Making the most of what you pay for by delaying tasks to improve overall cloud instance performance*

Daniel Bougleux Sodré[1], **Cristina Boeres**[1], **Vinod E.F. Rebello**[1]

[1] Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

danielbougleux@id.uff.br, {boeres,vinod}@ic.uff.br

Abstract. *Resource elasticity and server consolidation have long been among two of cloud computing's most relevant management tools. Yet, exemplified with a scientific application use case, this paper highlights how judicious scheduling of tasks can help maximize resource utilization and improve performance and costs for both users and cloud providers. Developing an efficient cloud service for DNA sequence comparisons is adopted as a motivating use case. Using the bioinformatics tool MASA that finds an optimal pair-wise sequence alignment, we propose a model for co-scheduling multiple alignments on a single cloud instance. The resulting, practically optimal, non-preemptive schedule can effectively double the throughput of MASA-based sequence alignment workflows.*

1. Introduction

The adoption of cloud computing continues to grow with many companies and research centers migrating their computing workloads from locally hosted physical infrastructures to virtualized resources hosted in public clouds such as Google Cloud Platform, Amazon Elastic Compute Cloud, or Microsoft Azure. These cloud providers offer compute servers, storage, databases, software, applications and tools, among others, as services over the Internet for different types and needs of users. One benefit of using cloud services is that users can avoid the upfront costs and the complexity of owning and maintaining their own IT infrastructure. Instead, they only have to pay for what they use, when they use it.

One of the oldest, but still very popular, cloud models is *Infrastructure as a Service* (IaaS). Here, cloud providers generally offer their infrastructure resources in the form of *virtual machine* (VM) *instances*, with different pre-configured capacities, at given cost per hour rates. Instances are classified based on the type of resources they contain. Typically, the rates of larger instance configurations of each class scale linearly, e.g., a VM instance with twice the number of CPUs and twice the RAM memory will be double the rate. However, the onus is then on users to choose the appropriate VM instance class and size to meet their needs. While the adoption of container services or *serverless computing* will perhaps move this difficult decision to the service provider, even then the same issues remain: Issue (a) identifying the appropriate instance class/size for the user's request, and; Issue (b) extracting the required or best performance from the chosen instance. Much recent research has focused on the former, proposing heuristics to search for the most appropriate cloud instance(s), e.g., [Brunetta and Borin 2019]. Our paper, however, focuses

*This work was funded in part by: CNPq, through an *Iniciação Científica* scholarship from PIBIC-UFF and the project CNPq/AWS 440014/2020-4, and; CAPES, through the *Programa Institucional de Internacionalização* (CAPES/PrInt) (Process numbers 88887.310261/2018-00 & 88887.570000/2020-00).

on the latter, given that Issue (a) cannot be solved without simultaneously addressing Issue (b), and the importance task scheduling plays not only in terms of performance but also resource utilization. The work presented here is applicable to any cloud cost model (on-demand or spot instances) and service model (IaaS, with VMs or containers, or FaaS).

It is challenging for scientists, without an understanding of cloud technologies, to define the best way to execute their workloads. In relation to Issue (a), estimating their resource requirements can be hard, especially without prior knowledge of the service or application's behavior. IaaS users tend to request more resources than necessary and thus incur additional expenses. This also leads to host servers in practice being either sub-utilized or perhaps even left idle. But even if clouds were to provide instances that adapt to fit the application (services based on containers and serverless computing are moving in this direction), tools are still required to dynamically manage an instance's resources.

Intuitively, when scheduling applications, users tend to prioritize maximizing CPU utilization, often by over-committing CPUs in order to use spare cycles. Maximizing memory utilization is more difficult and, in fact, riskier. A lack of available memory can cause significant performance degradation or even premature termination of applications. Thus, recently, a number of vertical memory elasticity management tools (e.g., [Calatrava et al. 2016]) have been proposed to allow the resource allocation for an instance (be it a VM or container) to shrink or expand, in response to the changes in resource usage by the applications running in the instance, without having to restart it. This allows a server's resources to be shared more efficiently between co-hosted instances. However, the underlying assumption behind over-committing resources is that applications do not use their peak consumption of a given resource for the majority of their execution. This means the possibility exists that a host or instance may have insufficient resources to attend its allocated applications. To avoid resulting slowdowns or failures, and additional costs, tools must either migrate instances or suspend them in an attempt to aggressively maximize the utilization of the memory available [Nicodemus et al. 2020].

