UNIVERSIDADE FEDERAL FLUMINENSE

Diego Machado da Conceição Soares

Choosing the Right Cloud Configuration for You and your Workload

NITERÓI

2023

Diego Machado da Conceição Soares

Choosing the Right Cloud Configuration for You and your Workload

This Master's Dissertation was presented to the Postgraduate Program in Computing of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Research Area: Computer Science.

Advisor:
Eugene Francis Vinod Rebello

NITERÓI

2023

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S676c Soares, Diego Machado da Conceição
Choosing the Right Cloud Configuration for You and your Workload / Diego Machado da Conceição Soares. - 2023.
68 f.: il.

Orientador: Eugene Francis Vinod Rebello.
Dissertação (mestrado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2023.

1. Cloud Computing. 2. Scientific Workloads. 3. Job Scheduling. 4. Multi-Objective Optimization. 5. Produção intelectual. I. Rebello, Eugene Francis Vinod, orientador. II. Universidade Federal Fluminense. Instituto de Computação.III. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

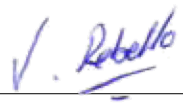
Diego Machado da Conceição Soares

Choosing the Right Cloud Configuration for You and your Workload

This Master's Dissertation was presented to the Postgraduate Program in Computing of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Research Area: Computer Science.

Approved on the 10th of July, 2023.

EXAMINATION BOARD



Prof. Eugene Francis Vinod Rebello - Advisor, IC/UFF

Prof. Maria Cristina Silva Boeres, IC/UFF

Prof. Alexandre da Costa Sena, IME/UERJ

Niterói

2023

Be ashamed to die until you have won some victory for humanity! (Horace Mann)

Acknowledgements

I want to thank my mother and father, without which nothing would be possible. Their sacrifice to give me the best education they could made these very words possible. Thank you, mother; thank you, father. I love you.

I want to thank my beloved wife for all her support during these difficult times. You know *you're like a dream come true*.

I want to thank my two daughters, Diana and Alice, for being the reason for my existence, my everything. I am very sorry for every night I had no time to play, every weekend I could not take you to the park, every holiday I had to study. Everything I do is for you girls.

I thank my brothers for all their support. Love you guys!

I thank my advisor for all comprehension, patience, and caring. For all nights of endless meetings. Thank you, Vinod!

I thank Prof. Cristina and Prof. Alexandre for being on the examining board; it was an honor! I also thank my research colleagues, Mario, Daniel, and José Victor. Daniel, it is up to you now! Best of luck!

I thank Led Zeppelin, Black Sabbath, Sepultura, Metallica, Slayer, and all the heavy metal bands that, together with my noise-canceling headphones, gave me the peace and "silence" I needed to make this work possible. I also thank myself and all the voices inside my head for all the dedication and hard work; this wouldn't be possible without you.

I thank CNPq and AWS for the grant that made this work possible.

Abstract

IaaS cloud service providers traditionally maintain a large portfolio of preconfigured VM instances (with static resource allocations) in an attempt to be able to offer each of their users an instance that is capable of meeting the resource requirements of their workloads. Unfortunately, with such an overwhelming variety of options, a user's ability to find their ideal choice is frequently obscured, resulting in the application taking longer to execute and incurring higher-than-expected costs. Although existing tools have been proposed in the literature to assist users in identifying an appropriate set of instances, they generally aim to simply identify the instance type with sufficient resources for a single execution of a given application. This work focuses on a different aspect of this problem. In many scientific studies, it is common that experimental workloads are composed of multiple executions of a parallel application, each with different input data or parameters, for example, parameter sweep investigations. This work presents a methodology to identify a set of Pareto optimal execution plans (called configurations) composed of the number and type of VM instances, the application's configuration, and per-instance schedules, that meet the user's chosen quality of service requirements. Accordingly, the methodology can provide the user or orchestrator with the fastest and the cheapest solutions, as well as a selection of compromises in between, so that the most suitable choice can be made at that moment.

Keywords: Cloud Computing, Scientific Workloads, Job Scheduling, Multi-Objective Optimization.

Resumo

Provedores de nuvem que oferecem o modelo de Infraestrutura como Serviço (*IaaS*) tradicionalmente mantêm um grande portfólio de instâncias de VM pré-configuradas (com alocações estáticas de recursos) na tentativa de oferecer a cada um de seus usuários uma instância capaz de atender aos requisitos de recursos de suas cargas de trabalho. Infelizmente, com uma variedade tão grande de opções, a capacidade do usuário de encontrar a escolha ideal é frequentemente obscurecida, fazendo com que a sua aplicação demore mais para ser executada e incorra em custos maiores do que o esperado. Embora as ferramentas tenham sido propostas na literatura para auxiliar os usuários na identificação de um conjunto apropriado de instâncias, elas geralmente visam simplesmente identificar o tipo de instância com recursos suficientes para uma única execução de uma determinada aplicação. Este trabalho trata de um aspecto diferente deste problema onde a carga de trabalho é composta por múltiplas execuções de uma aplicação paralela, com diferentes dados de entrada ou parâmetros, situação comum em experimentos científicos baseados em investigações de varredura de parâmetros. Este trabalho apresenta uma metodologia para identificar um conjunto de planos de execução (chamados configurações) Pareto ótimos, compostos pelo número e tipo de instância de VM, a configuração da aplicação e um escalonamento de tarefas por instância, que atendem aos requisitos de Qualidade de Serviço escolhidos pelo usuário. Assim, a metodologia é capaz de fornecer ao usuário ou orquestrador as soluções mais rápidas e econômicas, bem como uma escolha de compromissos intermediários para que seja feita a escolha mais adequada para aquele momento.

Palavras-chave: Computação em nuvem, Cargas de trabalho científicas, Escalonamento de jobs, Otimização multi-objectivo.

Contents

List of Figures

List of Tables

1 Introduction

The adoption of cloud computing (BUYYA et al., 2009) has brought numerous benefits to companies, research institutions that work in a wide variety of areas, and society in general. As with any commercial enterprise, to meet the diverse needs of even more clients, cloud providers frequently add to their portfolio of services and resources, adopt the most recent technologies, make costs more competitive, and introduce new pricing models or markets. With quick and easy access to relatively cheap and ready-to-use computing resources, providers aim to entice new, often increasingly less tech-savvy users to migrate their computational workloads to the cloud.

Given users' different degrees of familiarity, providers offer cloud computing services with varying control and administration abstraction levels. One of the oldest, but still very popular, cloud service models is *Infrastructure as a Service* (IaaS) (MORENO-VOZMEDIANO; MONTERO; LLORENTE, 2012). In the Infrastructure as a Service model, the cloud provider administers their underlying physical infrastructure (which is hidden from the user) and generally offers the resources for use in the form of a variety of *virtual machine instances* (also referred to as VMs or instances), composed of different types of computing resources like processors, memory, storage, networking, and hardware accelerators or co-processors. The variety of instances available may be significantly large given the combinations of resource capacities that can make up an instance's configuration in terms of the type and number of processing units, the amount of memory per processing unit, the capacity and speed of its storage, the networking speed, and the type and amount of hardware accelerators. Under this IaaS model, it is the user who is responsible for first choosing the most appropriate VM instance from a given list of pre-configured ones available, which offers the required combination of technologies and resource capacities for their workload or applications, and then managing the complete life cycle of the chosen VM instance (from instantiation and booting the operating system to its shutdown and termination). In addition, there is also the question of from which region (in the world) to instantiate the VM instance. Not all regions offer the same set of

instance types, and a given instance's cost may not necessarily be the same across regions.

More recent cloud service models, such as Platform as a Service (PaaS), Function as a Service (FaaS), and Software as a Service (SaaS) (LEAVITT, 2009) are becoming increasingly popular. Although these service models also simplify and decrease the IaaS decision-making burden on the end user, the burden still exists; it has just been transferred to the cloud provider. In these models, the cloud provider therefore needs to address these issues to make their infrastructure more efficient and profitable.

Computing costs in public clouds are predominantly based on a pay-as-you-go model where each instance type is priced at a predefined rate per hour (but charged per minimum billable time unit, e.g., per hour or per second). In the IaaS model, virtual machines with larger resource capacities typically have higher rates, and the total cost depends on how long the instance is in use, from the moment it is booted up until it is terminated. Cloud providers also often offer different pricing models or markets where the rate charged for the instance depends on availability guarantees or upfront payments. Most cloud providers utilize at least three market concepts: On-demand, Reserved, and Spot instance markets. The choice of which market to use typically depends on the predictability or frequency with which the user needs to execute their workloads - the more predictable or frequent, the more likely the Reserved market will be economically beneficial over the On-demand or Spot markets.

In practice, the On-demand market is still the most commonly used, despite being the most expensive. This is perhaps motivated by the fact that the user is not required to carry out additional preparations, unlike in the case of using the other two markets. In the On-demand market, the price of instances of the same type typically scales linearly with increasing capacities (e.g., a VM instance with twice the number of CPUs and twice the memory will normally be double the rate).

1.1 Motivation

The main challenge that arises from this cost model is how to choose the VM instance with enough resources to handle the user's workload without significantly overpaying for unused or underutilized resources. Often referred to as "*right-sizing*", making the correct decision can be more tricky than initially assumed. In the past, right-sizing was probably more of an educated guess based on the experience of the application developer and/or system administrator. Now, with the growing diversity of cloud services on offer, right-

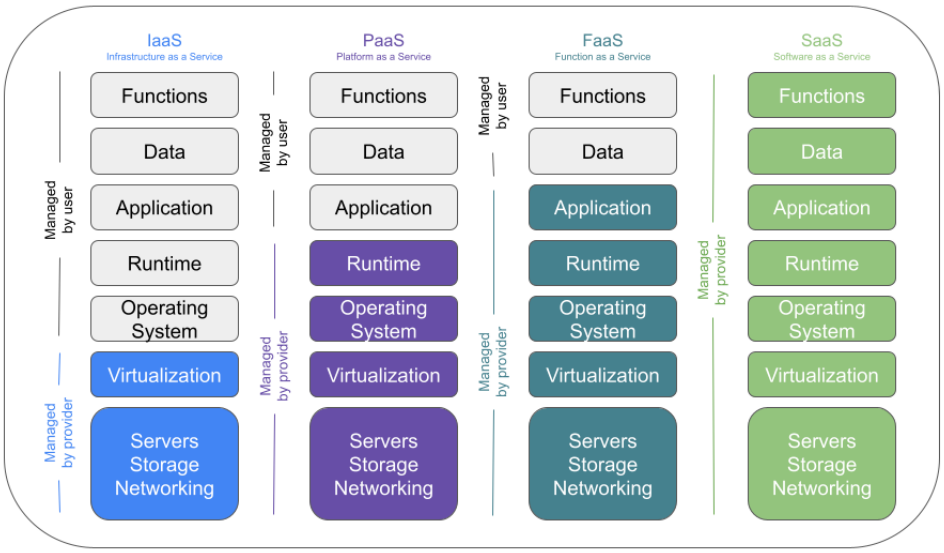


Figure 1: Distribution of the administrative burden for the different cloud computing service models.

sizing is no longer a simple task, given that there are a vast number of infrastructure and application-related options that must be taken into consideration, as well as the often conflicting objectives of maximizing the performance and minimizing the total cost of the execution.

As Figure 1 illustrates, while the adoption of other service models such as PaaS, FaaS, or SaaS will perhaps move this onus to the service provider, even then the same issues remain to be addressed: Issue (a) identifying the appropriate instance type for the user's request and; Issue (b) extracting the required or best performance from each chosen instance. Although these issues are often addressed independently, scenarios in which decisions taken regarding one issue can affect the other are common. Existing proposals to find the "best" VM instance configuration (also called *instance type*) generally focus solely on Issue (a) and on identifying the VM instance type that best fits the execution of a single instance of the target application, whose configuration has been independently predetermined. If the application needs to be instantiated more than once, these proposals may not provide the best solution to achieve the user's cost-performance objective.

The focus of the dissertation is thus different from the one typically adopted by previous approaches to identifying the best cloud resources to execute an application, such as those adopted by PARIS (YADWADKAR et al., 2017), Cherrypick (ALIPOURFARD et al., 2017), SCOUT (HSU; NAIR; MENZIES et al., 2018b), Micky (HSU; NAIR; MENZIES et al., 2018a), MOHEFT (DURILLO; PRODAN, 2014), and by the work of (BRUNETTA; BORIN, 2019) and (TAVARES; ASSIS; BORIN, 2021). All of these

approaches assume that cloud resources (the number of instances used) are not limited, and only a single instance of the application (or task of a parallel application) is allocated to a VM instance. In this scenario, these approaches do not address the role scheduling can play in improving the utilization of the chosen pre-configured VM instances.

Our approach differs by considering both of the previously mentioned issues simultaneously, providing the user with a set of Pareto optimal configurations (i.e., a set of VM instances, each with their allocated application instances, configurations, and schedules), which trade off performance and cost. The motive for providing the user or job orchestrator with options rather than a single solution is that these trade-offs can be quite subtle, while a small sacrifice in one objective can lead to a significant improvement in the other; the definition of what constitutes small and significant changes varies with each user and their objectives. Furthermore, this work assumes that applications may not be *rigid* and instead may be *moldable* (FEITELSON; RUDOLPH, 1996). As such, it can be configured to execute with different quantities of resources (most commonly varying numbers of CPUs, or amounts of memory), with more resources possibly leading to shorter execution times. This highlights an interesting four-way relationship between application and workload performance, the importance of scheduling not only in terms of performance, but also of resource utilization, and the cost and choice of VM instance to use.

1.2 Contributions

The main contribution of this work is the proposal of a methodology to help public cloud users (or system management tools such as workflow orchestrators or job schedulers) choose an appropriate set of instances to execute their workload while respecting the user's quality of service objective, be it to maximize performance or minimize their costs or both. Named PORSCHE (*Pareto Optimal Resource Selection sCHEme*), the methodology can be applied to any of the cloud cost markets (On-demand, Reserved, or Spot instances) and cloud service model (in addition to IaaS, container- and function-as-a-service, CaaS, and FaaS) thus helping users or service providers, respectively, to improve the utilization of the underlying VM instances.

As will be shown in Chapter 4, PORSCHE is capable of finding configurations that satisfy the classic objectives of minimizing the time or total cost of the workload execution, as well as finding configurations that represent interesting compromises between the total execution time and the total cost. For example, to find the Pareto optimal configurations

for one of our case study workloads considering just a single Instance Class with eight sizes (the AWS EC2 family C5), PORSCHE evaluated a total of almost fourteen thousand configurations and identified approximately thirty Pareto optimal configurations in sixty seconds.

The importance of generating the Pareto optimal solution instead of only a single configuration that optimizes for one objective becomes clear, for example, when comparing two Pareto optimal configurations suggested by our PORSCHE model for the execution of our chosen case study application within the same AWS EC2 instance class (the C5 instance family). For example, we found that while one of the optimal configurations was 6.6% more expensive than the other, it was 39 times faster. Although perhaps surprising, not every user may always be willing to pay for the additional cost. In fact, a user might only be willing to pay for increased performance in some specific circumstances and not in others. Thus, providing users with these options may be crucial to help them make more appropriate decisions. In contrast, note that the faster configuration may not even be presented to the user by existing methodologies that only optimize a single objective, such as minimizing the total cost. Comparisons of other Pareto configurations executed considering multiple AWS EC2 instance families resulted in similar trade-offs.

1.3 Organization

The remainder of this work is organized as follows. Chapter 2 discusses the problem and related work on finding cloud configurations and respective task schedules that optimize the execution of the application workload according to the user's chosen objectives. Chapter 3 presents the proposed PORSCHE methodology, while Chapter 4 describes the experimental analysis performed to evaluate the model that supports the methodology. Section 5 draws some conclusions and closes with a discussion of future directions.

