UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Understanding the Research World

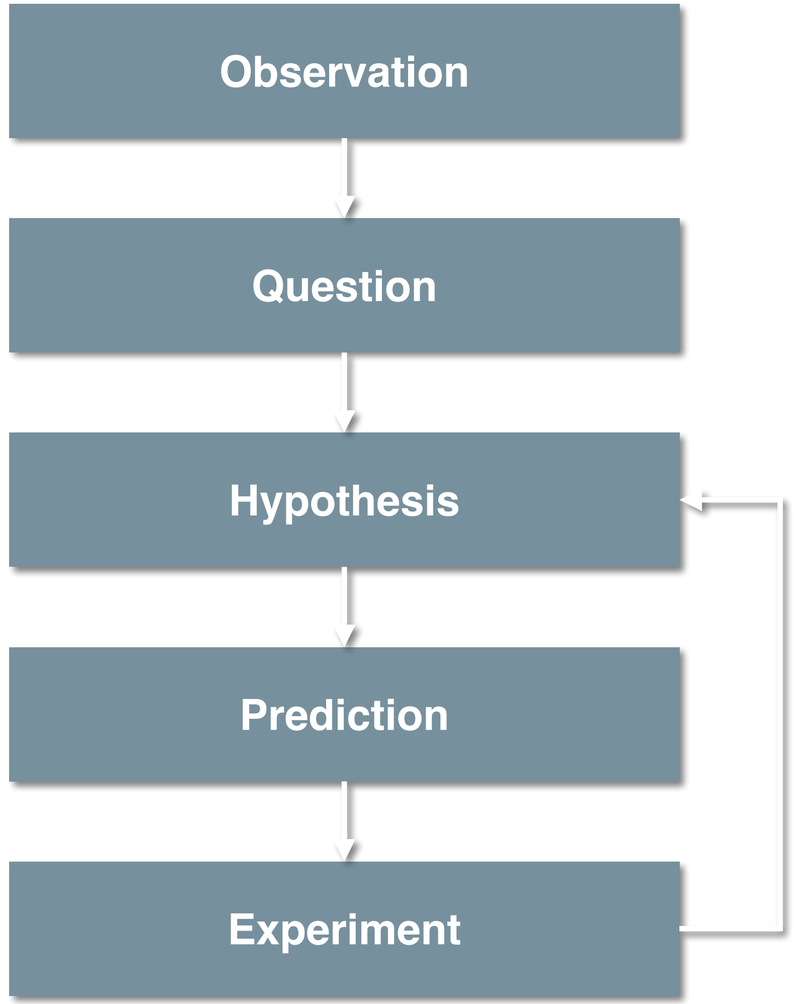
Seminar in Green Software Engineering

Dr. Pooja Rani
rani@ifi.uzh.ch, pooja.rani@uni-mannheim.de
University of Zurich, Switzerland
University of Mannheim, Germany

Scientific research papers

The scientific method

- A logical problem-solving approach
- A paper is scientific if it follows the method
- Otherwise it’s narrative
- An iterative process

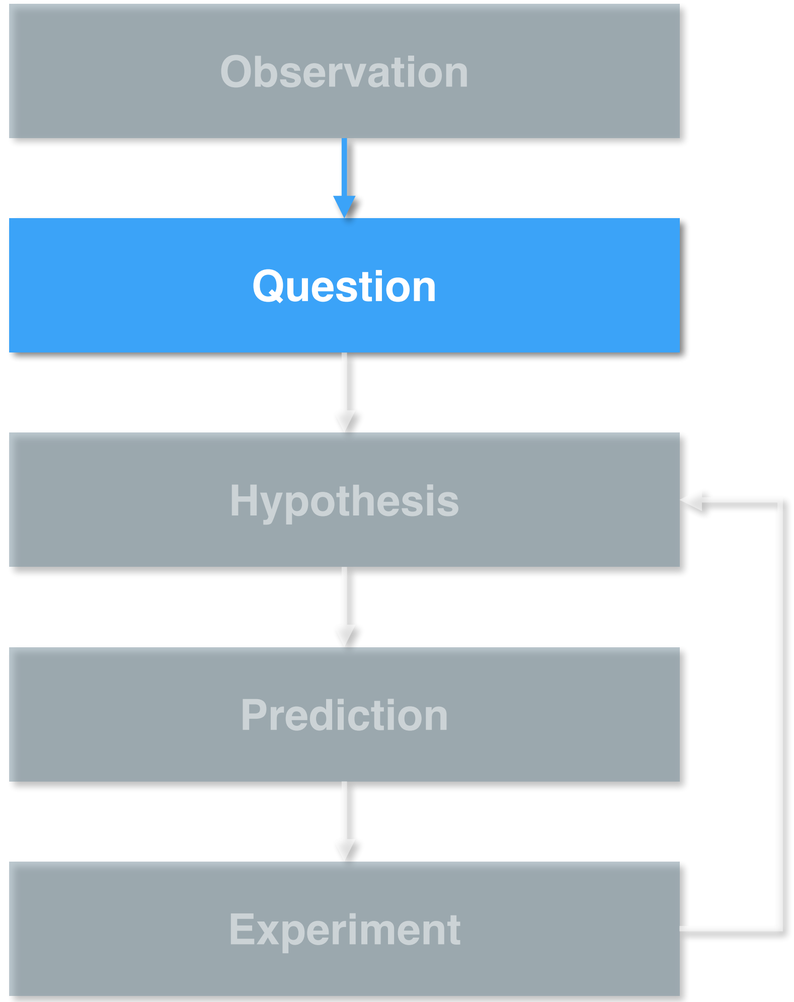


* From the experience, we observe that something happens

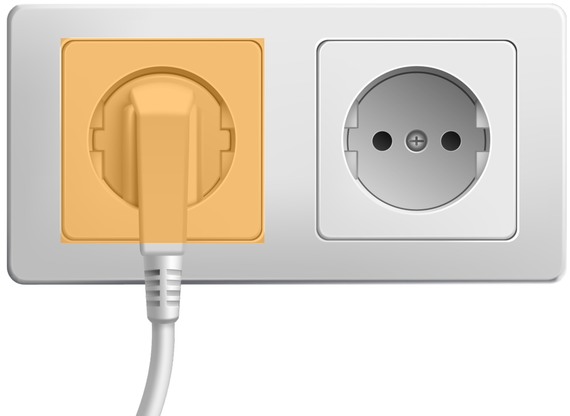
The coffee machine won’t make a coffee

* From the observation, we come up with a question

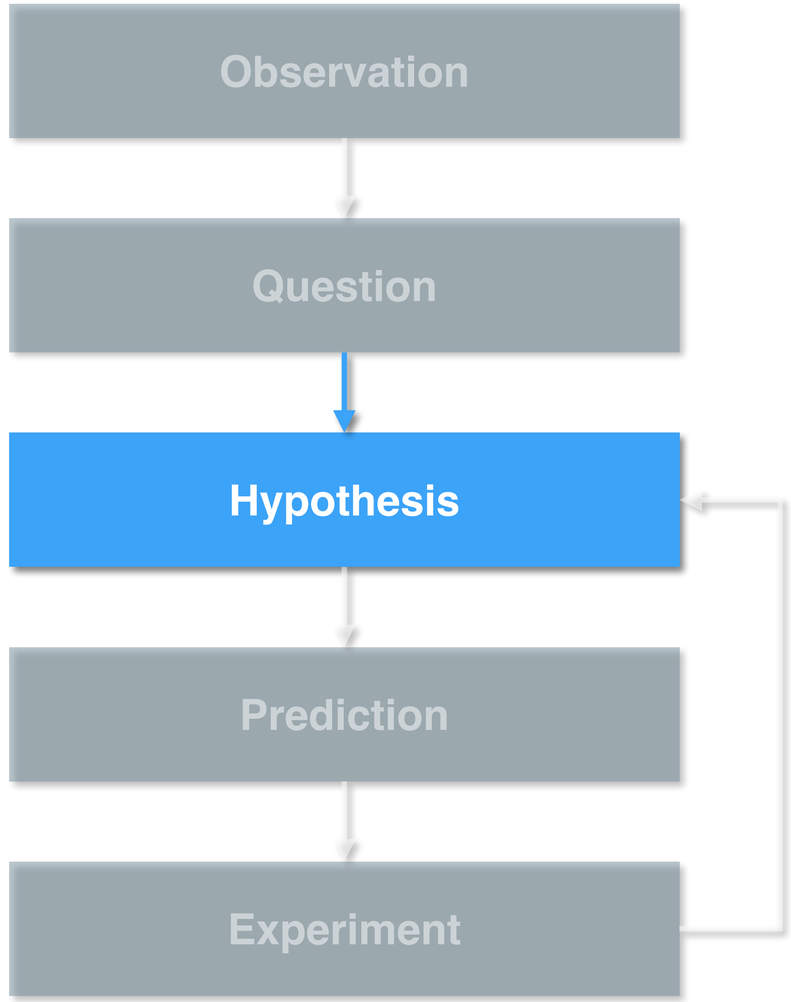
Why won't my coffee machine make a coffee?

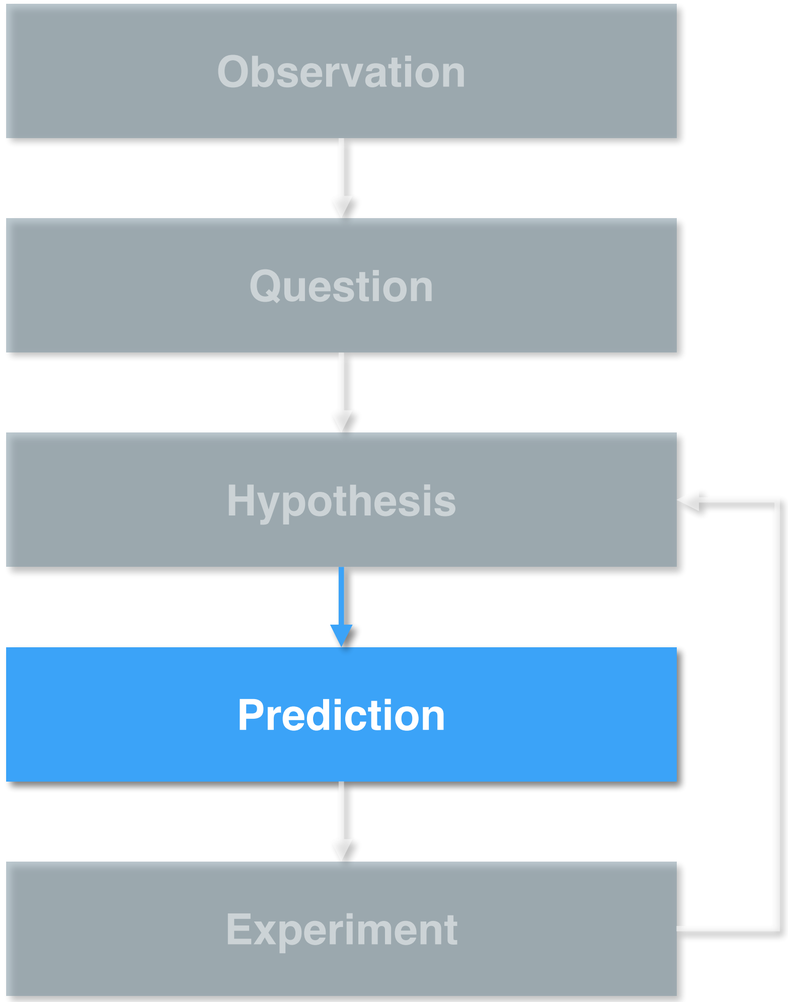


* We propose a tentative answer to the question that would be possible to test

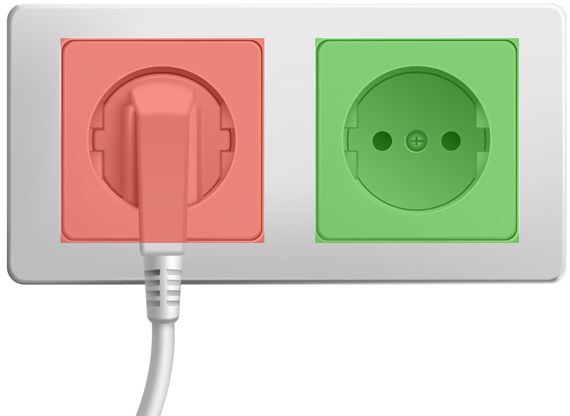


Maybe the outlet is broken





* An outcome we’d expect to see if the hypothesis is correct



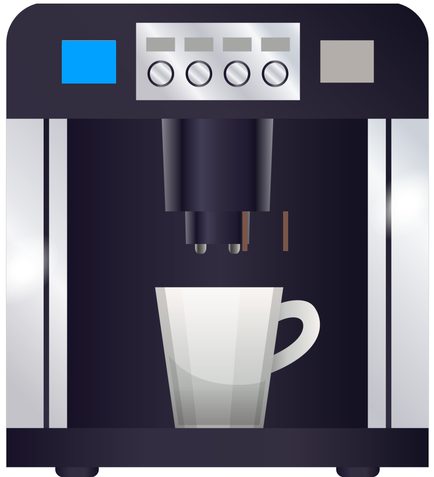
If I plug the coffee machine into a different outlet, then it’ll make a coffee

* We test the hypothesis by making an experiment associated to the prediction

I plug the coffee machine into a different outlet and try again

The hypothesis is supported

- The results of the test might support or contradict the hypothesis
- If we support it, it's likely that the hypothesis is correct
- We cannot prove it, but we could make a theory integrating the findings from other studies
- We can also be more specific with new questions

