

Checkpointing Parallel Applications for Azure Spot Virtual Machines: Tools, Runtimes, and Storage Trade-offs

Lucas Serrano¹, Miguel de Lima¹, Alan L. Nunes¹, Artur Pessoa², André F. Benevides¹
Lúcia M. A. Drummond¹, Felipe A. Portella³, Paulo J. B. Estrela³, Renzo Q. Malini³

¹ Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

² Departamento de Engenharia de Produção – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

³ Petróleo Brasileiro S.A. (PETROBRAS) – Rio de Janeiro – RJ – Brazil

{lucas_serrano, miguel_flj, alan_lira, arturpessoa, andrefb}@id.uff.br
lucia@ic.uff.br

{felipeportella, paulo.estrela, renzo}@petrobras.com.br

Abstract. *Spot virtual machines (VMs) provide access to spare computing capacity at lower prices, but cloud providers may evict them with short notice. We study transparent checkpoint/restart for CPU applications on Azure Spot VMs, using CRIU/Podman for OpenMP and DMTCP for single-node MPI. Local NVMe measurements validate both setups and reveal different checkpoint-time trends as parallelism grows. We also evaluate Managed Disks, Azure Files, and Blob Storage for checkpoint durability and recovery on a replacement VM. A Poisson revocation model relates checkpoint and recovery times to checkpoint intervals and expected VM execution time. Azure’s 30 s eviction notice motivates the proactive persistence of checkpoints.*

1. Introduction

Cloud platforms provide elastic access to processors, accelerators, and storage without the up-front investment required to build an on-premises cluster. Their pay-as-you-go model is particularly useful when a local HPC system is saturated or when an experiment needs a short-lived resource configuration. Spot virtual machines (VMs) further reduce the hourly price by selling unused capacity, but the provider may revoke them at any time. Azure Spot VMs have no availability service-level agreement and may be evicted because of capacity or price [Microsoft Azure 2026a, Microsoft Azure 2026b]. Consequently, lower instance price does not by itself imply a lower monetary cost: a long application that restarts from the beginning after every revocation may spend more time and money than an on-demand execution.

Checkpoint/restart (C/R) bounds this loss by saving an application’s state and resuming it after a failure. Application-level checkpoints are usually compact and portable, but require application changes. System-level tools instead aim for transparent capture of processes. CRIU (Checkpoint/Restore in Userspace) saves Linux process trees and is integrated with container runtimes such as Podman, while DMTCP (Distributed MultiThreaded CheckPointing) coordinates checkpoints across communicating processes

[CRIU Project 2026, Ansel et al. 2009]. Transparency, however, does not remove three practical constraints: the tool must support all external resources used by the application, the complete recovery procedure must fit the provider’s interruption model, and a durable copy must survive revocation of the VM. The experimental scope here is CPU-only OpenMP and single-node MPI.

This paper studies transparent C/R for parallel applications on Azure Spot VMs and relates its overhead to expected VM execution time under revocation. OpenMP and MPI belong in the same study because they expose different requirements of transparent recovery: CRIU captures the process tree of a shared-memory application, but does not by itself coordinate a globally consistent checkpoint across MPI ranks. DMTCP supplies that coordination [CRIU Project 2026, Ansel et al. 2009]. Comparing the two setups identifies which overheads arise from the parallel programming model and which arise from the tool and storage pipeline, without assuming that one tool covers both cases.

Building on the AWS storage study by Teylo et al. [Teylo et al. 2020], this work extends the comparison to Azure and the software stack that creates the checkpoint, and connects the measurements to an execution-time model. Three research questions organize the paper and its contributions:

- **RQ1: How do the checkpoint and restore times of the tested tools compare for OpenMP and single-node MPI applications?** The contribution is a performance comparison of CRIU/Podman and DMTCP on Azure, showing how checkpoint time changes with thread or rank count.
- **RQ2: How do the runtime pipeline and storage service affect checkpoint creation and recovery on a different VM?** The contribution is a separation of CRIU capture from Podman export and compression, together with a storage evaluation covering Managed Disks, Azure Files, and Blob Storage. Local NVMe supplies a performance baseline; recovery after revocation requires an independently retained checkpoint.
- **RQ3: How do checkpoint and restore times, application duration, and revocation rate determine the checkpoint interval and the expected VM execution time?** The contribution is a procedure that instantiates a periodic-checkpoint model with Poisson revocations from measured times, distinguishing the instant at which the application resumes from the instant at which the checkpoint becomes durable.

The remainder of the paper presents background and the time model in Section 2, related work in Section 3, the method in Section 4, results in Section 5, and conclusions in Section 6.