This work presents a scheduling scheme that aims to maximize both CPU and memory utilization of applications with a *monotonically increasing* memory consumption pattern without the need for time consuming suspensions. A careful execution plan is necessary due to the fact that processors may be left idle or memory may become scarce or insufficient depending on the size of the workload of the job being scheduled.

2. Problem Statement

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 may cause, how easily that virus spreads, the severity of associated symptoms, and the effectiveness of the respective vaccines. The recent pandemic has only highlighted the importance of this type of analysis, as exemplified by the over 18 million SARS-CoV-2 DNA sequences, obtained from human infections from 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/) from the National Center for Biotechnology Information (NCBI) in the USA and the European-based GISAID (https://www.gisaid.org/).

While cloud-based sequence alignment services exist (e.g., the one from NCBI is hosted on Amazon EC2), most have restrictive usage limits (in terms of DNA sequence

lengths < 200K) and little is known about the cost efficiency of their implementations. The final goal of this work is to design a provably efficient cloud-based alignment service, in terms of solution quality, total execution time and cloud costs. Furthermore, achieve this without a vertical elasticity controller having to resort to pausing (which is fast) or suspending/migrating (which is time consuming as this involves creating and recovering checkpoints) tasks or instances when memory usage is about to exceed availability.

We adopted the bioinformatics tool *Multi-Platform Architecture for Sequence Aligners* (MASA) for its ability to carry out pair-wise biological alignments of DNA sequences with more than 200 million base pairs, on a variety of hardware/software platforms [De O. Sanders et al. 2016]. The work here evaluates MASA-OpenMP, a version of MASA designed to harness the multiple CPU cores on a server and thus is likely to benefit from the large variety of architectures commonly offered by cloud providers. MASA-OpenMP compares two DNA sequences using a variation [Myers and Miller 1988] of the classic Smith-Waterman algorithm with the aim of finding the *optimal alignment* and the degree of similarity between a pair of sequences. A characteristic of this problem class is that the memory requirement grows quadratically with respect the average lengths of the two sequences being aligned. For reference, aligning sequences of around 60,000 residues with MASA-OpenMP requires approximately 30 MiB, while sequence pairs of $\approx$ 120-, 480-, and 960-thousand consume 114 MiB, 1.7 GiB, and 6.8 GiB, respectively.

To avoid having to make a set of all-to-all comparisons, a typical biological experiment involves comparing the genome sequences of interest against a known reference one. Nevertheless, the number of genomes of interest may be high, in the range of millions, for example, in the case of SARS-CoV-2. Thus, we define a MASA alignment experiment as being composed of a set W of independent tasks (each producing a pair-wise alignment of biological sequences of approximately similar sizes, using MASA-OpenMP) to be executed on a single cloud instance of a given type with P physical CPU cores and an available memory capacity of M. The goal is to find the shortest execution time, and thus its lowest cost, for an experiment of size $|W|$ given M, without preempting tasks.

Our work on addressing Issue (a), concluded that the degree of over-committing CPUs with concurrent MASA processes depends on the processor architecture of the instance in question (e.g., if symmetric multi-threading is available or not). Furthermore, for a sequence alignment experiment, exploiting CPU cores following Gustafson's fixed time model (scaling the number of concurrent MASA processes) is more efficient than following Amdahl's fixed workload model of parallelizing the execution of each MASA process to occupy the available cores.

3. Towards a scheduling model for MASA based DNA sequence alignments

While executing sequence alignments on traditional CPU cloud instances has been the focus of previous studies, e.g., [Carvalho et al. 2021], [Teylo et al. 2021], to the best of our knowledge, our work is one of the first to focus on also improving memory utilization to reduce cloud costs and increase throughput, in this context and for applications with a similar memory profiles. This is possible by scheduling alignment tasks so that the amount of under utilized memory during an execution is reduced. Figure 1 presents the memory consumption (in blue) as seen from the point of view of the virtual environment for a single MASA process carrying out an alignment of a pair sequences of approxi-

