2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) | 978-1-6654-5155-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/SBAC-PAD55451.2022.00037

Strategies for Fault-Tolerant Tightly-Coupled HPC Workloads Running on Low-Budget Spot Cloud Infrastructures

Vanderlei Munhoz
Fed. Univ. of Santa Catarina (UFSC)
Florianópolis, Brazil
munhoz@protonmail.ch

Márcio Castro
Fed. Univ. of Santa Catarina (UFSC)
Florianópolis, Brazil
marcio.castro@ufsc.br

Odorico Mendizabal
Fed. Univ. of Santa Catarina (UFSC)
Florianópolis, Brazil
odorico.mendizabal@ufsc.br

***Abstract*—Cloud providers can rent their spare computing capacity at substantial discounts, reclaiming it whenever there is a more profitable higher-priority request — a business model well known as spot infrastructure market. Users can attain significant cloud investment savings using spot machines, however with the caveat of increasing software complexity, given the fault tolerance requirements of this environment. Improvements in virtualization and network technology, combined with the development of key new software tools, may allow the HPC community to effectively take advantage of cheap cloud resources, cutting expensive maintenance costs. This study aims to evaluate the viability of budget-constrained cloud environments for tightly-coupled MPI applications, exploring both spot and traditional low-budget infrastructures from real public cloud platforms. We propose and evaluate two different fault tolerance strategies tailored for unreliable spot cloud environments: system-level rollback restart with Berkeley Labs Checkpoint/Restart (BLCR) and in-memory rollback restart with User-Level Failure Mitigation (ULFM). We also propose a provider-agnostic empirical method for testing and predicting MPI workloads execution times and cloud infrastructure costs. A detailed cost analysis and performance benchmark of a case-study application is provided, with data gathered from experiments with both spot and persistent machines from AWS and Vultr Cloud, respectively. Our results show that: (i) adequate cluster sizing plays an important role in the overall job execution performance and cost-effectiveness, regardless of the type of selected instances; (ii) fault tolerance strategies based on BLCR may have worse performance than ULFM, but still be cost-effective considering software migration costs; (iii) the use of spot infrastructure does not guarantee costs savings depending on the chosen machine flavors and discounts, as experiments with persistent low-budget options attained better cost-effectiveness in some conditions.**

***Index Terms*—High-Performance Computing, Cloud Computing, Spot Instances, Fault Tolerance, MPI, Virtual Clusters.**

I. Introduction

The cloud computing paradigm allows organizations to reduce information technology costs by offering compute infrastructure and software services through a pay-as-you-go pricing model, similar to public utilities. This pricing model reduces the capital and technological entry barrier for small organizations, who can create and destroy resources on-demand, paying only for what they use. Users can then dynamically scale resources in real-time as their application requirements change, reducing resource over-provisioning and under-provisioning. Cloud providers can further optimize resource usage by renting spare infrastructure under considerable discounts, in what is traditionally known as spot market. Spot machines are transient instances that can be revoked at any time without advance notice by the provider, being responsibility of the user to preemptively save data and recover applications in case of instance repossessions. Spot markets and alternative low-cost cloud providers may be attractive options for small organizations, however traditional High Performance Computing (HPC) still has a long way to go before being able to take advantage of cheap cloud resources efficiently [1]. Accelerators such as GPUs and FPGAs can also be rented at most cloud providers, although not all legacy HPC applications are ready to make use of this kind of hardware.

Despite the fact that several mainstream providers offer specialized hardware through on-demand cloud pricing models, renting these kinds of machines can be more expensive than buying and maintaining an on-premise HPC cluster, contrary to what may be expected [2]. Moreover, the hardware abstraction, which is considered one of the biggest improvements brought by the cloud computing paradigm, is actually a painful aspect for traditional HPC use cases, where fine-grained knowledge of the underlying hardware is usually required to optimize the execution of CPU-intensive applications [3]. There are continuous debate and ongoing studies on the viability of executing specialized HPC applications in an uncontrolled and abstracted environment, specially on public clouds [4].

Thus, to broaden the reach of this research, we focus on the ubiquitous, readily available CPU-only virtual machines. In particular, we are interested in spot instances, which are considerably cheaper resources, although ephemeral. Given that the cloud provider can revoke the spot infrastructure resources at any moment, fault-tolerant deployment strategies are a requirement in this context. Furthermore, as the number of nodes required for HPC workloads is possibly high and these workloads usually take a long time to execute, allocating spot clusters may seem unfeasible, as we can expect several instance repossessions by the provider. Additionally, given the performance requirements, any kind of fault tolerance mechanism must also be resource-efficient.

In this work, we evaluate two fault tolerance approaches for tightly-coupled MPI applications running in the cloud. The strategies are comprised of in-memory rollback restart at the MPI application-level through User-Level Failure Mitigation (ULFM) [5] and process-level rollback restart with the well established Berkeley Lab Checkpoint/Restart (BLCR) software package. Our experiments indicate that the proposed ULFM-

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

based approach has the best performance and lowest cloud infrastructure cost, although it also comes with a complex migration process. On the other hand, the BLCR strategy is based on a well-known software package and has a straightforward migration process, being a reasonable option for rapid migration of legacy HPC applications to public clouds.

When considering scalability, either when using spot or reliable on-demand instances, the machine flavors and the number of nodes in the virtual cluster must be carefully chosen based on the workload size and communication characteristics. Otherwise, the underlying public cloud hardware limitations and high spot failure rates can make execution prohibitively expensive or unfeasible, even considering the spot discounts and a low-overhead fault tolerance strategy. Thus, we also propose a model for empirically estimating the total job execution time.

The contributions of this work are mainly three-fold:

1) We propose and evaluate a new low-overhead fault tolerance strategy based on in-memory checkpoint restart using the recent ULFM API added in the MPI 5.0 version, considering the particularities of unreliable cloud computing environments.
2) We provide an empirical analysis of both fault tolerance strategies, showing to researchers and consumers an overview of how to achieve trade-offs between economic benefits and application availability.
3) We propose and implement much-needed open-source tools for assisting researchers and consumers on how to migrate legacy HPC code efficiently to the public cloud. In particular, our empirical approach for estimating cloud infrastructure costs can help consumers gain a quick impression of the pros and cons of the reserved and spot cloud options for CPU-intensive physics simulations and similar classes of applications.