2. Background

A Spot instance is unused provider capacity offered at a steep discount in exchange for the right to revoke it with little warning [Microsoft Azure 2026a]. On Azure, the revocation is announced by a `Preempt` scheduled event, published inside the VM through the metadata service; the scheduled minimum notice is 30 s [Microsoft Azure 2026b]. Notification delivery is best-effort [Microsoft Azure 2026a], so a recovery strategy cannot depend on receiving the full warning window. Azure supports `Deallocate` and `Delete` eviction policies: retained disks remain billable after deallocation, while deletion can also remove

associated disks [Microsoft Azure 2026a]. A checkpoint on Managed Disks must therefore have an explicit retention policy that preserves it independently of the evicted VM. Local temporary NVMe does not provide that recovery guarantee and is used only as a performance baseline or staging area [Microsoft Azure 2026c].

Table 1 contrasts the interruption and storage mechanisms relevant to this study. AWS documents a two-minute interruption notice for stopping or terminating a Spot instance, also delivered on a best-effort basis; hibernation begins immediately and has no two-minute warning [Amazon Web Services 2026b, Amazon Web Services 2026a]. Consequently, a checkpoint pipeline suitable for an AWS notice cannot be assumed to fit Azure’s shorter window.

Table 1. Azure and AWS Spot mechanisms relevant to checkpoint recovery.

Aspect	Azure Spot	AWS Spot
Eviction notice	Preempt, 30 s	Interruption notice, 2 min
Eviction policy	Deallocate or Delete	Terminate, Stop, or Hibernate
Block storage	Managed Disks	EBS
Network file storage	Azure Files	EFS
Object storage	Blob Storage	S3

The durable options span three storage systems: Managed Disks provide block volumes; Azure Files provides shared file access through SMB or NFS; and Blob Storage provides object storage, with API and filesystem access options [Microsoft Azure 2026d]. These correspond to AWS EBS, EFS, and S3 at the service level, but their protocols, tiers, and recovery procedures differ, so performance does not carry over between clouds.

CRIU freezes a process tree and serializes its memory, file descriptors, and kernel state into a set of image files; a container runtime such as Podman drives CRIU and then packages the dump into a compressed `.tar.gz` file for export. The two stages are separable and must be measured separately: compression and packaging are convenient but sit outside CRIU’s core dump time. MPI is harder still: a valid distributed checkpoint must capture a globally consistent state, including in-transit messages, which a per-process tool cannot guarantee. DMTCP solves this with a coordinator that synchronizes the capture across ranks [Ansel et al. 2009], and MANA (MPI-Agnostic Network-Agnostic Checkpointing) splits the application from the MPI library so each can be checkpointed independently, a promising route for multi-node work [Garg et al. 2019].

Whether periodic checkpointing pays off is a classical trade-off: frequent checkpoints add capture overhead, while rare ones increase the work lost at each failure [Daly 2006, Siavvas and Gelenbe 2019]. We adopt the model of Siavvas and Gelenbe [Siavvas and Gelenbe 2019], which expresses the expected execution time of a program subject to failures and lets us compute the interval that minimizes it. Let a program require T seconds of useful work on a VM whose revocations follow a Poisson process of rate λ (so the mean time between evictions is $1/\lambda$). All times and λ must use consistent units. Without checkpointing, a revocation forces a restart from the beginning, and the

expected wall-clock time to complete the run is

$$E[T_{\text{nocp}}] = \frac{e^{\lambda T} - 1}{\lambda}, \quad (1)$$

which grows without bound as λT increases. With a checkpoint of time C taken every τ seconds and a restore of time R after each revocation, the run is divided into T/τ intervals, and the expected wall-clock time is approximated by

$$E[T_{\text{cp}}] \approx T + \frac{T}{\tau} C + \lambda E[T_{\text{cp}}] \left(R + \frac{\tau + C}{2} \right), \quad (2)$$

where the last term counts, for the expected number of revocations $\lambda E[T_{\text{cp}}]$, one restore plus the work discarded since the last checkpoint, on average half of an interval plus the checkpoint in progress. To first order, minimizing this model gives the familiar Young-Daly checkpoint interval

$$\tau^* = \sqrt{\frac{2C}{\lambda}}, \quad (3)$$

which captures the basic trade-off: more expensive checkpoints favor longer intervals, whereas more frequent failures favor shorter ones [Daly 2006].

A single C assumes that the application resumes only once the checkpoint is durable. For a pipeline that resumes computation before persistence completes, we distinguish the checkpoint *stall* time C_s , measured from checkpoint initiation until application resumption, and the checkpoint *durability* time $C_d \geq C_s$, measured from the same initiation until the checkpoint is safely retained on the storage service. Recovery time R covers restoring that checkpoint on a different VM; provisioning and storage attachment must be accounted for separately if excluded from the restore timer. The previous checkpoint must remain available until its successor is durable. As a first-order approximation, we use C_s for the direct overhead and C_d in the expected-loss term:

$$E[T_{\text{cp}}] \approx T + \frac{T}{\tau} C_s + \lambda E[T_{\text{cp}}] \left(R + \frac{\tau + C_d}{2} \right), \quad (4)$$

which is an implicit equation for $E[T_{\text{cp}}]$. We can explicitly write it as:

$$E[T_{\text{cp}}] \approx \frac{T(1 + C_s/\tau)}{1 - \lambda(R + \frac{\tau + C_d}{2})} \quad (5)$$

