Towards Optimizing Cost and Performance for Parallel Workloads in Cloud Computing

William Maas[1], Fábio Diniz Rossi[2], Marcelo C. Luizelli[3], Philippe O. A. Navaux[1] and Arthur F. Lorenzon[1]

[1]*Institute of Informatics, Federal University of Rio Grande do Sul, Brazil*
[2]*Campus Alegrete, Federal Institute Farroupilha, Brazil*
[3]*Campus Alegrete, Federal University of Pampa, Brazil*
fabio.rossi@iffarroupilha.edu.br, marceloluizelli@unipampa.edu.br, {wbmaas, navaux, aflorenzon}@inf.ufrgs.br

Keywords: Parallel Computing, Cloud Computing, Cost, Performance.

Abstract: The growing popularity of data-intensive applications in cloud computing necessitates a cost-effective approach to harnessing distributed processing capabilities. However, the wide variety of instance types and configurations available can lead to substantial costs if not selected based on the parallel workload requirements, such as CPU and memory usage and thread scalability. This situation underscores the need for scalable and economical infrastructure that effectively balances parallel workloads' performance and expenses. To tackle this issue, this paper comprehensively analyzes performance, costs, and trade-offs across 18 parallel workloads utilizing 52 high-performance computing (HPC) optimized instances from three leading cloud providers. Our findings reveal that no single instance type can simultaneously offer the best performance and the lowest costs across all workloads. Instances that excel in performance do not always provide the best cost efficiency, while the most affordable options often struggle to deliver adequate performance. Moreover, we demonstrate that by customizing instance selection to meet the specific needs of each workload, users can achieve up to 81.2% higher performance and reduce costs by 95.5% compared to using a single instance type for every workload.

1 INTRODUCTION

Applications that handle massive data volumes, such as machine learning, data mining, and health simulations, are becoming increasingly prominent in cloud computing (Navaux et al., 2023). These workloads leverage the distributed processing capabilities of cloud servers to execute high-performance computing (HPC) tasks to reduce the execution time of the entire workload. However, the extensive variety of available instance types and configurations poses a challenge, often leading to substantial operational costs. This issue is exacerbated by the slowing efficiency gains in newer hardware generations resulting from the end of Dennard scaling (Baccarani et al., 1984). As a result, cloud servers require a scalable and cost-effective infrastructure to efficiently run applications, providing resources on demand via the Internet. Consequently, choosing the most suitable instance type for a given workload is critical for optimizing performance and cost-efficiency.

However, optimizing the cost-performance balance for parallel workloads in the cloud is challenging due to the wide variety of instance types, each with distinct computational capabilities and pricing models. AWS, Google Cloud, and Azure offer general-purpose and compute-optimized instances tailored for specific workloads (AWS, 2024). However, selecting the best instance becomes more complex when workloads with different characteristics must be matched to the most suitable option across multiple cloud platforms. Moreover, parallel execution improves performance by utilizing available computational resources, but gains are not always proportional to core count. Off-chip bus saturation, shared memory contention, and synchronization overhead (Suleman et al., 2008; de Lima et al., 2024) can limit scalability, making higher core counts ineffective for some workloads. For instance, increasing from 32 to 64 vCPUs may not accelerate machine learning tasks. Achieving an optimal cost-performance trade-off requires selecting instances based on workload-specific thread-level parallelism (TLP). However, varying workload behaviors and memory access patterns mean an instance efficient for one task may be suboptimal for another.

Given the scenario discussed above, we present

Maas, W., Rossi, F. D., Luizelli, M. C., Navaux, P. O. A. and Lorenzon, A. F.
Towards Optimizing Cost and Performance for Parallel Workloads in Cloud Computing.
DOI: 10.5220/0013421600003950

In *Proceedings of the 15th International Conference on Cloud Computing and Services Science (CLOSER 2025)*, pages 231-238
ISBN: 978-989-758-747-4; ISSN: 2184-5042

a comprehensive cost-performance evaluation when running eighteen parallel workloads with varying memory and CPU usage and TLP degree requirements across 52 HPC- and Compute-optimized instances of different types from three major cloud providers: Google Cloud Platform (GCP), Amazon Web Services (AWS), and Microsoft Azure. Through this extensive set of experiments, we show that: (*i*) Given the characteristics of each parallel workload regarding CPU and memory usage, no single instance can provide the best performance, the lowest cost, and the best trade-off between both across all eighteen parallel workloads simultaneously. (*ii*) Selecting the most suitable instance for each workload based on its specific characteristics regarding thread scalability, we show that it is possible to achieve a 35.7% improvement in performance and a 6.5% reduction in costs compared to using the same instance type that delivers the best results for the entire set of workloads. (*iii*) The provided Microsoft Copilot tool in Azure to recommend the best instance for running each parallel workload cannot find the most suitable instance as it does not consider the intrinsic characteristics of parallel applications, such as thread scalability.