2 Background and related work

This chapter discusses the problem of finding the cloud configuration and respective task schedule that optimizes the execution of the application according to the user's chosen objectives. Most of the related work found in the literature focuses on presenting the user with a single configuration. Our work argues that this may not be particularly useful given the difficulty that the user may have in defining those objectives precisely. Section 2.1 presents the paradox of choice that users face when having to execute workloads in the cloud. Section 2.2 provides an overview of the relevant research that has been conducted in the field and compares this with our proposed work. The remaining sections of this chapter focus on different aspects of our case study. Taking AWS EC2 as a use case and our chosen cloud provider, Section 2.3 briefly describes how their cloud infrastructure is architected and presented to the user. Next, Section 2.4 outlines the multi-objective optimization problem that is the basis of the proposed methodology. Finally, Section 2.5 identifies the scientific case study adopted in this work and provides a description of the bioinformatics problem and the application used.

2.1 The paradox of choice

Before running their workload in the cloud, users must make a series of decisions. Some decisions are technical, others economic, and some are more difficult than others. Although there are numerous cloud service providers today, according to Statista (STATISTA, 2023), three companies dominate 65% of the market: Amazon Web Services (32%), Microsoft Azure (23%), and Google Cloud (10%). Therefore, the first decision a user must make is with regard to the choice of service provider. Although providers tend to offer roughly the same set of services, there may be differences.

For various reasons, including reducing communication latency and economic costs, and government regulations, cloud service providers typically have multiple data centers worldwide. Having chosen their cloud provider, users must then choose where (in the

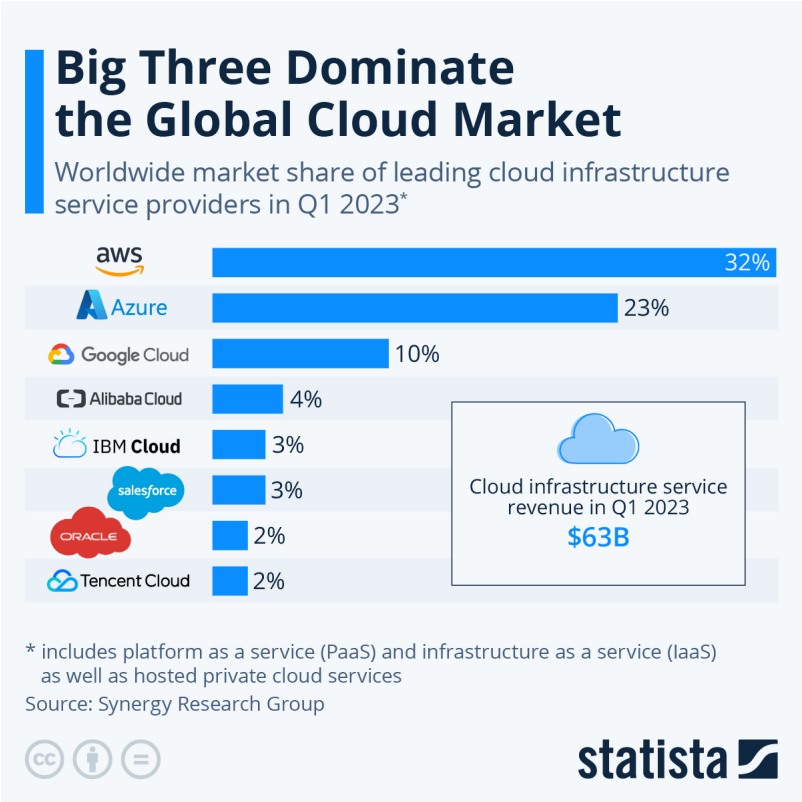


Figure 2: Cloud market (as of Q1 2023). Source: statista.com

world) to execute their workload. Modern cloud providers generally group their data centers, based on geographical proximity, into *Regions*. Users may choose the one closest to them for convenience, but the same resources may have different costs depending on the chosen region. Each data center in a region is sometimes called an *Availability Zone (AZ)* (or just *Zone*, for some providers). Providers tend to have multiple AZs per region to increase the capacity of that region and provide higher availability, reducing the risk of outage of an entire region. It is not uncommon for providers to have multiple copies of their services on each of the AZs in a region, and they encourage their users to do the same.

Although the choice of the region may appear straightforward, it does have implications for the execution of the workload. First, not all regions are created equally: not all services may be available in every region, or even when available, they may not have the same set of options, and different regions may charge differently for the desired service. Also, although the AZs and regions are usually interconnected through high-speed network connections, some classes of applications are highly data-locality sensitive. In this case, users may decide to execute their workload in the region closest to where the data are stored. Alternatively, they could opt to move their data to the preferred region for

the execution of the workload. Still, depending on the volume of data involved, this may incur additional data-movement costs and delays in the execution. Additionally, corporate users or users from publicly funded organizations may be obliged to choose a region within their country to meet legal or regulatory obligations.

The user then must choose the appropriate computing infrastructure for their workloads. While cloud providers often offer multiple computing services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Function as a Service (FaaS), and Software as a Service (SaaS), the underlying computing resources are typically preconfigured by the cloud provider. In the IaaS model, to meet the various demands of users, cloud providers offer an extensive range of VM instances, each with different predetermined capacities that can vary in terms of a variety of characteristics, such as CPU processor architectures from different manufacturers (e.g., Intel, AMD, and ARM), the amount of main memory available, the amount and location of storage, network bandwidth, and different types of hardware accelerators such as GPUs and FPGAs. Some of these characteristics are coupled; for example, VM instances with higher numbers of CPUs typically have higher memory capacities. The increase is usually linear, but the ratio depends on the VM instance type.

In the IaaS model, it is left to the user to identify the option that best meets their specific objective. Although the burden is transferred to the provider in other models, it still exists. Due to the pay-as-you-go pricing model of the cloud market, the consequences of choosing the wrong set of resources will be longer execution times or poorer execution efficiencies, and additional unnecessary financial costs. When a cloud provider decides to offer a service that will potentially be used by millions of users instead of offering the computing resources to execute it, efficiency is even more paramount due to scale. Several approaches have attempted to address this problem, some of which are discussed in the following section.

2.2 Related Work

Cloud providers have been motivated to offer an even more diverse range of VM instances, in part, by research that has shown that there is no universal cloud VM type that is an optimal fit for all workloads (ALIPOURFARD et al., 2017), (YADWADKAR et al., 2017), (HSU; NAIR; FREEH et al., 2018). Therefore, with the rapid adoption of cloud computing and its growing energetic demands, identifying the best VM configurations,

even across multiple cloud providers, to efficiently complete a given workload has become one of the most fundamental problems in cloud resource management.

Research on this problem has typically focused on two aspects, the accurate estimation of performance on a given cloud configuration and the cost of finding the ideal configuration. In the case of the latter, solutions might be effectively required to explore the entire search space to characterize the performance of a workload on all of the different cloud configurations. However, this may be prohibitively expensive or impracticable. To reduce the cost of the search, proposals employ techniques such as Bayesian optimization to reduce the number of evaluations necessary to find an efficient solution. Techniques for addressing the former aspect can be broadly divided into two approaches: prediction and sequential model-based optimization. Prediction techniques aim to build an accurate model to estimate, for example, the execution times or running costs of workloads, and thus determine the best architectural configuration for a given workload. However, the current consensus is that this method has two drawbacks. First, the accuracy of the model requires the collection of a significant amount of workload performance data. Second, performance in cloud environments is susceptible to a certain degree of variability due to multi-tenant hosting, where physical resources may be shared with workloads of other users. This means that performance data collected after benchmarking the workload on an instance type might not reflect the subsequent performance obtained by a later execution. A more recent trend has seen the wide adoption of machine learning techniques to build these prediction models.

To the best of our knowledge, we could not find any work in the literature that proposed a solution to the problem of finding the right instance configuration for the execution of BoT (or Bag-of-Task-like) parallel workloads in the cloud while also focusing on improving the efficiency of the execution within the chosen VM instance while attending to differing user's QoS requirements.

The approach most similar to ours is that of (DURILLO; PRODAN, 2014) where the authors propose the *Multi-Objective Heteregoneous Earliest Finish Time (MOHEFT)* algorithm and its cloud-aware version. MOHEFT is a multi-objective optimized version of the classic scheduling algorithm *HEFT* (TOPCUOGLU; HARIRI; WU, 2002). The original HEFT algorithm minimizes the total time to execute each task by iteratively mapping the workflow tasks onto the resources that present the earliest finishing time for that task (i.e., it is a mono-objective algorithm). In contrast, MOHEFT works by modifying the original HEFT algorithm to calculate the cost of using a chosen resource and

allows the generation of multiple trade-off solutions instead of building a single schedule. To generate its Pareto solution, MOHEFT, like our proposed PORSCHE methodology, uses the dominance concept in which solution A is said to *dominate* solution B when both its execution time and resource cost are smaller than the execution time and cost of B, respectively. Alternatively, solutions A and B are referred to as *non-dominated* in relation to each other when one has a shorter execution time at the same time as the other has a smaller cost, i.e., neither dominates the other. The Pareto set of solutions presented to the user is thus composed only of non-dominated solutions.

Our proposed methodology shares some concepts with MOHEFT like, for example, the generation of multiple trade-off solutions and the use of elimination by dominance to build the Pareto solution set. However, MOHEFT only generates schedules that allocate a single task per cloud instance. The authors either assume that sufficient quantities of cloud resources are always available and/or, as is typical in DAG scheduling, do not consider the concurrent allocation and execution of multiple tasks to the same resource, as this might delay a task's finishing time. Therefore, their work does not try to maximize the utilization of each cloud VM instance. As such, it is impossible to claim that MOHEFT generates optimal solutions in its Pareto set because there may be more efficient schedules (especially in terms of cost) that MOHEFT does not generate. Our methodology suggests the most adequate VM instance configurations to execute the entire workload according to the user's QoS objective by first generating custom schedules to optimize the execution of the tasks allocated to each required cloud instance. Our approach allows users to analyze the trade-offs between performance and cost according to their needs at that particular moment, while also considering additional restrictions, for example, the limit on the number of cloud resources that users can instantiate, a very common security policy applied by cloud providers to protect users and cloud services.

The PARIS approach proposed in (YADWADKAR et al., 2017) also aims to help users choose the best VM instance type for their workload by using a machine learning algorithm (Random Forest) to predict the workload performance according to a user-defined metric on all of the VM types of interest to the user. Their proposed algorithm has two phases: an offline phase, where benchmark applications of three different classes of workload are executed on all VM instance types available to build a performance profile for each VM type that will be used to train the machine learning model; and an online phase, where a pilot program that the user chooses as being representative of their actual workload is executed in two "reference" VM types. The machine learning model is then used with the metrics collected from the execution of the user workload on the reference VMs to predict

the performance of the workload on each of the VM types available, presenting to the user the VMs with average performance and those in the 90th percentile according to the metric defined by the user.

Although the proposed PARIS algorithm uses data collected by executing the pilot task provided by the user, if the applications used in the training offline phase are not sufficiently representative of the user's actual application, the model will be biased and may not offer the best configuration for the specific workload of its user. Since our PORSCHE methodology uses the user's application in an offline profiling phase, it is more likely that the offered configurations are more suitable for the users' workload. Also, PARIS only considers a single objective (user metric), and thus presents just one solution to the user. Our methodology considers multiple objectives when generating the Pareto optimal solution set for the user, a set that contains the configuration that minimizes the execution time, the configuration that minimizes the total execution cost and maximizes the throughput, and a selection of compromise configurations in between.

SCOUT, proposed in (HSU; NAIR; MENZIES et al., 2018b), adopts a machine learning technique called *transfer learning* that uses data collected from the execution of other workloads to train a model to predict the workload performance in different VM types. It uses Sequential Model-Based Optimization (SMBO), a technique that decides which configurations to investigate (HUTTER; HOOS; LEYTON-BROWN, 2011), reducing the cost of the search process to find the appropriate cloud configurations. The authors claim that SCOUT optimizes the search by limiting the search space to regions considered the most likely to have the best possible solution. The problem with that approach is that it is unclear which workload is actually used in the transfer learning phase, and its relationship to the user's workload may seriously decrease the model's accuracy. Another important aspect is that, even though updating the model may generate more accurate predictions, it does not guarantee the optimality of the solution generated by the current model, or at least not until iterations are made to the original model (which is based on previous workloads). Contrary to what is proposed in SCOUT, our methodology guarantees the optimality of the configurations in the Pareto solution.

2.3 AWS

AWS is the oldest and most well-established (STATISTA, 2023) cloud service provider on the market, offering an extensive range of services with a portfolio of over 200 services and

99 Availability Zones within 31 geographical regions around the world. Amazon *Elastic Compute Cloud* (EC2) (AMAZON AWS, 2022) is a cloud Infrastructure as a Service (IaaS) that allows users to create virtual machine instances on remote servers located in one of AWS's regions around the world. The cost or rate to hire each type of service is generally charged per second, but may differ between regions.

The North Virginia region (also known as us-east-1) was the first and is still one of AWS's most used regions. This region offers the widest variety of virtual machines and is generally the first to release the latest generations of Instance Types at a lower cost compared to other AWS regions. Regarding virtual machines (VMs), there are currently 665 Instance Types divided into 77 VM families in this region[1], each of which belongs to one of five categories (AMAZON AWS, 2021): General Purpose (with 18 families), whose instances provide a balance of computing, memory, and networking resources; Compute Optimized (11 families), designed for compute-intensive applications that benefit from high-performance processors; Memory Optimized (19 families) instances are intended for workloads that process large data sets in memory and, therefore, have a larger memory per processing unit ratio; Accelerated Computing (15 families), which uses hardware accelerators or co-processors, such as GPUs and FPGAs; Storage Optimized (10 families) designed for workloads that require high bandwidth disk access to massive data sets on local storage; and HPC Optimized (4 families) designed for memory-bound and data-intensive high-performance computing (HPC) workloads.

In this work, we refer to the Instance Class as a group of instance types from the same instance type category and same generation. Each Instance Class is identified by two sets of characters, composed of letters and numbers, separated by a dot, as in "c7g.large". The first letters represent the family of the instance; for example, "c" in "c7g.large" or the "hpc" in "hpc6a.48xlarge" represents the compute-optimized and HPC-optimized families, respectively. The number after the first letters is the generation of this family; for example, "c5.large" is the 5th generation of the compute-optimized family. The letters that follow the number and appear before the dot, if they occur, refer to secondary characteristics of the instances in the class. For example, instances with an "n" after the number have a faster network adapter, such as c5n.large, which is the 5th generation of the compute-optimized instances that also has faster network adaptors, or the c7g.large, which has a specific processor architecture - the custom Amazon's ARM processor Graviton, represented by the letter "g".

[1] As of August of 2023

The term following the dot indirectly identifies a given instance's size (or capacity) in terms of the CPU and its memory capacity, regardless of the family. The terms used are *medium, large, xlarge, Nxlarge*, where N is a number representing a multiplier over the number of processing units in the *xlarge* instance of the same instance class, and *metal*, in increasing order of the number of physical processing units available to the VM. Except for the term 'metal', which represents the entire underlying server, from medium to the Nth xlarge, each term represents double the number of physical processing units available in the VM. The amount of memory scales linearly with the number of CPUs, although the ratio of memory capacity to CPU can vary between different Instance Classes. Regarding the cost, each Instance Class has a predefined base rental cost per CPU per hour and scales linearly with the instance size, i.e., the double of resources represents double the hourly cost.

2.4 Multi-objective optimization

Multi-objective optimization, also known as Pareto optimization, is an area of decision-making that deals with optimization problems that involve more than one objective function. According to (MARLER; ARORA, 2004), the definition of Pareto optimality is as follows.

> A point $x^* \in X$, is Pareto optimal if there does not exist another solution, $x \in X$, such that $F(x) \leq F(x^*)$, and $Fi(x) < Fi(x^*)$ for at least one function.