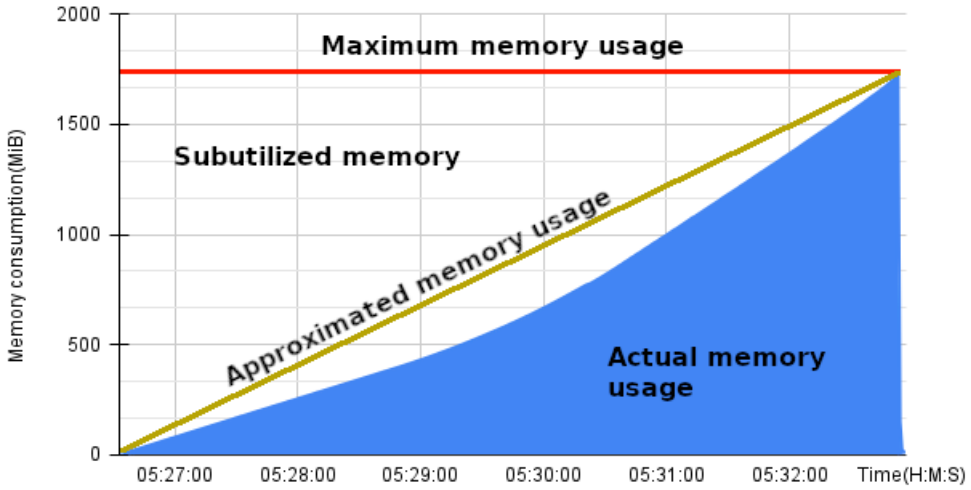

Figure 1. Memory consumption profile for the comparison of DNA sequences with lengths around 480 thousand nucleotides using MASA-OpenMP.

mately 480 thousand nucleotides. Notice that the amount of memory in use increases monotonically until the program nears its termination. The profile is similar when executing MASA-OpenMP with different sequences and lengths – longer sequences having longer execution times and higher memory requirements – in both VMs and containers. Furthermore, this pattern repeats even when using the option MASA has to limit its maximum memory consumption. While the red line identifies the maximum or peak memory usage, which occurs towards the end of the execution, the diagonal brown line represents a crude approximation to the actual memory usage that we used to develop our task scheduling model. Given the difference between these lines, on average, more than half of the memory that would be reserved to execute this MASA task is left idle over the period of its execution. Our goal is to exploit this sub-utilized memory. Simply executing more MASA tasks simultaneously, as many previous papers have done, will not achieve this.

3.1. Delaying MASA tasks improves memory utilization

Since a MASA experiment typically consists of a set of W independent alignments of sequences of similar lengths, the scheduling model assumes that each alignment task in W has the same execution time T and maximum memory consumption k. Therefore, if $M \geq k$ and the average memory utilization is at most 50% (as seen in Figure 1), an initial lower bound on the parallel execution time of a MASA experiment would be $ET_{LB} = |W| \times 50\% \times \frac{k}{M} \times T = \frac{|W|.k.T}{2 \times M}$, assuming a sufficient number of available processors. The aim is to find the schedule with a makespan as close as possible to this.

The total Memory Usage (TMU) of a task, the blue area in Figure 1, can be approximated to $\frac{k \times T}{2}$. As the tasks in W are independent, the experiment can be sub-divided into N equally (as possible) sized groups. Within each group or *workflow*, the tasks are executed sequentially, although for generality they need not be scheduled immediately after one another, thus the *cycle* between task start times, C, will be $e \times T$ with $e \geq 1$.

The value of N, the number of workflows in the schedule, is limited by the number of available CPUs, P, and the amount of available memory, M. Each workflow requires at least one CPU, so a quantity $N \times k$ of memory is sufficient to execute N workflows

simultaneously, with an execution time $ET_s(N) = T + (\lceil \frac{|W|}{N} \rceil - 1) \times C$. On the other hand, however, k is the minimum amount of memory required to execute $|W|$ tasks, although completely sequentially. Thus, the scheduling goal is to find the minimum amount of memory, M_{min}, necessary to allow the execution of N workflows in parallel.