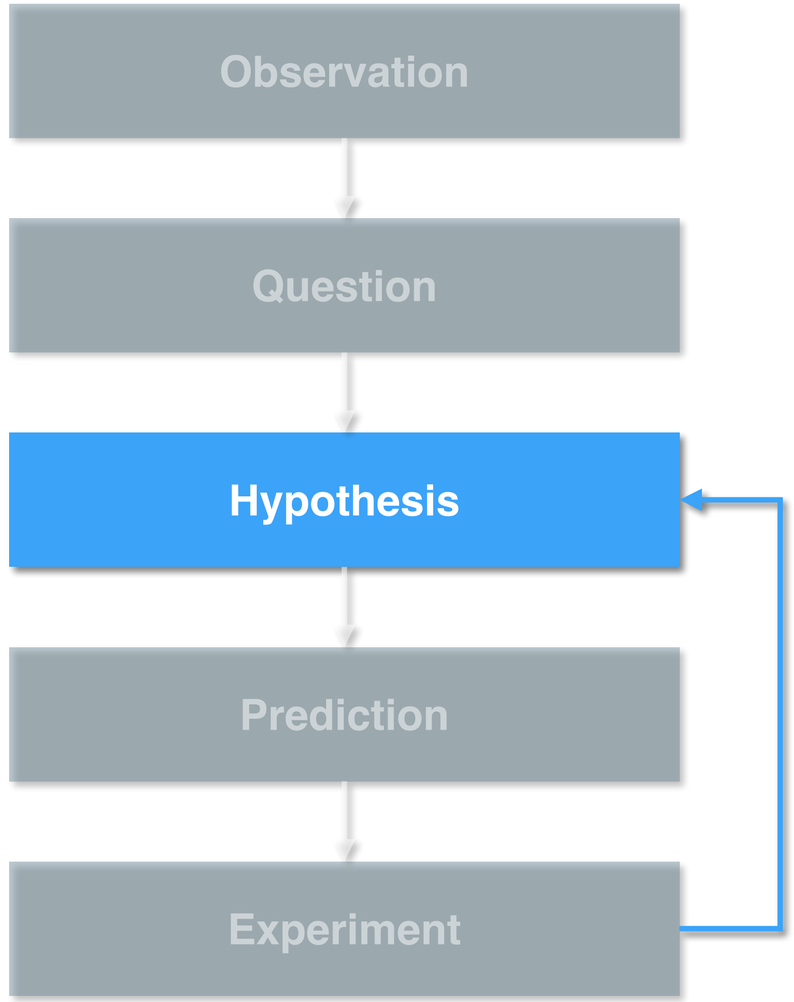


The coffee machine makes the coffee, but what is actually wrong with that outlet?

The hypothesis is <u>not</u> supported

- If we don’t support it, we can come up with a new hypothesis

The coffee machine still won’t make the coffee. Maybe there is a broken wire inside



The structure of a research paper

Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO

ABSTRACT

Title

- The shortest summary of the paper
- Best keywords means best visibility

Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO

Article history:
Received 25 October 2017
Received in revised form 12 September 2018
Accepted 30 September 2018
Available online xxxx

Keywords:
Genetic algorithms
Parallel genetic algorithms
Cloud computing
Software containers
Software engineering

ABSTRACT

Scalability issues might prevent Genetic Algorithms from being applied to real world problems. Exploiting parallelisation in the cloud might be an affordable approach to getting time efficient solutions that benefit of the appealing features of scalability, resource discovery, reliability, fault-tolerance and cost-effectiveness. Nevertheless, parallel computation is very prone to cause considerable overhead for communication. Also, making Genetic Algorithms distributed in an on-demand fashion is not trivial. Aiming at keeping under control the communication cost and, at the same time, supporting developers in the construction and deployment of parallel Genetic Algorithms, we designed and implemented a novel approach to distribute Genetic Algorithms in the form of a cloud-based application. It is based on the master/slave model, exploiting software containers, their cloud orchestration and message queues. We also devised a conceptual workflow covering each cloud Genetic Algorithms distribution phase, from resources allocation to actual deployment and execution, in an engineered fashion. Then, the application performance has been evaluated using a benchmark problem. According to the performance and setup times results, it emerged that the cloud can be considered a compelling way of scaling Genetic Algorithms and an excellent alternative to other technologies strictly related to the physically owned hardware.

1. Introduction

GAs are a powerful technique used in many different fields to search for a near-optimal solution when searching for the optimum is too expensive. Although attractive and elegant in the laboratory, scalability issues prevent GAs from being effectively applied to real world problems [1]. However, parallelisation may be a suitable way to improve the computational time and the effectiveness in the exploration of the search space. Indeed, GAs are 'naturally parallelisable', for instance, their population-based characteristics allow us to evaluate in a parallel way the fitness of each individual, i.e., the 'global parallelisation model', also known as the 'master/slave model' [2].

It is argued that a barrier to the wider application of parallel execution has been the high cost of parallel architectures and infrastructures and their management. GAs have been effectively parallelised on multi-core (i.e., CPUs) and many-core (i.e., GPUs) systems [3,4]. However, these solutions are often expensive and may obtain only a certain degree of parallelisation being strictly related to the number of multiple computational units available on the hardware. On the contrary, technologies based on network communication may hypothetically be scaled without limits, e.g., grid computing [5,6]. *Cloud computing* can represent an affordable solution to address the above issues because it breaks the barrier between employed resources and costs: in a short time, it is possible to allocate a cluster of the desired size without investing in expensive local hardware and its management [7,8].

Previous proposals for distributed GAs in the cloud exploited well-known technologies such as Hadoop MapReduce [9–12], some of them also providing framework/libraries [9,10,13,14] to support developers in building distributed GAs. Even though Hadoop offers some appealing features, the data exchange through a distributed file system may slow down the execution of parallel GAs [11,12]. Moreover, it is required to have dedicate skills for

* Corresponding author.
E-mail addresses: pasquale.salza@usi.ch (P. Salza), fferrucci@unisa.it (F. Ferrucci).

https://doi.org/10.1016/j.future.2018.09.066
0167-739X/

Authors

- The list can be alphabetical or by order of contribution
- For each of the authors an affiliation is mentioned
- We refer to the authors with the last name if they are less or equal than two
- Otherwise we use the expression *“First author’s last name et al.”* (from Latin *“et alia”*, meaning *“and others”*)

Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO

Article history:
Received 25 October 2017
Received in revised form 12 September 2018
Accepted 30 September 2018
Available online xxxx

Keywords:
Genetic algorithms
Parallel genetic algorithms
Cloud computing
Software containers
Software engineering

ABSTRACT

Scalability issues might prevent Genetic Algorithms from being applied to real world problems. Exploiting parallelisation in the cloud might be an affordable approach to getting time efficient solutions that benefit of the appealing features of scalability, resource discovery, reliability, fault-tolerance and cost-effectiveness. Nevertheless, parallel computation is very prone to cause considerable overhead for communication. Also, making Genetic Algorithms distributed in an on-demand fashion is not trivial. Aiming at keeping under control the communication cost and, at the same time, supporting developers in the construction and deployment of parallel Genetic Algorithms, we designed and implemented a novel approach to distribute Genetic Algorithms in the form of a cloud-based application. It is based on the master/slave model, exploiting software containers, their cloud orchestration and message queues. We also devised a conceptual workflow covering each cloud Genetic Algorithms distribution phase, from resources allocation to actual deployment and execution, in an engineered fashion. Then, the application performance has been evaluated using a benchmark problem. According to the performance and setup times results, it emerged that the cloud can be considered a compelling way of scaling Genetic Algorithms and an excellent alternative to other technologies strictly related to the physically owned hardware.

1. Introduction

GAs are a powerful technique used in many different fields to search for a near-optimal solution when searching for the optimum is too expensive. Although attractive and elegant in the laboratory, scalability issues prevent GAs from being effectively applied to real world problems [1]. However, parallelisation may be a suitable way to improve the computational time and the effectiveness in the exploration of the search space. Indeed, GAs are ‘naturally parallelisable’, for instance, their population-based characteristics allow us to evaluate in a parallel way the fitness of each individual, i.e., the ‘global parallelisation model’, also known as the ‘master/slave model’ [2].

It is argued that a barrier to the wider application of parallel execution has been the high cost of parallel architectures and infrastructures and their management. GAs have been effectively parallelised on multi-core (i.e., CPUs) and many-core (i.e., GPUs) systems [3,4]. However, these solutions are often expensive and may obtain only a certain degree of parallelisation being strictly related to the number of multiple computational units available on the hardware. On the contrary, technologies based on network communication may hypothetically be scaled without limits, e.g., grid computing [5,6]. *Cloud computing* can represent an affordable solution to address the above issues because it breaks the barrier between employed resources and costs: in a short time, it is possible to allocate a cluster of the desired size without investing in expensive local hardware and its management [7,8].

Previous proposals for distributed GAs in the cloud exploited well-known technologies such as Hadoop MapReduce [9–12], some of them also providing framework/libraries [9,10,13,14] to support developers in building distributed GAs. Even though Hadoop offers some appealing features, the data exchange through a distributed file system may slow down the execution of parallel GAs [11,12]. Moreover, it is required to have dedicate skills for

* Corresponding author.
E-mail addresses: pasquale.salza@usi.ch (P. Salza), fferrucci@unisa.it (F. Ferrucci).

https://doi.org/10.1016/j.future.2018.09.066
0167-739X/

Abstract

- States the principal objectives and scope of the investigation
- Describes the methods used and principal conclusions
- Usually, less than 250 words
- Very often, it's the only very occasion to get attention from the reader

Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO

Article history:
Received 25 October 2017
Received in revised form 12 September 2018
Accepted 30 September 2018
Available online xxxx

Keywords:
Genetic algorithms
Parallel genetic algorithms
Cloud computing
Software containers
Software engineering

ABSTRACT

Scalability issues might prevent Genetic Algorithms from being applied to real world problems. Exploiting parallelisation in the cloud might be an affordable approach to getting time efficient solutions that benefit of the appealing features of scalability, resource discovery, reliability, fault-tolerance and cost-effectiveness. Nevertheless, parallel computation is very prone to cause considerable overhead for communication. Also, making Genetic Algorithms distributed in an on-demand fashion is not trivial. Aiming at keeping under control the communication cost and, at the same time, supporting developers in the construction and deployment of parallel Genetic Algorithms, we designed and implemented a novel approach to distribute Genetic Algorithms in the form of a cloud-based application. It is based on the master/slave model, exploiting software containers, their cloud orchestration and message queues. We also devised a conceptual workflow covering each cloud Genetic Algorithms distribution phase, from resources allocation to actual deployment and execution, in an engineered fashion. Then, the application performance has been evaluated using a benchmark problem. According to the performance and setup times results, it emerged that the cloud can be considered a compelling way of scaling Genetic Algorithms and an excellent alternative to other technologies strictly related to the physically owned hardware.

1. Introduction

GAs are a powerful technique used in many different fields to search for a near-optimal solution when searching for the optimum is too expensive. Although attractive and elegant in the laboratory, scalability issues prevent GAs from being effectively applied to real world problems [1]. However, parallelisation may be a suitable way to improve the computational time and the effectiveness in the exploration of the search space. Indeed, GAs are 'naturally parallelisable', for instance, their population-based characteristics allow us to evaluate in a parallel way the fitness of each individual, i.e., the 'global parallelisation model', also known as the 'master/slave model' [2].

It is argued that a barrier to the wider application of parallel execution has been the high cost of parallel architectures and infrastructures and their management. GAs have been effectively parallelised on multi-core (i.e., CPUs) and many-core (i.e., GPUs) systems [3,4]. However, these solutions are often expensive and may obtain only a certain degree of parallelisation being strictly related to the number of multiple computational units available on the hardware. On the contrary, technologies based on network communication may hypothetically be scaled without limits, e.g., grid computing [5,6]. *Cloud computing* can represent an affordable solution to address the above issues because it breaks the barrier between employed resources and costs: in a short time, it is possible to allocate a cluster of the desired size without investing in expensive local hardware and its management [7,8].

Previous proposals for distributed GAs in the cloud exploited well-known technologies such as Hadoop MapReduce [9–12], some of them also providing framework/libraries [9,10,13,14] to support developers in building distributed GAs. Even though Hadoop offers some appealing features, the data exchange through a distributed file system may slow down the execution of parallel GAs [11,12]. Moreover, it is required to have dedicate skills for

* Corresponding author.
E-mail addresses: pasquale.salza@usi.ch (P. Salza), fferrucci@unisa.it (F. Ferrucci).

https://doi.org/10.1016/j.future.2018.09.066
0167-739X/

Introduction

- States the context of the research work
- Exposes a problem in that context
- Describes the state of the art, if any
- Motivates the current work and how it intends
- Highlights the main results achieved

Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO

Article history:
Received 25 October 2017
Received in revised form 12 September 2018
Accepted 30 September 2018
Available online xxxx

Keywords:
Genetic algorithms
Parallel genetic algorithms
Cloud computing
Software containers
Software engineering

ABSTRACT

Scalability issues might prevent Genetic Algorithms from being applied to real world problems. Exploiting parallelisation in the cloud might be an affordable approach to getting time efficient solutions that benefit of the appealing features of scalability, resource discovery, reliability, fault-tolerance and cost-effectiveness. Nevertheless, parallel computation is very prone to cause considerable overhead for communication. Also, making Genetic Algorithms distributed in an on-demand fashion is not trivial. Aiming at keeping under control the communication cost and, at the same time, supporting developers in the construction and deployment of parallel Genetic Algorithms, we designed and implemented a novel approach to distribute Genetic Algorithms in the form of a cloud-based application. It is based on the master/slave model, exploiting software containers, their cloud orchestration and message queues. We also devised a conceptual workflow covering each cloud Genetic Algorithms distribution phase, from resources allocation to actual deployment and execution, in an engineered fashion. Then, the application performance has been evaluated using a benchmark problem. According to the performance and setup times results, it emerged that the cloud can be considered a compelling way of scaling Genetic Algorithms and an excellent alternative to other technologies strictly related to the physically owned hardware.