Similarly, Pareto efficiency or optimality in task scheduling is finding the task schedule for which no other schedule or allocation is better or more efficient.

To find all Pareto optimal configurations - which, in the proposed PORSCHE model, includes custom task scheduling for the execution of a given workload on a given instance class, our proposed PORSCHE model applies the concept of dominance, also defined by (MARLER; ARORA, 2004) as follows.

> A vector of objective functions, $F(x^*) \in Z$, is non-dominated if there does not exist another vector, $F(x) \in Z$, such that$F(x) \leq F(x^*)$ with at least one $Fi(x) < Fi(x^*)$. Otherwise, $F(x^*)$ is dominated.

Therefore, our proposed model finds all optimal (or non-dominated) configurations in the search space to provide users with a Pareto solution set with only optimal configurations. It is worth noting that the search space comprises all possible configurations using

the different instance sizes in each of the instance classes of interest. Chapter 3 explains in detail how the configurations in the search space are determined and the generation of the Pareto solution set presented to the user.

2.5 A Bioinformatics Case Study

The proposed methodology is designed to be used in a broad range of scientific environments that require multiple executions of a given application, for example, often seen with SaaS and scientific portals. With more and more scientists relying on cloud resources instead of on-premise, both execution time and cost limitations have turned realizing experiments into a multi-objective cloud resource optimization problem. To exemplify the benefits of the proposal, with such bag-of-tasks (BoT) workloads, a bioinformatics case study has been adopted here: the pairwise alignment of biological sequences, using a tool called MASA-OpenMP (DE O. SANDERS et al., 2016). Sequence alignment is a key step in addressing various problems in bioinformatics, where pairs of sequences of DNA, RNA, or protein are compared to identify regions of differences that result from functional, structural, or evolutionary changes between sequences (ALURU, 2005).

The class of alignment algorithm used depends on the type of alignment required and what information one wants to know about the sequence. For example, global alignment algorithms are used when one wants to compare the whole sequence. On the contrary, local alignment algorithms are used when the scientist is interested in comparing only part of the sequence. Moreover, these algorithms can produce the optimal alignment or be a heuristic that, while it may execute comparatively faster, does not guarantee an optimal solution. Pairwise sequence alignment algorithms are essential to identify similarities or differences between two sequences. For example, during the recent pandemic, these algorithms were heavily used to help identify virus mutations and variants of the SARS-CoV-2 virus.

Our motivating case study is a bioinformatics cloud service that has gained significant importance recently, in part, due to the recent COVID-19 pandemic. Mutations allow viruses to change and evolve, and when these changes are significantly different from the original virus, they are known as *variants*. Performing DNA sequence comparisons to find, for example, new SARS-CoV-2 variants or variants of other viruses is essential for understanding the infection each virus can cause, how easily that virus spreads, the severity of associated symptoms, and the effectiveness of respective vaccines, among other

aspects.

To identify variants, scientists map the genetic material of viruses (known as DNA sequencing) and look for differences between sequences to identify regions that may have changed. Authorities are interested in identifying *variants of concern* that spread more easily within a population, cause more severe disease, escape the immune response of the body, change its clinical presentation, or decrease the effectiveness of protocols – such as public health measures, diagnostics, treatments, and vaccines. The recent pandemic has only highlighted the importance of this type of analysis, as exemplified by the more than 20 million SARS-CoV-2 DNA sequences obtained from human infections around the world, which have rapidly been made available to scientists in public genome databases such as GenBank (https://www.ncbi.nlm.nih.gov/genbank/) of the National Center for Biotechnology Information (NCBI) and GISAID (https://www.gisaid.org/).

There are already a few online sequence alignment services, such as Clustal Omega[2] and VectorBuilder[3]. Still, both, like most of the others, have severe limitations, such as the size of the sequences in the input, making it impossible to execute more extensive sequences. Another serious problem is that most services do not comply with any user Quality of Service (QoS) requirement or provide any information on the execution, such as the estimated execution time (which may vary depending on the number and size of the sequences). Our approach aims to enable the efficient execution of DNA Sequence Alignment (our study case) on the cloud, not only for end-users but also to enable cloud service providers to provide an efficient sequence alignment service, respecting users' QoS requirements.

Multi-Platform Architecture for Sequence Aligners (MASA) is a tool to carry out pairwise biological sequence alignments on various hardware/software platforms. It has been shown to support the alignment of massive DNA sequences with more than 200 million base pairs (DE O. SANDERS et al., 2016). This work focuses on using MASA-OpenMP, the freely available multithreaded OpenMP implementation[4] of MASA, designed to harness the computing power of multiple CPU cores on a server, but further investigation with other computing resources supported by the MASA architecture is listed in future work. MASA-OpenMP makes a pairwise comparison of two DNA sequences using the Myers and Miller (MYERS; MILLER, 1988) variation of the Smith-Waterman algorithm (SMITH; WATERMAN, 1981) to find the optimal alignment and the degree of similarity between

[2]https://www.ebi.ac.uk/Tools/msa/clustalo/
[3]https://en.vectorbuilder.com/tool/sequence-alignment.html
[4]Source code available at https://github.com/edanssandes/MASA-OpenMP.git

the pair of chosen sequences.

Each sequence in the alignment is represented by a text-based format in which nucleotides or amino acids (or lack of both) are represented using single-letter codes, in a format known as FASTA[5]. The "size" of a sequence is measured in terms of how many nucleotides or amino acids there are in the DNA sequence of an organism. This may vary from a few thousand (in the case of simple organisms such as viruses) to billions of characters (in the case of human or chimpanzee DNA sequences).

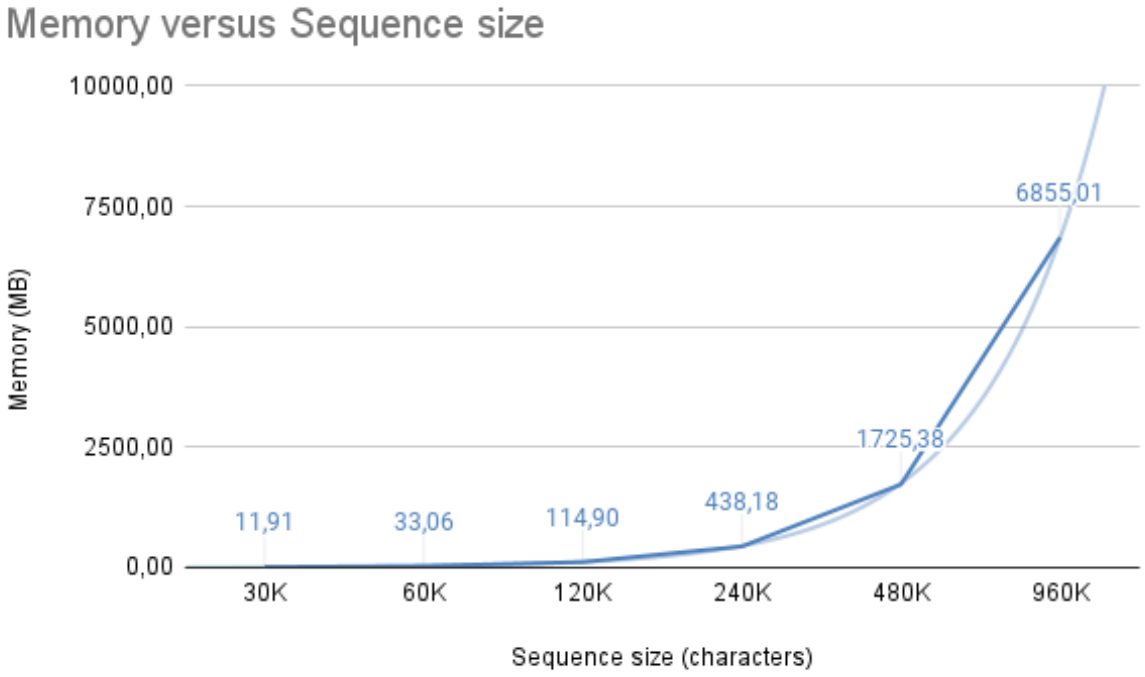


Figure 3: MASA-OpenMP's memory consumption (in MegaBytes) versus sequence length (in thousands (K) of characters)

When executed on a EC2 c5.24xlarge VM instance with sufficient memory capacity, the highly optimized MASA-OpenMP has a quadratic (or $O(n^2)$) complexity in relation to the sequence size (or more precisely the product of the lengths of the two sequences (MILLS et al., 2023)), in terms of both memory and (serial) execution time, as can be seen in Figures 3 and 4, respectively. A typical experiment, or MASA workload, is made up of multiple independent pairwise alignment tasks. Such experiments, composed of independent tasks, are thus known as Bag-of-Tasks (BoT) workloads. Due to these characteristics, a job scheduler or workflow manager has the burden of determining the optimal degree of concurrency and the degree of parallelism of each task, given the

[5]https://en.wikipedia.org/wiki/FASTA_format

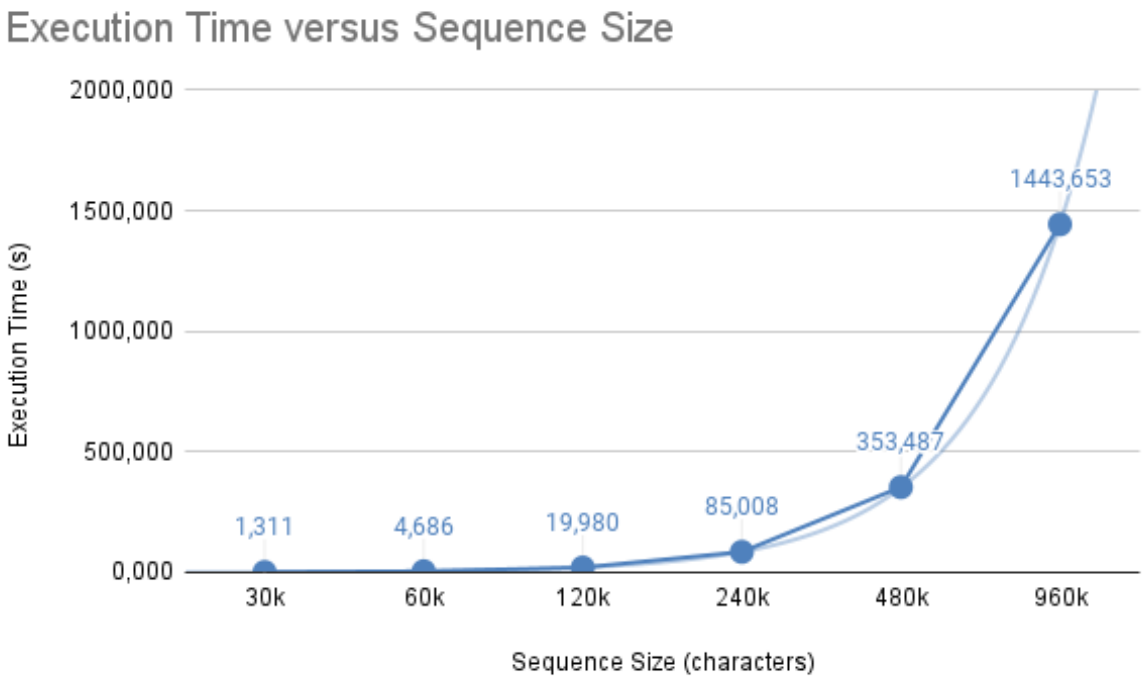


Figure 4: MASA-OpenMP's execution time (in seconds) versus sequence length (in thousands (K) of characters)

resources available, so that the workflow execution meets the user's QoS requirements. Unlike existing tools to find cloud configurations that only match resources to the workload, by being able to adjust the degree of parallelism of tasks, our approach is able to better match the workload to the resource type available. The methodology heavily explores this malleable characteristic to optimize the task schedule considering both degrees of parallelism and concurrency, on each instance, to offer a better opportunity of finding configurations that best satisfy the user's QoS requirements.

3 A Pareto Optimal Resource Selection Scheme

This chapter describes the design of our proposed methodology PORSCHE (*Pareto Optimal Resource Selection sCHEme*), which aims to provide the job manager of either the user or the service provider with a selection of, rather than just one, cloud instance configuration options for a given workload. This research focuses on workloads that consist of the execution of a set of independent, possibly parallel, application tasks. The different configuration options are Pareto optimal trade-offs between execution time and cloud cost and take into consideration user-defined limitations in terms of cloud instance types and the number of concurrent instances. This chapter presents the different stages of the PORSCHE methodology, the motivation and applicability, and the software developed to create the application-specific execution models used to determine good configurations.

3.1 PORSCHE - "There is No Substitute"

Cloud-based services are often offered to users at no cost or at no additional cost (other than for the computational resources they use). Therefore, such services must be implemented as efficiently as possible to be both viable and sustainable. This section presents the foundations of a methodology to identify "ideal" VM instance-and-schedule configurations for resource- and cost-efficient executions of workloads composed of parameter sweep executions of a given application. With such workloads becoming the workhorse of large-scale scientific experiments and simulation-based investigations, cloud providers are seeking to entice these scientists to use their ready-made cloud environments, e.g., ElasticBLAST for genomic sequence alignments on AWS or GCP (CAMACHO et al., 2023).

As explained in the previous chapter, unlike previous approaches that generally only identify a single cloud configuration to satisfy the user's requirements, our PORSCHE framework (the title of this section comes from the slogan that the Porsche car manufac-

turer used in the 1970s.) presents the user with multiple Pareto optimal cloud configurations in terms of cost, execution time, and the number of instances. Since optimizing various objectives can be difficult and somewhat subjective, the aim is to help the user visualize the trade-offs between objectives and empower them to make more informed decisions about the cloud configuration that best fits their current needs. This is especially useful when the objectives themselves are not clear to the user, are imprecisely defined, or are in fact flexible since, for example, the user might see that a less obvious configuration may adhere better to their objective at perhaps the cost of a minor relaxation of one of their restrictions.

Our methodology not only identifies configurations that meet the classic objectives of obtaining the fastest execution time or having the cheapest cost but also different configurations between these extremes that trade-off cost and performance to different degrees. For instance, it may be possible to find an alternative configuration that reduces the total execution time for a slight increase in cost, or vice versa. Furthermore, a given user may be subject to specific restrictions, for example, the total number of instances they can use simultaneously. The usual practice in such cases might be to say that not all of the configurations found may be feasible and simply remove those that do not meet the restriction. However, such an approach does not identify all Pareto optimal solutions. The restriction must be incorporated at the beginning of the process. Each configuration also has its own custom task schedule to execute the workload allocated to the VM instance as efficiently as possible. These configurations would not be present in a solution that presents the user with a single configuration that focuses only on meeting one objective or constraint.

PORSCHE derives an application-specific performance-cost model, to quickly evaluate cloud configurations, from a data collection phase that characterizes the application and the instance types available to the user, while identifying an appropriate workload execution plan for the execution of the experiment for each instance type and size considered. As illustrated in Figure 5, the PORSCHE methodology is comprised of three stages or phases: a one-off *Application Characterization Phase*, a one-off pre-processing (or offline) phase that builds *Instance Class Profiles* for the chosen application, and; an online phase that identifies viable *Service Request Configurations* for each submitted workload. A filtered subset with only the most efficient configurations generated in the online phase is presented to the user, so they may decide which best meets their QoS requirements.