2 BACKGROUND

2.1 Cloud Computing

Cloud computing has become the standard for running applications, offering on-demand resources (Liu et al., 2012). Initially, compute-intensive applications like Big Data faced challenges due to hypervisor overhead, despite efforts to ensure elasticity and availability (Barham et al., 2003). To address this, lightweight container technologies like Docker emerged, offering near-native performance. Docker has become the leading framework for developing and running containerized applications, encapsulating necessary components for efficient execution. Docker is widely used for deploying parallel workloads, providing isolated environments with all dependencies. Its lightweight nature minimizes virtualization overhead, making it ideal for HPC applications. This enables efficient scaling across multiple nodes, optimizing cloud-based HPC infrastructures. As a result, companies like NVIDIA, AMD, and Intel use Docker to optimize workloads for cloud execution. Cloud providers offer instance types tailored to specific workloads. Compute-optimized instances, such as AWS's C-series and Azure's F-series, prioritize CPU performance for data processing but come at a higher cost. HPC-optimized instances, like AWS's H-series and Azure's HBv3-series, feature advanced architectures and high-speed interconnects, offering maximum performance for scientific simulations and large-scale data analysis.

2.2 Thread Scalability

When executing parallel workloads in cloud environments, developers often maximize resource use, including vCPUs and cache memory. However, studies indicate that this approach may not yield optimal performance (Suleman et al., 2008; Subramanian et al., 2013). Performance limitations arise from software and hardware factors such as off-chip bus saturation, concurrent shared memory accesses, and data synchronization. **Off-Chip Bus Saturation:** Applications requiring frequent memory access may experience scalability issues when memory bandwidth bottlenecks. As more threads are added, the connection's bandwidth remains limited by I/O pins (Ham et al., 2013), leading to increased power consumption without performance gains. **Concurrent Shared Memory Accesses:** Performance can degrade when multiple threads frequently access shared memory locations. Since these accesses occur in distant memory areas like last-level cache, they introduce higher latency than private caches, potentially creating bottlenecks (Subramanian et al., 2013). **Data Synchronization:** Threads accessing shared variables require synchronization to maintain correctness. However, only one thread can access a critical section at a time, forcing others to wait. As the number of threads increases, serialization effects become more pronounced, limiting application scalability (Suleman et al., 2008).

2.3 Related Work

Recent studies have explored parallel computing in cloud environments and parallel workload optimization. Ekanayake et al. (Ekanayake and Fox, 2010) examined cloud-based parallel processing for scientific applications, highlighting distributed computing's ability to handle intensive workloads. Rak et al. (Rak et al., 2013) developed a cost-prediction method combining benchmarking and simulation, aiding in cloud cost management. Rathnayake et al. (Rathnayake et al., 2017) introduced CELIA, an analytical model optimizing cloud resource allocation by balancing execution time and costs. Wan et al. (Wan et al., 2020) proposed a cost-performance model using queuing systems to enhance workload management. Several works focus on parallel workload optimization. Zhang et al. (Zhang et al., 2020) de-

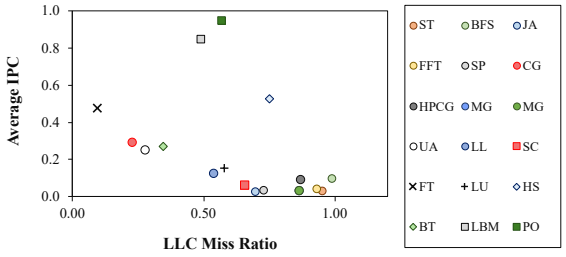

Figure 1: Characteristics of each parallel workload regarding the average IPC and LLC Miss Ratio on the *AWS c6a 32xlarge* instance type.