1. Introduction

GAs are a powerful technique used in many different fields to search for a near-optimal solution when searching for the optimum is too expensive. Although attractive and elegant in the laboratory, scalability issues prevent GAs from being effectively applied to real world problems [1]. However, parallelisation may be a suitable way to improve the computational time and the effectiveness in the exploration of the search space. Indeed, GAs are 'naturally parallelisable', for instance, their population-based characteristics allow us to evaluate in a parallel way the fitness of each individual, i.e., the 'global parallelisation model', also known as the 'master/slave model' [2].

It is argued that a barrier to the wider application of parallel execution has been the high cost of parallel architectures and infrastructures and their management. GAs have been effectively parallelised on multi-core (i.e., CPUs) and many-core (i.e., GPUs) systems [3,4]. However, these solutions are often expensive and may obtain only a certain degree of parallelisation being strictly related to the number of multiple computational units available on the hardware. On the contrary, technologies based on network communication may hypothetically be scaled without limits, e.g., grid computing [5,6]. *Cloud computing* can represent an affordable solution to address the above issues because it breaks the barrier between employed resources and costs: in a short time, it is possible to allocate a cluster of the desired size without investing in expensive local hardware and its management [7,8].

Previous proposals for distributed GAs in the cloud exploited well-known technologies such as Hadoop MapReduce [9–12], some of them also providing framework/libraries [9,10,13,14] to support developers in building distributed GAs. Even though Hadoop offers some appealing features, the data exchange through a distributed file system may slow down the execution of parallel GAs [11,12]. Moreover, it is required to have dedicate skills for

* Corresponding author.
E-mail addresses: pasquale.salza@usi.ch (P. Salza), fferrucci@unisa.it (F. Ferrucci).

https://doi.org/10.1016/j.future.2018.09.066
0167-739X/

Related work

- Can appear at the beginning or end of a paper
- Extensive search of the scientific literature
- It aims at showing what is known about the subject at the date of the paper

to physical locations. Also, it allows hiding under the same name multiple instances of the same service. In this way, a load balancer can decide which resource is less busy to accept a new request. Of course, even if very powerful, this capability needs to be specifically designed for the developed CBA. Regarding the scalability, this allows adding an undetermined number of resources to the computation, as in the case of the proposed CBA for GAs.

Therefore, the parallel GA can be finally executed. How the several containers interact is addressed in the next section, in which the application we devised is proposed.

3. Related work

A wide range of work is present in the literature about models and technologies for GAs parallelisation [2]. However, our work aims to parallelise GAs on a commercial cloud environment. Thus, we only report the most relevant work involving models, technologies, problems and conceptual deployment workflows in the GAs or *Evolutionary Algorithms* (EAs) fields.

Zheng et al. compared the multi-core (i.e., CPUs) and the many-core (i.e., GPUs) systems for GAs parallelisation [3]. Firstly, they found that the system based on GPUs is faster than the CPUs one. However, they observed that an architecture with a fixed number of parallel participants, such as GPU cores, might perform worse in terms of quality of solutions than another with more parallel nodes stating that distributed architectures, e.g., the cloud, are worth for GAs parallelisation.

Many authors used the *MapReduce* paradigm to implement parallel GAs [30,31] and some of them with *Hadoop MapReduce* in particular [11,12]. On the one hand, they claimed that GAs can efficiently scale on multiple Hadoop nodes. On the other hand, they highlighted the worrying presence of overhead, due to the communication with the data store, i.e., *Hadoop Distributed File System* (HDFS), suggesting their use only with large populations and intensive computation work for fitness evaluation. In general, Hadoop MapReduce represents one of the most mature and employed technologies to develop parallel algorithms since it provides a ready to use distributed infrastructure that is scalable, reliable and fault-tolerant [32]. Nevertheless, it requires high performance from the underlying hardware. Moreover, the scalability of the infrastructure is possible, but it requires a considerable amount of time before a new node becomes available and it is not suitable for all since specific skills for setup and maintenance activities are needed. Instead, other cloud technologies are affordable, and the scalability and fault-tolerance features can be obtained from the design of the distributed applications themselves.

Another aspect that is strictly connected to our work is the preservation of the metaheuristic nature of GAs, thus allowing the system to be adapted to a wide variety of problems. Even though being related to the EAs in general, the first attempt of generalisation was given from Fazenda et al. [13], who considered the parallelisation of EAs on the Hadoop MapReduce platform in a general purpose form of a library. The work has been further enhanced by Veeramachaneni et al. to produce *FlexGP* [33], which is probably the first large-scale Genetic Programming system that runs in the cloud, implemented over *Amazon EC2* with a socket-based client/server architecture. To the same aim, Ferrucci et al. [9,10,14,34] devised and implemented the *elephant56* framework for parallel GAs development, deployment and execution on the Hadoop MapReduce platform, based on the three models of GAs parallelisation (i.e., the global, grid and island model). They described the design of the framework and how a developer could interact in defining his/her genetic operators or using some provided samples. We addressed this aspect by using software containers that allow defining any environment for GAs execution.

With cloud computing is possible to address almost any problem, by mixing a large variety of technologies. Moreover, some software engineering methodologies are particularly effective in the cloud field, possibly easing the production of parallel GAs (see Section 2). As a first attempt at employing cloud technologies, Merelo Guervós et al. devised *SofEA* [35], a model for Pool-based EAs in the cloud, an evolutionary algorithm mapped to a central *CouchDB* object store. SofEA provides an asynchronous and distributed system for individuals evaluations and genetic operators application. Later, they defined and implemented the *EvoSpace Model* [36], consisting of two components: a repository storing the evolving population and some remote workers, which execute the actual evolutionary process. It is the first work to involve technologies on the *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS) level: *Heroku* as PaaS for the population store and *PiCloud* as SaaS for the computing operations. Not only does the work show how EAs can scale on the cloud, but also how the cloud can make EAs effective in a real world environment, speeding up the running time and lowering the costs.

This paper investigates how GAs can effectively take advantage of the cloud to speed up their execution.

4. Background

In this section, we give some background about the involved technologies and communication protocols. The container-based virtualisation and its related most famous utility, i.e., Docker, are presented, respectively, in Sections 4.1 and 4.2. Section 4.3 describes CoreOS, the technology of containers distributed orchestration we employed whereas Section 4.4 illustrates the *Advanced Message Queueing Protocol* (AMQP) we used for communication together with its most famous implementation, i.e., RabbitMQ.

4.1. Container-based virtualisation

The basic idea behind the classic *hypervisor-based virtualisation* is to emulate the underlying physical hardware, creating a new virtual one and installing a fully working *Operating System* (OS) on it. It is the typical model adopted by cloud providers, because of its ability to make hardware shareable and easily maintainable. Even though there are many existent techniques to optimise resources sharing (e.g., the *bare metal virtualisation*), the hypervisor-based virtualisation can be considered as limited in terms of performance. It is true especially when the aim is to execute cloud applications, where several service instances may require to be created and destroyed in seconds to guarantee the reliability and scalability of the entire system.

While with the hypervisor-based virtualisation everything is performed on the hardware level, the *container-based virtualisation* operates at the OS level. It provides a lightweight virtual environment, i.e., the *software container*, that groups and isolates a set of processes and hardware resources from the host and any other container. The main difference with the hypervisor-based virtualisation consists in the fact that all containers share the same kernel of the host system, instead of virtualising it, resulting in a high-performance resource utilisation. For this reason, containers are also much smaller and lightweight compared to an entire virtualised OS [17]. With the isolation, a process inside the container cannot directly see a process or resource outside the container itself, and the network is the only vehicle for communication.

Containers and its features are not such a new technology. Indeed, it was 1979 when it was made possible, for the first time, to create a new root filesystem inside an existing one, using a feature named ‘chroot’. This isolation feature was then evolved into the Linux ‘namespaces’ technology that not only does it offer the isolation of the filesystem, but also of other system resources such as network interfaces. In this way, processes can run in an environment where the resources appear to be dedicated to them.

Background

- Optional, it depends on the target audience of the paper
- Includes everything is needed to let the reader understand the rest of the contents

to physical locations. Also, it allows hiding under the same name multiple instances of the same service. In this way, a load balancer can decide which resource is less busy to accept a new request. Of course, even if very powerful, this capability needs to be specifically designed for the developed CBA. Regarding the scalability, this allows adding an undetermined number of resources to the computation, as in the case of the proposed CBA for GAs.

Therefore, the parallel GA can be finally executed. How the several containers interact is addressed in the next section, in which the application we devised is proposed.

3. Related work

A wide range of work is present in the literature about models and technologies for GAs parallelisation [2]. However, our work aims to parallelise GAs on a commercial cloud environment. Thus, we only report the most relevant work involving models, technologies, problems and conceptual deployment workflows in the GAs or *Evolutionary Algorithms* (EAs) fields.

Zheng et al. compared the multi-core (i.e., CPUs) and the many-core (i.e., GPUs) systems for GAs parallelisation [3]. Firstly, they found that the system based on GPUs is faster than the CPUs one. However, they observed that an architecture with a fixed number of parallel participants, such as GPU cores, might perform worse in terms of quality of solutions than another with more parallel nodes stating that distributed architectures, e.g., the cloud, are worth for GAs parallelisation.

Many authors used the *MapReduce* paradigm to implement parallel GAs [30,31] and some of them with *Hadoop MapReduce* in particular

efficiently sc
highlighted
munication
(HDFS), sugg
sive comput
MapReduce
technologies
ready to us
and fault-tol
from the ur
infrastructu
time before
for all since
needed. Inst
scalability a
design of th
Another
preservatior
the system
though bein
generalisatic
the paralleli
a general pu
enhanced by
is probably t
runs in the
based client
10,14,34] d
for parallel
Hadoop Ma
parallelisatic
scribed the
teract in defi
samples. W
that allow d
With clou
lem, by mix

software engineering methodologies are particularly effective in the cloud field, possibly easing the production of parallel GAs (see Section 2). As a first attempt at employing cloud technologies, Merelo Guervós et al. devised *SofEA* [35], a model for Pool-based EAs in the cloud, an evolutionary algorithm mapped to a central *CouchDB* object store. SofEA provides an asynchronous and distributed system for individuals evaluations and genetic operators application. Later, they defined and implemented the *EvoSpace Model* [36], consisting of two components: a repository storing the evolving population and some remote workers, which execute the actual evolutionary process. It is the first work to involve technologies on the *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS) level: *Heroku* as PaaS for the population store and *PiCloud* as SaaS for the computing operations. Not only does the work show how EAs can scale on the cloud, but also how the cloud can make EAs effective in a real world environment, speeding up the running time and lowering the costs.

This paper investigates how GAs can effectively take advantage of the cloud to speed up their execution.

4. Background

In this section, we give some background about the involved technologies and communication protocols. The container-based virtualisation and its related most famous utility, i.e., Docker, are presented, respectively, in Sections 4.1 and 4.2. Section 4.3 describes CoreOS, the technology of containers distributed orchestration we employed whereas Section 4.4 illustrates the *Advanced Message Queuing Protocol* (AMQP) we used for communication

The term 'process container' was first used around late 2006, then renamed to 'control groups' (abbreviated as 'cgroups') in 2007, as a Linux kernel feature available since v2.6.24. While namespaces isolates processes, cgroups lets the user limit the hardware resources for them. The combination of namespaces and cgroups is the basis of the modern *Linux Containers* (LXC) on which Docker was built on[1] and that Docker simplifies, especially when the aim is to containerise applications.

4.2. Docker

Docker is an open source container orchestration engine that separates applications from the underlying Linux OS. With Docker it is possible to manage software containers, which are intended to contain every component of an application. From the application perspective, there is no difference between an execution on a dedicated machine and inside the container: the application is run in a short time in a full isolated Linux environment and can find others only by using the network. This reduces drastically the activities of installation and maintenance of applications: configuration management methodologies can define the environments and the application can be tested during the process from development to actual production execution, in a CI fashion (see Section 2.1). Docker creates containers from the 'images', i.e., basically read-only templates. Docker also provides an on line registry called 'Docker Hub' where it is possible to push/pull images to/from it. Docker images and registry allow to instantiate containers without repeating installation and build operations. The images can be created through two different operations: 1. by executing operations directly on running containers and saving their state; 2. by executing 'Dockerfiles', a set of instructions which can be maintained in the same way as the source code. Docker is not the only alternative in the field of containers management (e.g., *runC*, and *rkt*), but it is currently the most mature product.

4.3. CoreOS

If Docker orchestrates containers in a single hosting machine, CoreOS can do it on a distributed cluster [16]. *CoreOS* is an open

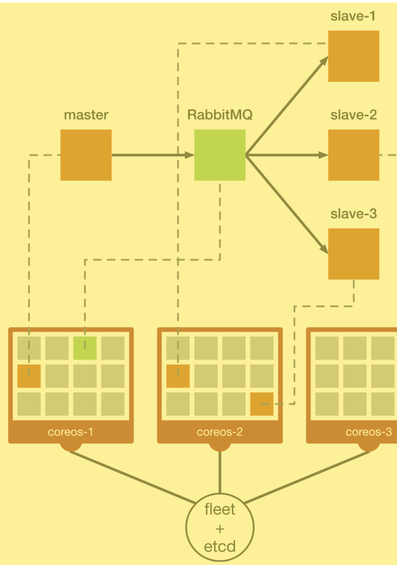


Fig. 2. CoreOS containers distribution.

from different cloud providers, surpassing the limitations
number of machines the cloud providers usually impose upo
users.