This approximation requires a positive denominator and assumes constant times and immediate replacement capacity. It does not model allocation delays or failures during recovery. Minimizing it gives:

$$\tau^* = -C_s + \sqrt{C_s^2 - 2C_s R - C_s C_d + \frac{2C_s}{\lambda}}. \quad (6)$$

When checkpoint and restore times are small relative to the mean time between revocations, the terms involving C_s , C_d , and R inside this correction are small compared with $2C_s/\lambda$, yielding the familiar first-order approximation $\tau^* = \sqrt{2C_s/\lambda}$.

A smaller stall time therefore lowers the direct checkpoint overhead and permits a shorter optimum checkpoint interval, while the durability and restore times affect the optimum through the expected loss associated with a revocation. Section 5.3 explains how to supply these inputs from the recovery protocol. When $C_s = C_d = C$, Eq. (4) reduces to the single- C model of Eq. (2).

3. Related Work

Teylo et al. [Teylo et al. 2020] developed CRIU-based checkpoint and recovery procedures for Spot VMs on AWS using S3, EBS, and EFS. On `c3.2xlarge` instances (8 vCPUs, 16 GB), they varied the memory footprint of a synthetic single-task application from 140 MB to 7,750 MB and measured dump times of 0.65–55.82 s on EBS and 2.12–78.73 s on EFS, while S3 was the slowest, exceeding EBS by roughly 89% on average across footprints. The ordering inverted during recovery: reassociating the EBS volume with a replacement VM made its recovery about 9% slower than S3 and 26% slower than EFS, because the volume cannot be attached elsewhere until the revoked instance terminates, which alone took 148 s on average.

One set of experiments therefore produced two different rankings, fastest to write and fastest to recover, and the service that won the first lost the second. That separation, rather than the measured values, is what carries to another provider, and it motivates measuring storage-access preparation and application restore separately on Azure. Our study keeps the separation but changes platform and workload: Azure’s services and 30 s notice, parallel NPB applications whose process state reaches 10.6–23.3 GiB, and a CRIU pipeline driven by Podman, which adds an export stage absent from their direct CRIU invocation. Storage timings measured on AWS cannot be transferred to Azure without its own tier-specific configurations and cross-VM measurements.

The decision of how often to checkpoint has a long history: Young and then Daly derived successive approximations of the optimum interval [Daly 2006], and Siavvas and Gelenbe [Siavvas and Gelenbe 2019] developed a model of expected execution time under failures that we adopt in Section 2.

For MPI on AWS Spot, Gong et al. [Gong et al. 2015] combined checkpoints with replicated execution to cut cost, while Munhoz et al. [Munhoz et al. 2022] compared BLCR (Berkeley Lab Checkpoint/Restart) [Duell et al. 2002] rollback with in-memory recovery based on User-Level Failure Mitigation (ULFM), showing that preemptive signals reduce overhead and that Spot is not always cheapest. Recent fleet studies from our group reinforce that the lowest hourly price is not the lowest cost to solution [Sodré et al. 2025, Serrano et al. 2025]. Transparent MPI checkpointing has also been studied: Azeem and Helal [Abdel Azeem and Helal 2014] found DMTCP faster than BLCR/Open MPI on EC2, and MANA targets portability across MPI and network implementations [Garg et al. 2019].

4. Experimental Method

The method follows the three questions introduced in Section 1: validate the tool configurations (RQ1), separate run and storage times with recovery on a different VM (RQ2), and supply those times to the revocation model (RQ3). Single-node execution provides a controlled baseline for process capture, runtime processing, and storage I/O, without

the additional inter-node communication and coordination overhead of a distributed MPI job. The MPI results therefore establish feasibility for the tested configuration, rather than multi-node scalability.

4.1. Workloads and Platforms

The experimental workloads use the class-D NAS Parallel Benchmarks (NPB) [NASA Advanced Supercomputing Division 2026, Jin et al. 1999]: BT (Block Tridiagonal) is a structured-grid solver, and CG (Conjugate Gradient) has irregular sparse accesses. Their standardized inputs and built-in verification support repeatable comparisons and checks of restored execution. Their different memory-access patterns and multi-gigabyte process states exercise the C/R pipeline, while BT’s OpenMP and MPI versions permit a comparison within one benchmark family. These controlled workloads measure mechanism overhead; they do not establish representativeness for industrial applications or validate multi-hour execution under actual revocations.

All CPU experiments run on the Azure `Standard_L64s_v4`, a 64-vCPU instance based on an Intel Xeon 8573C with approximately 512 GiB of RAM. The shared-memory setup uses Ubuntu 24.04, Linux 6.8, Podman 4.9.3, and CRIU 4.2; the message-passing setup adds GCC/GFortran 13.3, MPICH 5.0 (Hydra, `ch3:nemesis`), and DMTCP 4.1. Table 2 summarizes these configurations.