The remainder of this paper is organized as follows. Section II presents background knowledge regarding the class of MPI applications tackled in this work, and the details and peculiarities of the infrastructure market of two public cloud providers. Section III presents the proposed strategies for fault tolerance in MPI applications, as well as implementation details on the two chosen approaches for experimentation. Section IV presents our proposed strategy for cloud infrastructure cost prediction. Section V describes in detail the experimental environment and brings a detailed discussion regarding the results. Section VI discusses related works. Finally, Section VII concludes this paper with a recap of the results and a brief discussion regarding future work.

II. Background

A. HPC Applications and Cloud Computing Challenges

Some HPC workloads such as MapReduce and Monte Carlo simulations can be decomposed into multiple independent tasks, which require few to no communication, being able to more easily benefit from parallelism with heterogeneous cloud resources [1]. In contrast, tightly-coupled applications such as heat transfer, particle physics, fluid mechanics simulations and many others are usually modeled through Partial Differential Equation (PDE) systems which are then solved numerically. Some traditional approaches for solving PDE systems are based on Finite Difference Method (FDM), Finite Element Method (FEM), or Finite Volume Method (FVM). These methods essentially consist on computing the system variables in discrete places inside a meshed geometry [6].

Problems tackling huge geometrical domains with high resolution and multiple variables pose a challenge even to the most powerful computers. The main approach for parallelism in this context consists of dividing such geometrical mesh into chunks for simultaneous processing. Consequently, as the computation of each mesh cell requires information on its immediate neighbors, message passing between nodes becomes intense as the number of nodes scales, which can easily make the application of MPI-based parallelism impractical depending on the available inter-node communication speed. Furthermore, the use of heterogeneous resources, in this case, will certainly generate crippling balancing issues, given that all nodes must complete their tasks and synchronize before starting the next solver iteration.

B. A Test Case Application

In this work, we apply fault tolerance mechanisms to a parallel solver implementation for Laplace's heat diffusion equations as a study case, which is based on a popular alternative FDM technique called Forward-Time Central Space (FTCS) method [7]. However, even as we focus on this specific application, our mechanisms can be applied to any other tightly-coupled grid application implemented with MPI.

For the experiments, we consider a square region of n^2 mesh cells with non-periodic borders as the physical domain to be simulated. This domain is then broken into subdivisions and distributed to each MPI rank for iterative computation, resulting in a Cartesian topology of MPI processes. Subdivisions are made following a one-dimensional decomposition strategy and global boundaries have a constant temperature value. Each subdivision needs information from its neighbors to compute each iteration at the internal borders. To reduce messaging, we employ a ghost-cell pattern updated at the end of each iteration [8]. Thus, given an initial state, the application will iteratively solve the discrete heat diffusion recurrence formula in each mesh point until a convergence criterion or a defined maximum number of iterations is reached. An action diagram of the parallel algorithm is presented in Figure 1.

At the initial stages, the MPI world is initialized and memory is allocated locally for each rank. The workload distribution is made through a Cartesian MPI process topology employing a slab decomposition strategy, where each process will compute an equal chunk of the total mesh. Local updates are computed applying the heat diffusion recurrence equation at each mesh cell. After updating the entire local grid, we check a stopping criteria (maximum number of iterations reached, for example) and then write the solution to disk if

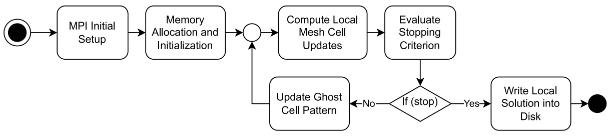
MPI Initial Setup
Memory Allocation and Initialization
Compute Local Mesh Cell Updates
Evaluate Stopping Criterion
Update Ghost Cell Pattern
No
If (stop)
Yes
Write Local Solution into Disk

Fig. 1. Action diagram of the parallel algorithm for Laplace's heat diffusion equations.

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

completed. In case the simulation did not reach the solution, we update the ghost cell pattern by exchanging border cells information between neighbors and start a new iteration.

C. Spot Cloud Infrastructure

Spot machines are ephemeral instances with a considerably lower cost per hour than standard on-demand machines, running on top of the same type of infrastructure with identical characteristics and performance. The downside comes from the fact that spot machines can be revoked on short notice at any time by the cloud provider, usually whenever capacity needs to be reclaimed for other reserved on-demand instances. AWS pioneered this service model in 2009 to sell idle infrastructure and increase revenue [9]. It is offered today by major cloud providers, such as Microsoft, Google, IBM, Oracle, Tencent, and many others. Spot machines are considerably cheaper than their on-demand counterparts, and since launched, they have attracted a good amount of attention from enterprises and organizations looking to reduce infrastructure costs.

When requesting a spot machine, users inform capacity in terms of vCPUs or the number of instances of a specific type, bidding the maximum price they are willing to pay per hour for the requested spot infrastructure. Spot prices for each machine type often fluctuate dynamically based on market demand in each availability zone, although this depends on the provider. Spot instances from Oracle for example just have a fixed discount. The provider only meets requests if the specified spare capacity is available at a price lower or equal to the bid value. This model is completely different from on-demand pricing, where an instance request is met immediately. This characteristic is an important detail in contexts where workloads have a defined deadline for completion, as spot requests have no guarantee by the provider ever to be met. As the available capacity also varies between different instance types, careful consideration is required when choosing machine flavors. Furthermore, estimating optimal bids for minimum costs is a hard problem, given the lack of available information from most cloud providers, which also often lack transparency regarding pricing history.

1) Peculiarities of the AWS Spot Market: Amazon Web Services infrastructure resources are arguably expensive, however they have one of the most matured spot markets. AWS is also the most far-reaching cloud provider, being one of the first options for researchers and software architects that venture in the cloud world.