We preferred to use CoreOS over other alternatives (e.g., *

Methodology: Approach

- The research methods to verify the hypothesis
- Includes the proposed approach to solve the problem
- Figures and algorithms can be used to improve the understandability

High Availability (HA) capabilities, many client and developer tools available for the majority of programming languages, besides being easily deployable as Docker containers. Furthermore, differently from other communication technologies, RabbitMQ can easily sustain a distributed infrastructure without requiring any other discovery technology.

5. The proposed cloud-based application

In this section, we present the design and implementation of the approach we devised to parallelise GAs in the cloud as a CBA, which we then used as a benchmark for the experimentation described in Section 6. We refer to the development, deployment and execution workflow presented in Section 2, covering each of the expected phases.

We implemented the parallel GA as a distributed algorithm based on containers, following the global parallelisation model (also known as the master/slave model) where a master node executes the GA generations on the whole population except for the fitness evaluation, which is demanded to distributed slave nodes. We named this implementation as *AMQPGA*.

Since we needed other utility components, we structured the application in microservices [37] using separate containers, decoupling functionalities and exploiting existing and more reliable services for communication and report activities. We involved a total of 4 types of services: 1. AMQPGA master, to manage the computation of the whole parallel GA; 2. AMQPGA slave, to compute the fitness evaluation function in parallel for the distributed individuals; 3. RabbitMQ, as the communication protocol and technol
benchmarki
In the fol
of the parall
of the whole
cation proto
the algorithm

5.1. Architec

We descr
allel GA, incl
indicated in
Fig. 3 sh
base layer is
virtual insta
manager. W
and flexible
cloud provic
and private
since the on
the CoreOS i
usage that si
both the ma
and etcd as t
As menti
machines in
the downloa
assigned by
an entire en
genetic oper
any external
The appli
interfaces of
bitMQ, and
named AMQ
slave contai

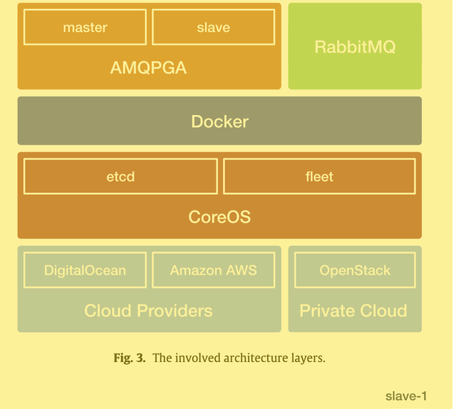
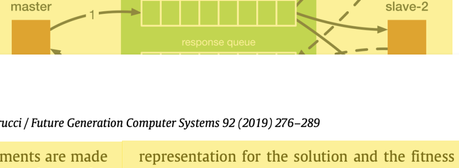


Fig. 3. The involved architecture layers.



slave nodes, in a round-robin fashion (i.e., assignments are made in equal portions and circular order); 3. the AMQPGA slave nodes process the individuals by computing the fitness function values and publish them on the response queue; 4. the only consumer of the response queue, i.e., the AMQPGA master node, takes back all the individuals and continues the computation until the next generation. Using the message broker as the central point for the computation, we were able to add any number of further slave nodes to the GA, even at runtime, making the application scalable. In terms of discovery, we used the specific name of 'rabbitmq' for the service container executing RabbitMQ. In this way, the master and slave nodes only have to reach that node to be part of the distributed computation.

5.3. Parallel genetic algorithm master and slaves

For the implementation, we chose *Go*, an open-source programming language by Google. In our case, Go was used to simplify the GA processes and build small containerised environments. It is worth noting that it would be possible to switch the Go clients with clients developed with any programming language. The only requirements consist in being able of communicating with RabbitMQ, serialising individuals with the same codification algorithm and respecting the devised communication protocol.

Algorithm 1: The AMQPGA master

```
1  population ← Initialisation(popsize);
2  repeat
3      foreach individual ∈ population do
4          ProduceMessage(individual);
5      population ← ConsumeMessages();
6      selectedCouples ← Selection(population);
7      foreach parent1, parent2 ∈ selectedCouples do
8          child1, child2 ← Crossover(parent1, parent2);
9          offspring ← offspring ∪ {child1} ∪ {child2};
10     foreach individual ∈ offspring do
11         Mutation(individual);
12     population ← SurvivalSelection(population
```

representation for the solution and the fitness evaluation
Of course, the fitness value will be filled when computed
slaves only. Lines 6–12 applies the other genetic operato
`Selection`, `Crossover`, `Mutation`, and `SurvivalSele`
functions, in the same process of the master, thus being lo
respect to the parallel GA.

Algorithm 2: The AMQPGA slave

```
1  while true do
2      individual ← ConsumeMessage();
3      individual ← FitnessEvaluation(individual);
4      ProduceMessage(individual);
```

The AMQPGA slave consists of a straightforward algorith
gorithm 2). Until the process is not terminated (line 1), it r
edly consumes a new message as soon as a new message is
on the queue on which it is listening (`ConsumeMessage` in l
Then, it applies the `FitnessEvaluation` function and fi
fitness value field of the individual in line 3. Each individual i
sent back to the master as a message, `ProduceMessage` in
The way the individuals are distributed between the sla
dictated by the configuration of the message queues and com
cation protocol. Moreover, we set a data exchange with Mo
to the purpose of collecting experimentation reports.

6. Empirical study design

Our aim was to understand if the proposed approach c
an effective solution to improve the scalability of GAs. Ther
we had to verify if GAs parallelised using cloud technologies
us to get a better execution time compared to the sequ
version. Moreover, we were interested in quantifying the
time required to have the infrastructure ready to execute th
Thus, we sought to answer the following research question:

RQ. *Is the use of AMQPGA based on a combination of software c*

Methodology: Experiments

- Usually, it clearly states the *research questions*
- Explains the experiments that were conducted
- Also, it describes the evaluation methods for the results

slave nodes, in a round-robin fashion (i.e., assignments are made in equal portions and circular order); 3. the AMQPGA slave nodes process the individuals by computing the fitness function values and publish them on the response queue; 4. the only consumer of the response queue, i.e., the AMQPGA master node, takes back all the individuals and continues the computation until the next generation. Using the message broker as the central point for the computation, we were able to add any number of further slave nodes to the GA, even at runtime, making the application scalable. In terms of discovery, we used the specific name of 'rabbitmq' for the service container executing RabbitMQ. In this way, the master and slave nodes only have to reach that node to be part of the distributed computation.

5.3. Parallel genetic algorithm master and slaves

For the implementation, we chose *Go*, an open-source programming language by Google. In our case, Go was used to simplify the GA processes and build small containerised environments. It is worth noting that it would be possible to switch the Go clients with clients developed with any programming language. The only requirements consist in being able of communicating with RabbitMQ, serialising individuals with the same codification algorithm and respecting the devised communication protocol.

Algorithm 1: The AMQPGA master

```
1  population ← Initialisation(popsize);
2  repeat
3      foreach individual ∈ population do
4          ProduceMessage(individual);
5      population ← ConsumeMessages();
6      selectedCouples ← Selection(population);
7      foreach parent1, parent2 ∈ selectedCouples do
8          child1, child2 ← Crossover(parent1, parent2);
9          offspring ← offspring ∪ {child1} ∪ {child2};
10     foreach individual ∈ offspring do
11         Mutation(individual);
12     population ← SurvivalSelection(population,
       offspring);
13 until termination criterion;
```

Algorithm 1 shows the pseudocode of the AMQPGA master, in charge of managing the communication with the slaves, which compute the fitness evaluation, and the execution of the other genetic operators. It is worth noting that this is just one of the possible implementations for GAs, since we could have also used other genetic operators and customised the order of execution. Moreover, every single operator is generalised since there are several versions, with different parameters each. However, since we mainly aimed at demonstrating the correct operation of the CBA based on software containers and benchmark the parallel GA with the global parallelisation model, the selection and configuration of operators are irrelevant. Let us suppose to have a specific representation for the problem to solve, meaning that is possible to encode the solutions in a data structure, the `Initialisation` function in line 1 produces the initial random population. Until a termination criterion is satisfied, e.g., a specific number of iterations has been reached, the population is evolved through the application of genetic operators lines 2–13, i.e., a generation. At every generation, the master encapsulates each individual in a message that is sent to a queue (`ProduceMessage` function in lines 3–4), ready to be consumed and evaluated from the listening slaves. Then, in line 5 the master collects back all the evaluated individuals through the `ConsumeMessages` function. It is worth noting that the individuals are objects including both the data representation for the solution and the fitness evaluation value. Of course, the fitness value will be filled when computed by the slaves only. Lines 6–12 applies the other genetic operators, i.e., `Selection`, `Crossover`, `Mutation`, and `SurvivalSelection` functions, in the same process of the master, thus being local in respect to the parallel GA.

Algorithm 2: The AMQPGA slave

```
1  while true do
2      individual ← ConsumeMessage();
3      individual ← FitnessEvaluation(individual);
4      ProduceMessage(individual);
```

The AMQPGA slave consists of a straightforward algorithm (Algorithm 2). Until the process is not terminated (line 1), it repeatedly consumes a new message as soon as a new message is ready on the queue on which it is listening (`ConsumeMessage` in line 2). Then, it applies the `FitnessEvaluation` function and fills the fitness value field of the individual in line 3. Each individual is then sent back to the master as a message, `ProduceMessage` in line 4.

The way the individuals are distributed between the slaves is dictated by the configuration of the message queues and communication protocol. Moreover, we set a data exchange with MongoDB to the purpose of collecting experimentation reports.

6. Empirical study design

Our aim was to understand if the proposed approach can be an effective solution to improve the scalability of GAs. Therefore, we had to verify if GAs parallelised using cloud technologies allow us to get a better execution time compared to the sequential version. Moreover, we were interested in quantifying the setup time required to have the infrastructure ready to execute the GAs. Thus, we sought to answer the following research question:

RQ *Is the use of AMQPGA based on a combination of software containers, message queues and cloud orchestration effective for parallel GAs against the sequential execution?*

Considering that the global parallelisation model is parallelised only during the fitness evaluation, to address the **RQ** we considered as a benchmark a dummy fitness evaluation function that does nothing except receiving individuals, sleep for a specified time and return a random fitness value to the master [38]. The choice of this dummy function, together with the variation of the network load (i.e., the chromosome size), was motivated by the fact that it allowed us to assess the GAs scalability considering different problem sizes by just varying the sleep time [5]. Moreover, we tested the actual time required to have the cloud infrastructure ready to execute the parallel GAs. It is worth noting that in global parallelisation model the populations evolve in the same way as the sequential version. For this reason, we do not mention quality results.

Details about the problem and GAs configuration are provided in Section 6.1. The hardware employed to run the experiments is reported in Section 6.2. To understand the effectiveness of the approach, we applied the experimental method described in Section 6.3 and employed several evaluation criteria, namely the execution time, speedup, overhead and setup time, described in Section 6.4. Finally, Section 6.5 analyses some threats to validity that may have affected our experimentation and how we tried to alleviate them.

Results

- Analyzes the outcome data from the experiments
- Tables and figures to help the description and discussion
- Refers to the results to answer the research questions

the *Wilcoxon Signed Rank* test [41], as recommended in the literature [2,20,42]. The Wilcoxon Signed Rank test verifies, as the null hypothesis, if two considered populations have equal distributions. It is particularly useful when no assumptions about the normality of the distributions are possible, as for our case. For all the statistical tests, we accepted a probability of 5 % of committing a Type-I-Error, i.e., the significance level.

Furthermore, we used the *Vargha–Delaney* $\hat{A}_{12}$ test for effect size [43] to characterise the magnitude of difference. The $\hat{A}_{12}$ test is an estimation of the probability the algorithms have against each other in obtaining better results of the considered measure. When two algorithms are compared, and their results are equivalent, then $\hat{A}_{12} = 0.5$. $\hat{A}_{12} > 0.5$ means that, on the average over all the runs, the first algorithm obtains better results than the one with which is compared. The magnitude values can be summarised by 4 nominal values, namely the 'negligible', 'small', 'medium' and 'large'.

6.5. Threats to validity

Threats to *construct validity* concern the relationship between the theory behind the experiments and the observations. In order to alleviate possible threats related to measurement, the GAs execution time was quantified using the system clock, because it represents the speed of a technique to the end-user. We could not make use of any machine independent measure, e.g., the evaluations number, since the sequential and master/slave models behave in the same way.

Threats to
that could infl
the randomne
tional/network
deed, GAs are i
by executing a
each, and prese
nodes may hav
and the multip
well.

Threats to e
findings outsid
the fact that we
provider (i.e., D
fer from other p
times concerni
we considered
tion trends' ins
varying the cor

7. Results

In this secti
ison between s
cution time, is
and overhead a
setup time is a

7.1. Execution

Fig. 5 shows
fitness evaluat
time for the co
with different
single individu
chromosome (
employed a tot

Table 2
The intervals for which the execution time decreases when increasing the number of parallel nodes.

c	$T_f = 1$		$T_f = 10$		$T_f = 100$	
	P_{min}	P_{max}	P_{min}	P_{max}	P_{min}	P_{max}
128	2	8	2	32	2	128
256	4	8	2	32	2	128
512	4	8	2	32	2	128
1 024	4	8	2	32	2	128
2 048	4	8	2	32	2	128
4 096	4	8	2	32	2	128
8 192	4	8	2	32	2	64
16 384	4	8	2	16	2	64
32 768	4	8	2	16	2	64
65 536	–	–	2	16	2	64

generations each, registering a total of 100 observations for each combination.