veloped the EPRD algorithm to minimize scheduling durations for precedence-constrained tasks. Li et al. (Li et al., 2020) introduced AdaptFR, accelerating distributed machine learning training through scalable resource allocation. Ohue et al. (Ohue et al., 2020) demonstrated cloud computing's adaptability for protein-protein docking simulations. Bhardwaj et al. (Bhardwaj et al., 2020) tackled task scheduling in parallel cloud jobs, improving efficiency. Haussmann et al. (Haussmann et al., 2019b) evaluated cost-performance trade-offs using reserved and volatile processors. Their later work (Haussmann et al., 2019a) optimized processor counts for irregular parallel applications. They also introduced an elasticity description language to enhance resource allocation in cloud deployments (Haussmann et al., 2020).

3 METHODOLOGY

3.1 Parallel Workloads

Eighteen parallel workloads from assorted benchmark suites already written in C/C++ and parallelized with OpenMP (OpenMulti Processing) API were selected for evaluation. **Seven** benchmarks are from the NAS Parallel Benchmark (Bailey et al., 1995), used to evaluate the performance of parallel hardware and software: *BT* (Block Tridiagonal), *CG* (Conjugate Gradient), *FT* (Fourier Transform), *LU* (LU Decomposition), *MG* (Multigrid), *SP* (Scalar Pentadiagonal), and *UA* (Unstructured Adaptive). **Three** applications from the Rodinia benchmark suite (Che et al., 2009), used to identify performance bottlenecks and compare different computing platforms: *HS* (HotSpot), *LUD* (LU Decomposition with Data Redistribution), and *SC* (Streamcluster). **Two** workloads from the Parboil suite (Stratton et al., 2012), a set of applications used to study the performance of throughput computing architectures: *BFS* (Breadth-First Search) and *LBM* (Lattice Boltzmann Method). **Six** from distinct domains: *FFT* (Fast Fourier Transform) (Brigham and Morrow, 1967), *HPCG* (High-Performance Conjugate Gradient)(Heroux et al., 2013), *JA* (Jacobi iteration), *LULESH* (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics) (Karlin et al., 2013), *PO* (Poisson Equation), and *ST* (Stream).

The applications were executed with the standard input set defined on each benchmark suite. We compiled each application with gcc/g++ 12.0, using the *-O3* optimization flag. The chosen benchmarks cover a wide range of parallel workloads when considering the average instructions per cycle (IPC) and last-level cache (LLC) memory behavior on the target architectures, as shown in Figure 1 for the AWS *c6a 32xlarge* instance type. We consider the IPC metric as it reflects how CPU- or memory-intensive each application is. On the other hand, the amount of LLC misses represents the number of DRAM memory accesses. Hence, the higher the LLC miss ratio, the more time the threads spend accessing data from the main memory, impacting its IPC.

3.2 Target Instances

We consider a set of fifty-two cloud instances from the compute- and HPC-optimized domains (Table 1). The selected instances consider different processors, including AMD, Intel, and ARM (*AWS Graviton*). In addition to choosing various instance types, we selected different configurations (i.e., variations in Table 1) within the same instance type, each with varying numbers of available vCPUs. This setup allows us to examine how performance and cost efficiency are impacted by scaling computational resources within the same instance family. For example, within the AWS *c7g* series, we considered instances with 8, 16, 32, 48, and 64 vCPUs to assess the ability of parallel applications to leverage additional cores and memory bandwidth efficiently. Additionally, all instances were configured to use the same operating system (*Ubuntu 22.04*) and kernel version. Our analysis used the on-demand pricing model to maintain consistency and comparability among various instance types and cloud providers.

3.3 Execution Environment

The process for running each workload on the cloud instance was as follows: first, the workload binary was placed within a Docker container to ensure isolated execution, regardless of the architecture. Then, the container was deployed for execution on the corresponding instance. During execution, the number of cores assigned to the container (and thus to the work-

Table 1: Characteristics of the Compute- and HPC-Optimized instances.