Regarding its spot market, AWS provides historical pricing data for the public, although only during a limited past time frame. Supply and demand changes are opaque to the cloud consumers during execution time, and AWS spot prices can be highly volatile even during a single day. Machines with large numbers of vCPUs or accelerators (GPUs) typically have few spot instances available at a time, making the smaller standard x86 machines the most commonly used spot instance types. AWS also provides a messaging system that alerts the consumer within a short notice of two minutes before a spot instance repossession occurs. Such alerts are helpful for preemptively preparing a recovery strategy, even without a complex failure prediction model. Users are also able to define expiration times for their spot requests (the maximum wait time before cancelling a bid and giving up). This feature is explored in our spot cluster creation strategy.

2) Peculiarities of Vultr Cloud: Vultr is an alternative cloud provider known for low prices when compared to major market players, and their target audience are individual software developers and startups. Unfortunately, Vultr does not offer spot virtual machines at the time of this research. Nevertheless, we conduct some experiments using non-transient infrastructure from Vultr, as this provider is also a popular low-budget infrastructure option and have substantial discounts compared to other cloud platforms.

III. Fault Tolerance Strategies for Spot Clusters

In this section, we present our fault tolerance strategies for MPI-based HPC workloads running on cloud spot instances. First, we describe the distributed system model considered in this work. Then, we detail the proposed fault tolerance strategies.

A. System Model

We assume a distributed system composed of interconnected processes running on top of a cloud infrastructure. The system is asynchronous, *i.e.*, there is no bound on message delays and on relative process speeds. The underlying infrastructure comprises instance nodes that can be allocated or deallocated on the fly. Computational nodes may differ in computing, memory, and network capacities.

The unbounded set of processes $P = \{p_1, p_2, \dots\}$ is automatically mapped to the unbounded set of nodes $N = \{n_1, n_2, \dots\}$, where each process $p_i \in P$ is mapped to a node n_j and all nodes $n_j \in N$ host at least one process. Computational nodes are classified as *spot* or *on-demand instances*. We assume there is no resource's scarcity for *on-demand* instances. Thus, although *spot instances* might be unavailable, *on-demand instances* are available whenever they are requested.

Spot instances may fail, or they can be revoked by the cloud provider, while the *on-demand instances* never fail or are revoked. Any process executing on a faulty or revoked node eventually fails. We assume the *fail-stop* failure model and exclude malicious and arbitrary behavior (e.g., no Byzantine failures). This means that a faulty processes running on *spot instances* are detected by correct processes in the system. We assume the existence of a notification system that alerts system processes before a spot instance revocation occurs. Typically, these alerts are notified with short notice, but processes can miss the alert. Processes are equipped with a volatile memory, a local storage, and they share a persistent storage. Upon a failure, a process loses its volatile memory and local storage content, but the information written in stable storage survives the failure.

B. Process-Level Rollback Recovery with BLCR

One of the strategies evaluated is based on the Berkeley Labs Checkpoint Restart (BLCR) software package [10], a well-established library for executing system-level checkpoint restarts, and the MVAPICH MPI implementation developed by the Ohio State University [11], which has built-in support for BLCR in the latest release and supports AWS Elastic Fabric Adapter [12]. In this approach, no changes to the MPI

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

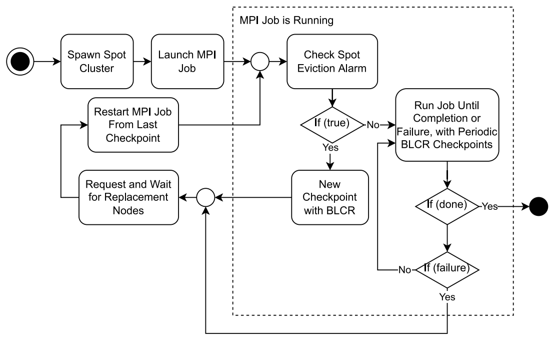
MPI Job is Running
Spawn Spot Cluster
Launch MPI Job
Check Spot Eviction Alarm
If (true)
No
Yes
Run Job Until Completion or Failure, with Periodic BLCR Checkpoints
Restart MPI Job From Last Checkpoint
Request and Wait for Replacement Nodes
New Checkpoint with BLCR
If (done)
Yes
No
If (failure)
Yes

Fig. 2. Action diagram for the BLCR-based rollback recovery strategy.

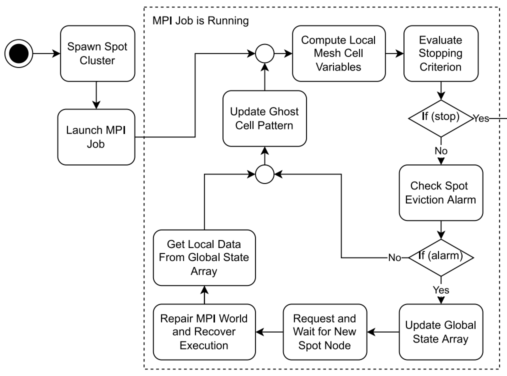
MPI Job is Running
Spawn Spot Cluster
Launch MPI Job
Compute Local Mesh Cell Variables
Evaluate Stopping Criterion
If (stop)
Yes
No
Update Ghost Cell Pattern
Check Spot Eviction Alarm
Get Local Data From Global State Array
If (alarm)
No
Yes
Repair MPI World and Recover Execution
Request and Wait for New Spot Node
Update Global State Array

Fig. 3. Action diagram for the ULFM-based rollback recovery strategy.

application code are required, although the BLCR package must be installed in every cluster node, and MVAPICH must be built from sources with compiler flags for BLCR support enabled.

Fault tolerance with BLCR is well researched and analyzed by several studies, and our purpose on evaluating this approach is mainly guided to collect data for comparison with the state-of-the-art ULFM-based strategy. Nevertheless, to take advantage of spot repossession notifications from AWS, we improve the basic BLCR strategy to be able to orchestrate checkpoints, launch new spot requests and listen to spot eviction alarms sent by the cloud provider. These new functionalities are handled by a job manager running externally to the spot cluster, which can be configured to trigger checkpoints periodically or only upon notification by AWS. Periodical checkpoints are required when spot eviction alarms are unavailable or unreliable, which may be the case for most providers. Some applications may also need more time than the short notice alarm to complete a checkpoint.