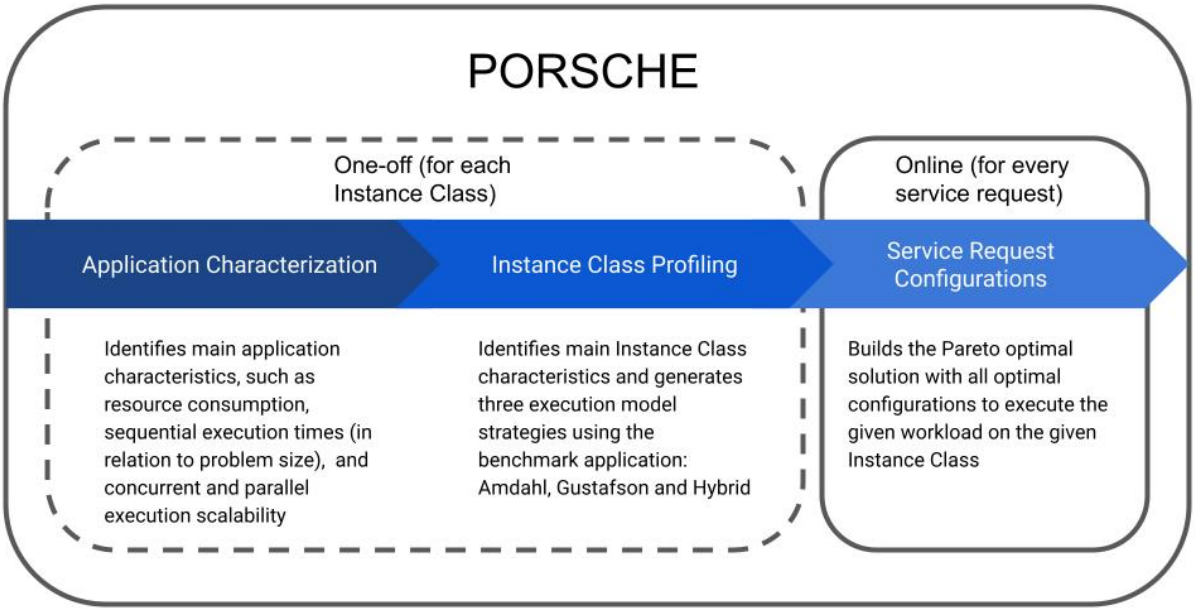


Figure 5: PORSCHE's three-stage approach

3.2 Characterizing the application

Identifying time- and cost-efficient configurations in the online phase for a given experiment requires first determining an ideal workload scheduling strategy for the tasks to be allocated to each VM instance. Clearly, the resulting schedule depends on the optimization criterion used (e.g., minimizing time, cost, or both), which is only defined by the user at the end of the process. Therefore, the strategy must identify schedules that optimize different criteria. This depends not only on the number of tasks in the workload, but also on the degree of parallelization of each task since the application is considered to be moldable.

This first characterization phase thus aims to understand the application's behavior in terms of resource consumption, the sequential execution times in relation to problem size (or different parameter values), and the scalability of concurrent and parallel executions. The methodology achieves this characterization by executing the application under three different scenarios: (1) While using the largest VM size for a given instance type (so that the application will not be restricted due to a lack of resources and will not suffer interference from co-hosted applications of other users), execute the application a few times each time with different inputs to measure the sequential execution times and identify changing resource demands during their executions, within the domain of possible problem sizes. A benchmark instance of the application is chosen from one of these

executions as a representative of the application; (2) The chosen benchmark instance is executed in parallel with an increasing number of threads to understand the parallel efficiency and scalability of the application, and; (3) Multiple instances of this benchmark are executed concurrently to analyze this form of scalability. Scenario (1) defines the reference execution with which the efficiencies of the two parallelization scenarios (2) and (3) will be compared. While Scenario (2), the Amdahl strategy, aims to identify faster executions, and Scenario (3), the Gustafson strategy, looks for higher throughputs, combining both approaches (the hybrid strategy) might improve the utilization of the instance being used.

This application characterization phase also helps eliminate instance type/size configurations, which cannot meet the application's resource requirements, from the search space prior to the online phase. For example, suppose that a single instance of the application needs a minimum amount of 10 GiB of local RAM memory to execute with a given problem size, then all instance sizes within an Instance Class that do not comply with this need not be considered. The limited resource capacities of the VM instances will also influence the type of scheduling strategy to be used for the portion of the workload allocated to each instance. For example, since cloud providers typically maintain a fixed ratio of CPUs and RAM memory for each Instance Class, this means that, depending on the size of the problem and the characteristics of the application, the degree of parallelism or concurrency may be limited by the number of CPUs or the amount of memory available (SODRÉ; BOERES; REBELLO, 2022).

3.3 Building the Instance Class Execution Profiles

Cloud providers offer a variety of VM instance configurations in the form of Instance Classes, each might employ a different processor architecture, have different amounts of memory per core, and, of course, have different costs. These variations are likely to affect applications differently. The second phase of the methodology aims to understand how the application might perform on each of the Instance Classes available to the user by building *Instance Class Execution Profiles*. A profile is created for the Instance Class by executing the benchmark application on its largest instance size. It is important to note that when the Instance Class has more than one CPU architecture available, executing the application on each architecture available in the Instance Class may be necessary to characterize the class fully. As in the previous phase, the evaluation is carried out in the largest-sized instance of the class to lower the chance of interference with other VM

instances co-hosted on the same underlying hardware.

Each instance profile is composed of three execution models for the benchmark application. Each model predicts the execution time under three parallelization strategies: Amdahl (parallel), Gustafson (concurrent), and hybrid (a combination of the concurrent and parallel execution strategies). In the Amdahl strategy, the application instances are executed sequentially, but each is a single process with multiple threads. The Gustafson strategy assumes that multiple instances of the application are executed concurrently, each as a single thread process in the VM instance. Finally, the hybrid strategy combines the previous ones and executes multiple application processes, each with multiple threads, per VM instance.

To generate the Amdahl strategy model, a single instance of the parallel application is executed in the largest VM instance of the class with an increasing number of threads, starting with two and increasing with factors of the number of physical processing units available on every size of the instance class until the number of threads equals the number of physical processing units. Experience has shown that, even in architectures with hyper-threading (or equivalent technologies), performances rarely show improvement beyond this upper limit. For example, the C5 instance class has eight different sizes available, as shown in Table 3.3. To generate the complete Amdahl strategy model for the C5 instance class, the performance of the benchmark application with 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, and 48 threads is measured. This does not necessarily mean that the benchmark application has to be executed explicitly with each of these quantities of thread, since the performance of intermediate values can be interpolated from neighboring values.

Instance Size	Number of CPUs
large	1
xlarge	2
2xlarge	4
4xlarge	8
9xlarge	18
12xlarge	24
18xlarge	36
24xlarge	48

Table 1: C5 instance sizes and respective physical processing units available.

The collected execution times are stored as the C5 dataset that will be used in the online phase to estimate the total execution time of instance workloads when using this strategy. After assembling the dataset, the parallel scalability of the benchmark applica-

tion on the chosen Instance Class is obtained by calculating the efficiency of each parallel execution in relation to the serial execution of the same application, using Equation 3.1, where SET represents the serial execution time, PET_t is the parallel execution time with t threads, and N_{CPU} is the number of physical processing units used by the process. Note that when calculating the efficiency of this Amdahl strategy, some additional executions may be needed to identify more precisely when changes to the efficiency trends occur. These efficiencies will also be used in the online phase as one of the measures to choose an adequate strategy to generate custom schedules for configurations with the fastest execution times and for configurations with the most efficient executions.

$$Ef_{Amdahl} = ((SET/PET_t)/N_{CPU}) \tag{3.1}$$

To generate the Gustafson strategy model, the benchmark application must also be executed in the largest VM instance in the Instance Class, with increasing degrees of concurrency (also using factors of the number of physical processing units), starting from one for every physical processing unit available to the VM instance up to a variable upper limit which depends on the processor architecture, the number of physical processing units available to the VM instance, and the application's resource consumption from the application characterization phase. Again, the execution time of each execution configuration is collected and stored in a dataset that will be used in the online phase to estimate the total execution time of a given workload when using this Gustafson strategy. The concurrent scalability of the benchmark application on the chosen Instance Class is obtained by calculating the efficiency in relation to its sequential single core execution using Equation 3.2, where SET is the serial execution time, CET_p is the execution time obtained when running p application processes concurrently, and N_{CPU} is the number of physical processing units available in the VM instance.

$$Ef_{Gustafson} = ((SET/CET_p)/N_{CPU}) \tag{3.2}$$

The aim of calculating these concurrency efficiencies is to identify the best obtainable efficiency and the configurations that fall in this highest range of values (i.e., where the efficiency peaks). This efficiency range is determined by the relation between the number of concurrent processes and the number of physical processing units used with the processes. The range values will be used in the online phase as one of the measures to choose the appropriate strategy to adopt when determining the custom schedule to obtain

configurations with the highest throughput.

The hybrid strategy model looks for configurations that trade-off shorter execution times for higher throughput. Since this strategy combines the two previous ones, one might find it tempting to use the datasets already generated in parallel and concurrent executions to lower the profiling costs. The problem with this approach is that there is typically a non-negligible slowdown when executing multiple instances simultaneously in relation to executing a single one. In compensation, increasing the number of threads of a single parallel execution generally improves the time. Using the datasets generated in previous stages without considering the combined effect may lower the methodology's accuracy because the hybrid configurations may be incorrectly favored or prejudicated when generating the Pareto solutions.

Although it may be possible to predict an accurate slowdown rate, it appears difficult to extend this accuracy to other instance sizes purely through modeling. Therefore, the methodology opts to obtain the information experimentally by running the benchmark application on the largest instance with combinations of x concurrent processes with y multiple threads such that $x \times y = 2.N_p$. Since it may not be the cheapest or most efficient way to achieve this, more work on this aspect is proposed as future work.

All three datasets are used to generate equations in the online phase, employing multivariate regression to estimate missing intermediate values if necessary. While at least three data points for each strategy are necessary, more data points improve the accuracy of an equation's fit. On the other hand, obtaining more data points requires more benchmarking executions in this pre-processing phase, which incurs a higher cost to generate these strategy profiles. Note that this pre-processing phase is only executed once for each instance class, usually when a new generation of VM instances is released by the cloud provider. Such instances generally bring novel processing architectures and technologies whose impact can vary from application to application. Nevertheless, there is a tradeoff between the initial cost of obtaining more accurate models during this second pre-processing phase against the gains from better choices of instances and task schedules for the user's job.

3.4 Building the Service Request Configurations

The online phase of the proposed methodology is responsible for generating a Pareto optimal solution containing the Service Request Configurations that will be presented to

the users so that they may choose the most appropriate one, according to their own QoS requirements. The solution contains configurations that minimize the execution time, obtain high throughputs to reduce the total cost, and multiple combinations that represent compromises between execution time and cost. The solutions presented can also be designed to present configurations subject to additional constraints, such as a maximum number of instances to be employed. The inputs of this phase are the number of tasks and other relevant application-specific characteristics obtained during the application characterization phase, including the duration of each task's execution and resource requirements. The user may also include the preferred instance class, the maximum total cost, the maximum number of instances to be used, or the maximum total execution time as optional additional parameters. These additional parameters may be used to refine the generation of the Service Request Configurations before presenting the final configurations to the users in later stages of the framework.

Each Service Request Configuration (also referred to as a configuration) is composed of: the instance size in the instance class; the number of instances to be used; the estimated total execution time; and the estimated total cost of the execution. Additionally, the configuration includes a custom task schedule that identifies the number of tasks per instance, the maximum number of concurrent tasks per round, the number of threads per task, and the number of rounds, which identifies the number of application processes to be executed sequentially on that instance independently of the degrees of parallelism and concurrency.

To derive the custom task schedule according to the strategy being evaluated, an estimate of the execution time for each round is required, taking into consideration the number of concurrent tasks, the number of threads per task, and the number of processing units available. To discover the estimated execution time, a profiled execution is identified that coincides with the corresponding parameters (i.e., the number of concurrent tasks, the number of threads per task, and the number of processing units) or at least is extrapolated from the information available in the dataset for the given strategy, collected in the offline characterization phase. If an experimental value for the given parameters was not found, multivariate regression is used to generate a best-fit function to estimate the execution time based on the data points available. The profile data collected previously are used to validate the precision of the function, choosing a maximum limit of 10% error in order to accept the function.

One of the most important aspects of the methodology is that it aims to be time-

Algorithm 1 Configuration creation algorithm

1: $InstSet \leftarrow AllInstInInstClasses - InstEliminatedByPreviousSteps$
2: **for** $InstType \in InstSet$ **do**
3: $N_{CPU} \leftarrow PhysicalCPUCount(InstType)$
4: $N_{instances} \leftarrow 1$
5: **while** $N_{instances} \leq N_{tasks}$ **do**
6: $N_{taskPerInst} \leftarrow N_{tasks}$ / $N_{instances}$
7: **if** $N_{tasksPerInst} \leq (N_{CPU}$ / 2) **then**
8: $N_{threads} \leftarrow N_{tasksPerInst}/N_{CPU}$
9: **if** $N_{tasksPerInst} \geq 2$ **then**
10: $T_{exec} \leftarrow EstimateExecutionTime(N_{taskPerInst}, N_{threads}, Hybrid)$
11: **else**
12: $T_{exec} \leftarrow EstimateExecutionTime(1, N_{threads}, Amdahl)$
13: **end if**
14: $Cost \leftarrow T_{exec} * (InstanceHourlyCost(InstType)$ / 3600)
15: **else**
16: $P_{Max} \leftarrow N_{CPU} * ArchEffUpperLimit(InstType)$
17: $P_{Min} \leftarrow N_{CPU} * ArchEffLowerLimit(InstType)$
18: $Schedule \leftarrow GenSched(N_{taskPerInst}, P_{max}, P_{min})$
19: $T_{exec} \leftarrow CalcExecutionTime(Schedule)$
20: $Cost \leftarrow T_{exec} * (InstanceHourlyCost(InstType)$ / 3600)
21: **end if**
22: $N_{instances} \leftarrow N_{instances} + 1$
23: **end while**
24: **end for**

and cost-efficient. To accomplish that, the methodology tries to eliminate configurations considered "unfit" or impossible as early as possible. This strategy greatly reduces the number of calculations made while generating the configurations, reducing the cost of generating solutions. It is worth remembering that some configurations may already have been considered unfit based on the data generated in the previous phases. Therefore, the first step in this phase is to eliminate all configurations considered to be unviable using the information generated in previous phases. One criterion used in the elimination is excluding Instance Classes without datasets from previous phases. It may be possible to reuse the datasets of other Instance Classes with some adaptation, and more work on this is proposed as future work, but without the dataset for the Instance Class, the inaccuracy of the generated solution may be high. Another criterion to eliminate a configuration at this stage is when the instance in the configuration does not have enough resources to execute a single task of the workload, as identified in the application characterization phase.

Algorithm 1 describes the steps of the methodology to generate the configurations.

In line 1, one can see that the set of instances available to generate the configurations (*InstSet*) for the given Instance Class is the set of all instances in the Instance Class (*AllInstInInstClasses*) minus the set of instances considered unfit in previous steps (*InstEliminatedByPreviousSteps*). After eliminating the unfit configurations, all remaining instances in the list are evaluated using task schedules to optimize the execution, as seen in line 2 of the Algorithm 1.

Next, for each instance in the set of remaining instances, the algorithm generates configurations with different numbers of that instance type, starting from one, i.e., executing all tasks in a single instance, up to the number of instances equal to the number of tasks, i.e., executing one task per instance, as seen in line 5 of the Algorithm 1. For each configuration, the number of tasks per instance that need to be executed is obtained by dividing the number of tasks by the number of instances, as shown in line 6 of the Algorithm 1. If the calculated number of tasks per instance is smaller than half the number of physical processing units available to the instance and bigger than one, we use Hybrid's strategy model to estimate the execution time (line 10), which will be used to calculate the total cost (line 14). If it is smaller than half the number of physical processing units available to the instance and equals one, we use the Amdahl scheduling strategy (line 12) for the same purpose.

When calculating the number of threads per task to increase the overall efficiency of the configuration, we always consider using all available processing units and respecting the parallel efficiency scalability limits found in the application characterization phase (i.e., not to use more threads per task than the predetermined maximum number of threads per task for the benchmark application's input after which there is a decrease in the performance).