Instance Type	Processor	Variations	vCPU's	Cost per Hour (USD)
gcp-h3	Intel Xeon Platinum 8481C	std-88	88	4.93
gcp-c2	Intel Xeon Gold 6253CL	std-8; std-16; std-30; std-60	8; 16; 30; 60	0.34; 0.67; 1.25; 2.51
gcp-c2d	AMD EPYC 7B13	std-8; std-16; std-32; std-56; std-112	8; 16; 32; 56; 112	0.36; 0.73; 1,45; 2.54; 5.09
aws-c6a	AMD EPYC 7R13	2xlarge; 4xlarge; 8xlarge; 16xlarge; 32xlarge; 48xlarge	8; 16; 32; 64; 128; 192	0.31; 0.61; 1.22; 2.45; 4.90; 7.94
aws-c7i	Intel Xeon 8488C	2xlarge; 4xlarge; 8xlarge; 12xlarge; 16xlarge; 32xlarge; 48xlarge	8; 16; 32; 48; 64; 96; 192	0.36; 0.71; 1.43; 2.14; 2.86; 4.28; 8.57
aws-c7g	AWS Graviton3	2xlarge; 4xlarge; 8xlarge; 12xlarge; 16xlarge	8; 16; 32; 48; 64	0.29; 0.58; 1.16; 1.73; 2.31
aws-hpc7g	AWS Graviton3E	4xlarge; 8xlarge; 16xlarge	16; 32; 64	1.69
aws-hpc6a	AMD EPYC 7R13	48xlarge	96	2.88
aws-hpc6id	3rd Gen Intel Xeon Scalable	32xlarge	64	8.64
aws-hpc7a	AMD EPYC 9R14	12xlarge; 24xlarge; 48xlarge; 96xlarge	24; 48; 96; 192	7.20
azure-fsv2	Intel Xeon Platinum 8370C	f8s_v2; f16s_v2; f32s_v2; f64s_v2; f72s_v2	8; 16; 32; 64; 72	0.34; 0.68; 1.35; 2.71; 3.05
azure-hx176	AMD EPYC 9V33X	24rs; 48rs; 96rs; 144rs; 176	24; 48; 96; 144; 176	8.64
azure-hb176	AMD EPYC 9V33X	24rs_v4; 48rs_v4; 96rs_v4; 144rs_v4; 176rs_v4	24; 48; 96; 144; 176	7.20

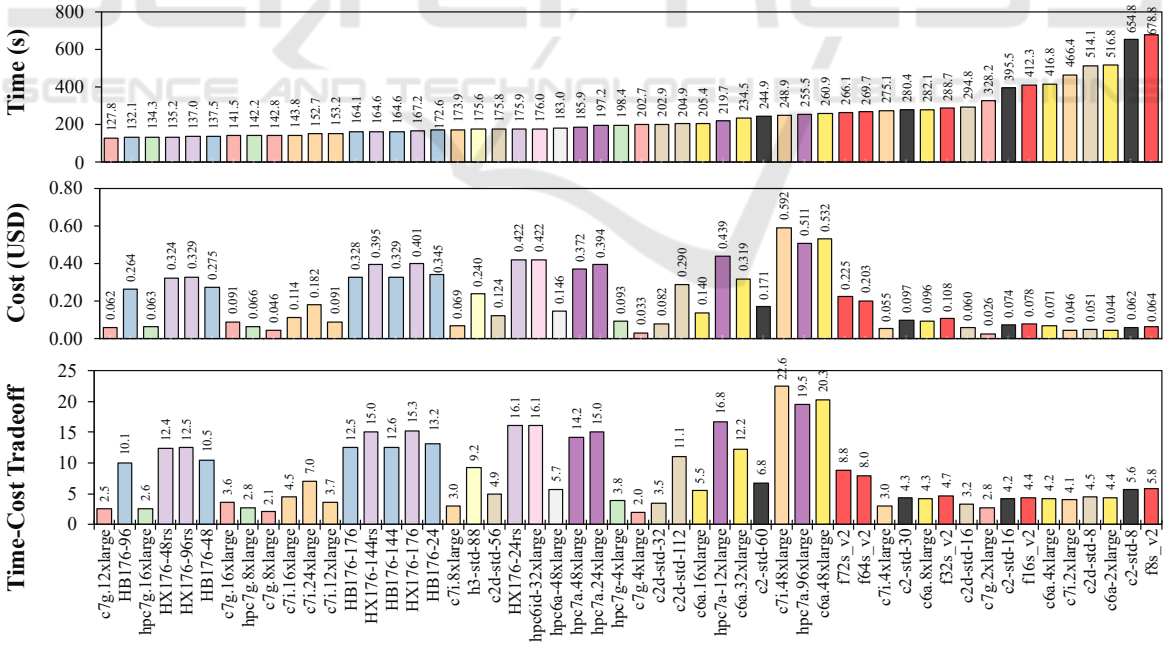

Figure 2: Total time (top), cost (middle), and the trade-off between both (bottom) to execute the batch of workloads on each cloud instance. Each set of bar colors represents a different instance type. The lower the value, the better.