When placing bids for spot machines, the job manager launches a parameterized Terraform plan to build the virtual cluster through system calls to its command-line tool. The job manager waits for a configurable time interval before assuming no supply is available and then proceeds to increment the bids by a configurable percentage. This cycle repeats until all spot requests are met and the cluster is created. In this approach, the job manager keeps track of the spot requests states and launches a new bid as soon as an eviction occurs. After recreating all missing nodes, the job manager restarts the MPI application from the last checkpoint file. This cycle repeats until execution is completed. An overview of this strategy is presented in Figure 2.

The job manager launches MVAPICH jobs through a system call to the `mpirun_rsh` command-line tool. Checkpoints are enabled by default, and we configure them to be saved in a file at the virtual cluster shared file system. To trigger a checkpoint, the job manager uses the `cr_checkpoint` command to save the MPI application context into the previously defined file, passing the `mpirun_rsh` process identifier as an argument. Resuming execution from a saved context file is done through the `cr_restart` command. In this approach, when restarting execution from a checkpoint all MPI communications are reconstructed from scratch.

For the specific case of AWS, the job manager keeps track of *Amazon CloudWatch* events, from where the provider sends notifications before spot instance evictions. This is done by polling the CloudWatch API periodically. Upon detecting a scheduled termination event, the job manager immediately registers a bid for a new spot instance by re-executing the cluster Terraform plan and triggering a checkpoint save with BLCR. After Terraform finishes respawning the terminated nodes, the MPI job is re-launched from the previously saved checkpoint file.

C. In-Memory Rollback Recovery with ULFM

User-Level Failure Mitigation (ULFM) is the most recent effort from the Message Passing Interface Forum (MPIF) [13] to standardize semantics for fault tolerance in the MPI specification. It is based on a new flexible recovery API with a set of routines for error detection and propagation, making the recovery of failed MPI ranks possible. After communications are restored, applications can employ a fault tolerance strategy of their choosing to repair data and resume execution. Thus, fault tolerance in this context is the user's responsibility and happens at the MPI application-level [14].

Using the new set of ULFM routines currently available in the latest OpenMPI releases, we implemented a mechanism for rollback recovery in the Laplace's heat diffusion equation solver discussed in Section II-B. In this strategy, each rank keeps a duplicate array in memory with the last global mesh state information. Care must be taken given that a node must have sufficient memory to allocate a global state copy for each rank. Otherwise, the swap space usage will severely degrade the execution time. Similar to the BLCR approach, the application can be configured to checkpoint periodically if a notification system is not made available by the cloud provider, or to mitigate failures when the checkpoint routine takes more time than the short spot eviction notice. A high-level overview of the overall strategy is presented in Figure 3.

Although applying fault tolerance with ULFM requires considerable changes in the original application code and may not be easily implemented, there is great flexibility in terms of approaches that can be implemented.

Building on top of the parallel algorithm briefly described in Section II-B, we implement the following new routines. First, routines to allocate and update an array with global mesh checkpoint data were added. Secondly, we set a custom

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

error handler routine for the application. By default, MPI errors are handled by `MPI_ERRORS_ARE_FATAL`, which terminates the `MPI_COMM_WORLD` and stops the entire application. With the ULFM API, we are able to set a new `MPI_ERRORS_RETURN` handler, which will return an error code in case of failure [15].

After a spot eviction, all MPI processes currently hosted at that instance are unusable. By setting a custom `MPI_ERRORS_RETURN` handler, we define a set of routine calls to restore the MPI world and resume execution from the last in-memory checkpoint. To remove failed processes and stop current operations after a failure, we call the `MPIX_Comm_shrink` routine to create a new communicator by excluding all known failed processes from the parent communicator. For fully recovering our application, we need to restore our MPI world to the same previous size while also maintaining the same exact rank mapping. To re-spawn missing processes preserving rank mapping we use the `MPI_Comm_spawn` routine.

Algorithm 1 shows the application logic to fully recover execution as pseudo-code.

Algorithm 1 ULFM-based application recovery strategy

```
 1: set error handlers
 2: setjmp() for recovery
 3: build row and column communicators
 4: if recovering then
 5:     get data from last checkpoint
 6:     longjmp() to computation
 7: repeat
 8:     update ghost pattern                ▷ data exchange
 9:     if periodic checkpoint then
10:         save state checkpoint
11:     else if spot eviction alarm then
12:         save state checkpoint
13:         re-execute Terraform plans      ▷ asynchronous
14:     setjmp() for computation
15:     compute local updates and residual error
16:     all-reduce residual error with all ranks
17:     if criterion (residual or iterations) reached then
18:         converged = true
19: until converged
20: function ERROR-HANDLER(MPI_Comm, ...)
21:     if Terraform plan in progress then
22:         wait                            ▷ block execution
23:     replace failed communicators
24:     re-order communicators to original rank ordering
25:     longjmp() to recovery
```

IV. Predicting Execution Costs

Cost estimation is an important feature for enabling the migration of legacy workloads to public clouds [4]. Traditional HPC applications are usually tailored for a specific type of hardware where they will be executed, while in our case, we want to tailor the underlying infrastructure instead, building cost-effective virtual clusters in the cloud for specific applications. Cost estimation plays an important role in this regard. It can help users assess viability, balance the pros and cons of migrating to public clouds, and help them configure the best set of resources for their workloads.

We propose an empirical approach for estimating cloud infrastructure costs for specific workloads. Our proposal is based on executing an instrumented version of the target application for a short time span, then calculating the average iteration execution time. This data can then be used to estimate the total execution time and total variable infrastructure costs. For the case of spot clusters, unpredictable instance evictions can also affect the workload execution time and, consequently, the infrastructure costs. In this case, we first estimate costs for a *best-case scenario* (execution without failures), then estimate a *single failure impact on the total execution time*. We use the most recent provider-specific spot eviction rates to estimate the total number of failures.