We know that the average Memory Usage per cycle for N simultaneous workflows is $N \times \frac{TMU}{C}$. Also, when a task terminates it frees up k of memory, so the smallest obtainable oscillation in the memory consumption around the average usage is $\pm\frac{k}{2}$. Thus, the minimum amount of memory, M_{min}, for any schedule of degree N is $\frac{k \times T \times N}{2 \times C} + \frac{k}{2} = \frac{k}{2}(\frac{N \times T}{C} + 1)$. To find a N-parallel schedule that meets this memory requirement, we define for each respective workflow, a starting delay of $T \times \delta_{1..N}$ in relation to the beginning of the experiment's execution, where $0 \leq \delta_1 \leq \delta_j \leq \delta_N < \frac{C}{T}$, $\forall\ j$ such that $1 < j < N$. We omit the proof due to space constraints, but given only one task terminates at a time and all workflows have the same value of e, the schedule requires that $\delta_i < \delta_{i+1}$ and $(\delta_{i+1} - \delta_i) \times T = \frac{C}{N}$. That is, each workflow should start at a distinct time, equally spaced at a constant interval that is inversely proportional to number of workflows to be initiated during a cycle. Using the equation for pipelining, the execution time of the schedule that meets the minimum memory requirement $ET_p(N) = (|W| - 1) \times \frac{C}{N} + T$. As the value of $|W|$ increases, $ET_p(N)$ tends asymptotically to $\frac{|W| \times C}{N}$. After substituting M_{min} for M in ET_{LB}, one can see that $ET_p(N)$ is almost equivalent to $ET_{LB} = \frac{|W| \times C}{N+e}$. As N increases, $\frac{ET_p(N)}{ET_{LB}} = \frac{N+1}{N}$ approximates to 1, given $e = 1$, as does $\frac{ET_p(N)}{ET_s(N)} = \frac{|W|-1+N}{|W|}$, for large $|W|$. In other words, for large experiments, the pipelined schedule can be as fast as commonly adopted greedy job scheduling policies while requiring only $\frac{N+1}{2 \times N}$ of the memory.

4. Experimental Evaluation on Amazon's AWS EC2

Amazon's AWS EC2 cloud provides a wide variety of services based on different virtualization technologies, including VMs, containers and serverless environments. The experiments here used on-demand VM instances, with pre-configured combinations of hardware, on remote servers localized in the us-east-1 region. One common trend across instance types, markets and virtualization services is that the amount of memory available per virtual instance is proportional to the number of vCPUs and scales linearly from smaller to the larger instances. Since MASA's memory consumption scales quadratically with the length of the input sequences, schedules will have to be aware of which of the instance's resources (CPUs or memory) are being sub-utilized and optimize accordingly.

We present a small set of experiments with the aim of briefly illustrating the potential practical effectiveness of the proposed pipeline or delayed scheduling approach. With an eye on applying this technique across cloud virtualization services, in particular, container services and FaaS solutions, we have implemented our own container-based Sequence Alignment service on top of IaaS VM instances which is managed by VEMoC [Nicodemus et al. 2020]. One drawback of this vertical memory elasticity controller is that it employs container suspension to be able to maximize memory utilization. For MASA, theory suggests that the best non-preemptive schedule can be better. Thus, a container image has been created with MASA and all its dependencies which also allows the container's memory consumption metrics to be collected every second by VEMoC.

Each MASA experiment is executed in a single container deployed in an EC2 c5.12xlarge VM instance with $P = 24$ CPUs and $M = 96$ GiB. The workload W is a

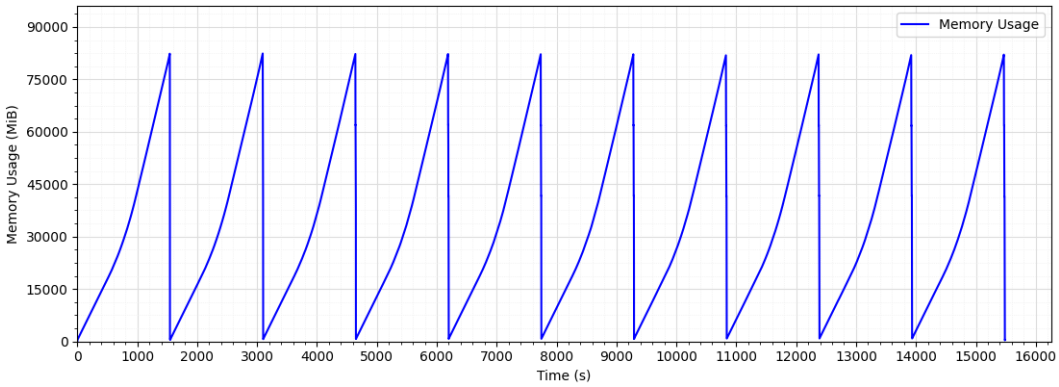

Figure 2. Memory consumption for the execution of 12 simultaneous workflows.