load) was set to the number of system cores (*vCPUs* in the instance), which is the standard practice for executing parallel applications. To measure the execution time of each run, we used the `omp_get_wtime` function from OpenMP. The cost to execute each container (and therefore the workload) was calculated by

dividing the cost per hour (see Table 1) by the amount of time taken to execute it. Moreover, to calculate the trade-off between performance and cost, we considered the Euclidean distance from the pair of execution time and cost for each instance to the origin point, representing the ideal scenario of zero cost and zero execution time. This metric quantitatively measures how close each instance is to the optimal balance between cost and performance, enabling a fair comparison across all tested instances. The smaller the distance, the better the trade-off the instance achieves for the given workload. The results discussed next are the average of 30 executions of each container and instance type, with a standard deviation of less than 0.5%.

4 EXPERIMENTAL EVALUATION

4.1 Performance and Cost Evaluation

In this Section, we discuss the performance and cost of running the workload set on each target instance. Figure 2 shows the total execution time (top), cost (middle), and the balance between them (bottom) for running the eighteen workloads on every instance, respectively. All the results are sorted by the lowest execution time. Moreover, for all plots, lower values indicate a better outcome.

When processing all workloads in batch, the AWS *c7g.12xlarge* instance achieves the best performance, surpassing the Azure *hb176-96* by 3.3%. In contrast, the Azure *f8s_v2* is the worst-performing instance, increasing execution time by 81.1% due to its lower vCPU count (8 vs. 48 in *c7g.12xlarge*). However, *c7g.12xlarge* was not the most cost-effective option, costing 2.33× more than *c7g.2xlarge*, while the *c7g.2xlarge* was 95.6% cheaper than the *c7i.48xlarge*. Although *c7g.12xlarge* delivered the best performance and *c7g.2xlarge* the lowest cost, neither was optimal for every workload. Table 2 shows that no single instance consistently outperformed others. For instance, *BT* ran best on *hb176-144* with 144 vCPUs, *FFT* on *hpc7g.4xlarge* with 16 vCPUs, and *BFS* on *c6a-2xlarge* with 8 vCPUs due to thread scalability limitations. The *BFS* workload suffers from synchronization overhead, where adding more threads increases waiting time instead of improving performance (Suleman et al., 2008). Similarly, workloads like *FFT* and *LL* that require frequent memory access face off-chip bus saturation beyond a certain thread count. For *FFT*, optimal performance was achieved with 16 vCPUs on *m7g.4xlarge* and *hpc7g.4xlarge*. Increasing the number of vCPUs beyond this point led

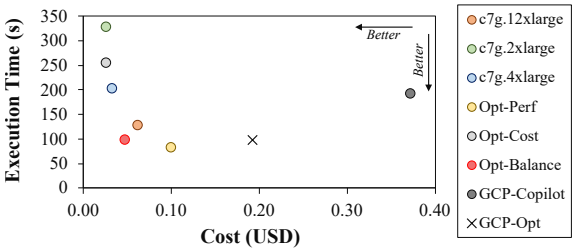

Figure 3: Performance and cost comparison between the best results found by the best instances for each metric and the strategies that optimize for each metric.

to a 5.6% execution time increase on *m7g.16xlarge* and 5.01% on *hpc7g.16xlarge* due to bandwidth limitations.

Furthermore, because threads communicate by accessing data in shared memory regions, the overhead imposed by the shared memory accesses may outweigh the gains achieved by the parallelism exploitation, as discussed in Section 2. This behavior was observed for workloads with high communication demands, including *SC* and *HPCG*. In the case of *HPCG*, the *hx176-48rs* with 48 *vCPUs* performed better than the *hx176* with 176 *vCPUs*. While we identified thread communication as the primary reason for this behavior, other factors such as thread scheduling, data affinity, or the inherent characteristics of the workload may also influence L3 cache performance. For instance, a workload with a high rate of private access to L1 and L2 caches may also result in a rise in L3 accesses. There are also scenarios of workloads that were able to scale by increasing the number of hardware resources in the instances, such as *JA*, *LUD*, and *LBM* (refer to Table 2).