Although this empirical approach can be useful, it has some limitations:

1) As public cloud environments are unreliable and have no consistency guarantees, predicting the execution time is challenging as the gathered data will be as reliable as the cloud environment.
2) This approach works only for specific HPC application classes with constant CPU requirements during its entire execution. This is true for our case-study application and other fixed geometry physics simulations, with a time-invariant number of operations.
3) Empirically estimating the total execution time is not cost-effective on cloud providers that bill users by hour. Some Amazon EC2 instances are billed by minutes, depending on the instance size and deployed region. On the other hand, all Vultr instances are billed by hour. This means that an instance that lives for just a couple of minutes will still be billed for a full hour, making our testing strategy moot for small workloads.

Besides the actual virtual machines, infrastructure costs may include Virtual Private Clouds (VPCs), persistent shared storage, storage for custom machine images, and more, depending on the provider. We focus our analysis on *virtual machine costs*, which are by far the largest expenditure and are proportional to the application execution time.

V. Experimental Results

In this section, we first provide an overview of the methodology and experimental environment adopted in this work (Section V-A). This is followed by an analysis of the results obtained with the case study application running on virtual clusters composed of persistent (standard) AWS and Vultr machines (Section V-B1) and ephemeral (spot) AWS machines (Section V-B2). Finally, we evaluate our empirical cost prediction strategy on the same virtual clusters in Section V-C.

A. Experimental Methodology and Environment

Clusters for the experiments are designed based on a fixed head node from which MPI jobs are launched, and a Network File System (NFS) server is hosted. Persistent block storage devices are attached to the head node to implement the shared file system, and all resources are isolated inside a single-zone Virtual Private Cloud (VPC) resource. Worker nodes are comprised of separate homogeneous virtual machine instances,

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

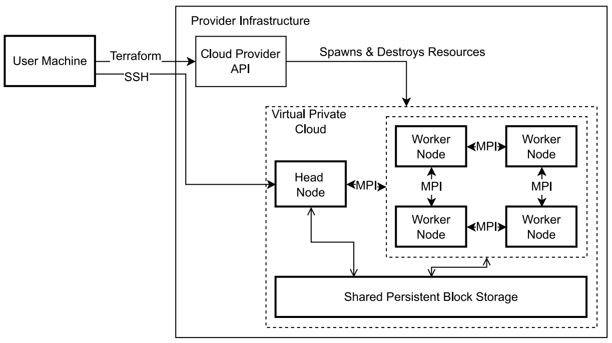

Fig. 4. High-level overview of cloud resources and setup of virtual clusters for the experiments.

which can be ephemeral (spot) or persistent (standard). The overall cluster setup is presented in Figure 4.

We focus our experiments on the low-end machine types with four to twelve virtual cores. We also run some experiments with one of the cheapest bare-metal infrastructure options in the market, the Vultr `vbm-6c-32gb` machine with Intel E-2286G processors. AWS machines are created at the `us-east-1` availability zone located at Washington DC (USA), and Vultr machines are created at the `dfw` availability zone located at Dallas (USA).

Table I presents all machine types used in this research, including some of their details as informed by each cloud provider. We prepared a series of cluster configurations with sizes varying between 1, 2, 4, and 8 worker nodes. For each experiment execution, we adopted one MPI rank per vCPU assignment. Every experiment is executed with three different problem sizes, as shown in Table II.

B. Performance Analysis

In this section, we analyse the execution time of the study case application (parallel solver implementation for Laplace's heat diffusion equations) when running on a virtual cluster composed of persistent AWS and Vultr machines (Section V-B1) and ephemeral AWS machines (Section V-B2).

TABLE I
TESTED PROVIDERS AND INSTANCE TYPES.

Cloud Provider	Instance Type	vCPUs	Base Cost (USD/h)	Spot Cost* (USD/h)
AWS	`c5a.2xlarge`	8	0.3080	0.2172
	`t3.2xlarge`	8	0.3328	0.1561
	`c5.2xlarge`	8	0.3400	0.1986
	`c6i.2xlarge`	8	0.3400	0.2265
	`c4.2xlarge`	8	0.3980	0.2216
Vultr	`vhp-4c-8gb`	4	0.0710	-
	`vhp-8c-16gb`	8	0.1430	-
	`vhp-12c-24gb`	12	0.2140	-
	`voc-c-8c-16gb`	8**	0.2380	-
	`vbm-6c-32gb`	6***	0.2750	-

*Week-average costs at time of publication.
**Dedicated vCPUs.
***Bare-metal CPUs.

TABLE II
TESTED CASE-STUDY PROBLEM SET.

Problem Label	Grid Size (Mesh Cells)	Iterations
`Problem Size A`	16,384	10,000,000
`Problem Size B`	1,048,576	10,000,000
`Problem Size C`	4,194,304	10,000,000

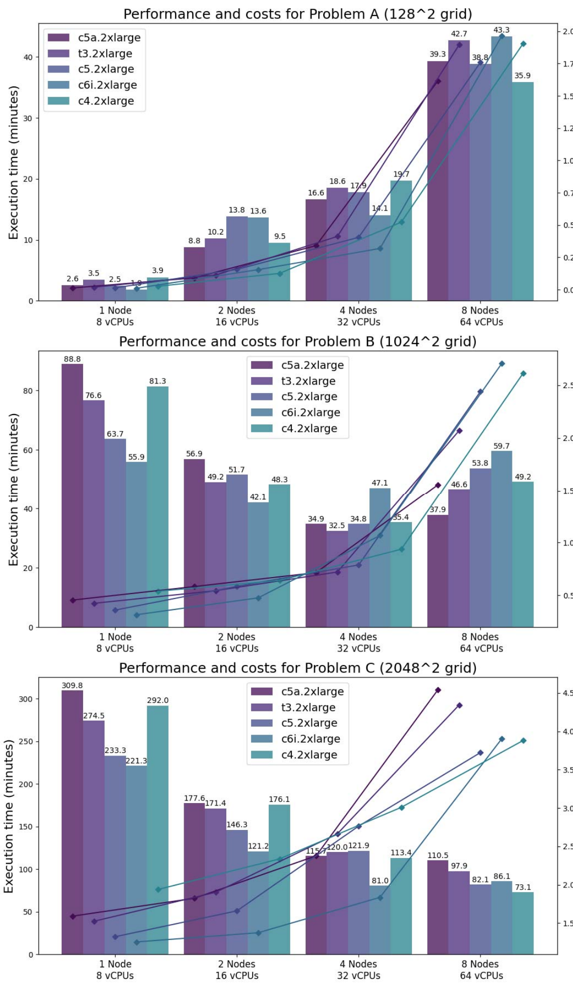

Fig. 5. Experiment results with AWS persistent machines.