Table 2. Experimental setups. All MPI experiments are single-node.

Setup	Workload	Parallelism	Mechanism / platform
CPU: shared memory	NPB BT.D	OMP 4, 32 threads	Azure <code>L64s_v4</code> ; Podman + CRIU
CPU: message passing	NPB BT.D	16, 25, 36 ranks	Azure <code>L64s_v4</code> ; MPICH + DMTCP
Azure storage	NPB BT.D, CG.D	OMP 32 threads	Azure <code>L64s_v4</code> ; Podman + CRIU

In this setup, the operating-system disk is slower than the local NVMe SSD. To avoid making it a bottleneck, Podman’s working directory (`/var/lib/containers`) was explicitly moved to the fast local NVMe SSD. For the durable storage service comparison, only the exported checkpoint is written to the remote storage service. The selected L-series VM provides both this local-device baseline and sufficient memory for the class-D workloads.

4.2. Storage Service Configuration

A service name does not fix performance or price on Azure: the tier, the provisioned size, the protocol version, the host caching mode, and the client used to reach the service each change the measured times and the resulting charges. Table 3 records these settings for the three durable services, so that the timings of Section 5.2 and the time inputs of Section 5.3 can be reproduced and priced. VM A and VM B are both `Standard_L64s_v4` instances in `East_US`, and each run writes its checkpoint to a single service.

The client cache decides what a measured write means, so we state it rather than infer it from the service. The Managed Disk is attached with host caching `None` and

Table 3. Configuration of the durable storage services.

Parameter	Managed Disk	Azure Files	Blob Storage
SKU / tier	Premium SSD	Standard_LRS, Trans- actionOptimized	Standard_LRS, Hot, block blob
Provisioned size	128 GiB	128 GiB quota	Elastic
IOPS / throughput	500 IOPS, 100 MB/s	1,000 IOPS, 60 MiB/s	40,000 req/s, 60 Gbps
Protocol / client	ext4, host caching none	SMB	blobfuse2 file_cache

the blobfuse2 cache directory is `/mnt/blob-cache/cache`. A file cache placed on the local NVMe device would return a write once the data reached that device rather than Blob Storage, reporting local-device latency under a Blob label. For each client we therefore name the event that ends the checkpoint durability time C_d of Section 5.3: for the Managed Disk, `fsync` of the file and of its parent directory; for Azure Files and Blob Storage, the service’s own REST confirmation that the object exists with the expected size, because `close` or `fsync` on the mounted client does not guarantee the data left the host.

4.3. Protocol and Metrics

For the local-device comparison, every configuration is checkpointed at three nominal progress points (30%, 50%, and 80% of its own baseline), with three repetitions per point, yielding nine measured runs per configuration. The OpenMP four-thread case uses two repetitions per point, yielding six runs. A CPU container is started, allowed to make progress, checkpointed to a `.tar.zst` file, stopped, restored, and allowed to finish. Every restored run is checked for correctness; we retain only those where the application completes and validates its own output. The shared-memory case checkpoints with CRIU through Podman. The message-passing case uses DMTCP: the original job is terminated after coordinated checkpoint creation and then restored using the generated restart script. Each tool configuration and thread or rank count is compared with its own uninterrupted baseline.

The storage evaluation targets checkpoint creation on VM A and application restore on a different VM B. For Managed Disks, recovery requires retaining the checkpoint volume, detaching it from A, and attaching and mounting it on B. For Azure Files and Blob Storage, B must establish access to the same retained share or container using the selected protocol. These are storage-access steps, distinct from the application restore operation. Their durations, VM allocation delay, and the start and end events of the restore timer must be reported separately so that the model includes each time once.

Because checkpoint and restore times vary little with the position at which the checkpoint is taken, we pool the runs of each configuration and report each metric as a mean $\pm$ sample standard deviation. Alongside checkpoint duration, restore duration, and checkpoint size, we report the baseline uninterrupted run time and the total wall-clock time of the workflow. This structure allows the time of C/R to be read directly as the difference between the total and the baseline times. CRIU dump time and the full Podman checkpoint command time are distinguished when discussing runtime overhead.

5. Results and Discussion

Section 5.1 addresses RQ1 through the local-device tool comparison. Section 5.2 addresses RQ2 through runtime overhead and the requirements for recovery on another VM. Section 5.3 addresses RQ3 by defining how cross-VM measurements determine checkpoint intervals and expected VM execution time.