As expected, when increasing the chromosome size, the execution time for the same cluster size and individual fitness time increases proportionally. Focusing on the execution with 1 slave node, for which the communication is not affected by the message broker scheduling policy, we carried out the following statistical tests. We iteratively compared the distributions of consecutive chromosome sizes looking for the first threshold where the *p-value* of the two-tailed Wilcoxon signed-ranks test was less than the level of significance of 0.05 (i.e., accepting the alternative hypothesis of equality). To strengthen the differences, we also employed the Vargha–Delaney test considering as different only the couples

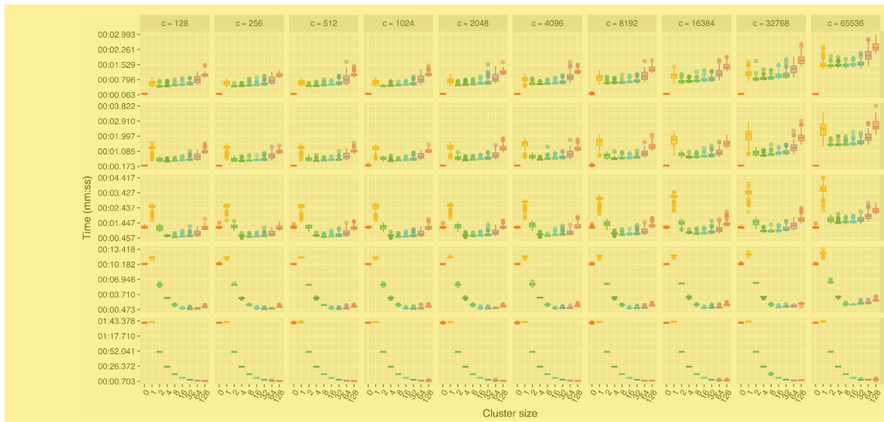


Fig. 5. The execution times achieved by the sequential and parallel GAs, reporting all the combinations for the chromosome size (c), fitness evaluation time (T_f) size.

On the other hand, the increment of the individual fitness time makes the communication time irrelevant against the computation one thus splitting more linearly the execution times between parallel nodes. Except for the cluster with one node (i.e., size of 1), where the resulting execution time is obviously greater, the clusters with multiple nodes outperform the sequential execution (i.e., the cluster size of 0). It means that the execution time is directly proportional to the individual fitness evaluation time and, as we hoped, inversely proportional to the number of cluster nodes.

The effect of the increment of the number of slave nodes is better discussed in the following, where the speedup is analysed.

7.2. Speedup

it is more evident that the chromosome size only influe
execution time from a certain number of nodes on, a t
that is shifted accordingly to the individual fitness evaluat
growth.

7.4. Setup time

To understand the feasibility of the approach in a re
context, we analysed the setup time discriminating into
ation and deployment times. As depicted in Fig. 8, both the
and deployment times are proportional to the target nu
nodes but in a light way. In the case of the creation
strictly depends on the specific capability of the cloud

Conclusions

- Often a summary of the whole work
- Focus on the achieved results after the experiments
- The authors raise more questions for possible future work

a single JVM. However, sacrificing the test of HDFS could be not desirable because it is one of the key points of the Hadoop jobs. Moreover, it is not possible to use multiple reduce executions that is fundamental to test grid and island models. Instead, the second option that is called 'pseudo-distributed' simulates all the required Hadoop daemons, each one running on an independent JVM, thus resulting in very heavy executions possibly not runnable with CI.

8.1.3. Continuous deployment

As for the CD, we consider and discuss the possibility of using a cluster of multiple machines and extending the size of it when and if needed.

AMQPGA is specifically devised to run in the cloud using a container orchestration platform, e.g., CoreOS. The orchestration platform is directly in charge of scheduling containers on the connected resources. Moreover, as described in Section 4.3, a new node can join the cluster by simply installing CoreOS on it. As shown in Section 7.4, the installation of a cloud cluster of 128 nodes takes around 5 min. Furthermore, being AMQPGA based on RabbitMQ, a new node can join a running computation even during a GA is running, in a plug and play fashion. The AMQPGA nodes use the discovery feature of CoreOS to reach the RabbitMQ queue, and then they are ready to divide the computation with the others.

DEAP uses SCOOP for distributed tasks. SCOOP is based on *Secure SHell* (SSH) to connect to remote hosts, thus using SSH keys as a join condition to the same cluster. Even if SSH protocol might result troublesome, DEAP can be potentially easily deployed. It only expects the resulting machine to have installed a Python environment, which is a very common configuration offered by commercial cloud providers. However, the master node needs to know a priori the list of the available resources with the relative addresses, meaning that is not possible to join the distributed execution while a GA is running.

elephant56, being a Hadoop application, is strictly related to the underlying platform. Maintaining and extending a Hadoop cluster means having specific expertise with YARN. Indeed, every single node needs to be set up by installing the OS first, the YARN platform and enabling proper SSH connection with the rest of the nodes. Therefore, it can result in an effort-prone activity.

8.2. Performance discussion

Here we further discuss the results of the empirical study we conducted, by comparing with the way DEAP and elephant56 operate.

8.2.1. Execution time

In Sections 7.1 and 7.3 we analysed the execution time and overhead when running several GA experiments. We varied the chromosome size, i.e., the quantity of data required to represent a single individual, and the fitness evaluation time, i.e., the required time to evaluate a single individual. As also demonstrated by related work [11,12,14,30,31], distributed GA are expected to be particularly effective with intensive computation, when the fitness evaluation time overcome the overhead. Intuitively, the chromosome size seems to be the main factor for overhead. It is strictly related to the activity of communication, consisting of the data that should be moved from one node to another. However, also the specific characteristics of the communication protocol can largely impact performance. In the case of AMQPGA, the presence of a message broker causes every message to pass through a middle point, be stored inside a queue, i.e., a data structure, before being sent to the final destination. On the other hand, it guarantees full reliability in message exchange, and there is no risk if a message gets lost during transmission. Moreover, as described above, the nodes can join the computation in a plug and play way at any moment.

In the case of DEAP, the presence of SCOOP as distributed manager lightens a bit the communication. It depends on the fact that SCOOP uses ZeroMQ, which implements a message queue protocol for asynchronous communication but without using a message broker. However, a master node needs to be fully in charge of resource scheduling reducing the reliability of the whole cluster.

As for elephant56, the data passes through HDFS. On the one hand, data is distributed over the nodes in the Hadoop cluster with a redundancy protocol that guarantees the reliability of data. On the other hand, as confirmed by the literature [14], HDFS is an important bottleneck for communication. The application JAR files have also to be transmitted all over the network when the job starts, adding further overhead.

9. Conclusions and future work

In this paper, we distributed *Genetic Algorithms* (GAs) based on the master/slave model with technologies specifically devised for the cloud, i.e., the software containers, cloud orchestration and message queues. We presented a novel implementation, called AMQPGA, that exploits message queues to schedule parallel GAs tasks. We also devised a conceptual workflow for development, deployment and execution activities of distributed GAs as *Cloud-Based Applications* (CBAs), exploiting modern software engineering methodologies and tools. Then, we empirically assessed the effectiveness of the approach in terms of execution time, speedup, overhead, using a dummy fitness function as a benchmark problem. Finally, we compared AMQPGA with state-of-the-art approaches, highlighting the pros and cons of using an architecture based on containers.

We succeeded in accelerating the execution time of our GA application up to a total number of 128 slave nodes. From the results, it emerged that there is a dependency between computation load and communication cost. We observed that the execution time is directly proportional to the individual fitness evaluation time and inversely to the number of cluster nodes. There is an inferior limit for the evaluation time for the fitness function that makes the parallelisation effective. Also, there is also a superior limit regarding the chromosome size that, together with the population size, determines the network load and thus influences the final execution time. Moreover, we observed that the setup time can be quantified to a few minutes even if the request is of many nodes (e.g., 128). It is worth noting that this time is related to a completely automatised activity, which does not require the human presence as in the case of other methodologies, e.g., Hadoop [10].

The performance and setup times place the cloud positively between other employed technologies for GAs parallelisation, e.g., multi-core systems, GPUs [3,4] and Hadoop MapReduce. Cloud orchestration and software containers surpass limitations of existing approaches for parallel GAs distribution, offering exclusive features such as environment configuration as part of the development and plug and play join to the computation. Also, it is clear that cloud orchestration for parallel GAs can be considered affordable in an economical way, against other technologies strictly related to the hardware physically owned.

One avenue for future work is to evaluate other models of parallel GAs such as the cellular and island model [2]. The empirical study should be replicated with other fitness functions, and we want to put into operation our structure by solving real world optimisation problems, such as Test Suite generation [12,31] or machine learning problems. To make the approach more flexible and easy to use, we also plan to abstract the concepts further and propose it in the form of a framework. In this way, the developer would have to deal exclusively with the activity of problem encoding. As for the cloud aspects, we want to measure other metrics

References

- The list of references cited within the paper
- Depends on the bibliography style of the publisher
- Always reports at least the authors, title, year, and location for each of the references

such as energy efficiency and security of the algorithms [47]. We aim at exploiting and investigating more the features of multi-cloud [18], enhancing our architecture with an Intention Workflow Model [15,48] to reach full self-management of GAs execution in the cloud, and considering the cost-effect factor of cloud spot instances [49].

