# A modern, event-based architecture for distributed evolutionary algorithms

Anonymous Author 1
Anonymous institute 1
Dublin, Ohio
aa1@ai1.com

Anonymous Author 2
Anonymous institute 2
Dublin, Ohio
2aa@ai2.com

## ABSTRACT

In this paper we introduce KafkEO, a cloud native evolutionary algorithms framework that is prepared to work with population-based metaheuristics by using micro-populations and stateless services as the main building blocks; KafkEO is an attempt to map the traditional evolutionary algorithm to this new cloud-native format.

## CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; • **Computer systems organization → Cloud computing**; • **Computing methodologies** → *Distributed algorithms*;

## KEYWORDS

Cloud computing, microservices, distributed computing, event-based systems, stateless algorithms, functions as a service.

**ACM Reference Format:**
Anonymous Author 1 and Anonymous Author 2. 2018. A modern, event-based architecture for distributed evolutionary algorithms. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3205651.3205719

## 1 INTRODUCTION

In general, Cloud based framworks have tried to achieve functional equivalence with parallel or sequential versions of EAs [2–4, 6]. Besides these implementations using well known cloud services, there are new computation models for evolutionary algorithms that are not functionally equivalent to a canonical EA, but have proved to work well in these new environments. Pool based EAs, [1], have been used for new frameworks such as EvoSpace [5], and proved to be able to accommodate all kinds of ephemeral and heterogeneous resources. In the serverless, event based types of architectures we are going to be targeting in this paper, there has been so far no work that we know of. Similar setups including microservices have been employed by Salza et al. [6]; however, the serverless system adds a layer of abstraction to event-based queuing systems such as the one employed by Salza by reducing it to functions, messages and rules or triggers.


*GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan*

ACM ISBN 978-1-4503-5764-7/18/07.
https://doi.org/10.1145/3205651.3205719

In this paper we want to introduce KafkEO, a serverless framework for evolutionary algorithms and other population-based systems. The main design objective is to leverage the scaling capabilities of a serverless framework, as well as create a system that can be deployed on different platforms by using free software. Our intention has also been to create an algorithm that is functionally equivalent to an asynchronous parallel, island-based, EA, which can use parallelism and at the same time reproduce mechanisms that are akin to migration. The island-based paradigm is relatively standard in distributed EA applications, but in our case, we have been using it since it allows for better parallelism and thus performance, at the same time it makes keeping diversity easier while needing less free parameters to tune. The rest of the paper is organized as follows. Next we present The KafkEO Event based framework and in section 3 conclusions and future lines of work.

## 2 THE KAFKEO EVENT BASED FRAMEWORK

The evolutionary algorithm mapped over this architecture is represented in Figure 1. The main design challenge is to try and map an evolutionary algorithm to a serverless, and then stateless, architecture. That part is done in points 1 through 5 of Figure 1. The beginning of the evolution is triggered from outside the serverless framework (1) by creating a series of Population objects, which we pack (2) to a message in the new-populations *topic*. The arrival of a new population package sets off the MessageArrived trigger (3), that is bound to the actions that effectively perform evolution. In this case we feature GA and PSO algorithms, although only GA has been implemented for this paper. Any number of actions can be triggered in parallel by the same message, and new actions can be triggered while others are still working; this phase is then self-scaling and parallel by design.

Population objects are extracted from the message and, for each, a call to an *evolve* process is executed in parallel. The *evolve* process consists of two sequential *actions* (5), first, the *GA Service* function that runs a GA for a certain number of generations, producing a new evolved object, which is then sent to the second action called *Message Produce* responsible of sending the object to the evolved-population-objects message queue. The new Population object (6) includes the evolved population and also metadata such as a flag indicating whether the solution has been found, the best individual, and information about each generation. With this metadata a posterior analysis of the experiment can be achieved or simply generating the files used by the BBOB Post-processing scripts.

This queue is polled by a service outside the serverless framework, called Population-Controller. This service needs to be stateful, since it needs to wait until several populations are ready to then mix them (in step #9 in Figure 1) to produce a new population, that