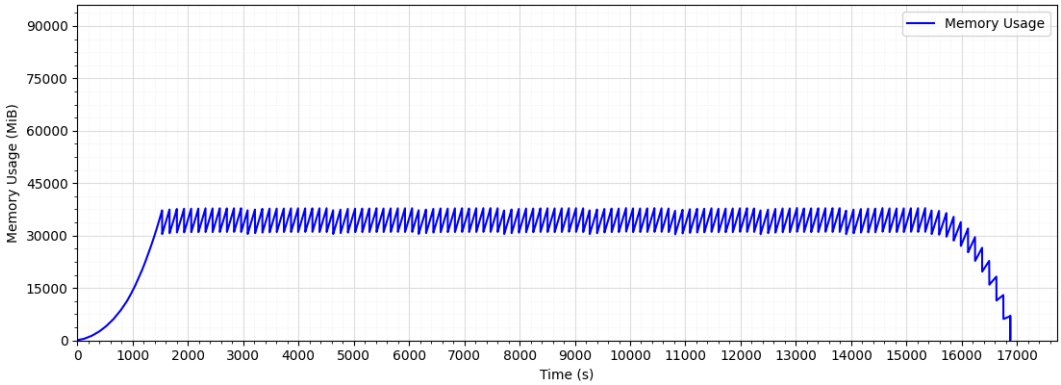

Figure 3. Memory consumption for the execution of 12 workflows with 10 tasks each and respective starting delay increments of 128.83 seconds.

set of 120 alignments of sequence pairs with lengths around 960 thousand DNA bases. The execution time of each alignment is $T \approx 1546$ seconds with k, its peak memory usage, being 6862 MiB. The workload is divided into N workflows of S tasks ($|W| = N \times S$). The experiments varied N and their respective starting time δ_i and present the total execution time, peak and average memory usage as well as a plot of the memory consumption of the container during its execution. The financial costs of the schedules are proportional to their execution times (ignoring the $60s$ minimum VM billing cost).

Figure 2 illustrates the execution of a MASA experiment with $N = 12$ simultaneous workflows of $S = 10$ tasks each, which were all started at the same time, i.e., $\delta_i = 0,\ \forall i \in \{1 \dots N\}$. This is the most common schedule found in the literature, e.g., [Carvalho et al. 2021], [Teylo et al. 2021], as it leads to shortest execution time for N. As a consequence, during each period T, the memory consumption of the 12 workflows rises monotonically to a value around $12 \times k + q$ (where q is the memory of the container), and decreases back to q. The experiment executed in 15485 seconds with a peak memory consumption of 82407 MiB and an average consumption of 34435 MiB.

In order to reduce the memory required to realize the same MASA experiment, a pipeline approach is applied. The only change to the schedule is to delay the start times of each of the subsequent 11 workflows by additional intervals of $\frac{C}{N}$, i.e., 128.83 seconds, given $C = T$ in the shortest schedule. The behavior presented in Figure 3 shows that while the number of workflows is the same, the peak memory consumption has decreased

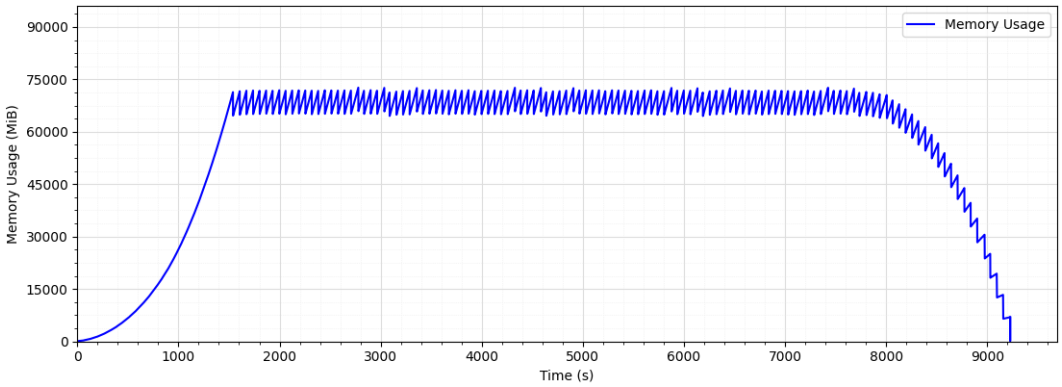

Figure 4. Memory consumption for the execution of 24 workflows with 5 tasks each and respective starting delay increments of 64.41 seconds.

54% to 37879 MiB, at the cost of a 9% increase in the execution time, i.e., 1394 seconds slower. This slowdown is caused by the pipeline, the last of the 12 workflows begins its execution approximately $11 \times 128.83s$ after the first. For larger values of $|W|$, this percentage overhead will continue to fall while the memory utilization, at 80%, will rise. This execution is significantly cheaper under FaaS, given the financial cost depends on both the total execution time and the memory consumption.