References

[1] M. Harman, The current state and future of search based software engineering, in: 2007 Future of Software Engineering, IEEE Computer Society, 2007, pp. 342–357.
[2] G. Luque, E. Alba, Parallel Genetic Algorithms: Theory and Real World Applications, in: Studies in Computational Intelligence, no. 367, Springer, 2011.
[3] L. Zheng, Y. Lu, M. Guo, S. Guo, C.-Z. Xu, Architecture-Based design and optimization of genetic algorithms on multi- and many-core systems, Future Gener. Comput. Syst. 38 (2014) 75–91.
[4] S. Yoo, M. Harman, S. Ur, GPGPU test suite minimisation: search based software engineering performance improvement using graphics cards, Empir. Softw. Eng. 18 (3) (2013) 550–593.
[5] M. Ivanovic, V. Simic, B. Stojanovic, A. Kaplarevic-Malisic, B. Marovic, Elastic grid resource provisioning with wobingo: a parallel framework for genetic algorithm based optimization, Future Gener. Comput. Syst. 42 (2015) 44–54.
[6] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, B.-S. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, Future Gener. Comput. Syst. 23 (4) (2007) 658–670.
[7] P. Salza, F. Ferrucci, F. Sarro, Develop, deploy and execute parallel genetic algorithms in the cloud, in: Evolutionary Computation Software Systems Workshop at GECCO, EvoSoft, 2016, pp. 121–122.
[8] I. Sadooghi, J.H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T.P.P. de Lacerda Ruivo, G. Garzoglio, S. Timm, Y. Zhao, I. Raicu, Understanding the performance and potential of cloud computing for scientific applications, IEEE Trans. Cloud Comput. 5 (2) (2017) 358–371.
[9] F. Ferrucci, M.-T. Kechadi, P. Salza, F. Sarro, A framework for genetic algorithms based on hadoop, Computing research repository (CoRR) abs/1312.0086.
[10] F. Ferrucci, P. Salza, M.-T. Kechadi, F. Sarro, A parallel genetic algorithms framework based on hadoop mapreduce, in: ACM/SIGAPP Symposium on Applied Computing, SAC, 2015, pp. 1664–1667.
[11] A. Verma, X. Llorà, D.E. Goldberg, R.H. Campbell, Scaling genetic algorithms using mapreduce, in: International Conference on Intelligent Systems Design and Applications, ISDA, 2009, pp. 13–18.
[12] L. Di Geronimo, F. Ferrucci, A. Murolo, F. Sarro, A parallel genetic algorithm based on hadoop mapreduce for the automatic generation of junit test suites, in: IEEE International Conference on Software Testing, Verification and Validation, ICST, 2012, pp. 785–793.
[13] P. Fazenda, J. McDermott, U.-M. O'Reilly, A library to run evolutionary algorithms in the cloud using mapreduce, in: European Conference on Applications of Evolutionary Computation, EvoApplications, 2012, pp. 416–425.
[14] F. Ferrucci, P. Salza, F. Sarro, Using hadoop mapreduce for parallel genetic algorithms: a comparison of the global, grid and island models, Evolutionary Computation.
[15] T. Baker, M. Mackay, M. Randles, A. Taleb-Bendiab, Intention-Oriented programming support for runtime adaptive autonomic cloud-based applications, Comput. Electr. Eng. 39 (7) (2013) 2400–2412.
[16] D. Weerasiri, M.C. Barukh, B. Benatallah, Q.Z. Sheng, R. Ranjan, A taxonomy and survey of cloud resource orchestration techniques, ACM Comput. Surv. 50 (2) (2017) 1–41.
[17] Z. Kozhirbayev, R.O. Sinnott, A performance comparison of container-based technologies for the cloud, Future Gener. Comput. Syst. 68 (2017) 175–182.
[18] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg, Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems, in: IEEE International Conference on Cloud Computing, Cloud, 2014, pp. 887–894.
[19] M. Harman, S.A. Mansouri, Y. Zhang, Search-Based software engineering: trends techniques and applications, ACM Comput. Surv. 45 (1) (2012) 1–61.
[20] M. Harman, P. McMinn, J.T. de Souza, S. Yoo, Search Based Software Engineering: Techniques, Taxonomy, Tutorial, in: Empirical Software Engineering and Verification, Vol. 7007, Springer, Berlin Heidelberg, 2012, pp. 1–59.
[21] M. Harman, B.F. Jones, Search-Based software engineering, Inf. Softw. Technol. 43 (2001) 833–839.
[22] X. Yu, M. Gen, Introduction to Evolutionary Algorithms, Decision Engineering, Springer London, London, 2010.
[23] elephant56, A genetic algorithms framework for hadoop mapreduce. https://github.com/pasqualesalza/elephant56.
[24] jMetal: a framework for multi-objective optimization with metaheuristics. https://github.com/jMetal/jMetal.
[25] Jenetics, Java genetic algorithm library. http://jenetics.io.
[26] DEAP, Distributed evolutionary algorithms in Python. https://github.com/DEAP/deap.
[27] M. Fowler, VersionControlTools. https://martinfowler.com/bliki/VersionControlTools.html (February 2010).
[28] M. Fowler, Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html (January 2006).
[29] Agile Alliance, Continuous Deployment, 2018. https://www.agilealliance.org/glossary/continuous-deployment.
[30] C. Jin, C. Vecchiola, R. Buyya, MRPGA: an extension of mapreduce for parallelizing genetic algorithms, in: IEEE International Conference on E-Science (e-Science), 2008, pp. 214–221.
[31] S. Di Martino, F. Ferrucci, V. Maggio, F. Sarro, Towards migrating genetic algorithms for test data generation to the cloud, in: Software Testing in the Cloud: Perspectives on an Emerging Discipline, IGI Global, 2013, pp. 113–135.
[32] I.A.T. Hashem, N.B. Anuar, A. Gani, I. Yaqoob, F. Xia, S.U. Khan, MapReduce: review and open challenges, Scientometrics 109 (1) (2016) 389–422.
[33] K. Veeramachaneni, I. Arnaldo, O. Derby, U.-M. O'Reilly, FlexGP: cloud-based ensemble learning with genetic programming for large regression problems, J. Grid Comput. 13 (3) (2015) 391–407.
[34] P. Salza, F. Ferrucci, F. Sarro, Elephant56: design and implementation of a parallel genetic algorithms framework on hadoop mapreduce, in: Genetic and Evolutionary Computation Conference, GECCO, 2016, pp. 1315–1322.
[35] J.J. Merel Guervós, A.M. Mor. García, C.M. Fernandes, A.I. Esparcia-Alcázar, SofEA, a pool-based framework for evolutionary algorithms using couchDB, in: Genetic and Evolutionary Computation Conference, GECCO, 2012, pp. 109–116.
[36] M. García-Valdez, L. Trujillo, J.J. Merel Guervós, F. Fernandez de Vega, G. Olague, The evospace model for pool-based evolutionary algorithms, J. Grid Comput. 13 (3) (2015) 329–349.
[37] J. Lewis, M. Fowler, Microservices, a definition of this new architectural term. https://martinfowler.com/articles/microservices.html (March 2014).
[38] E. Cantú-Paz, D.E. Goldberg, On the scalability of parallel genetic algorithms, Evol. Comput. 7 (4) (1999) 429–449.
[39] E. Alba, A.J. Nebro, J.M. Troya, Heterogeneous computing and parallel genetic algorithms, J. Parallel Distrib. Comput. 62 (9) (2002) 1362–1385.
[40] E. Alba, J.M. Troya, Analyzing synchronous and asynchronous parallel distributed genetic algorithms, Future Gener. Comput. Syst. 17 (4) (2001) 451–465.
[41] W.J. Conover, Practical Nonparametric Statistics, third ed., John Wiley & Sons, 1999.
[42] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: IEEE/ACM International Conference on Software Engineering, ICSE, 2011, pp. 1–10.
[43] A. Vargha, H.D. Delaney, A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong, J. Educ. Behav. Stat. 25 (2) (2000) 101–132.
[44] D. Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: a python framework for evolutionary algorithms, in: Genetic and Evolutionary Computation Conference (GECCO), ACM, 2012, pp. 85–92.
[45] F.-A. Fortin, F.-M.D. Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: evolutionary algorithms made easy, J. Mach. Learn. Res. 13 (2012) 2171–2175.
[46] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2012.
[47] B. Aldawsari, T. Baker, D. England, Trusted energy-efficient cloud-based services brokerage platform, Int. J. Intell. Comput. Res. 6 (4) (2015) 630–639.
[48] T. Baker, O.F. Rana, R. Calinescu, R. Tolosana-Calasanz, J.A. Bañares, Towards autonomic cloud services engineering via intention workflow model, in: International Conference on Economics of Grids, Clouds, Systems, and Services, GECON, 2013, pp. 212–227.
[49] Z. Li, H. Zhang, L. O'Brien, S. Jiang, Y. Zhou, M. Kihl, R. Ranjan, Spot pricing in the cloud ecosystem: a comparative investigation, J. Syst. Softw. 114 (2016) 1–19.

Pasquale Salza is a Postdoctoral Research Assistant at USI Università della Svizzera italiana, Switzerland. He received his Ph.D. degree in Computer Science from the University of Salerno, Italy, in 2017. His research interests are mainly focused on cloud computing, search-based software engineering, evolutionary computation and mobile computing, with the aim of efficiently joining solutions and approaches to improve Information Technology systems and supply software solutions of better quality, cost-effectively.

Filomena Ferrucci is professor of software engineering and software project management at University of Salerno, Italy. Her main research interests include software metrics, effort estimation, search-based software engineering, empirical software engineering, and human–computer interaction. She has been program co-chair of the International Summer School on Software Engineering.


Scientific conferences

What is a conference?

- An annual event
- Present the state of the art in specific fields
- Keynotes
- Networking
- Journal first presentations

https://youtu.be/I_57gjvek7Q

Organization

- A conference is based on a complex organization of people and resources
- The majority of international conferences changes the venue every year
- Also the organization for the instances of the conference changes

Steering committee

- Organizes the conference at global level
- Decides important future directions
- Selects the location for the next instances
- Nominates the *general chair(s)*

Organizing committee

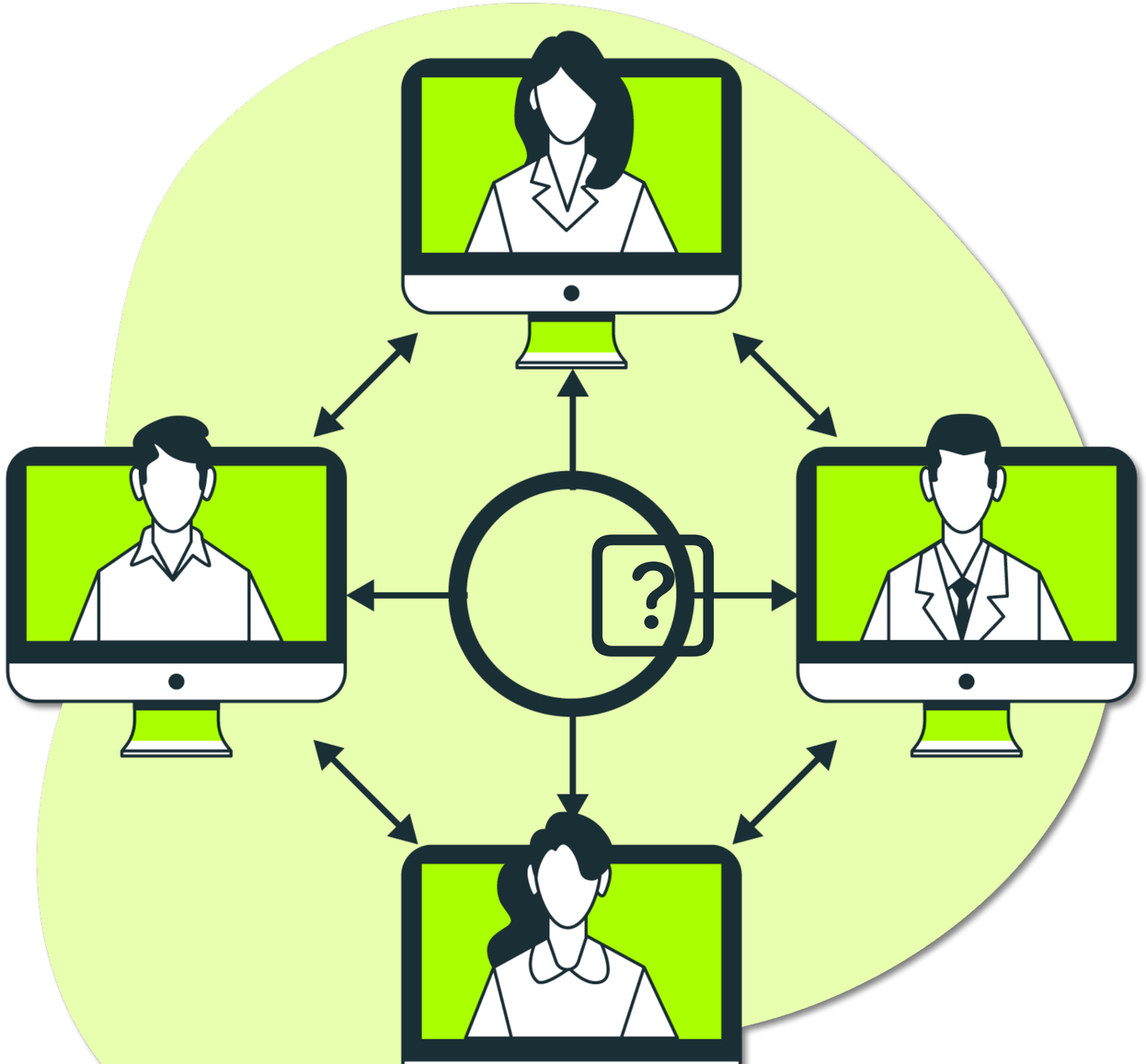
- It's directed by the *general chair(s)*
- The general chair nominates the organizing committee
- Several figures who help to organize the local event and its publicity
- In particular, the *program chairs* are responsible for the scientific program

Program committee

- Selects which papers to accept and reject
- Each of the members reviews several papers
- They are coordinated by the *program chairs*

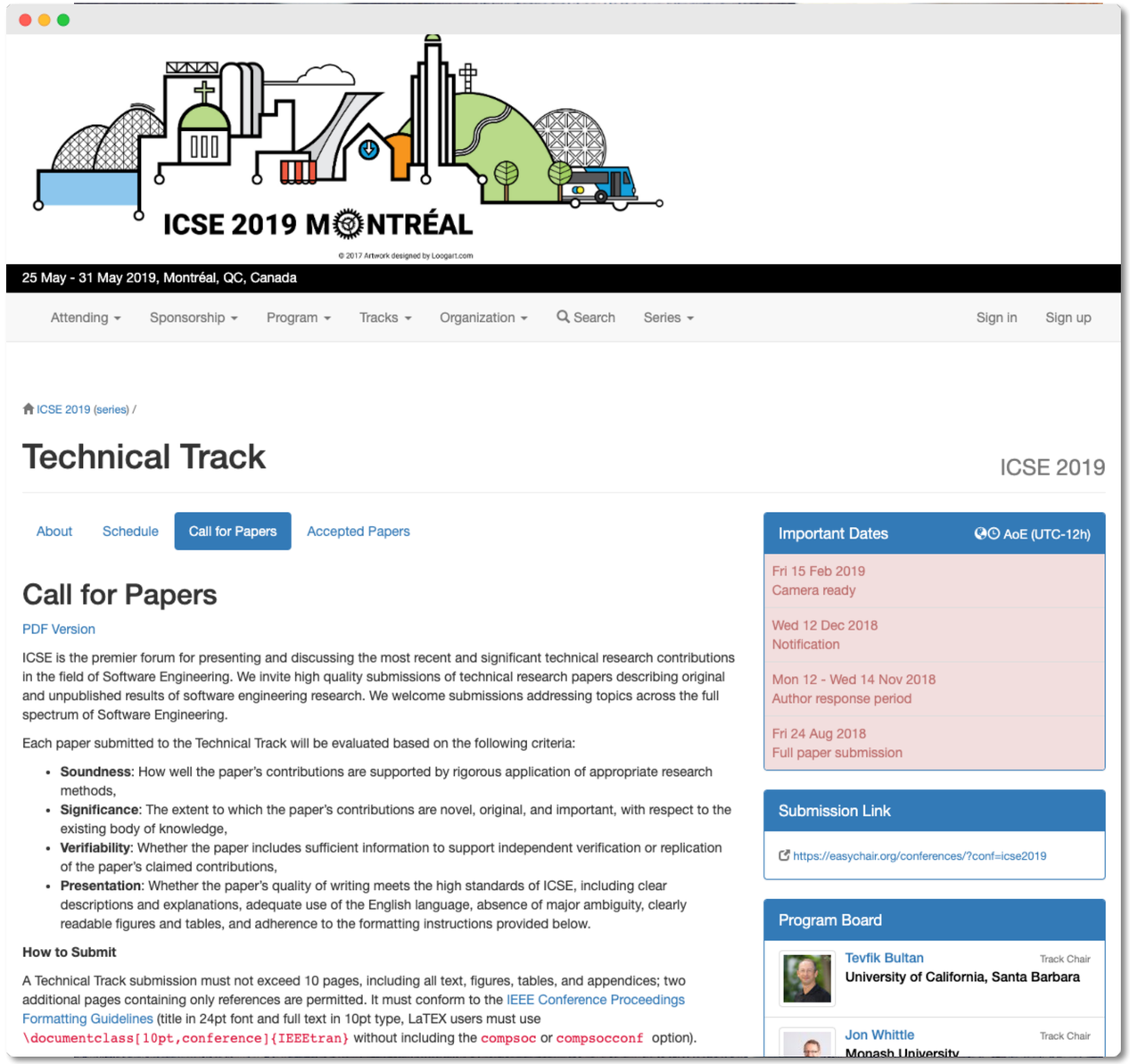
Peer reviewing process

- The papers are reviewed by peers
- A peer is an expert in the same field of the authors