1) Persistent Cluster Experiments: Figure 5 shows experimental results using standard AWS machines with the aforementioned cluster configurations without fault tolerance requirements. Bars show execution times (lower is better) and lines show the experiment costs (lower is better).

As expected, results show a clear picture indicating that, *for small problems at least*, horizontal scaling is not effective. For `Problem A`, horizontally scaling up our cluster slows down the execution considerably. The execution of `Problem B`, which in our context can be considered a medium-sized

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

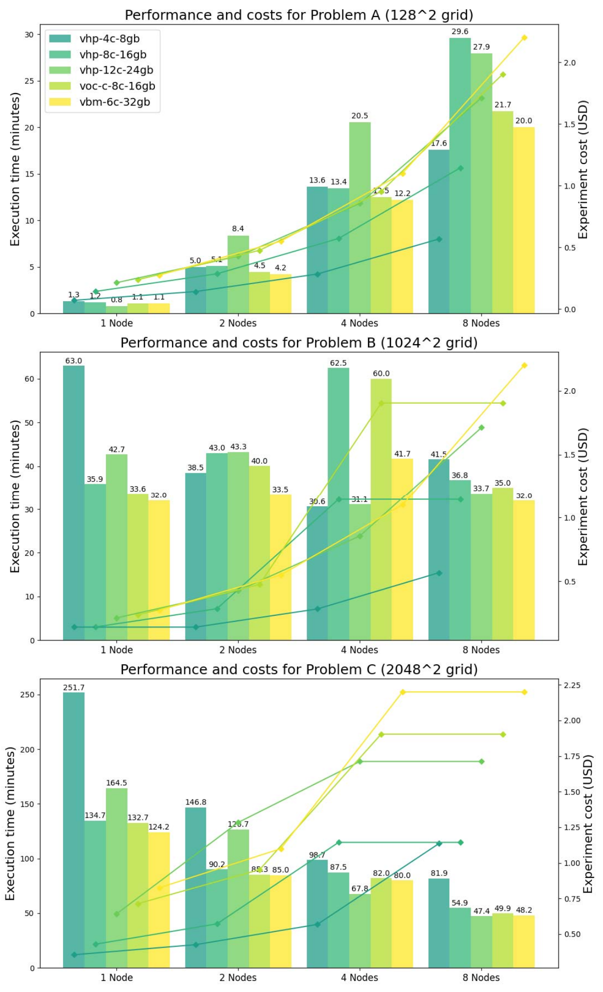

Fig. 6. Experiment results with Vultr persistent machines.

workload, attains a positive speed-up by scaling our virtual cluster up to 4 nodes. For medium workloads, further increasing the number of nodes is neither cost-effective nor better performance-wise. The horizontal scalability is much better for larger problems, as results for `Problem C` demonstrate. `Problem C` attains performance improvements with horizontal scaling up to the tested 8-node clusters, although with a considerable infrastructure cost increase. For some types of machines (especially ones with low bandwidth such as `c5a.2xlarge`), increasing nodes is not cost-effective, as performance gains are minimal and expensive.

The same problem set is executed using Vultr infrastructure detailed at Table I, and results are shown in Figure 6.

Experimental results with Vultr infrastructure cement previous results with AWS machines. The best performance is attained with dedicated Vultr bare-metal hardware, that is not much more expensive than virtualized multi-tenant infrastructure from the major cloud providers.

Regarding costs, persistent Vultr infrastructure is substantially cheaper than persistent AWS infrastructure considering public on-demand AWS prices. Vultr lower costs also come without sacrificing performance, as results show. With AWS, Azure, and other major cloud providers, most large organizations sign long-term usage contracts in exchange for considerable discounts, a common business model offered by the providers [26]. Alternative providers with low-budget options are an important factor for small organizations, which usually neither have interest in long-term usage contracts nor pass the infrastructure volume requirements to be able to receive discounts.

2) Ephemeral Spot Cluster Experiments: To evaluate our fault tolerance strategies, we execute two series of experiments with AWS spot infrastructure:

1) Fixed 8-node clusters executing three workload sizes (problems A, B, and C) with varying numbers of failures (i.e., spot instance evictions).
2) Clusters with varying sizes (1, 2, 4, and 8 nodes) executing three workload sizes (problems A, B, and C) with a fixed proportional number of failures.

For both series of experiments, we used `c4.2xlarge` machines to build the 8-node clusters, as they attained the best horizontal scaling and performance results in our previous evaluation. We also applied the bid strategy described in Section III-B, where we increase our bids by 1% after waiting 20 seconds without claiming a spot node. These parameters should be carefully set not to degrade application performance excessively, as checkpoints are expensive, specially ones based on process-level rollback restart such as BLCR.

Problems A, B, and C are the same workloads described in Table II. Spot instance evictions are induced in a controlled manner through the destruction of machines with a prepared automated script. We repeat both experiments evaluating the BLCR-based and ULFM-based fault tolerance strategies described in Section III-B and Section III-C.

Both strategies have two variations: periodic and preemptive. In the preemptive strategy, checkpoints are executed only when spot eviction alarms are triggered. In the periodic strategy, checkpoints are executed at configurable fixed intervals. We set the checkpoint interval as 360 seconds (6 minutes), a reasonable value given the performance results obtained with the standard persistent AWS machines. Results for the fixed 8-node spot cluster experiments are shown in Figure 7, where we compare the four fault tolerance strategies with the base results. Figure 8 shows the obtained results when varying the number of nodes of the spot cluster.

As expected, the BLCR-based strategies have considerable worse performance than the proposed ULFM-based strategies, which execute checkpoints in-memory. Previous research regarding BLCR note that the checkpoint overhead can be improved by using solid-state drives [16], although we notice that there is a substantial cloud infrastructure cost increase in this approach.