5.1. Shared-Memory and Message-Passing Comparison

Table 4 reports, for the same BT.D benchmark on Azure `L64s_v4`, the shared-memory case (OpenMP with CRIU through Podman) and the message-passing case (single-node MPI with DMTCP). The column T/r stands for the number of threads/ranks used. Both cases were verified to restore and validate every configuration, so the question is not whether a tool can restore a given form of parallelism but how its time behaves as parallelism grows. An important characteristic of this benchmark set is that it divides its input data as evenly as possible among multiple ranks in the message-passing case so that the amount of RAM used by each process decreases as the number of ranks increases. To keep the comparison focused on the parallelism model rather than the storage medium, both cases write their checkpoint to the local device; one difference is nonetheless intrinsic to the tools, as the CRIU times include the Podman export and compression of the checkpoint whereas DMTCP writes its own images with no such step.

Table 4. Single-node C/R on Azure (BT.D), checkpoint written to the local device. Times are mean $\pm$ standard deviation over the pooled runs of each configuration (nine per row, six for OpenMP/4 threads). *Total* is the full checkpoint-and-restore workflow. For MPI, *Size* is the sum of checkpoint sizes across ranks. These local-device restore times validate the tools; they are not cross-VM recovery times.

Tool	Model	T/r	Baseline (s)	Total (s)	Checkpoint (s)	Restore (s)	Size (GiB)
CRIU	OpenMP	4	2826.9	2928.9 ± 2.3	20.5 ± 0.0	52.4 ± 0.1	10.60
CRIU	OpenMP	32	426.3	546.3 ± 0.6	23.9 ± 0.2	52.9 ± 0.2	10.61
DMTCP	MPI	16	904.6	1054.3 ± 6.4	59.2 ± 5.8	21.1 ± 0.2	22.52
DMTCP	MPI	25	712.6	848.5 ± 1.6	50.3 ± 0.3	19.5 ± 0.5	22.92
DMTCP	MPI	36	735.0	865.1 ± 2.8	41.8 ± 2.7	21.2 ± 1.1	23.33

The OpenMP checkpoint size is essentially fixed at about 10.6 GiB regardless of thread count, because all threads share one address space, while the MPI checkpoint grows smoothly with the number of ranks, reaching roughly 23 GiB at 36 ranks as runtime buffers and boundary data are replicated in each process. For the same benchmark the message-passing state is more than twice the shared-memory one. The capture times move in opposite directions with the degree of parallelism. In the shared-memory case CRIU stays around 20-24 s whether the run uses 4 or 32 threads, so adding threads does not lengthen the capture. However, in the message-passing case DMTCP’s capture instead *falls* as ranks grow—from about 59 s at 16 ranks to 42 s at 36. This trend is consistent with each process writing a smaller image concurrently. Note that, because the `L64s_v4` has only 32 physical cores, the total runtime increases when moving from 25 to 36 ranks in the message-passing case; interestingly, the checkpointing time keeps decreasing in this scenario.

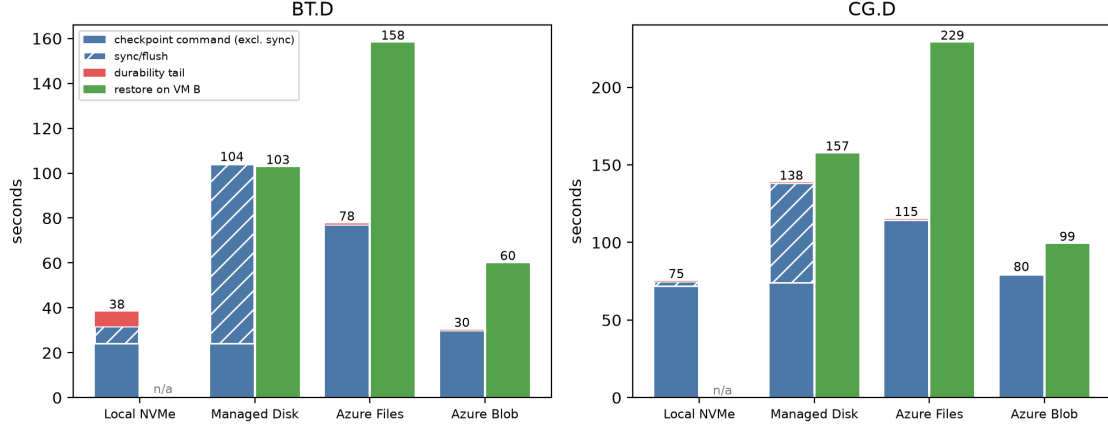


Figure 1. Checkpoint on VM A (blue, with the *sync* hatched and the durability tail in red) and restore on VM B (green). Local NVMe has no restore on VM B. Means; the numbers above the bars are seconds.

5.2. Runtime Overhead and Recovery on a Different VM

The storage study fixes OpenMP parallelism at 32 threads and considers Managed Disks, Azure Files, and Blob Storage, with local NVMe as a performance baseline. The recovery comparison requires a checkpoint created on VM A and restored on VM B, as described in Section 4. Its purpose is to measure the complete data path needed after losing the original VM, including access to the retained checkpoint.