On the other hand, if the calculated number of tasks per instance is larger than half the number of physical processing units available to the instance, we use Gustafson's scheduling strategy, as seen in lines 16 to 20 of the Algorithm 1. When using the Gustafson strategy model, we generate a custom schedule for the execution of the tasks in the instance, considering both the lower- and upper-efficiency bounds found in the instance characterization phase (lines 16 to 18). The function that generates the schedule tries to distribute the tasks so that there are always more processes in execution concurrently than the lower efficiency bound and less than the upper efficiency bound, guaranteeing that the execution is always in the best efficiency range. The algorithm can then calculate the total execution time based on the generated schedule (line 19) and multiply it by the

instance cost per second to find the total execution cost (line 20).

3.5 Generating the Pareto solution

After all of the configurations in the search space have been generated, PORSCHE analyses each of them to identify those configurations that are optimal, including the configurations that attend to the classical objectives, that is, the two that minimize the execution time and minimize the total cost, respectively, along with solutions that trade cost for performance. It is important to note that of all the solutions generated, the intermediary (or trade-off) solutions are the most interesting because they represent neither the lowest cost nor the shortest execution time but a compromise, while also considering the number of instances in the solution, which may or may not be an additional restriction. To the best of our knowledge, most of the other proposed solutions to this problem found in the literature try to maximize only one of these objectives, rarely both, and none offer multiple trade-off solutions focused on efficiency. Our methodology offers configurations that identify both classic objectives and several compromise solutions in between that the user may not otherwise be aware of.

The search for Pareto optimal solutions involves evaluating all generated configurations and eliminating the non-dominant configurations to assemble the solution set. This is achieved by comparing all of the configurations generated against each other and eliminating those for which there exists another configuration with the same or lower cost but a shorter execution time (i.e., dominated in terms of the execution time) or a configuration that has the same or lower execution time but a smaller cost (i.e., dominated on cost). A final step involves removing instances of dominant configurations with equivalent execution times and total costs from the optimal solution set. When equivalent configurations are detected, a secondary objective, chosen by the user, may used to give preference to a certain type of solution. For example, a secondary objective may be to opt for the configuration that requires the fewest VM instances (the current default objective) in order to improve the likelihood that sufficient resources will be available in the Cloud to meet the request.

It is also worth noting that the limits or conditions used by the model to consider a configuration dominant or equivalent configuration are not exact. It is necessary to allow for a certain degree of "fuzziness", for example, configurations with a difference in cost of less than one cent of a dollar (the smallest billable unit) are in fact considered equal. The

same applies to configurations with differences in execution time of less than 10%. In this case, this is to allow for the inherent variation in the performance of cloud executions, which in turn may be caused by a number of factors that, for example, influence the state of operation of the underlying hardware on which the VM is allocated, causing distortions in some executions.

To close this chapter, an interesting observation that can be made concerning the characteristics of the configurations generated and those that make up the Pareto solution set is that each configuration generally falls into one of five groups related to the overall efficiency of the workload execution. The first group is comprised of configurations that only use the parallel Amdahl strategy, where only a single task is executed per VM instance, using multiple threads to harness the processing capacity available, and thus, there are no concurrent tasks in execution within an instance. These are the configurations that usually yield the fastest execution times and are the most expensive found in the Pareto solution set. The number of tasks assigned to the instance can vary between 1 and $\frac{N}{2}$, where N is the number of CPUs of the instance. The second and third groups are each comprised of configurations where the number of tasks per instance is less than the number of physical processing units available to the VM instance, i.e., less than N. In the second group, the configurations execute parallel tasks concurrently under a hybrid scheduling strategy. The degree of parallelism is smaller than that of the first group, but often the execution is more efficient. In contrast, the third group reduces the degree of parallelism to one, i.e., the tasks have no parallelism (the Gustafson strategy). Since the number of tasks is smaller than the number of CPUs, the VM instance is underutilized, which tends to harm the general efficiency of the solution. Thus, configurations of this third group generally do not make the Pareto solution set. The fourth and fifth groups are comprised of configurations where the number of tasks per VM is equivalent to or greater than the number of physical processing units (N) and use the Gustafson scheduling strategy. In the fourth group, the number of tasks that can be executed concurrently is insufficient for the instance to execute at its most efficient rate, the range identified during the offline profiling phase. However, the fifth group has configurations with a sufficient number of allocated tasks to allow the execution to occur in the most efficient range, maximizing the use of the VM's resources. This is usually where the most cost-efficient configurations are found, and if they exist, normally dominate configurations in the fourth group.

4 A Bioinformatics Case Study

This chapter discusses the experiments performed to evaluate and highlight the advantages, benefits, and possible caveats of using our methodology to aid in choosing the ideal cloud configuration for a realistic scientific case study: A DNA Sequence Alignment Service using MASA-OpenMP. *Amazon Web Services* (AWS) was used as the cloud provider for the evaluation, mainly because it is the oldest and most widely used (i.e., the largest in terms of market share) public cloud provider. The computational resources used in the experimentation were AWS's EC2 virtual machine instances hosted in the AWS region of North Virginia (us-east-1). This is the region that generally has the lowest hourly cost per virtual machine instance, and is the one where new AWS instances are usually launched first[1].

Two example workloads were used in the evaluation; one to generate the instance profiles and another to simulate a realistic user workload, generating a solution with the configurations. The workload used to generate the instance profiles, denoted here as SP_{align}, was composed of 200 variants of SARS-CoV-2 DNA sequences, chosen at random from the NCBI database, which were each aligned against the established reference genome of SARS-CoV-2 (identified as s_{nc_045512}). The workload used to validate the model, denoted as SU_{align}, was composed of the alignment of 1738 SARS-CoV-2 sequences randomly chosen from the NCBI database against the SARS-CoV-2 reference genome. This SU_{align} workload mimics the work of a biologist trying to identify potentially new strains or variants of the SARS-CoV-2 virus. The number of sequences in the SU_{align} workload was also chosen at random to avoid biasing configurations in which the workload might fit perfectly in some given instance type, but in general, this is not important to the objective of the experiment.

Although the framework may be applied to any instance or group of instances in the cloud, for simplicity and illustration, the experiments presented here focus on examples

[1]For example, during the modeling of our framework, the latest third generation of Graviton processors were first debuted on the us-east-1 region

of the fifth, sixth, and seventh generations of EC2 instance types from the Compute Optimized family, one generation of Intel processors and two generations of AWS Graviton processors. For this bioinformatics application, the C5a instances with AMD processors performed worse than the Intel instances and thus were excluded from the experiment for motives of clarity (as can be seen by the size of some of the following tables). All virtual machine instances for the experiment have the operating system Ubuntu Server 20.04 LTS 64-bit installed (which showed to have better performance than both the newer Ubuntu Server 22.04 and Amazon Linux 2) and were executed in the On-demand market. Table 2 summarizes the main characteristics of the Compute Optimized families, while Tables 3 and 4 presents the available instance sizes and costs of the Intel-based C5 family in the us-east-1 region for which instance profiles are generated.

Family	Processor Architecture	Base Price
C5	Intel Xeon Scalable Processors (8124M / 8275CL)	US$0.085
C6g	AWS Graviton2 Processor (64-bit Arm Neoverse)	US$0.034
C7g	AWS Graviton3 Processor (64-bit Arm Neoverse)	US$0.036

Table 2: Main processing characteristics and On-Demand price per hour of the cheapest instance of the respective AWS EC2 Compute Optimized Instance Families on us-east-1 region, with Linux OS (as of July 2023).

Instance Size	Number of CPUs	RAM (GiB)	Price (USD/h)
large	1	4	$0.085
xlarge	2	8	$0.17
2xlarge	4	16	$0.34
4xlarge	8	32	$0.68
9xlarge	18	72	$1.53
12xlarge	24	96	$2.04
18xlarge	36	144	$3.06
24xlarge	48	192	$4.08

Table 3: C5 instance sizes with their respective numbers of physical CPUs and amounts of RAM, as well as the on-demand price per hour with a Linux OS, in us-east-1 (as of July 2023).

4.1 Application characterization and building the Instance Class Execution Profiles

As most OpenMP applications, MASA-OpenMP by default, creates as many threads per process as there are processing units available (as seen by the operating system). If pro-

			c6g	c7g
Instance Size	Number of CPUs	RAM (GiB)	Price (USD/h)	Price (USD/h)
medium	1	2	$0.034	$0.0361
large	2	4	$0.068	$0.0723
xlarge	4	8	$0.136	$0.1445
2xlarge	8	16	$0.272	$0.2890
4xlarge	16	32	$0.544	$0.5781
8xlarge	32	64	$1.088	$1.1562
12xlarge	48	96	$1.632	$1.7342
16xlarge	64	128	$2.176	$2.3123

Table 4: C6g and C7g instance sizes with the number of physical CPUs and amount of RAM available, and on-demand price per hour with Linux in us-east-1 (as of July 2023).

cessor architecture has multithreading capabilities, the number of threads may not necessarily be the number of physical cores. Therefore, to build the parallel execution strategy dataset, the profiling method externally adjusts the number of threads a single MASA-OpenMP process employs using OpenMP's OMP_NUM_THREADS variable. The execution times of alignments with different numbers of threads per process are collected, as explained in Chapter 3, to identify the best configurations in terms of performance, cost, and efficiency.

As described by the documentation of the instance class provided by AWS on its website at [2], the C5 family employs more than one Intel processor architecture: Instances may come with an Intel Xeon Scalable Platinum 8124M (first generation) or the newer Intel Xeon Scalable Platinum 8275CL (second generation). While the latter is generally the faster processor, the documentation also states that when a family has more than one processor architecture, the user does not normally get to choose which processor architecture it will receive with its allocated VM instance. In the case of the C5 family, the only thing that AWS guarantees is that the *12xlarge* and *24xlarge* instance sizes will have the newer Intel Xeon Scalable Platinum 8275CL. Thus, instance profiles for both processor architectures have to be generated, as explained in Chapter 3. On the other hand, our experiments showed that whenever one instantiated instances with the sizes large, xlarge, 4xlarge, 9xlarge, and 18xlarge of the C5 Instance Class in the *us-east-1* region (no matter the Availability Zone), they always had the 1st generation of the Intel Xeon Scalable Platinum 8124M as the underlying processor architecture. While the 2xlarge instance size sometimes had the 1st generation of the Intel Xeon Scalable Platinum 8124M, other times it might have the 2nd generation of the Intel Xeon Scalable

[2] https://aws.amazon.com/ec2/instance-types/c5/

Platinum 8275CL.

As mentioned in Chapter 3, it is necessary to build a dataset, one for each processor architecture in an Instance Class. For C5, Table 5 presents the dataset of the Amdahl strategy on the largest instance with the 2nd generation of the Intel Xeon Scalable Platinum 8275CL, while Table 6 also presents the execution times for the benchmark application, which form the dataset for the Amdahl strategy on the largest instance with an Intel Xeon Scalable Platinum 8124M. As one might notice, executing more configurations helps to identify more precisely how the efficiency changes in relation to the degree of parallelism. In both architectures, using 36 threads obtains the shortest execution times.

Number of threads per task	Execution time (s)	Efficiency
1	1.311	1.000
2	0.746	0.879
4	0.461	0.711
8	0.308	0.532
16	0.247	0.332
18	0.232	0.314
24	0.207	0.264
36	0.189	0.193
48	0.199	0.137
64	0.237	0.115
72	0.281	0.097
96	0.390	0.070

Table 5: Amdahl's strategy dataset on C5.24xlarge (8275CL processor architecture, with 48 CPUs) executing MASA-OpenMP's comparison with SARS-Cov-2 sequence.

Throughout the experimentation with the C5 family, the newer 2nd generation of the Intel Xeon Scalable Platinum 8275CL performed consistently better than the older 1st generation of the Intel Xeon Scalable Platinum 8124M. This indicates that the configurations with instances in which datasets were collected with the newer processor will dominate the configurations with the older and slower processor architecture, given that both have the same cost per hour. Therefore, they will be preferred when generating the Pareto Optimal solution. Unfortunately, we did not know that until we executed the instance profiling phase. This will be the case with any Instance Class that eventually has multiple processor architectures as the underlying infrastructure since users are not previously informed of which processor architecture will be available to their VM instances when there is more than one possibility. We cannot ignore the different processor architectures in the underlying infrastructure when they exist. More work on how to reuse data from different processor architectures to reduce the cost of generation of the instance

Number of threads per task	Execution time (s)	Relative Efficiency
1	1.446	0.907
2	0.818	0.801
4	0.494	0.663
8	0.321	0.511
16	0.264	0.310
18	0.255	0.286
24	0.239	0.229
36	0.233	0.156
48	0.283	0.097
64	0.315	0.065
72	0.368	0.049

Table 6: Amdahl's strategy dataset on C5.18xlarge (8124M processor architecture, with 36 CPUs) executing MASA-OpenMP's comparison with SARS-Cov-2 sequence. The efficiency presented is calculated in relation to the fastest sequential execution time obtained on any instance, which in this case was 1.311s on a C5.24xlarge instance (with an 8275CL processor architecture and 48 CPUs).

profiles will be proposed as future work.

After collecting the data and assembling the dataset for Amdahl's strategy, the parallel efficiency of the benchmark application: MASA-OpenMP with SARS-CoV-2 sequences can be calculated. As seen in Figure 6, the parallel efficiency of MASA-OpenMP rapidly decreases as the number of threads per process increases. It is also important to note that the C5.24xlarge instance has 48 physical processing units available (96 threads/vCPUs with Intel Hyper-threading technology enabled). That is why the efficiency decreases slightly slower after the number of threads hits 48. This shows that although the execution times for the alignment of SARS-CoV-2 sequences, using a single MASA-OpenMP process, improve using an increasing number of threads (up to 36), the usage of these resources becomes increasingly inefficient. This is an indicator that executing more comparisons using fewer threads each may be a more cost-effective use of this C5.24xlarge instance, at least for comparisons of the SARS-CoV-2 virus.

The execution of multiple alignment tasks concurrently, one process per comparison with one thread per process, is then carried out to build the Gustafson strategy dataset. To ensure that the environment was indeed being used concurrently, each of the workloads was configured to execute two hundred comparisons sequentially, with the average execution time of a comparison of the slowest workload being recorded. The experiment is repeated with increasing numbers of concurrent processes starting from a baseline of one process up to four times the total number of physical CPUs available in the VM instance. To

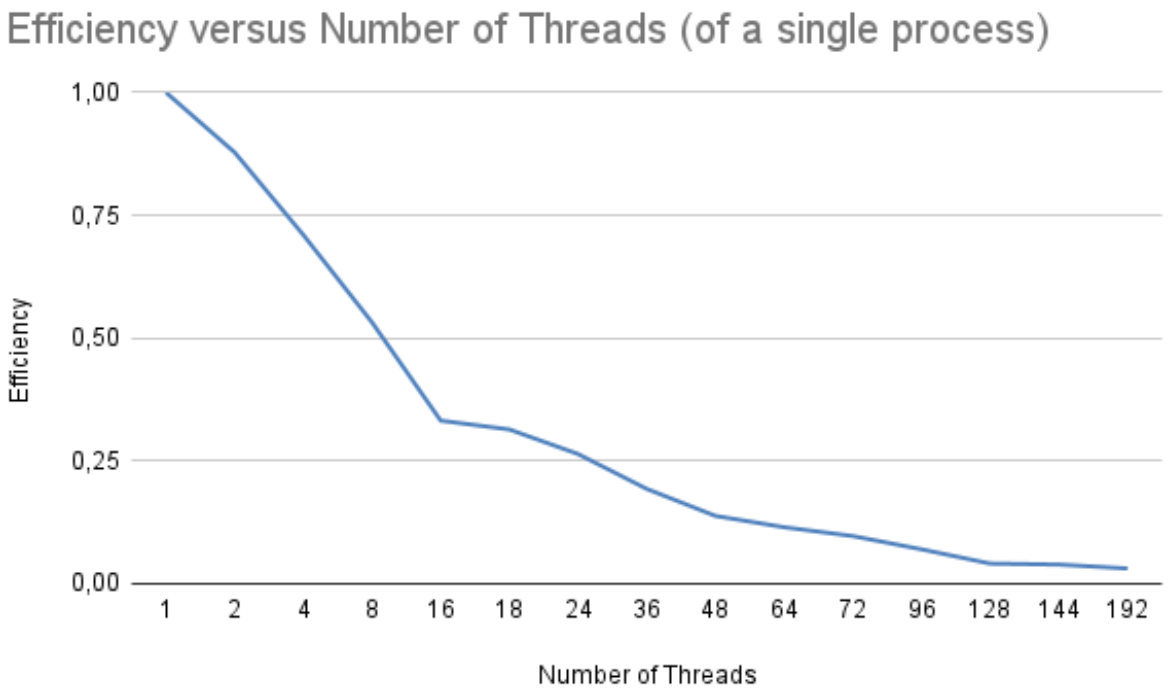


Figure 6: MASA-OpenMP parallel efficiency using increasing numbers of threads on EC2's C5.24xlarge instance (8275CL processor architecture, with 48 CPUs).

better understand the scalability of an instance family, the largest instance in this case of the C5 family (with Intel Hyper-Threading enabled) was used. Table 7 shows the dataset obtained for the Gustafson strategy using a C5.24xlarge instance.