As the BLCR-based strategies must save entire process states into persistent storage regardless of the application problem size, there is a meaningful performance hit when using the periodic checkpoint approach. It is also worth mentioning that the BLCR checkpoint size depends on the machine image size and its characteristics, not being influenced by the application problem size. This is the main reason for

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

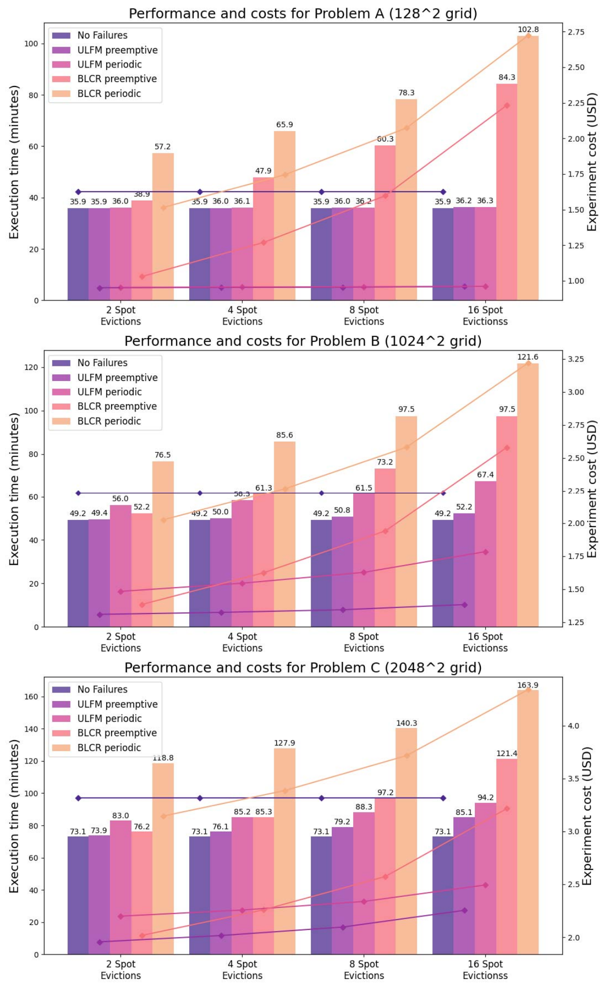

Fig. 7. Fault tolerance strategies experimental results (varying number of failures and fixed cluster size).

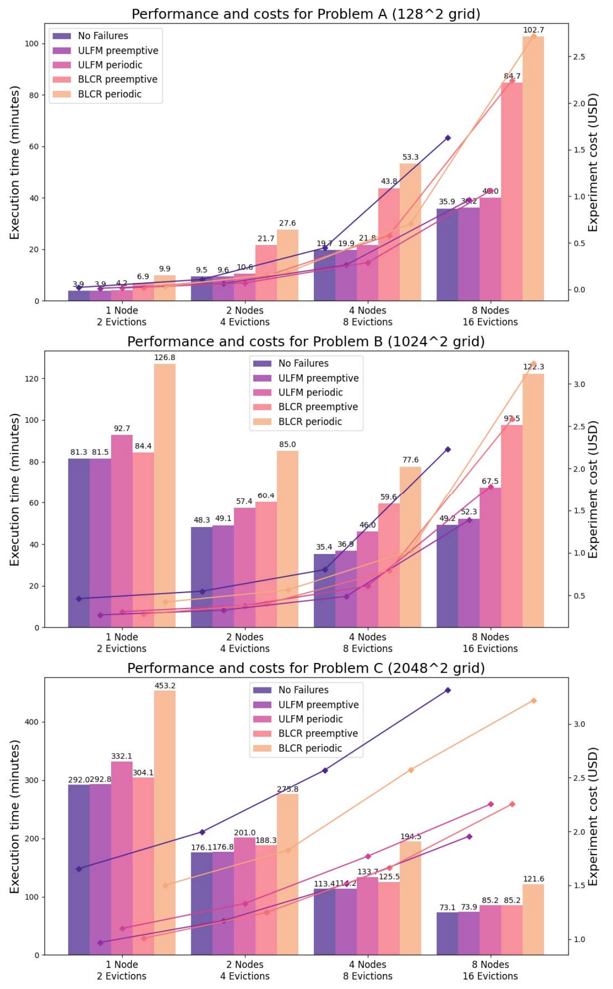

Fig. 8. Fault tolerance strategies experimental results (varying cluster sizes with proportional eviction events).

the degraded performance even when running the smaller Problem A. Corroborating with previous research, BLCR approaches (both periodic and preemptive) do not scale well with a growing number of failures. Costs can be even higher than using standard persistent clusters, as it can be noticed from the 8 and 16 spot evictions experiments for problems A, B, and C (Figure 7). This even further highlights the need for novel efficient fault tolerance strategies and the value of ULFM for the MPI specification.

The ULFM technique demonstrated good results for all three problems, and the cost-effectiveness is perceptible specially when analyzing Problem A results, where there is minimal performance overhead and a considerable cost reduction (44% infrastructure savings). Both ULFM techniques (periodic and preemptive) scaled well with the growing number of failures, not reaching a cost breakeven point with standard clusters in any of the experiments. Both preemptive strategies for BLCR and ULFM have better performance than their periodic counterparts, although the periodic BLCR approach has considerable worse performance than the preemptive one, making spot eviction notification systems important for the cost-effectiveness of the BLCR strategy.

Results show the same horizontal scaling pattern reached by using persistent infrastructure, and almost in all cases the fault tolerance strategies attained considerable cost savings, which were larger and more apparent for the bigger problems. Exceptions are for BLCR-based approaches executed with 8-node clusters and 16 failures, where spot costs surpassed persistent infrastructure costs when executing Problem B. While spot costs were smaller than persistent costs in Problem C with 8-node BLCR, the performance degradation is apparent (65% larger running time).