The client cache was invalidated on VM B before each read. Local NVMe has no restore on VM B, because it disappears with VM A. Each cell pools nine checkpoints on VM A and three restores on VM B (one per wait time); with $n = 3$ the standard deviation only indicates spread. Durability follows the events defined in Section 4.2. The time to attach the disk or mount the share on VM B was not measured, so the values below exclude it.

Table 5. Checkpoint on VM A and restore on VM B (mean $\pm$ standard deviation). Checkpoint is the Podman command plus the *sync*, which is already contained in it. Durable is Checkpoint plus the persistence tail confirmed by the storage (Section 4.2). Upload and Read are checkpoint size divided by the Durable and Restore times; the Managed Disk has no network upload.

Storage	Checkpoint (s)	of which sync (s)	Durable (s)	Upload (MB/s)	Restore (s)	Read (MB/s)
<i>BT.D</i>						
Managed Disk	104.0 $\pm$ 20.5	79.8	104.0	–	102.7 $\pm$ 0.2	111
Azure Files	77.1 $\pm$ 3.6	0.1	77.5	141	158.1 $\pm$ 2.9	72
Azure Blob	30.0 $\pm$ 0.6	0.1	30.2	362	59.9 $\pm$ 0.3	190
<i>CG.D</i>						
Managed Disk	138.4 $\pm$ 4.1	64.3	138.5	–	157.3 $\pm$ 0.1	105
Azure Files	114.7 $\pm$ 6.3	0.1	115.1	138	228.7 $\pm$ 5.3	72
Azure Blob	79.5 $\pm$ 1.1	0.0	79.6	199	99.1 $\pm$ 0.5	167

Figure 1 illustrates the measured checkpoint and restore times across the storage configurations, distinguishing the *sync* and persistence-tail components of checkpoint creation. As shown in Table 5, Azure Blob has the shortest times to durability and restore for

both benchmarks. This ranking applies only to the measured storage configurations and timing boundaries and does not imply a monetary-cost advantage.

The *sync* accounts for the Managed Disk’s longer checkpoint times. For BT.D, the Podman command takes about 24 s, as on local NVMe, but the *sync* adds 79.8 s, or 77% of the checkpoint time, with the largest variation in the study (standard deviation 20.5 s; individual runs range from 90 to 140 s). For CG.D, the *sync* accounts for 64.3 s of 138.4 s. On Azure Files and Azure Blob, the *sync* is negligible and the persistence tail is below 0.6 s; most of the checkpoint time is spent uploading data as part of the command. Upload rates are about 140 MB/s on Azure Files and 362 and 199 MB/s on Azure Blob for BT.D and CG.D, respectively (Upload column).

The restore reads (Read column) at about 190 (BT.D) and 167 MB/s (CG.D) from Azure Blob, 105–111 MB/s from the Managed Disk and 72 MB/s from Azure Files. The Managed Disk and Azure Files rates are close to their provisioned throughput in Table 3 (100 MB/s and 60 MiB/s), so the restore on these storages is bounded by what was provisioned.

The runtime also contributes to the checkpoint time. On local NVMe, CRIU completes its dump in about 3.4 s for BT.D while the Podman checkpoint command returns after 23.9 s; for CG.D the times are 5.1 s and 73.9 s. Work outside the CRIU dump thus accounts for approximately 86% and 93% of the command. In the CG.D run without export, the dump returns in about 5 s. This work includes export, compression and writing the checkpoint to the storage, and the last part depends on the storage: for BT.D the command takes 23.9, 23.9, 29.4 and 76.7 s on local NVMe, the Managed Disk, Azure Blob and Azure Files. These observations do not isolate compression alone; a comparison with uncompressed export is needed before attributing the difference to the compression algorithm.

Compared with Teylo et al. [Teylo et al. 2020], the structural point is the ordering, not the values. Their EBS recovery waited for the revoked instance to terminate before the volume could be attached elsewhere. On Azure, under `Deallocate` the VM is stopped rather than removed (Table 1), and a Managed Disk can also be attached to several VMs at once (shared disks): at least three on the lowest eligible tier and up to ten on the highest. A replacement VM can then attach the volume before the evicted VM releases it, which takes the detach wait off the critical path, provided the file system is cluster-aware or only one VM uses it at a time. EBS Multi-Attach is restricted to Provisioned IOPS volumes. Our measurements do not include this attach step, so the mechanism is not tested here.

The write and recovery orders also differ on Azure. On BT.D the Managed Disk writes slower than Azure Files (104.0 against 77.5 s) yet restores faster (102.7 against 158.1 s); on CG.D it writes slower (138.5 against 115.1 s) and restores faster (157.3 against 228.7 s). The storage that writes a checkpoint fastest need not be the one that restores it fastest, and Azure Blob is the only one that leads both here. As Teylo et al. reached S3 through S3FS-FUSE, part of their object-storage penalty belongs to the FUSE layer, and the same caveat applies to blobfuse2. Section 5.3 uses the Durable and Restore columns as $C_s = C_d$ and R .