4.2 Optimizing the Trade-off Between Performance and Cost

In the previous section, we evaluated the performance and cost of each instance when the entire workload set is executed on the same instance. However, as shown in Table 1, because no single instance can deliver the lowest cost and execution time for all workloads simultaneously, determining the ideal instance to execute each workload based on its characteristics is essential to improve performance and reduce costs. Hence, in this section, we compare the results of the best instances for the overall performance (*c7g.12xlarge*), cost (*c7g.2xlarge*), and the trade-off between cost and performance (*c7g.4xlarge*) to the following strategies: **Opt-Perf**: considers the total execution time to execute the batch of workloads by running each workload on the instance that delivers the best performance, as shown in Table 2.

Table 2: Ideal compute- and HPC-optimized instances found by an exhaustive search to execute each parallel workload. Each instance has associated with it the time (in seconds) and cost (in USD) to execute the workload according to the target metric.

	Opt-Performance	**Opt-Cost (USD)**	**Opt-Balance**
FFT	hpc7g.4xlarge (19.9s)	c7g.2xlarge (0.00162)	c7g.8xlarge (*20.0s - 0.00642*)
HPCG	hx176-48rs (1.9s)	c7g.2xlarge (0.00051)	hb176-48 (*2.0s - 0.00398*)
JA	hb176 (0.4s)	c7g.2xlarge (0.00051)	hb176 (*0.4s - 0.00078*)
LL	c7i.8xlarge (19.0s)	c7g.2xlarge (0.00259)	c7i.8xlarge (*19.0s - 0.00755*)
PO	c7g.12xlarge (0.18s)	c7g.2xlarge (0.00004)	c7g.12xlarge (*0.18s - 0.00009*)
ST	hpc7g.16xlarge (0.06s)	c7g.8xlarge (0.00003)	hpc7g.16xlarge (*0.06s - 0.00003*)
BT	hb176-144 (2.0s)	c7g.2xlarge (0.00185)	hpc7g.16xlarge (*3.9s - 0.00186*)
CG	hpc7a.96xlarge (0.4s)	c7g.2xlarge (0.00029)	hpc6a (*0.6s - 0.00046*)
FT	hb176-144 (0.2s)	c7g.2xlarge (0.00028)	hpc7g.16xlarge (*0.7s - 0.00033*)
LU	hb176-144 (2.1s)	c7g.2xlarge (0.00094)	hpc7g.16xlarge (*2.5s - 0.00117*)
MG	hb176-144 (0.5s)	c7g.2xlarge (0.00016)	c7g.8xlarge (*1.2s - 0.00037*)
SP	hx176-144rs (1.5s)	c7g.2xlarge (0.00080)	hpc6a (*2.7s - 0.00215*)
UA	hx176-96rs (2.1s)	c7g.2xlarge (0.00132)	hpc7g.16xlarge (*3.1s - 0.00145*)
HS	c7g.16xlarge (3.1s)	c7g.2xlarge (0.00101)	hpc7g.16xlarge (*3.1s - 0.00146*)
LUD	hb176 (2.2s)	c7g.2xlarge (0.00164)	c7g.8xlarge (*5.3s - 0.00170*)
SC	hb176 (19.3s)	c7g.2xlarge (0.00614)	c7g.12xlarge (*24.5s - 0.01179*)
BFS	c6a.2xlarge (2.08s)	c6a.2xlarge (0.00018)	c6a.2xlarge (*2.08s - 0.00018*)
LBM	hx176-96rs (5.5s)	hpc6a (0.00561)	hpc6a (*7.0s - 0.00561*)

Opt-Cost: the same as the previous, but considering the Cost as the optimization metric. **Opt-Balance**: the same as the *Opt-Perf*, but considering the balance between cost and performance. **GCP-Copilot**: employs the Microsoft Copilot in Azure to find the best instance type for each workload. The provided prompt to Copilot was: `Which VM size will best suit for the workload with IPC = XX, LLC Misses Ratio = XX, and TLP degree = XX?`, where `XX` was replaced by the values shown in Figure 1. Table 3 depicts the instances recommended to execute each workload and compares it to the ones that deliver the best performance found by an exhaustive search (*GCP-Opt*).