Submission

- A *call for papers* is published on the official website
- A specific LaTeX template is indicated
- There are also page limits, usually 10 pages and 2 pages for references, two columns
- By a *deadline*, one of the authors has to submit the PDF of the paper on a specific platform, e.g., *EasyChair*



Blindness

- Sometimes, the reviewing process is *blind*
- For conferences, it’s at least a *single blind*
- In single blind, the reviewers’ identities are kept hidden from authors
- In *double blind*, the identities of both authors and reviewers are kept hidden

Conflicts

- Both authors and reviewers have to declare *conflicts*
- Lifelong relationship between Ph.D. student and Ph.D. supervisor
- Personal or family relationships
- Professional relationships within the last two years or reasonably expected within the next year

Bidding

- The program chairs control for *desk rejects* issues, e.g., plagiarism, clones
- The program committee starts the *bidding* phase
- Every member can see at least the title and abstract of papers
- They express their preferences
- The program chairs, helped by the platform, assign the reviewers to the papers
- Usually, 3 reviewers per paper

Reviewing

- Then, the reviewers have access to the PDF files of the papers
- They have to write a *review* where they evaluate the paper
- It’s mostly a written feedback
- They also have to choose a score on a scale
- E.g., *strong reject*, *weak reject*, *weak accept*, *accept*, *award level*

Discussion

- The reviewers have to find a consensus
- Usually, a program chair joins to moderate the discussion
- A final decision is then made

Notification

- The e-mails with the notification are sent on a defined date
- The notification, either for acceptance or reject, contains the produced reviews
- The reviews serve as indications for improvement

Camera ready submission

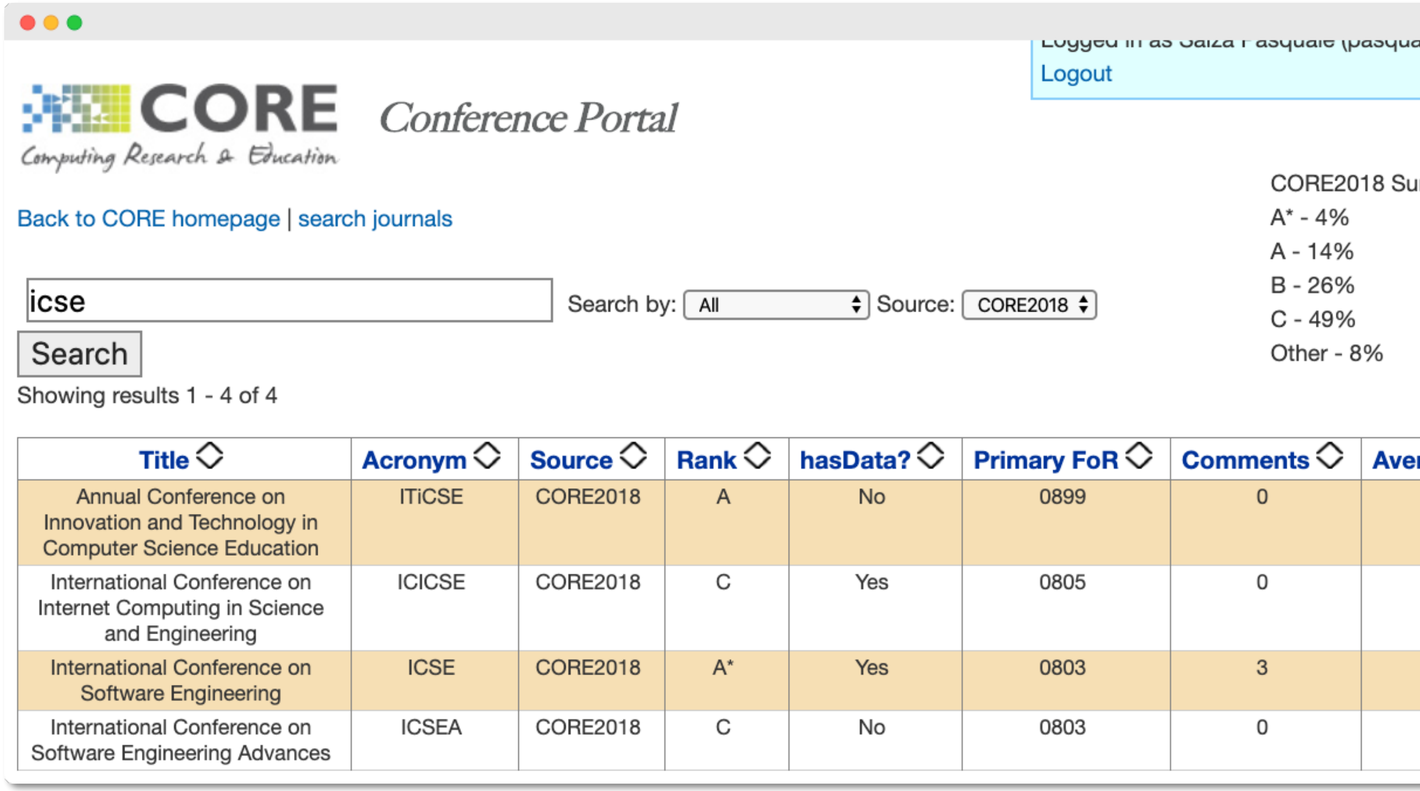
- In the case of acceptance, the authors have a few days to prepare the final version of the paper
- Very few changes are allowed, mostly for the format
- Clarifications are allowed
- No additional changes

Publish in proceedings

- Usually, just before the conference starts, the papers are available in a digital format
- During, or just after, the conference, the proceedings are officially published
- It’s always mandatory for at least one of the authors to attend the conference and present the paper
- A paper is considered as published when it has a DOI (Digital Object Identifier)

CORE conference ranking

- An ongoing activity sponsored by the *Computing Research and Education Association of Australasia*
- Provides assessments of major conferences in computing disciplines
- The ranking is updated from time to time based upon a mix of indicators, e.g., citation rates, paper submission, and acceptance rates



Title	Acronym	Source	Rank	hasData?	Primary FoR	Comments	Aver
Annual Conference on Innovation and Technology in Computer Science Education	ITiCSE	CORE2018	A	No	0899	0	
International Conference on Internet Computing in Science and Engineering	ICICSE	CORE2018	C	Yes	0805	0	
International Conference on Software Engineering	ICSE	CORE2018	A*	Yes	0803	3	
International Conference on Software Engineering Advances	ICSEA	CORE2018	C	No	0803	0	

Scientific journals

What is a journal?

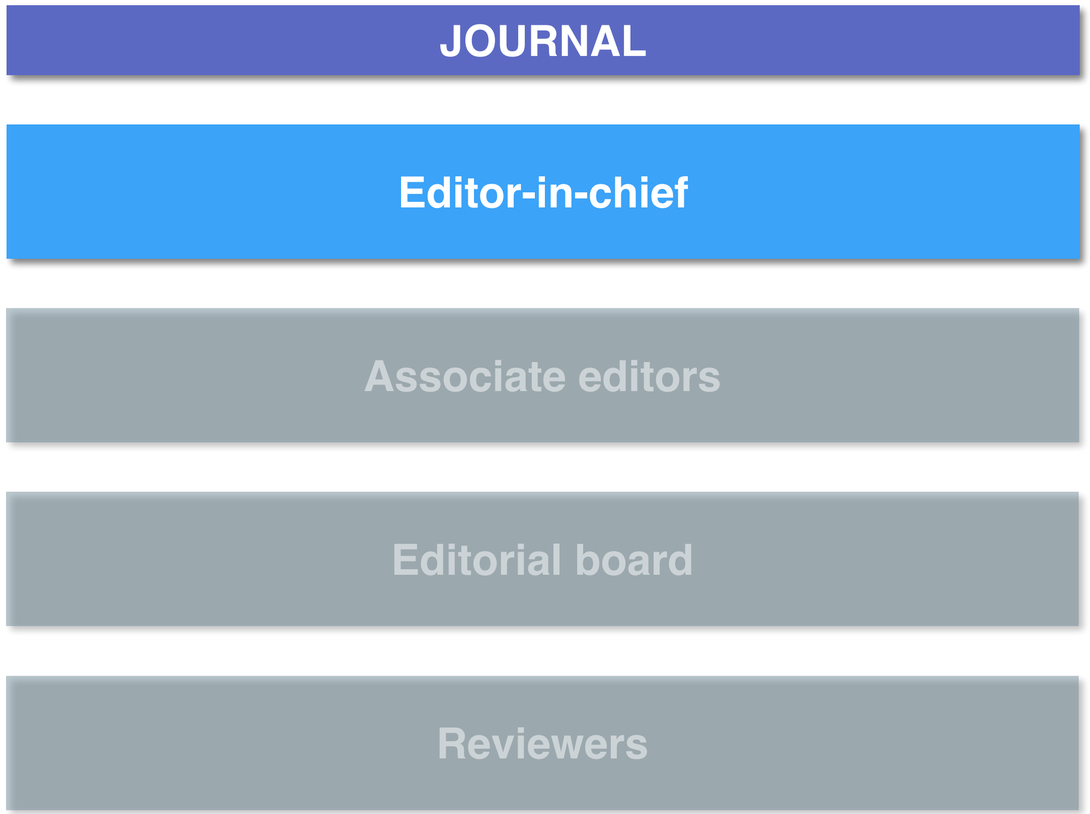
- Multiple issues during the year
- Long-lasting process
- Multiple review phases
- More pages allowed (usually)
- Extensions of publications to conferences
- Brand new works, i.e., journal first

Organization

- The organization of a journal is more or less stable for a certain amount of years
- The *publisher*, e.g., *IEEE*, decides the *editor-in-chief*

Editor-in-chief

- Directs the journal
- Nominates the *associate editors* and *editorial board*
- Is responsible for the academic content of the journal

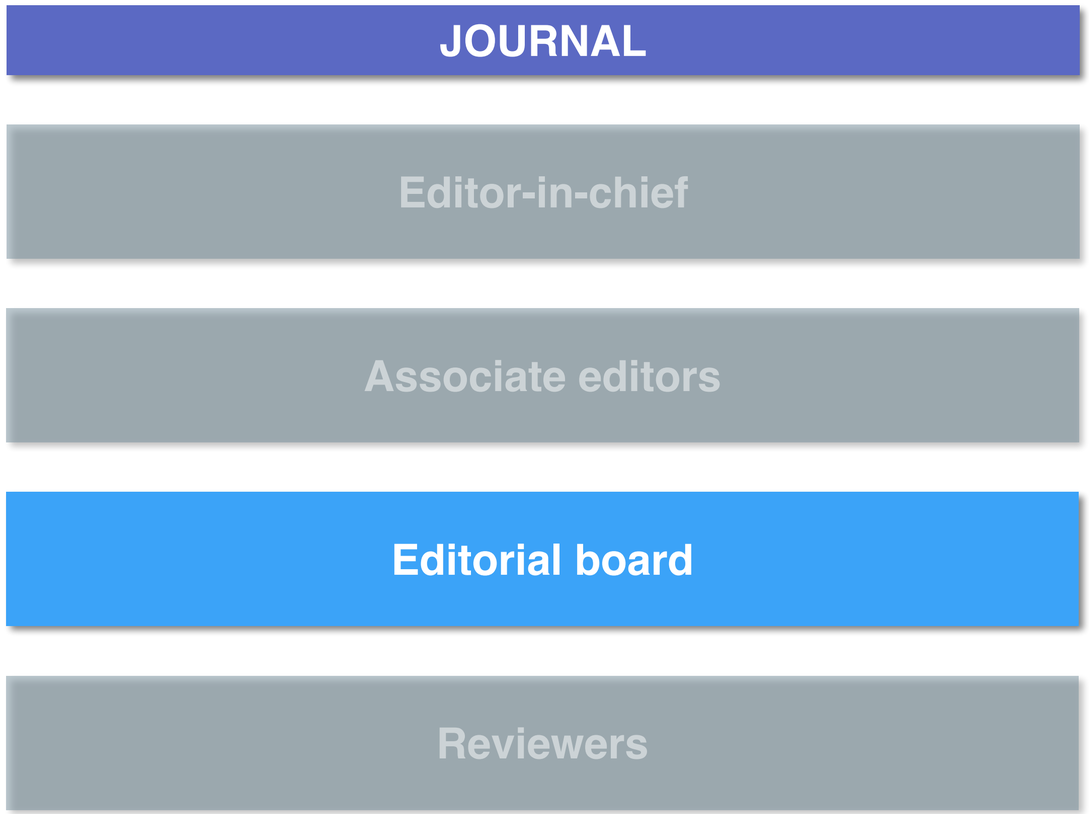


Associate editors

- Help the *editor-in-chief* in administration matters
- They are usually directly responsible for the assignment of papers

Editorial board

- A pool of researchers who are expert in several subfields
- They are chosen by the editor-in-chief and associate editors
- Once a paper is submitted, if there is not a desk reject, the associate editors assign an editor from the board to it
- They are responsible for the reviewing process of the papers