5.3. Expected Execution Time under Revocation

The model of Section 2 requires checkpoint stall C_s , durability time C_d , and recovery time R for a specified storage configuration. Managed Disks provide a block-storage reference corresponding to EBS in the AWS study, and can retain a checkpoint independently of VM A when configured accordingly. This motivates using them as a reference configuration; it does not establish either a performance or a monetary advantage over Files and Blob. The same procedure applies to each service once its cross-VM timings are available.

For each benchmark, C_s must come from the measured application pause, C_d from completion of persistence to the retained service, and R from restoring the checkpoint on VM B. If the protocol resumes the application only after persistence completes, use $C_s = C_d$. VM provisioning, disk attachment, and mounting must either be included in a documented recovery-time boundary or added as separate terms. The local-device restore values in Table 4 cannot supply R for this comparison.

Revocations are represented by a homogeneous Poisson process, an analytical assumption rather than a measured Azure eviction policy. Azure publishes eviction-rate ranges, including a highest band of 20+% [Microsoft Azure 2026c]; this band alone does not identify a constant hazard λ for a particular VM type and region. An illustrative $\lambda = 0.50 \text{ h}^{-1}$ means a mean time between revocations of 2 h, not a 50% probability of revocation within an hour. The latter probability under the model is $1 - e^{-0.50} \approx 39.3\%$. This deliberately severe scenario can expose the effect of revocation on short runs, but must not be presented as an observed rate.

Sensitivity follows directly from the first-order interval $\tau^* \approx \sqrt{2C_s/\lambda}$: for fixed C_s , multiplying the rate by a factor k divides the interval by $\sqrt{k}$. Relative to 0.50 h^{-1} , rates of 0.10 and 0.05 h^{-1} lengthen the interval by factors of approximately 2.24 and 3.16, respectively. This analytical dependence explains the direction of the effect; numerical expected-time and cost comparisons still require the new timing inputs.

For a finite job, the interval must also be compared with useful run time T . If $\tau^* \geq T$, no periodic checkpoint is scheduled before completion. The fractional checkpoint count T/τ in the continuous approximation is not an instruction to create a final checkpoint for a completed job. The policy comparison must include this no-checkpoint option and use a positive denominator in Eq. (5).

NPB runs supply measurements of mechanism overhead, while any multi-hour scenario is an extrapolation that assumes those times remain applicable. At a constant VM price p_{VM} per hour, expected VM execution time is

$$E[M_{\text{VM}}] = p_{\text{VM}} \frac{E[T_{\text{policy}}]}{3600}, \quad (7)$$

where time is in seconds and the policy is either no checkpointing or periodic checkpointing. This quantity excludes storage capacity, transactions, data transfer, and separately billed recovery resources. It therefore cannot support a claim about the cheapest storage service or total monetary savings. A full comparison must add those charges using a stated region, tier, price date, retained capacity, and retention time, as well as the number of storage operations. Numerical dollar estimates and storage-cost rankings are deferred until the cross-VM timings and billing inputs are available.

6. Conclusions and Future Work

Cloud providers offer elastic resources as an alternative to on-premises clusters, allowing users to access computing capacity without acquiring and maintaining local infrastructure. Spare virtual machines (VMs) are available at lower prices in the Spot market, however as they can be reclaimed at any time by the providers, often with short notice, users are exposed to interruptions and potential loss of work, motivating fault-tolerance mechanisms such as checkpoint/recovery (C/R). In this work, we evaluated transparent C/R tools for single-node OpenMP and MPI applications on Azure Spot VMs, comparing checkpoint and restore times across the tested configurations (RQ1), examining how the runtime pipeline and storage service affect checkpoint creation and recovery on a replacement VM (RQ2), and assessing how these times, application duration, and revocation rate shape the checkpoint interval and expected VM execution time (RQ3).

Regarding RQ1, CRIU/Podman successfully checkpointed and restored the OpenMP runs, while DMTCP did for single-node MPI. On local NVMe, checkpoint time remained nearly constant from 4 to 32 threads and decreased from 16 to 36 MPI ranks. Regarding RQ2, the runtime pipeline added substantial checkpoint overhead: the CG.D CRIU dump took about 5 s, compared with 74 s for the full Podman command on local NVMe. In the cross-VM tests, Azure Blob was fastest for durability and restore for both benchmarks: 30.2/79.6 s to durability and 59.9/99.1 s to restore for BT.D/CG.D, respectively. BT.D's Blob checkpoint reached durability in about the time of Azure's 30 s notice, while Managed Disk *sync* accounted for 77% of its checkpoint time. Regarding RQ3, the analysis shows that higher revocation rates favor shorter checkpoint intervals, while no periodic checkpoint is scheduled when the optimal interval is at least the job duration. However, the rates considered are illustrative rather than observed Azure rates, so numerical time and cost comparisons require a revocation rate estimated from Azure eviction data for the relevant VM type and region, complete cross-VM timing measurements, and billing inputs, since VM execution cost alone does not establish total savings.