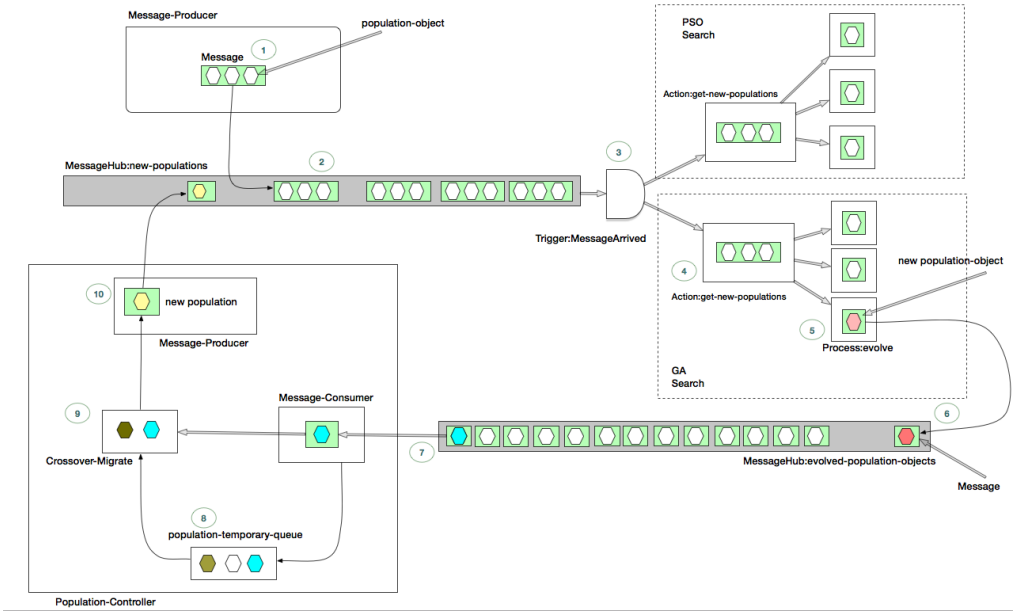


**Figure 1: A flow diagram of KafkEO, showing message routes, MessageHub topics and the functions that are being used.**

is the result of selection and crossover between several populations coming from the evolved-population-objects message queue. Eventually, these mixed populations are returned to the initial queue to return to the *serverless* part of the application. Another task of the Population-Controller is to start and stop the experiment. The service must keep the number of Population objects received, then after a certain number is reached, the controller stops sending new messages to the new-populations *topic*. It is important to note, that because of the asynchronous nature of the system, several messages could still arrived after the current experiment is over. The controller must only accept messages belonging to the current process.

This would be functionally equivalent to a sequential algorithm except for the fact that, between two calls to the get-new-populations function, several population-messages have been received in the message queue. In fact, every call the Crossover-migrate function receives several populations, which have to be merged to generate several new populations. This *merging* step before starting evolution takes the place of the *migration* phase and allows this type of framework to work in parallel, since several instances of the function might be working at the exact same time; the results of these instances are then received back by every one of the instances.

## 3 CONCLUSIONS

This paper is intended to introduce a simple proof of concept of a serverless implementation of an evolutionary algorithm. The main problem with this algorithm, shared by many others, is to turn something that has state (in the form of loop variables or anything else) into a stateless system. In this initial proof of concept we have opted to create a stateful *mixer* outside the serverless (and thus stateless) platform to be able to perform *migration* and mixing among populations. A straightforward first step would be to parallelize this service so that it can respond faster to incoming evolved populations; however, this scaling up should be done by hand and a second step will be to make the architecture totally serverless by using functions that perform this mixing in a stateless way. This might have the secondary effect of simplifying the messaging services to a single topic, and making deployment much easier by avoiding the desktop or server back-end we are using now for that purpose.

## ACKNOWLEDGMENTS

The author would like to acknowledge the support of grants
taking
this

## REFERENCES

[1] A. Bollini and M. Piastra. 1999. Distributed and persistent evolutionary algorithms: a design pattern. In *Genetic Programming, Proceedings EuroGP'99 (Lecture notes in computer science)*. Springer, 173–183.
[2] Andrea De Lucia and Pasquale Salza. 2017. Parallel Genetic Algorithms in the Cloud. (2017), 1–166.
[3] Gilberto Viana de Oliveira and Murilo Coelho Naldi. 2015. Scalable Fast Evolutionary k-Means Clustering. In *Intelligent Systems (BRACIS), 2015 Brazilian Conference on*. IEEE, 74–79.
[4] Włodzimierz Funika and Paweł Koperek. 2016. Towards a Scalable Distributed Fitness Evaluation Service. In *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Ewa Deelman, Jack Dongarra, Konrad Karczewski, Jacek Kitowski, and Kazimierz Wiatr (Eds.). Springer International Publishing, Cham, 493–502.
[5] Mario García-Valdez, Leonardo Trujillo, Juan-J Merelo, Francisco Fernández de Vega, and Gustavo Olague. 2015. The EvoSpace Model for Pool-Based Evolutionary Algorithms. *Journal of Grid Computing* 13, 3 (01 Sep 2015), 329–349. https://doi.org/10.1007/s10723-014-9319-2
[6] Pasquale Salza, Erik Hemberg, Filomena Ferrucci, and Una-May O'Reilly. 2017. cCube: a cloud microservices architecture for evolutionary machine learning classification. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 137–138.