It is important to note that we could not compare our results with the recommendation tools provided by AWS and GCP due to the following reasons: (*i*) The AWS Compute Optimizer generates recommendations by analyzing resource specifications and utilization metrics collected through Amazon CloudWatch over a 14-day period, making it unsuitable for scenarios lacking historical usage data ((AWS), 2024). (*ii*) Similarly, the machine-type recommendations from GCP rely on system metrics gathered by the Cloud Monitoring service during the previous 8 days, which also requires prior resource utilization data (Cloud, 2024). These constraints made integrating these tools into our evaluation impractical, as they depend on prior workload execution history, which was not the case in our experimental setup.

Figure 3 depicts the results for executing all workloads with the discussed instances and strategies. Each mark in the plot represents the execution time (*y-axis*) and cost (*x-axis*) for a given instance or strategy. The closer the point is to the origin (*0*, *0*), the better the balance between performance and cost. When prioritizing performance, using the best instance to execute each workload (*Opt-Perf*) yields 35.5% of performance improvements compared to the instance that delivers the best overall performance for all workloads (*c7g.12xlarge*). When the cost is considered as the target optimization metric, selecting the ideal instance to execute each workload (*Opt-Cost*) leads the user to save 4% compared to the best single instance (*c7g.2xlarge*) while also reducing the total execution time by 32.5%. When both performance and cost are equally important for evaluation, choosing the ideal instance to execute each workload brings performance improvements and a lower cost to execute the entire set of applications: *Opt-Balance* is 2.06 times faster while it is 1.45% more expensive than if the application was executed on the single instance that provides the best tradeoff between cost and performance.

Regarding Microsoft Copilot in Azure Cloud, it failed to select optimal instances for parallel workloads because it does not integrate critical metrics like TLP Degree, average IPC, and LLC Miss Ratio, which are essential for characterizing inter-thread communication and memory access patterns. Copilot relies mainly on historical data and general resource usage, neglecting detailed microarchitectural features

Table 3: Instances recommended by the Microsoft Azure Copilot compared to the instance that delivered the best outcome, found by an exhaustive search.

	Microsoft Azure Copilot	GCP-Opt
FFT	*HB176-144*	*hb176-24*
HPCG	*HB176-144*	*hx176-48rs*
JA	*f32s_v2*	*HB176-24*
LL	*hx176-144rs*	*hb176-24*
PO	*hb176-24*	*NB176-48*
ST	*hx176-48rs*	*HB176-96*
BT	*hx176rs-176*	*HB176-144*
CG	*hx176-96rs*	*hx176rs-176*
FT	*f32s_v2*	*HB176-144*
LU	*HB176-144*	*HB176-144*
MG	*hx176-96rs*	*HB176-144*
SP	*HB176-24*	*hx176-144rs*
UA	*f64s_v2*	*hx176-96rs*
HS	*HB176-96*	*HB176-96*
LUD	*HB176-144*	*HB176-176*
SC	*hx176-144rs*	*HB176-176*
BFS	*NB176-48*	*f8s_v2*
LBM	*HB176-96*	*hx176-96rs*

and workload-specific requirements. As a result, it failed to identify instances that optimize performance, cost, or both. Comparing GCP-Copilot with selecting the best GCP instance for each workload (GCP-Opt), Copilot was 93% more expensive and 96% slower, emphasizing the need to incorporate thread scalability metrics in its recommendation model.

5 CONCLUSION

Optimizing performance and cost for parallel workloads in cloud computing is crucial for handling data-intensive applications efficiently. Cloud providers like AWS, GCP, and Azure offer various HPC-optimized instances, each with unique cost and performance characteristics. However, selecting the optimal instance type based on CPU utilization, memory needs, and thread scalability remains challenging. This study analyzed 18 parallel workloads on 52 HPC instances to evaluate performance-cost trade-offs. Results show that no single instance type is universally optimal. High-performance instances excel in compute-heavy tasks but are not always cost-efficient, while lower-cost options often fail to meet performance needs. By matching instances to workload requirements, users achieved up to 81.2% better performance and 95.5% cost savings over a generic approach. Future work will explore GPU-based instances and other pricing models to refine cost-performance optimization strategies in cloud environments.

ACKNOWLEDGEMENTS

This study was financed in part by the CAPES - Finance Code 001, FAPERGS - PqG 24/2551-0001388-1, and CNPq.

REFERENCES

AWS (2024). Amazon ec2. https://aws.amazon.com/ec2/. [Online; accessed 19-February-2024].