Still, the evaluation covered only CPU NPB workloads and single-node configurations, so it does not establish performance for production applications, GPU workloads, or multi-node MPI. The cross-VM tests included three restores per storage and benchmark on one replacement VM in one region. The checkpoint and restore runs were unpaired, and attachment and mounting times were excluded. The multi-hour results extrapolated measured NPB overheads using illustrative revocation rates. Accordingly, future work will evaluate industrial applications, GPU workloads, and multi-node MPI with DMTCP and MANA, expand paired cross-VM tests across VMs and regions, measure attachment and mounting, including Managed Disk shared attachment, compare compressed and uncompressed export, test the blobfuse2 block cache, and use real revocation traces and billing data to compare policies and costs across VM types, regions, and storage tiers.

Acknowledgments

The authors thank Petrobras for funding this work.

References

- Abdel Azeem, B. and Helal, M. (2014). Performance evaluation of checkpoint/restart techniques for MPI applications on Amazon cloud. In *Proceedings of the 9th International Conference on Informatics and Systems*, pages PDC-49-PDC-56.

- Amazon Web Services (2026a). Behavior of spot instance interruptions. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/interruption-behavior.html>. Accessed October 15, 2026.
- Amazon Web Services (2026b). Spot instance interruption notices. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-instance-termination-notices.html>. Accessed October 15, 2026.
- Ansel, J., Arya, K., and Cooperman, G. (2009). DMTCP: Transparent checkpointing for cluster computations and the desktop. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12.
- CRIU Project (2026). Checkpoint/restore. <https://criu.org/Checkpoint/Restore>. Accessed July 13, 2026.
- Daly, J. T. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312.
- Duell, J., Hargrove, P., and Roman, E. (2002). The design and implementation of Berkeley Lab’s Linux Checkpoint/Restart. Technical Report LBNL-54941, Lawrence Berkeley National Laboratory.
- Garg, R., Price, G., and Cooperman, G. (2019). MANA for MPI: MPI-agnostic network-agnostic transparent checkpointing. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pages 49–60. ACM.
- Gong, Y., He, B., and Zhou, A. C. (2015). Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: Checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. ACM.
- Jin, H., Frumkin, M., and Yan, J. (1999). The OpenMP implementation of NAS Parallel Benchmarks and its performance. Technical Report NAS-99-011, NASA Ames Research Center.
- Microsoft Azure (2026a). About Azure Spot Virtual Machines. <https://learn.microsoft.com/en-us/azure/virtual-machines/spot-vms>. Accessed October 15, 2026.
- Microsoft Azure (2026b). Azure Metadata Service: Scheduled events for Linux virtual machines. <https://learn.microsoft.com/en-us/azure/virtual-machines/linux/scheduled-events>. Accessed October 15, 2026.
- Microsoft Azure (2026c). Build workloads on spot virtual machines. <https://learn.microsoft.com/en-us/azure/architecture/guide/spot/spot-eviction>. Accessed October 15, 2026.
- Microsoft Azure (2026d). Introduction to Azure Storage. <https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction>. Accessed October 15, 2026.
- Munhoz, V., Castro, M., and Mendizabal, O. (2022). Strategies for fault-tolerant tightly-coupled HPC workloads running on low-budget spot cloud infrastructures. In *Pro-*

ceedings of the 34th International Symposium on Computer Architecture and High Performance Computing, pages 263–272. IEEE.

NASA Advanced Supercomputing Division (2026). NAS Parallel Benchmarks. <https://www.nas.nasa.gov/software/npb.html>. Accessed July 13, 2026.

Serrano, L., Lima, M., Portella, F., and Drummond, L. (2025). A multi-cloud approach to cost optimization with aws and azure fleet services. In *Anais Estendidos do XXXVII International Symposium on Computer Architecture and High Performance Computing*, pages 61–68, Porto Alegre, RS, Brasil. SBC.

Siavvas, M. and Gelenbe, E. (2019). Optimum checkpoints for programs with loops. *Simulation Modelling Practice and Theory*, 97:101951.

Sodré, D., Serrano, L., Lima, M., Boeres, C., Drummond, L., and Rebello, V. (2025). Spotting the right cloud instances with multiple aws ec2 fleets. In *Anais do XXXVII International Symposium on Computer Architecture and High Performance Computing*, pages 69–79, Porto Alegre, RS, Brasil. SBC.

Teylo, L., Brum, R. C., Arantes, L., Sens, P., and Drummond, L. M. d. A. (2020). Developing checkpointing and recovery procedures with the storage services of Amazon Web Services. In *Proceedings of the 49th International Conference on Parallel Processing Workshops*, pages 1–8. ACM.