In the final experiment, W has now been divided into 24 workflows of 5 tasks each. With a smaller delay interval of $\frac{T}{24} = 64.41s$, the schedule executes the same number of comparisons with a reduction in the execution time of 40% for this experiment size. This is less than the upper bound of 50% due to the pipeline's prolog and epilog stages as seen by the rising curve at the beginning and falling one at the end in Figure 4. The average memory usage has increased to 57443 MiB, 79% of the peak 72632 MiB, again showing an improvement in memory utilization, and which will increase with higher $|W|$ values.

5. Related Work

Tools exist for container/VM auto-tuning with vertical memory elasticity to adjust the allocation of available memory. When the memory demand cannot be met by the host, some tools opt to suspend containers/VMs to avoid starvation and performance degradation of other running containers/VMs. The suspended containers/VMs are restored when memory becomes available again, if not on the same host, on another with spare capacity. [Qazi 2019] considers the execution of containers in a number of heterogeneous VMs with different memory configurations on Amazon EC2. If the memory allocated to a container becomes insufficient, their strategy suspends and checkpoints it, and then re-starts the container in another instance with sufficient memory. The work of [Nicodemus et al. 2020] manages container memory allocations by dynamically adjusting memory limits in conjunction with strategies that combine pausing and preempting containers when short of resources. As a result, improvements in memory utilization, cloud costs, and job throughput can be provided in comparison to existing commercial orchestration tools. While these tools are beneficial, and in theory very efficient, the overheads of suspending VMs or containers increase with the amount of memory involved. For big data applications, this can be prohibitively expensive both in terms of time and financially. Therefore, this paper proposes a solution (although for a specific use case, other applications also have

similar memory consumption behaviors) without these overheads while optimizing memory requirements. Given predictable execution times and memory demands, schedules that optimize the CPU and memory utilization of an instance can be provided.

6. Conclusions and Future Work

Taking full advantage of an instance's resources benefits both: the user, by providing time and costs advantages, and; cloud providers, promoting more effective resource consolidation, improving availability and reducing energy costs. This work has shown that there exist applications where delaying task execution can improve throughput. Given the increasing impact memory has on performance, and that memory demands continue to increase, effective strategies to minimize requirements and maximize memory utilization are fundamental. Ongoing work includes devising related preemption-based schedules, and also evaluating both schemes against solutions for different cost models: the spot market, elastic container services and FaaS/serverless computing, where to date, resource scheduling within containers (and function aggregation) appears to have been ignored.

Acknowledgments

The authors would like to thank Dr. Olivier Beaumont and Dr. Lionel Eyraud-Dubois of Inria Bordeaux Sud-Ouest Research Center in France for their collaboration in this work.

References

Brunetta, J. R. and Borin, E. (2019). Selecting efficient cloud resources for HPC workloads. In *2019 IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC)*, page 155–164. ACM.

Calatrava, A., Romero, E., Moltó, G., Caballer, M., and Alonso, J. M. (2016). Self-managed cost-efficient virtual elastic clusters on hybrid cloud infrastructures. *Future Generation Computer Systems*, 61:13–25.

Carvalho, L., Melo, A., and Araujo, A. (2021). A framework for executing protein sequence alignment in cloud computing services. In *Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 48–59, Porto Alegre, RS, Brasil. SBC.

De O. Sanders, E. F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., and De Melo, A. C. M. A. (2016). MASA: A multiplatform architecture for sequence aligners with block pruning. *ACM Transactions on Parallel Computing*, 2(4).

Myers, E. W. and Miller, W. (1988). Optimal alignments in linear space. *Bioinformatics*, 4(1):11–17.

Nicodemus, C. H. Z., Boeres, C., and Rebello, V. E. F. (2020). Managing vertical memory elasticity in containers. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 132–142. IEEE.

Qazi, K. (2019). Vertelas - Automated user-controlled vertical elasticity in commercial clouds. In *2019 4th Int. Conf. on Computing, Comm. and Security (ICCCS)*, pages 1–8.

Teylo, L., Nunes, A. L., Melo, A. C. M. A., Boeres, C., de A. Drummond, L. M., and Martins, N. F. (2021). Comparing SARS-CoV-2 sequences using a commercial cloud with a spot instance based dynamic scheduler. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 247–256.