(AWS), A. W. S. (2024). What is aws compute optimizer? Accessed: 2024-12-24.

Baccarani, G., Wordeman, M. R., and Dennard, R. H. (1984). Generalized scaling theory and its application to a $1/4$ micrometer mosfet design. *IEEE Transactions on Electron Devices*, 31(4):452–462.

Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., and Yarrow, M. (1995). The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177.

Bhardwaj, A. K., Gajpal, Y., Surti, C., and Gill, S. S. (2020). Heart: Unrelated parallel machines problem with precedence constraints for task scheduling in cloud computing using heuristic and meta-heuristic algorithms. *Software: Practice and Experience*, 50(12):2231–2251.

Brigham, E. O. and Morrow, R. E. (1967). The fast fourier transform. *IEEE Spectrum*, 4(12):63–70.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54.

Cloud, G. (2024). Apply machine type recommendations for instances. Accessed: 2024-12-24.

de Lima, E. C., Rossi, F. D., Luizelli, M. C., Calheiros, R. N., and Lorenzon, A. F. (2024). A neural network framework for optimizing parallel computing in cloud servers. *Journal of Systems Architecture*, 150:103131.

Ekanayake, J. and Fox, G. (2010). High performance parallel computing with clouds and cloud technologies. In Avresky, D. R., Diaz, M., Bode, A., Ciciani, B., and Dekel, E., editors, *Cloud Computing*, pages 20–38, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ham, T. J., Chelepalli, B. K., Xue, N., and Lee, B. C. (2013). Disintegrated control for energy-efficient and heterogeneous memory systems. In *IEEE HPCA*, pages 424–435.

Haussmann, J., Blochinger, W., and Kuechlin, W. (2019a). Cost-efficient parallel processing of irregularly structured problems in cloud computing environments. *Cluster Computing*, 22(3):887–909.

Haussmann, J., Blochinger, W., and Kuechlin, W. (2019b). Cost-optimized parallel computations using volatile cloud resources. In Djemame, K., Altmann, J., Bañares, J. Á., Agmon Ben-Yehuda, O., and Naldi, M., editors, *Economics of Grids, Clouds, Systems, and Services*, pages 45–53, Cham. Springer International Publishing.

Haussmann, J., Blochinger, W., and Kuechlin, W. (2020). An elasticity description language for task-parallel cloud applications. In *CLOSER*, pages 473–481.

Heroux, M. A., Dongarra, J., and Luszczek, P. (2013). Hpcg benchmark technical specification.

Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. (2013). Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)*, Boston, USA.

Li, M., Zhang, J., Wan, J., Ren, Y., Zhou, L., Wu, B., Yang, R., and Wang, J. (2020). Distributed machine learning load balancing strategy in cloud computing services. *Wireless Networks*, 26:5517–5533.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2012). *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology*. CreateSpace Independent Publishing Platform, USA.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

Ohue, M., Aoyama, K., and Akiyama, Y. (2020). High-performance cloud computing for exhaustive protein-protein docking.

Rak, M., Cuomo, A., and Villano, U. (2013). Cost/performance evaluation for cloud applications using simulation. In *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 152–157.

Rathnayake, S., Loghin, D., and Teo, Y. M. (2017). Celia: Cost-time performance of elastic applications on cloud. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 342–351.

Stratton, J. A., Rodrigues, C., Sung, I.-J., Obeid, N., Chang, L.-W., Anssari, N., Liu, G. D., and Hwu, W.-m. W. (2012). Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127:27.

Subramanian, L., Seshadri, V., Kim, Y., Jaiyen, B., and Mutlu, O. (2013). MISE: Providing performance predictability and improving fairness in shared main memory systems. In *IEEE HPCA*, pages 639–650.

Suleman, M. A., Qureshi, M. K., and Patt, Y. N. (2008). Feedback-driven Threading: Power-efficient and High-performance Execution of Multi-threaded Workloads on CMPs. *SIGARCH Computer Architecture News*, 36(1):277–286.

Wan, B., Dang, J., Li, Z., Gong, H., Zhang, F., and Oh, S. (2020). Modeling analysis and cost-performance ratio optimization of virtual machine scheduling in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1518–1532.

Zhang, L., Zhou, L., and Salah, A. (2020). Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. *Information Sciences*, 531:31–46.