transactions on Evolutionary Computation. 1999;3(4):257–271.

[20] Van Veldhuizen DA, Lamont GB. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. Evolutionary computation. 2000;8(2):125–147.