The execution data show that increasing the number of concurrent processes in the VM instance beyond the number of CPUs available (in this case, 48) increases the system's efficiency and, consequently, its throughput. It is worth noting that there is a small slowdown, as one might expect, due to concurrency and a possible lowering of the clock frequency to compensate for the increased energy consumption. Figure 7 shows the efficiency of the execution on the largest instance of the C5 family, instance c5.24xlarge, and highlights a "range of efficiency", where the efficiency is at its highest, as mentioned in the previous chapter. For the AWS´s third generation of Computed Optimized Intel Instances, the best efficiency range begins when there are at least 2.5 processes per physical CPU and remains fairly constant up to as many as 10 processes per CPU in some instances (e.g., c5.12xlarge), especially when using the larger ones that have a lower chance of suffering interference from other co-hosted VMs. This finding is interesting because the number of vCPUs available to instances using Intel processors (that come with Intel´s Hyper-Threading Technology enabled by default) is twice the number of physical CPUs

Number of concurrent processes	Execution time (s)	Efficiency
1	1.31	1.000
2	1.32	0.992
4	1.36	0.963
8	1.38	0.949
16	1.39	0.942
32	1.40	0.936
48	1.45	0.903
64	1.74	1.004
72	1.86	1.056
96	2.20	1.191
120	2.68	1.222
128	2.85	1.226
144	3.20	1.228
168	3.89	1.179
192	6.03	0.869

Table 7: Dataset for Gustafson's scalability on a C5.24xlarge (8275CL processor architecture, with 48 CPUs).

inducing users to run only 1 or 2 processes per CPU (and not 2.5!). This indicates that executing more processes concurrently than the number of vCPUs would be necessary for this benchmark application to achieve peak efficiency and increase the system's throughput. This may be extremely counter-intuitive to a user, who would instinctively, at most, submit as many processes as there are vCPUs. Especially for a service that potentially executes millions of DNA sequence alignments in the cloud, this inefficiency would cause a significant increase in cost. Therefore, Figures 6 and 7 clearly show that, in this case, to achieve the objective of obtaining the highest throughput, it is important to use more processes with a single thread, i.e., to use Gustafson's strategy instead of a single process with multiple threads, i.e., Amdahl's strategy.

4.2 Generating the Pareto optimal solution for the real workload of MASA-OpenMP

After the instance profiles were generated, the service request configurations were generated based on the data collected using the C5 family in the profile generation phase. Given the sizes and costs of instances in the C5 family shown in Table 3, the PORSCHE methodology ends up evaluating 13911 configurations. The total number of configurations evaluated is the product of the number of tasks in the workload (i.e., "problem size") and

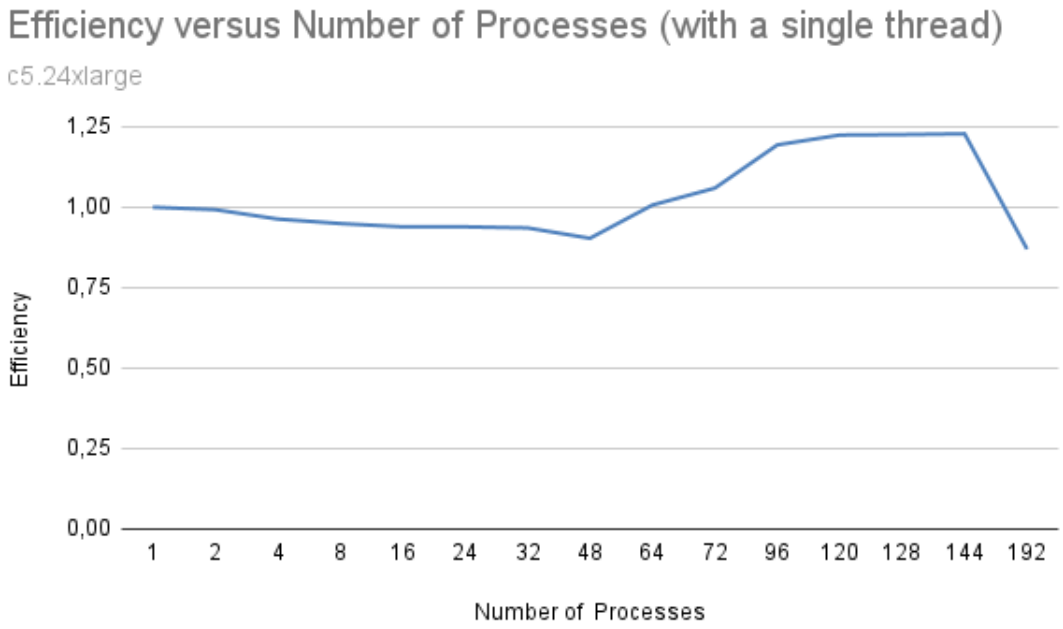


Figure 7: The efficiency of executing an increasing number of concurrent MASA-OpenMP processes, each with a single thread each, on a C5.24xlarge Instance with 48 CPUs.

the number of instance sizes per each instance family under consideration. In this case, there were 1738 tasks (alignments) in the workload, and eight instance sizes of the C5 family were considered; therefore, 13911 configurations were evaluated. From the original 13911 configurations evaluated, only 30 configurations were actually non-dominated, therefore, they were selected to enter the Pareto optimal solution as explained in Section 3.5. Table 8 presents an overview of each of the 30 configurations, ordered by cost, identifying the number and type of instance, and the predicted execution time. Table 9 presents the corresponding scheduling part of the identified configurations and describes the parallelisation approach to be used to execute the workload on the given instance type. In both tables, lines with white backgrounds represent the cheapest configurations with concurrent executions, i.e. using Gustafson's strategy; lines with light gray backgrounds represent configurations with hybrid executions, i.e. using Hybrid's strategy; and lines with dark gray backgrounds show the fastest configurations with parallel executions, i.e. using Amdahl's strategy.

Table 8 highlights the importance of presenting the user with multiple configurations. Line 1 shows the configuration with the cheapest execution cost. When comparing lines 1 and 2, it is possible to see that using the same instance size with different scheduling strategies yields very different results. The configuration in line 2 is 1.2% more expensive than the solution in line 1 but it is also 3.9 times faster. Comparing lines 1 and 3, the

configuration in line 1 is 6.6% more expensive, but it is more than 39 times faster, using an instance size different from the one suggested in the configuration in line 1. An approach that focuses on minimizing the execution cost might ignore the configurations in both lines 2 and 3 because those configurations are not the ones with the cheapest execution cost. However, for a user with flexibility in terms of expenses, perhaps an increase of only 1.2% or even 6.6% in the cost would be worth a return of a 3.9 or 39-fold improvement in performance, respectively.

Configuration Number	Instance Size	Number of instances	Estimated execution time (s)	Estimated total cost ($)
1	4xlarge	3	74.919	0.0425
2	4xlarge	12	18.982	0.0431
3	24xlarge	20	1.914	0.0453
4	24xlarge	21	1.842	0.0476
5	24xlarge	22	1.775	0.0499
6	24xlarge	23	1.728	0.0521
7	24xlarge	24	1.684	0.0544
8	24xlarge	25	1.642	0.0567
9	24xlarge	26	1.603	0.0589
10	24xlarge	27	1.578	0.0612
11	24xlarge	28	1.555	0.0635
12	24xlarge	29	1.522	0.0657
13	24xlarge	30	1.502	0.0680
14	24xlarge	31	1.492	0.0703
15	24xlarge	32	1.474	0.0725
16	24xlarge	33	1.456	0.0748
17	24xlarge	34	1.448	0.0771
18	24xlarge	35	1.433	0.0793
19	24xlarge	36	1.425	0.0816
20	24xlarge	37	1.412	0.0839
21	4xlarge	46	4.936	0.0434
22	4xlarge	47	4.827	0.0444
23	24xlarge	76	0.552	0.0861
24	24xlarge	92	0.544	0.1043
25	24xlarge	109	0.536	0.1235
26	24xlarge	134	0.528	0.1519
27	2xlarge	1738	0.394	0.1641
28	4xlarge	1738	0.283	0.3283
29	9xlarge	1738	0.163	0.7387
30	12xlarge	1738	0.149	0.9849

Table 8: Partial view of the solution set generated by the PORSCHE framework for the execution of 1738 alignments using MASA-OpenMP on instances from the C5 family. Each configuration identifies the size of the instance, the number of instances required, and the estimated execution time and cost.

The same happens when comparing the configurations that result in the fastest execution times. Comparing lines 30 and 29, for example, shows that an increase of 10% in the execution time results in a decrease of 25% in total cost. These are just a few examples of the importance of generating and presenting multiple solutions to the users.

A common misconception when using the cloud is that the number of instances available to a user is unlimited. In practice, cloud providers impose hard limits on the maximum number of instances (or CPUs) that a user may use at the same time. Additionally, resources at each availability zone are limited, and if the chosen region is particularly busy, there may not be a sufficient number of instances of the required type available to meet the request. When there exists a limit on the number of instances that can be used simultaneously, one might assume that it would be correct to simply ignore configurations that do not meet the criterion. Using the example experiment and its set of solutions presented previously in Table 8, Table 10 aims to show that this common assumption is flawed. The limit must be passed as a parameter to the algorithm before the configurations are generated. This is because, contrary to what one might think, the Pareto solution generated with a limit on the number of instances may be entirely different from the one without this limit. For example, the set of Pareto solutions generated for the same experiment, with 1738 alignments but with an imposed limit of 40 simultaneously executed instances, and shown in Table 10, differs from the one without the limit shown in Table 8. Not only is the number of solutions fewer, as expected, but some of the solutions were not in the previous solution set (e.g., configurations 3, 4, 5, 24, 25, and 26), i.e., Table 10 is not a subset of Table 8. This happens because configurations that were dominated by configurations with more instances would not enter the Pareto solution set without a limit. On the other hand, when the limit is considered in the generation of the configurations (i.e. before the Pareto solution is generated), those solutions with more instances would not even be calculated, allowing the now non-dominated configurations to be included in the Pareto solution set. The user must then know this limit value prior to generating the configurations.

A possible limitation of using our methodology is that the number of configurations in the Pareto solution has not been limited to a predefined manageable number during this generation stage. When presented to a user, this number of options may be too many, which, in turn, may make the decision overwhelming and confusing for the user. To mitigate this problem, several refinements may be made in the generation of the Pareto solution. For example, having seen the range of performances and costs in an initial Pareto solution set, the user may then choose to pass a maximum threshold for the total

Configuration Number	Number of comparisons per instance	Maximum number of parallel processes	Number of threads per per process	Number of rounds
1	580	20	1	29
2	145	19	1	8
3	87	87	1	1
4	83	83	1	1
5	79	79	1	1
6	76	76	1	1
7	73	73	1	1
8	70	70	1	1
9	67	67	1	1
10	65	65	1	1
11	63	63	1	1
12	60	60	1	1
13	58	58	1	1
14	57	57	1	1
15	55	55	1	1
16	53	53	1	1
17	52	52	1	1
18	50	50	1	1
19	49	49	1	1
20	47	47	1	1
21	38	19	1	2
22	37	19	1	2
23	23	23	4	1
24	19	19	5	1
25	16	16	6	1
26	13	13	7	1
27	1	1	8	1
28	1	1	16	1
29	1	1	36	1
30	1	1	48	1

Table 9: The schedule portion of each configuration within the solution set presented in Table 8. The second column identifies the workload of the instance, while the third and fourth define the degree of parallelism in terms of the Gustafson and Amdahl models, respectively.

Configuration Number	Instance Size	Number of instances	Estimated execution time (s)	Estimated total cost ($)
1	4xlarge	3	74.919	0.0425
2	4xlarge	12	18.982	0.0431
3	4xlarge	23	9.873	0.0434
4	4xlarge	33	6.981	0.0436
5	9xlarge	21	4.917	0.0446
6	24xlarge	20	1.914	0.0453
7	24xlarge	21	1.842	0.0476
8	24xlarge	22	1.775	0.0499
9	24xlarge	23	1.728	0.0521
10	24xlarge	24	1.684	0.0544
11	24xlarge	25	1.642	0.0567
12	24xlarge	26	1.603	0.0589
13	24xlarge	27	1.578	0.0612
14	24xlarge	28	1.555	0.0635
15	24xlarge	29	1.522	0.0657
16	24xlarge	30	1.502	0.0680
17	24xlarge	31	1.492	0.0703
18	24xlarge	32	1.474	0.0725
19	24xlarge	33	1.456	0.0748
20	24xlarge	34	1.448	0.0771
21	24xlarge	35	1.433	0.0793
22	24xlarge	36	1.425	0.0816
23	24xlarge	37	1.412	0.0839
24	24xlarge	38	1.405	0.0861
25	24xlarge	39	1.399	0.0884
26	24xlarge	40	1.394	0.0907

Table 10: Solutions proposed by the framework for the execution of 1738 alignments using MASA-OpenMP on the C5 instance family with the additional constraint of a maximum of 40 instances. New configurations not present in Table 8 are indicated in gray.

execution time or total cost and generate a refined Pareto solution set.

4.3 Validating the Methodology

Section 4.2 presented the results of applying the PORSCHE methodology to the AWS EC2 C5 instance class (that is, the 5th generation of compute-optimized instances with Intel processors). This section aims to validate the extensibility and feasibility of applying this methodology to another distinct AWS EC2 instance class. We opted to use another instance class of the compute-optimized family but with an entirely different processor architecture. The family C6g, the sixth generation of compute-optimized instances, is based on Amazon's custom-built ARM processor, Graviton2.

The experiment uses the same workload as the previous section with 1738 DNA sequence alignments of the SARS-CoV-2 sequences (i.e., each sequence with approximately 30,000 characters long) to allow some insights into the differences between the two instance families to be identified. Since the application is being used, there is no need to repeat the application characterization phase. Instead, the previous experimentation knowledge (and data) about the application collected with C5 instances will be reused.

The instance class execution profile is built using the largest instance size of the C6g family, the 16xlarge, with 64 CPUs and 128 GB of RAM. It is worth noting that the Graviton2 processor architecture present on the C6g instance family does not support Hyper-Threading technology; therefore, each CPU corresponds to one vCPU. Unlike the C5 instance class, C6g employs a single processor architecture, and therefore, only one profile is needed. Tables 11, 12, and 13, present the datasets generated for each of the three scheduling strategies.