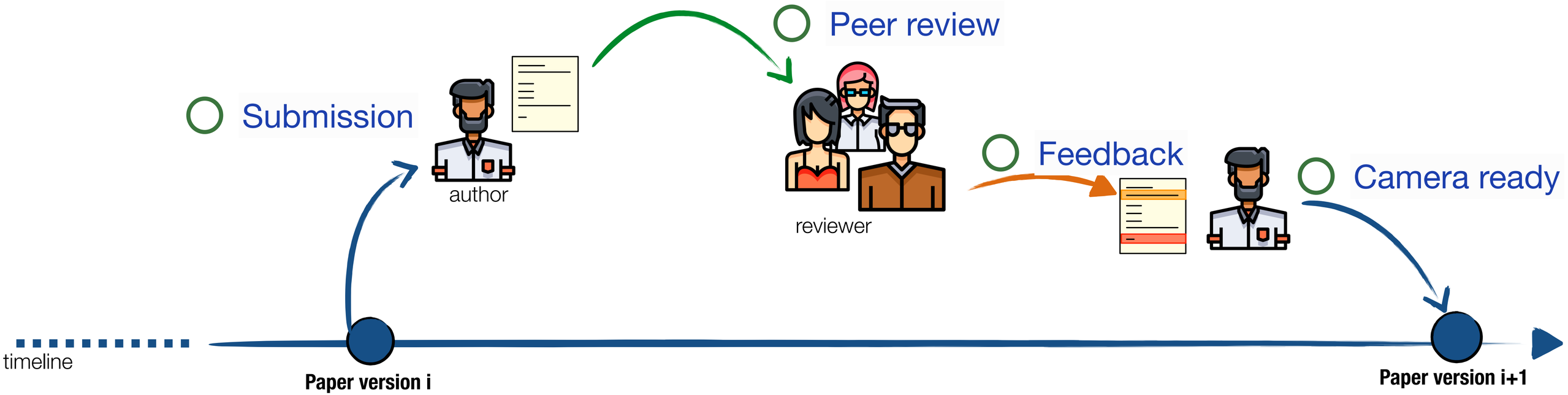


Reviewers

- The assigned editors can invite *reviewers*
- They could change for different steps of revisions of a paper

Reviewing process

1. A paper is submitted
2. An associate editor chooses an editor from the editorial board
3. The assigned editor invites around 3 reviewers
4. When they accept, the process for review starts (usually 3 months)


Submission
author
Peer review
reviewer
Feedback
Camera ready
timeline
Paper version i
Paper version i+1

Decision

- The reviewers propose a decision for the paper
- The editor analyzes the proposals and has the final decision on the paper
- 😖 If *reject*, the paper is excluded from future publications with the journal
- 😐 If *revise and resubmit*, the authors are invited to considerably revise the paper and resubmit it again, causing a new full process of review
- 🙂 If *major revision*, the authors will have a certain amount of time to implement the indications from the reviewers
- 😏 If *minor revision*, the indications will require a bit of effort since the paper is almost ready for acceptance
- 🥳 If *accepted*, the authors can work with the publisher for final proofreading and publication

Reviewing

- Second, you can start doing your 3 reviews
- Brief Summary for the report (3-4 lines)
- Technical Quality, Originality, and Significance
    - Are the arguments in the paper logically sound and well-supported?
    - How original and novel are the findings compared to existing research?
    - How significant is the research question in advancing the field?
    - Does the paper provide a comprehensive review of the research area?
    - What are the key strengths, and what areas need improvement?
    - Are there any gaps or limitations that should be addressed?
- Logical Structure, Presentation, and Style
- Overall Feedback

Reviewing

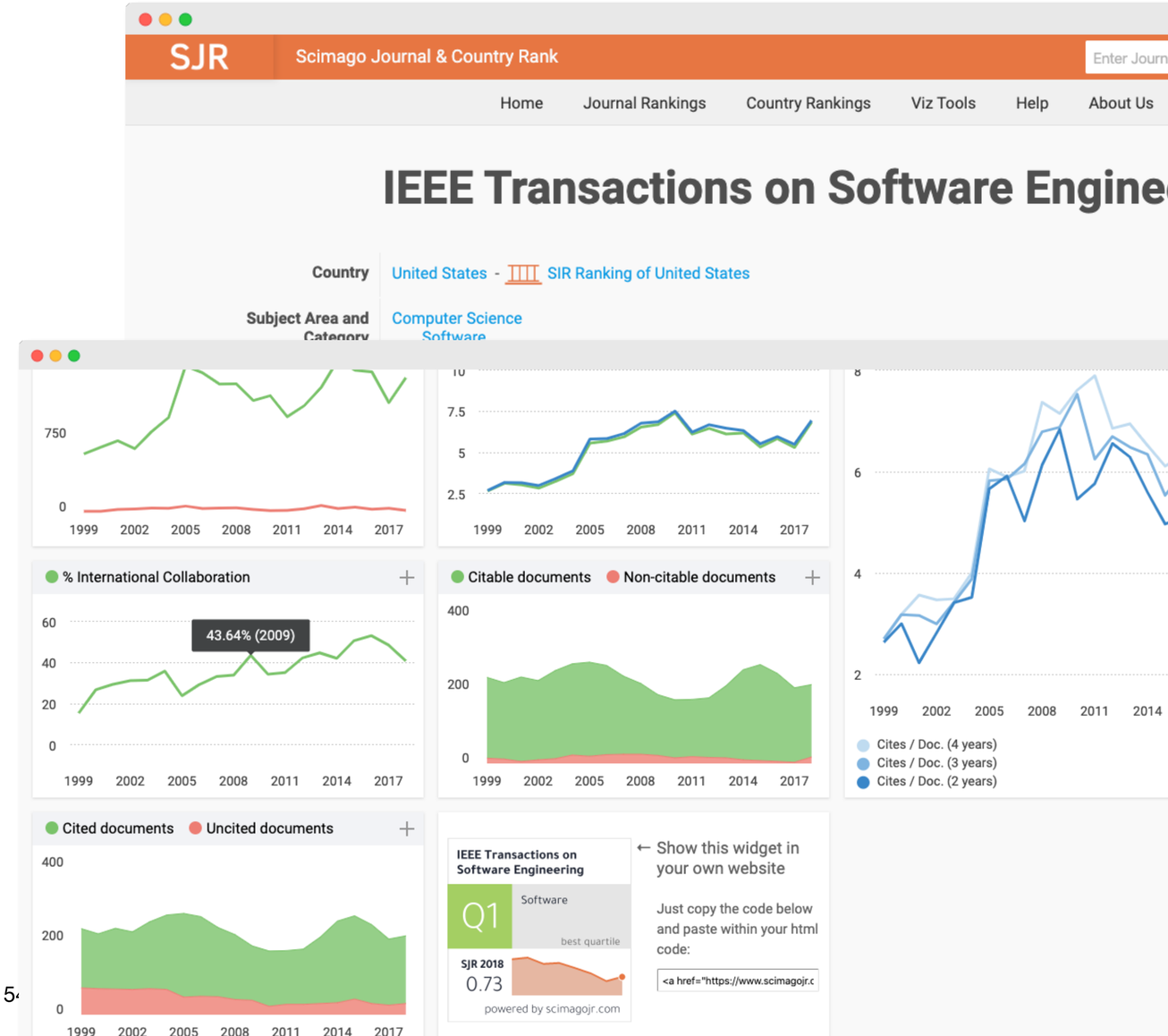
- Second, you can start doing your 3 reviews.
- Brief Summary for the report (3-4 lines)
- Technical Quality, Originality, and Significance
- Logical Structure, Presentation, and Style
    - Is the paper well-structured, with a clear and logical flow?
    - Are the key concepts and methodologies clearly explained?
    - Are graphics, tables, and figures used effectively to support the discussion?
    - Is the writing clear, professional, and easy to follow?
    - What specific improvements could enhance readability and coherence?
- Overall Feedback

Reviewing

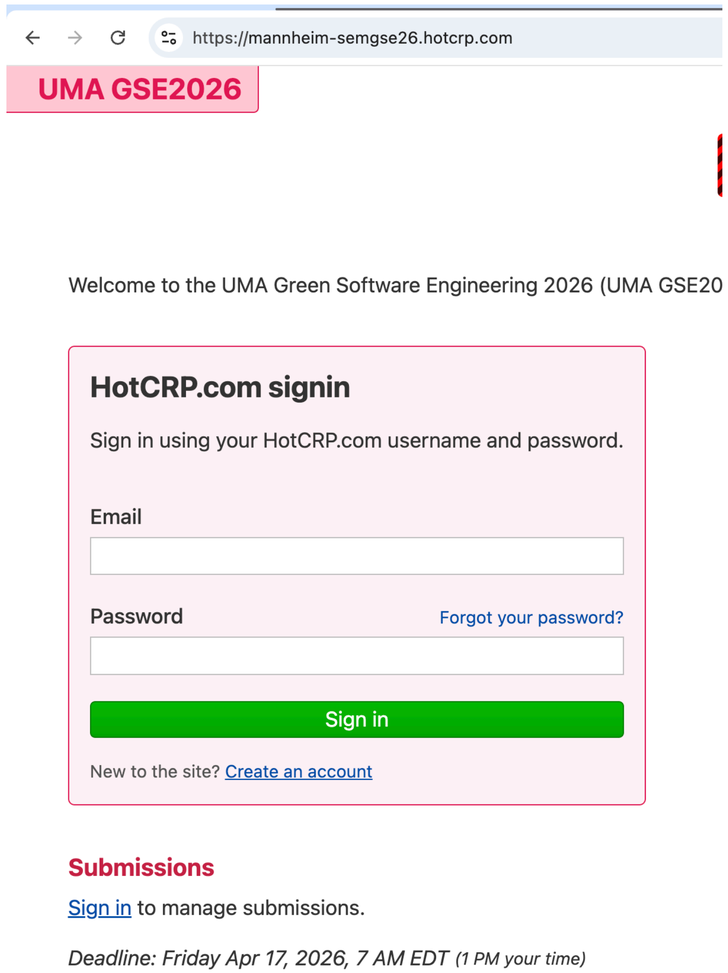
- Second, you can start doing your 3 reviews.
- Brief Summary for the report (3-4 lines)
- Technical Quality, Originality, and Significance
- Logical Structure, Presentation, and Style
- Overall Feedback
    - What are the main takeaways, and how does this report contribute to the field?
    - What constructive suggestions can be provided to strengthen the paper?
    - How can the paper better communicate its significance and findings?
    - How can feedback be delivered constructively, politely, and professionally?

Scimago Journal Rank

- A portal including the journals indicators developed from the information contained in the *Scopus* database
- The *SJR* metric is a numeric value indicating the average number of weighted citations received on a select year per document published in that journal during the previous three years
- The higher SJR, the better
- Usually the *quartiles* are used to identify best journals



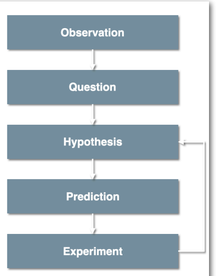
Do not forget to sign with Mannheim address



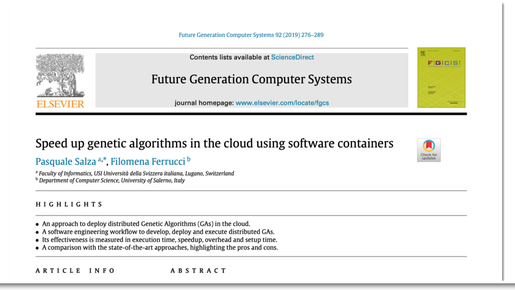
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

The scientific method

- A logical problem-solving approach
- A paper is scientific if it follows the method
- Otherwise it's narrative
- An iterative process



The structure of a research paper



Future Generation Computer Systems 92 (2019) 276–289

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speed up genetic algorithms in the cloud using software containers

Pasquale Salza [a,*], Filomena Ferrucci [b]

[a] Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland
[b] Department of Computer Science, University of Salerno, Italy

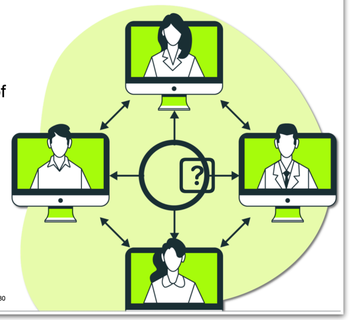
HIGHLIGHTS

- An approach to deploy distributed Genetic Algorithms (GAs) in the cloud.
- A software engineering workflow to develop, deploy and execute distributed GAs.
- Its effectiveness is measured in execution time, speedup, overhead and setup time.
- A comparison with the state-of-the-art approaches, highlighting the pros and cons.

ARTICLE INFO ABSTRACT

Peer reviewing process

- The papers are reviewed by peers
- A peer is an expert in the same field of the authors

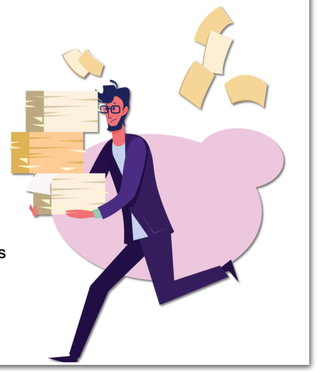


What is a conference?

- An annual event
- Present the state of the art in specific fields
- Keynotes
- Networking
- Journal first presentations

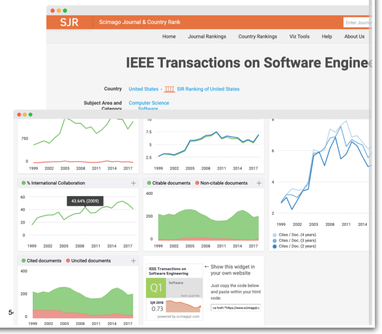
What is a journal?

- Multiple issues during the year
- Long-lasting process
- Multiple review phases
- More pages allowed (usually)
- Extensions of publications to conferences
- Brand new works, i.e., journal first



Scimago Journal Rank

- A portal including the journals indicators developed from the information contained in the *Scopus* database
- The *SJR* metric is a numeric value indicating the average number of weighted citations received on a select year per document published in that journal during the previous three years
- The higher SJR, the better
- Usually the *quartiles* are used to identify best journals



Dr. Pooja Rani
rani@ifi.uzh.ch, pooja.rani@uni-mannheim.de
University of Zurich, Switzerland
University of Mannheim, Germany