After creating the Instance Class Execution Profile, the next step is to generate and analyze the possible configurations to identify the Pareto solution set. As in the case of the C5 instance class, the C6g family also has eight different instance sizes available, and therefore the same number of configurations, 13911, will be generated. Finally, a Pareto solution is generated by finding the non-dominated configurations within these 13911 configurations. Table 14 presents the 53 non-dominated configurations in the Pareto solution set.

As with the Pareto solution for the C5 instance class, comparing the configurations on the C6g Pareto solution set with each other again throws up some interesting configurations that might otherwise not be presented to the user. For example, comparing the

Number of threads per task	Execution time (s)	Relative Efficiency
1	1.792	0.732
2	0.986	0.665
4	0.589	0.556
8	0.385	0.426
16	0.314	0.261
32	0.227	0.180
48	0.218	0.125
64	0.251	0.082
128	1.105	0.009

Table 11: Data set for the Amdahl strategy on a C6g.16xlarge VM Instance. The efficiency presented is calculated in relation to the fastest sequential execution time obtained on any instance, which in this case was 1.311s on a C5.24xlarge instance (with an 8275CL processor architecture and 48 CPUs).

estimated time and cost obtained by the configuration on the first line with the one on the second line, i.e., comparing the configurations that produce the cheapest results, it is possible to see that, while the configuration on the second line is only three one-hundredths of a cent ($0.0003) more expensive, it is twice as fast. However, the configuration on the first line uses eight times fewer instances. Comparing the configurations in lines three and four with the configuration in line one also results in similar results. Both are less than $0,001 and more expensive, but are more than twice as fast, at the expense of using many more VM instances.

There are even more interesting findings when comparing the configurations on both Pareto solutions for the C5 and C6g instance families. For example, although the Pareto solution set for the C5 instance family has the configuration with the shortest execution times, it is the set for the C6g instance family that has the configuration with the lowest cost. Therefore, a Pareto solution considering both instance families would present configurations with instances from both instance families. The fastest configurations would certainly have instances from the C5 instance family, while the cheapest configurations would certainly have instances from the C6g instance family.

Number of concurrent processes	Execution time (s)	Relative Efficiency
1	1.807	0.726
2	1.910	0.686
4	1.910	0.686
8	1.904	0.689
16	1.889	0.694
32	1.880	0.697
48	1.891	0.693
64	1.900	0.690
96	2.660	0.739
128	3.650	0.718
160	4.538	0.722
192	5.503	0.715

Table 12: Data set for the Gustafson strategy on a C6g.16xlarge VM Instance. The efficiency presented is calculated in relation to the fastest sequential execution time obtained on any instance, which in this case was 1.311s on a C5.24xlarge instance.

Number of concurrent processes	Number of threads per process	Execution time (s)	Relative Efficiency
2	32	0.221	0.186
4	16	0.308	0.266
8	8	0.383	0.428
16	4	0.586	0.559
32	2	1.018	0.644

Table 13: Data set for the Hybrid strategy on a C6g.16xlarge VM Instance.

	Instance size	Number of instances	Estimated total execution time (s)	Estimated total cost (USD$)
1	8xlarge	12	7.974	0.0290
2	2xlarge	97	3.976	0.0293
3	4xlarge	65	2.995	0.0295
4	12xlarge	22	2.934	0.0299
5	4xlarge	67	2.899	0.0304
6	8xlarge	34	2.889	0.0308
7	12xlarge	23	2.840	0.0313
8	16xlarge	26	1.982	0.0314
9	16xlarge	27	1.928	0.0326
10	large	869	1.910	0.0328
11	4xlarge	109	1.889	0.0329

12	8xlarge	55	1.880	0.0332
13	12xlarge	76	0.979	0.0345
14	8xlarge	116	0.960	0.0351
15	12xlarge	79	0.941	0.0358
16	16xlarge	60	0.935	0.0363
17	12xlarge	83	0.903	0.0376
18	16xlarge	65	0.881	0.0393
19	12xlarge	87	0.865	0.0394
20	8xlarge	134	0.846	0.0405
21	12xlarge	92	0.828	0.0417
22	16xlarge	70	0.827	0.0423
23	8xlarge	145	0.791	0.0438
24	16xlarge	76	0.774	0.0459
25	12xlarge	103	0.755	0.0467
26	8xlarge	158	0.737	0.0478
27	12xlarge	109	0.719	0.0494
28	8xlarge	174	0.684	0.0526
29	16xlarge	92	0.668	0.0556
30	12xlarge	125	0.650	0.0567
31	8xlarge	194	0.633	0.0586
32	12xlarge	134	0.616	0.0607
33	12xlarge	145	0.586	0.0657
34	16xlarge	116	0.564	0.0701
35	12xlarge	158	0.549	0.0716
36	8xlarge	249	0.533	0.0753
37	12xlarge	174	0.516	0.0789
38	16xlarge	134	0.512	0.0810
39	4xlarge	580	0.483	0.0876
40	16xlarge	158	0.461	0.0955
41	12xlarge	218	0.453	0.0988
42	16xlarge	174	0.436	0.1052
43	12xlarge	249	0.422	0.1129
44	16xlarge	194	0.411	0.1173
45	4xlarge	869	0.383	0.1313

46	16xlarge	249	0.361	0.1505
47	16xlarge	290	0.336	0.1753
48	12xlarge	435	0.332	0.1972
49	16xlarge	348	0.311	0.2103
50	8xlarge	869	0.308	0.2626
51	16xlarge	580	0.261	0.3506
52	16xlarge	869	0.221	0.5253
53	12xlarge	1738	0.218	0.7879

Table 14: The solution set suggested by the model for the execution of 1738 alignments using MASA-OpenMP on the C6g instance class with the instance size, number of instances, and estimated execution time and costs.

To validate the precision of the methodology on the C6g instance class, five of the cheapest, five of the fastest, and 20 other configurations were executed on AWS EC2. The variation between the measured and predicted execution times ranged from -12,48% (i.e., the actual execution time was greater than the one estimated by the model) to +3,05% (i.e., the actual execution time was shorter than the estimated by the model). Table 15 shows the difference between the estimated execution times and the actual execution time measured by executing the suggested configuration in C6g instances.

	Instance size	Number of instances	Execution time per round (s)		Difference (%)
			Estimated	Actual	
1	8xlarge	12	4.016	4.123	-2.60%
2	2xlarge	97	3.976	4.090	-2.79%
3	4xlarge	65	2.995	3.055	-1.95%
4	12xlarge	22	2.934	2.974	-1.35%
5	4xlarge	67	2.899	2.940	-1.40%
6	8xlarge	34	2.889	2.935	-1.56%
7	12xlarge	23	2.84	2.859	-0.66%
8	16xlarge	26	1.982	1.923	3.05%
9	16xlarge	27	1.928	1.873	2.92%
11	4xlarge	109	1.889	1.846	2.33%
12	8xlarge	55	1.88	1.840	2.18%
13	12xlarge	76	0.979	0.995	-1.57%
14	8xlarge	116	0.96	0.989	-2.98%
29	16xlarge	92	0.668	0.726	-7.99%
31	8xlarge	194	0.633	0.723	-12.48%
34	16xlarge	116	0.564	0.586	-3.74%
36	8xlarge	249	0.533	0.582	-8.49%
38	16xlarge	134	0.512	0.584	-12.33%
39	4xlarge	580	0.483	0.508	-4.96%
40	16xlarge	158	0.461	0.509	-9.50%
42	16xlarge	174	0.436	0.456	-4.48%
44	16xlarge	194	0.411	0.414	-0.74%
45	4xlarge	869	0.383	0.427	-10.21%
46	16xlarge	249	0.361	0.364	-0.92%
47	16xlarge	290	0.336	0.351	-4.39%
49	16xlarge	348	0.311	0.338	-7.93%
50	8xlarge	869	0.308	0.309	-0.46%
51	16xlarge	580	0.261	0.264	-1.08%
52	16xlarge	869	0.221	0.222	-0.25%
53	12xlarge	1738	0.218	0.213	2.49%

Table 15: Comparison of the actual and the estimated execution time of the suggested configurations for executing the study case application on the EC2 C6g instance class.

4.3.1 Further testing on Graviton's new generation - C7g

The recent release of instances with AWS's third generation of Graviton processor offered the opportunity to investigate if and how a new generation of processor might change the resulting Pareto solution set. Tables 16, 17, and 18 present the new datasets for the three respective strategies using the largest C7g instance available.

Number of threads per task	Execution time (s)	Relative Efficiency
1	1.222	1.073
2	0.671	0.977
4	0.394	0.832
8	0.257	0.638
16	0.208	0.394
32	0.153	0.268
64	0.2	0.102

Table 16: Amdahl's strategy dataset on C7g.16xlarge. The efficiency presented is calculated in relation to the sequential execution time obtained on the C5.24xlarge instance, which was 1.311.

Number of concurrent processes	Execution time (s)	Relative Efficiency
1	1.222	1.073
2	1.219	1.075
4	1.218	1.076
8	1.218	1.076
16	1.22	1.075
32	1.222	1.073
64	1.266	1.036
96	1.847	1.065
128	2.476	1.059
160	4.254	0.770
192	6.214	0.633

Table 17: Gustafson's strategy dataset on C7g.16xlarge. The efficiency presented is calculated in relation to the fastest sequential execution time obtained on any instance, which in this case was 1.311s on a C5.24xlarge instance.

After generating the datasets, they are used to generate Pareto solutions for the same

Number of concurrent processes	Number of threads per process	Execution time (s)	Relative Efficiency
2	32	0.153	0.268
4	16	0.212	0.386
8	8	0.261	0.628
16	4	0.402	0.815
32	2	0.700	0.936

Table 18: Hybrid's strategy dataset The efficiency presented is calculated in relation to the fastest sequential execution time obtained on any instance, which in this case was 1.311s on a C5.24xlarge instance.

1738 alignments. Fifty-eight configurations were found. Table 19 presents an excerpt of the solution that contains the size of the instances, the number of instances, the estimated total execution time, and the total cost.

When comparing the Pareto solution set for the C7g instance family (Table 19) with the Pareto solution sets for the other instance families (Tables 8 and 14) tested so far (i.e., C5 and C6g), the set for the C7g instance family presents the configuration with the shortest execution time and also the configuration with the lowest cost. This is interesting because it shows that AWS's latest generation of Graviton instances, which feature a custom ARM processor architecture, perform better than the more commonly used C5 instances, which feature 8275CL and 8124M Intel processor architectures. Furthermore, note that this is in spite of the fact that the 8275CL processor obtained the fastest sequential execution. When AWS launched its C6g family powered by Arm-based AWS Graviton2 processors, they claimed that these instances deliver up to 40% better price performance over C5 instances. The cheaper Pareto configurations do indeed use c6g instances, while faster execution configurations employ c5 instances. It is perhaps worth noting that this direct comparison may not exactly be fair given that the processor in the C7g instance family is far newer than the processors in the C5 instance family. In fact, AWS has released two further generations of Intel-based instances, c6i and, most recently, c7i. However, end users may not be aware of this or their price performance differences. This is yet more evidence of the importance of a tool like PORSCHE to aid the user in choosing the right configuration for their workload and budget.

	Instance size	Number of instances	Estimated total execution time (s)	Estimated total cost (USD$)
1	2xlarge	83	2.963	0.0200
2	12xlarge	22	1.968	0.0212
3	4xlarge	67	1.934	0.0215
4	16xlarge	17	1.912	0.0218
5	12xlarge	23	1.874	0.0222
6	4xlarge	70	1.842	0.0225
7	16xlarge	18	1.783	0.0231
8	8xlarge	37	1.720	0.0238
9	12xlarge	25	1.708	0.0241
10	4xlarge	76	1.680	0.0244
11	12xlarge	26	1.636	0.0250
12	2xlarge	158	1.557	0.0254
13	4xlarge	83	1.544	0.0267
14	16xlarge	21	1.542	0.0270
15	8xlarge	43	1.518	0.0276
16	xlarge	348	1.399	0.0279
17	2xlarge	194	1.283	0.0311
18	medium	1738	1.222	0.0349
19	12xlarge	76	0.673	0.0366
20	8xlarge	116	0.660	0.0373
21	12xlarge	79	0.647	0.0381
22	16xlarge	60	0.643	0.0385
23	12xlarge	83	0.620	0.0400
24	16xlarge	65	0.605	0.0417
25	12xlarge	87	0.594	0.0419
26	8xlarge	134	0.581	0.0430
27	12xlarge	92	0.568	0.0443
28	8xlarge	145	0.543	0.0466
29	16xlarge	76	0.531	0.0488
30	12xlarge	103	0.518	0.0496
31	8xlarge	158	0.506	0.0507
32	12xlarge	109	0.493	0.0525

33	8xlarge	174	0.469	0.0559
34	16xlarge	92	0.458	0.0591
35	12xlarge	125	0.445	0.0602
36	8xlarge	194	0.433	0.0623
37	12xlarge	134	0.422	0.0646
38	xlarge	1738	0.394	0.0698
39	16xlarge	116	0.386	0.0745
40	12xlarge	158	0.376	0.0761
41	8xlarge	249	0.365	0.0800
42	12xlarge	174	0.354	0.0838
43	16xlarge	134	0.351	0.0861
44	4xlarge	580	0.331	0.0931
45	16xlarge	158	0.316	0.1015
46	12xlarge	218	0.310	0.1050
47	16xlarge	174	0.299	0.1118
48	12xlarge	249	0.289	0.1199
49	16xlarge	194	0.282	0.1246
50	2xlarge	1738	0.257	0.1395
51	16xlarge	249	0.247	0.1599
52	16xlarge	290	0.230	0.1863
53	12xlarge	435	0.228	0.2095
54	16xlarge	348	0.214	0.2235
55	4xlarge	1738	0.208	0.2791
56	16xlarge	580	0.180	0.3725
57	16xlarge	869	0.153	0.5582
58	16xlarge	1738	0.145	1.1163

Table 19: Excerpt of a solution offered by the model to the execution of 1738 alignments using MASA-OpenMP on the C7g instance class demonstrating the instance size, number of instances suggested by the model, and the estimated execution time and costs.

Table 20 shows an excerpt of the solution with the scheduling information.

	Number of comparisons per instance	Maximum number of parallel processes	Number of threads per process	Number of rounds
1	21	11	1	2
2	79	79	1	1
3	26	26	1	1
4	103	103	1	1
5	76	76	1	1
6	25	25	1	1
7	97	97	1	1
8	47	47	1	1
9	70	70	1	1
10	23	23	1	1
11	67	67	1	1
12	11	11	1	1
13	21	21	1	1
14	83	83	1	1
15	41	41	1	1
16	5	5	1	1
17	9	9	1	1
18	1	1	1	1
19	23	23	2	1
20	15	15	2	1
21	22	22	2	1
22	29	29	2	1
23	21	21	2	1
24	27	27	2	1
25	20	20	2	1
26	13	13	2	1
27	19	19	2	1
28	12	12	2	1
29	23	23	2	1
30	17	17	2	1

31	11	11	2	1
32	16	16	3	1
33	10	10	3	1
34	19	19	3	1
35	14	14	3	1
36	9	9	3	1
37	13	13	3	1
38	1	1	4	1
39	15	15	4	1
40	11	11	4	1
41	7	7	4	1
42	10	10	4	1
43	13	13	4	1
44	3	3	5	1
45	11	11	5	1
46	8	8	6	1
47	10	10	6	1
48	7	7	6	1
49	9	9	7	1
50	1	1	8	1
51	7	7	9	1
52	6	6	10	1
53	4	4	12	1
54	5	5	12	1
55	1	1	16	1
56	3	3	21	1
57	2	2	32	1
58	1	1	48	1

Table 20: Excerpt of a solution offered by the model to the execution of 1738 alignments using MASA-OpenMP on the C7g instance class demonstrating the scheduling information (Number of comparisons per instance, Maximum number of parallel processes, Number of threads per process, Number of rounds).

As before, 30 of the suggested solutions were executed: five of the cheapest, five of the fastest, and 20 other configurations. Again, the executions were reasonably accurate and presented the user with configurations that might have otherwise been obscured. Table 21 shows the percentual difference between the estimated and actual execution times found. Further improvements in the accuracy of the generated solution are necessary and are proposed as future work. This is important because, as accurate as the configurations may seem when executing the configurations in the solution, a slight difference in the estimated execution time could generate an entirely different Pareto solution. Therefore, the optimization of the model's accuracy is of paramount importance. On the other hand, the intrinsic performance fluctuations inherent to the Cloud may make these optimizations difficult. Consequently, deciding the right granularity of the improvements in the accuracy must always consider a trade-off between the accuracy of the data used by the model and the cost of generating them, as explained before.

	Instance size	Number of instances	Execution time per round (s)		Difference (%)
			Estimated	Actual	
1	2xlarge	83	2,963	3,405	13,0%
2	12xlarge	22	1,968	2,001	1,7%
3	4xlarge	67	1,934	1,996	3,1%
4	16xlarge	17	1,912	1,982	3,5%
5	12xlarge	23	1,874	1,927	2,8%
8	8xlarge	37	1,72	1,807	4,8%
9	12xlarge	25	1,708	1,782	4,1%
11	12xlarge	26	1,636	1,707	4,1%
12	2xlarge	158	1,557	1,703	8,6%
13	4xlarge	83	1,544	1,622	4,8%
19	12xlarge	76	0,673	0,706	4,7%
20	8xlarge	116	0,66	0,675	2,3%
23	12xlarge	83	0,62	0,697	11,0%
24	16xlarge	65	0,605	0,677	10,7%
25	12xlarge	87	0,594	0,697	14,8%
32	12xlarge	109	0,493	0,544	9,3%
33	8xlarge	174	0,469	0,495	5,2%
39	16xlarge	116	0,386	0,399	3,3%
40	12xlarge	158	0,376	0,463	18,8%

41	8xlarge	249	0,365	0,398	8,3%
44	4xlarge	580	0,331	0,346	4,3%
45	16xlarge	158	0,316	0,348	9,1%
50	2xlarge	1738	0,257	0,263	2,3%
51	16xlarge	249	0,247	0,250	1,1%
52	16xlarge	290	0,23	0,239	3,9%
54	16xlarge	348	0,214	0,231	7,3%
55	4xlarge	1738	0,208	0,229	9,4%
56	16xlarge	580	0,18	0,181	0,6%
57	16xlarge	869	0,153	0,152	-0,4%
58	16xlarge	1738	0,145	0,142	-1,9%

Table 21: Pareto solution Comparison of the actual and the estimated execution time of the suggested configurations for executing the case study application on the EC2 C7g instance class.

To finish, we generated a Pareto solution considering all three instance families tested: C5, C6g, and C7g. Table 22 shows the resulting Pareto solution. As expected, the configuration with the shortest execution time and the configuration with the cheapest execution feature C7g instances.

	Instance size	Number of instances	Execution total execution time (s)	Estimated total cost (USD$)
1	c7g.2xlarge	83	2.963	0.020
2	c7g.12xlarge	22	1.968	0.021
3	c7g.4xlarge	67	1.934	0.022
4	c7g.16xlarge	17	1.912	0.022
5	c7g.12xlarge	23	1.874	0.022
6	c7g.4xlarge	70	1.842	0.023
7	c7g.16xlarge	18	1.783	0.023
8	c7g.8xlarge	37	1.720	0.024
9	c7g.12xlarge	25	1.708	0.024
10	c7g.4xlarge	76	1.680	0.024
11	c7g.12xlarge	26	1.636	0.025
12	c7g.2xlarge	158	1.557	0.025

13	c7g.4xlarge	83	1.544	0.027
14	c7g.16xlarge	21	1.542	0.027
15	c7g.8xlarge	43	1.518	0.028
16	c7g.xlarge	348	1.399	0.028
17	c7g.2xlarge	194	1.283	0.031
18	c6g.12xlarge	76	0.979	0.035
19	c6g.8xlarge	116	0.960	0.035
20	c6g.12xlarge	79	0.941	0.036
21	c6g.16xlarge	60	0.935	0.036
22	c7g.12xlarge	76	0.673	0.037
23	c7g.8xlarge	116	0.660	0.037
24	c7g.12xlarge	79	0.647	0.038
25	c7g.16xlarge	60	0.643	0.039
26	c7g.12xlarge	83	0.620	0.040
27	c7g.16xlarge	65	0.605	0.042
28	c7g.12xlarge	87	0.594	0.042
29	c7g.8xlarge	134	0.581	0.043
30	c7g.12xlarge	92	0.568	0.044
31	c7g.8xlarge	145	0.543	0.047
32	c7g.16xlarge	76	0.531	0.049
33	c7g.12xlarge	103	0.518	0.050
34	c7g.8xlarge	158	0.506	0.051
35	c7g.12xlarge	109	0.493	0.053
36	c7g.8xlarge	174	0.469	0.056
37	c7g.16xlarge	92	0.458	0.059
38	c7g.12xlarge	125	0.445	0.060
39	c7g.8xlarge	194	0.433	0.062
40	c7g.12xlarge	134	0.422	0.065
41	c7g.xlarge	1738	0.394	0.070
42	c7g.16xlarge	116	0.386	0.075
43	c7g.12xlarge	158	0.376	0.076
44	c7g.8xlarge	249	0.365	0.080
45	c7g.12xlarge	174	0.354	0.084
46	c7g.16xlarge	134	0.351	0.086

47	c7g.4xlarge	580	0.331	0.093
48	c7g.16xlarge	158	0.316	0.102
49	c7g.12xlarge	218	0.310	0.105
50	c5.4xlarge	580	0.264	0.110
51	c7g.2xlarge	1738	0.257	0.140
52	c7g.16xlarge	249	0.247	0.160
53	c5.4xlarge	869	0.212	0.164
54	c7g.4xlarge	1738	0.208	0.279
55	c7g.16xlarge	580	0.180	0.373
56	c7g.16xlarge	869	0.153	0.558
57	c7g.16xlarge	1738	0.145	1.116

Table 22: Excerpt of a solution offered by the model to the execution of 1738 alignments using MASA-OpenMP on the C5, C6g, and C7g instance classes demonstrating the instance size, number of instances suggested by the model, and the estimated execution time and costs.

It is important to note that while generating the Pareto solution, both the execution time and cost of the configurations were rounded to the nearest unit (second or cent, in case of execution time and cost, respectively). This had an intended impact on the Pareto solution generated because this way, equivalent configurations were made equal, therefore eliminating one of them, by default the one with the bigger number of instances, as mentioned in Chapter 3. This is an example of the use of secondary objectives as refinements, also mentioned in Chapter 3, in this case, to have a lower number of configurations in the solution (by eliminating configurations considered equivalent). Another important consideration is that the minimum billable time, which is 60 seconds on AWS but may vary for other cloud providers, was left out of the cost calculations for simplicity's sake. It is important because considering this minimum billable time could change the Pareto solution generated since configurations with execution times shorter than 60 seconds would have their execution time rounded to 60 seconds.

In this chapter, we have tested the PORSCHE methodology with a real-world bioinformatics workload, the DNA Sequence Alignment, using MASA-OpenMP, with different instance families. We have proved that the PORSCHE methodology is easily applicable to different instance families, even from newer generations, like the C7g instance family, and that it is also flexible enough to attend to different user objectives.

5 Conclusions and Future work

Generating and offering multiple optimal configurations to users is important because it allows them not only to use configurations that meet the classic objectives of finding the fastest execution possible or the one with the lowest cost but also to identify configurations that may not be as obvious and yet offer interesting compromises in relation to the originally chosen objective. Furthermore, these additional configurations may be important when having to comply with certain constraints.

Simply choosing the "right" instance for the workload is insufficient to guarantee the efficient use of the selected resources. Task scheduling plays an important role when trying to maximize execution efficiency, and therefore in identifying the "right" instance and obtaining an optimal configuration to execute the given workload in the cloud. To obtain such schedules, it is important to characterize the application in terms of both its parallel scalability and concurrent scalability. Often, the ideal schedule may combine both to different degrees.

A core aspect of the execution model used by the PORSCHE methodology is the fact that it uses efficiency as the principal metric to make comparisons between alternative configurations across resources. With this, the model is able to identify the transition points when one scheduling strategy should be abandoned and another adopted to identify alternative configurations that might better address the users' objectives for the instance classes being considered.

It is worth noting that in this work this black-box methodology has only been extensively tested with MASA-OpenMP workloads (in the context of Bag-of-task-like scientific experiments), using pairs of sequences of similar sizes, on AWS EC2. Expanding the evaluation to other applications and scientific experiments, or to other MASA-OpenMPs workloads with sequences of different sizes, and additional public cloud providers is important to understand the effectiveness and the practical applicability of the approach. Equally essential is minimizing the cost of the profiling phase of the methodology by reducing the

number of evaluations required during both offline phases - application characterization and instance profiling - to obtain a model with an acceptable degree of accuracy for use during the online phase that is executed every for every experiment to be executed.

REFERENCES

ALIPOURFARD, Omid et al. Cherrypick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In: PROCEEDINGS of the 14th USENIX Conference on Networked Systems Design and Implementation. Boston, MA, USA: USENIX Association, 2017. (NSDI'17), p. 469–482. ISBN 9781931971379.

ALURU, S. **Handbook of Computational Molecular Biology**. [S. l.]: Chapman e Hall/CRC, 2005. DOI: 10.1201/9781420036275.

AMAZON AWS. **Amazon Elastic Compute Cloud (EC2)**. [S. l.: s. n.], 2022. https://aws.amazon.com/ec2/. Accessed January 2023.

_____. **Instance Types**. [S. l.: s. n.], 2021. https://aws.amazon.com/ec2/instance-types/. Accessed January 2023.

BRUNETTA, Jeferson R.; BORIN, Edson. Selecting Efficient Cloud Resources for HPC Workloads. In: PROCEEDINGS of the 12th IEEE/ACM International Conference on Utility and Cloud Computing. Auckland, New Zealand: Association for Computing Machinery, 2019. (UCC'19), p. 155–164. ISBN 9781450368940. DOI: 10.1145/3344341.3368798. Disponível em: <https://doi.org/10.1145/3344341.3368798>.

BUYYA, Rajkumar et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, v. 25, n. 6, p. 599–616, 2009. ISSN 0167-739X. DOI: https://doi.org/10.1016/j.future.2008.12.001. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X08001957>.

CAMACHO, C. et al. ElasticBLAST: accelerating sequence search via cloud computing. **BMC Bioinformatics**, 24(1):117, 2023.

DE O. SANDERS, Edans F. et al. MASA: A Multiplatform Architecture for Sequence Aligners with Block Pruning. **ACM Trans. Parallel Comput.**, Association for Computing Machinery, New York, NY, USA, v. 2, n. 4, fev. 2016.

DURILLO, J.J.; PRODAN, R. Multi-objective workflow scheduling in Amazon EC2. **Cluster Comput**, v. 17, p. 169–189, 2014. DOI: https://doi.org/10.1007/s10586-013-0325-0.

FEITELSON, D. G.; RUDOLPH, L. Towards convergence in job schedulers for parallel supercomputers. In_____. **Job Scheduling Strategies for Parallel Processing**. [S. l.]: Springer, 1996. v. 1162. (LNCS), p. 1–26.

HSU, Chin-Jung; NAIR, Vivek; FREEH, Vincent W. et al. Arrow: Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). [S. l.: s. n.], 2018. P. 660–670. DOI: 10.1109/ICDCS.2018.00070.

HSU, Chin-Jung; NAIR, Vivek; MENZIES, Tim et al. Micky: A Cheaper Alternative for Selecting Cloud Instances. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). [S. l.: s. n.], 2018. P. 409–416. DOI: 10.1109/CLOUD.2018.00058.

HSU, Chin-Jung; NAIR, Vivek; MENZIES, Tim et al. Scout: An Experienced Guide to Find the Best Cloud Configuration. **ArXiv**, abs/1803.01296, 2018.

HUTTER, Frank; HOOS, Holger H.; LEYTON-BROWN, Kevin. Sequential Model-Based Optimization for General Algorithm Configuration. In_____. **Learning and Intelligent Optimization**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. P. 507–523. ISBN 978-3-642-25566-3.

LEAVITT, Neal. Is Cloud Computing Really Ready for Prime Time? **Computer**, v. 42, n. 1, p. 15–20, 2009. DOI: 10.1109/MC.2009.20.

MARLER, R. T.; ARORA, J. S. Survey of multi-objective optimization methods for engineering. **Structural and Multidisciplinary Optimization**, v. 26, n. 6, p. 369–395, abr. 2004. ISSN 1615-1488. DOI: 10.1007/s00158-003-0368-6. Disponível em: <https://doi.org/10.1007/s00158-003-0368-6>.

MILLS, Gabriel et al. Estimativa de Tempo de Alinhamentos Biológicos na Nuvem. In: ANAIS da VIII Escola Regional de Alto Desempenho do Rio de Janeiro. Rio de Janeiro/RJ: SBC, 2023. P. 14–16. DOI: 10.5753/eradrj.2023.231704. Disponível em: <https://sol.sbc.org.br/index.php/eradrj/article/view/24818>.

MORENO-VOZMEDIANO, Rafael; MONTERO, Rubén S.; LLORENTE, Ignacio M. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. **Computer**, v. 45, n. 12, p. 65–72, 2012. DOI: 10.1109/MC.2012.76.

MYERS, E. W.; MILLER, W. Optimal alignments in linear space. **Bioinformatics**, v. 4, n. 1, p. 11–17, 1988. DOI: 10.1093/bioinformatics/4.1.11. eprint: /oup/backfile/content_public/journal/bioinformatics/4/1/10.1093/bioinformatics/4.1.11/2/4-1-11.pdf. Disponível em: <http://dx.doi.org/10.1093/bioinformatics/4.1.11>.

SMITH, T.F.; WATERMAN, M.S. Identification of common molecular subsequences. **Journal of Molecular Biology**, v. 147, n. 1, p. 195–197, 1981. ISSN 0022-2836. DOI: http://dx.doi.org/10.1016/0022-2836(81)90087-5. Disponível em: <http://www.sciencedirect.com/science/article/pii/0022283681900875>.

SODRÉ, Daniel; BOERES, Cristina; REBELLO, Vinod. Making the most of what you pay for by delaying tasks to improve overall cloud instance performance. In: ANAIS Estendidos do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho. Florianópolis: SBC, 2022. P. 9–16. DOI: 10.5753/wscad_estendido.2022.226672. Disponível em: <https://sol.sbc.org.br/index.php/wscad_estendido/article/view/21898>.

STATISTA. **"Big Three Dominate the Global Cloud Market"**. [S. l.: s. n.], 2023. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. Accessed June 2023.

TAVARES, William F. C.; ASSIS, Marcio R. M.; BORIN, Edson. Leveraging VCPU-Utilization Rates to Select Cost-Efficient VMs for Parallel Workloads. In: PROCEEDINGS of the 14th IEEE/ACM International Conference on Utility and Cloud Computing. New York, NY, USA: Association for Computing Machinery, 2021. ISBN 9781450385640. Disponível em: <https://doi.org/10.1145/3468737.3494095>.

TOPCUOGLU, H.; HARIRI, S.; WU, Min-You. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE Transactions on Parallel and Distributed Systems**, v. 13, n. 3, p. 260–274, 2002. DOI: 10.1109/71.993206.

YADWADKAR, Neeraja J. et al. Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach. In: PROCEEDINGS of the 2017 Symposium on Cloud Computing. Santa Clara, California: Association for Computing Machinery, 2017. (SoCC '17), p. 452–465. ISBN 9781450350280. DOI: 10.1145/3127479.3131614. Disponível em: <https://doi.org/10.1145/3127479.3131614>.