BLCR approaches (both periodic and preemptive) do not scale well with a growing number of failures. Costs can be even higher than using standard persistent clusters, as it can be noticed from the 8 and 16 spot evictions experiments

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

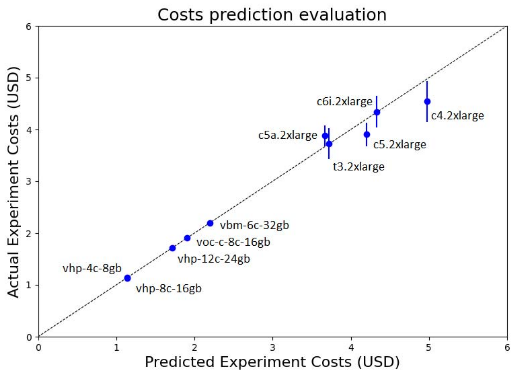

Fig. 9. Empirically estimated and actual execution costs for persistent clusters.

for problems A, B, and C (Figure 7). The ULFM technique demonstrated good results for all three problems, and the cost-effectiveness is perceptible specially when analyzing `Problem A` results, where there is minimal performance overhead and a considerable cost reduction (44% infrastructure savings).

C. Evaluation of the Empirical Costs Prediction Strategy

To evaluate our proposed cost prediction strategy, we executed an instrumented version of the case-study application with the same problem configurations described in Table II, except for the number of iterations, which was set to 10,000 (1% of the complete problem). The instrumented version has methods to compute each iteration execution time, allowing us to roughly estimate the total execution costs with a simple linear regression based on the gathered statistical data.

Figure 9 presents a comparison between estimated and actual total execution times for the experiments conducted with standard persistent infrastructure.

Results show that our approach is reasonable for problems with persistent clusters and no expected failures. Predictions for Vultr (left-hand side) costs are exact, as machines are billed by hour and costs can easily be estimated in hours timescales at our context. AWS machines that are billed by seconds (right-hand side) have inaccuracies, as our estimation mechanism is not accurate enough for predicting execution time at seconds timescales. Error bars represent the range of predictions calculated during our series of experiments.

Estimating problem running times at transient spot clusters is unreliable, as there is no current real-time prediction strategy embedded in our proposed solution for estimating how many spot evictions will happen during execution. Other unknown variables play a role on total execution time, such as time between placing bids and allocation of spare infrastructure, which has no guarantees of ever being met by the provider. Aggressive bid strategies can help reduce allocation time, however they also come with increased cost. In cases where there is no spare infrastructure for spot placements, there is nothing that the user can do besides waiting for an unknown amount of time, or changing machine types.

VI. Related Work

Recent studies on spot markets are abundant, although most focus on embarrassingly parallel applications or web services, while our research evalutes tightly-coupled HPC applications which face different challenges. Qu et al. [17] discussed both reliability aspects and cost-effectiveness of spot instances, however focusing on enterprise web applications. Teylo et al. [18] presented an extensive evaluation of spot machines for scheduling bag-of-tasks applications, which is also a different class of workloads from the ones considered in this paper. Zheng et al. [19] presented a comparative bibliographical investigation on the pros and cons of the spot and fixed-pricing schemes for cloud computing, whereas our study presents a *practical analysis* regarding spot market cost-effectiveness.

Research regarding BLCR performance for HPC applications running on traditional on-premise HPC clusters is also abundant, as the software package was launched in the early 2000s. Some recent studies with a similar approach that of our research were made by Gong et al. [20] and Voorsluys et al. [21], covering the usage of spot machines and an analysis of fault tolerance techniques using AWS infrastructure. Both studies investigate the effectiveness of BLCR-based checkpoint and replicated execution on reducing monetary costs. However, our research brings new contributions in terms of *new fault tolerance strategies based on ULFM*, and an *improvement upon BLCR with preemptive checkpoints through the AWS notification system*, which was only initially launched in 2015 and updated to the current state in 2018 [22].

Jangjaimon et al. [23] presented a BLCR-based checkpoint strategy based on a Markov model for predicting AWS spot revocations. Di et al. [24] introduced a BLCR-based checkpoint analysis on a large Google Cloud cluster but they did not consider spot instances and their peculiarities, focusing mostly on the BLCR checkpoint overhead for generic failures. Azeem et al. [16] also brought a practical study regarding BLCR and another system-level checkpoint mechanism called Distributed MultiThreaded Checkpointing (DMTCP) using AWS infrastructure. However, the authors did not consider spot instances and did not provide any cost-effectiveness data evaluation or analysis. Our research brings new fault tolerant strategies based on ULFM and improves the BLCR-based strategy with spot market specific features. We also discuss and test the actual *costs* of implementing these fault tolerance strategies in the cloud in addition to their performance analysis.

Finally, Yi et al. [25] presented an analysis of spot AWS resources cost-reductions granted by different checkpointing policies. However, the authors did not give any details regarding which technology they used for checkpointing. Our work is complementary, as it adds new policies based on preemptive failure alarms, which were not considered in that research.

Research regarding ULFM and HPC is mostly focused on traditional on-premise HPC clusters [5], [15]. To the best of our knowledge, there is a lack of research and practical evaluation of fault tolerance strategies based on this approach *on real spot cloud computing infrastructures*. Moreover, existing solutions do not take advantage of provider-specific features such as instance repossession alarms to reduce checkpointing overhead. Our work further pushes the state-of-the-art with a proposal and evaluation of empirical testing method to aid users on migrating legacy applications to public clouds.

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.

VII. Conclusion

Scaling cloud infrastructures for tightly-coupled HPC applications requires caution, as smaller problems do not scale well horizontally. For these kinds of problems, single-node infrastructures may be better than clusters and vertical scaling is advisable until memory limitations become a concern. Regarding the viability of using public cloud resources, we note that for maximum costs savings, spot markets are the only viable option for major providers, making fault tolerance a requirement. In this paper, we presented and analyzed in detail two fault-tolerant techniques tailored for cloud environments, proven useful for improving the cost-effectiveness of public clouds for HPC.

All fault tolerance strategies attained infrastructure cost savings of up to 55.5%, *with the exception of BLCR with periodic checkpoints*. More aggressive cost savings were reached using our proposed preemptive ULFM-based strategy, which had minimal performance degradation. Although the ULFM approach yielded excellent results in this regard, it is not an out-of-the-box fault tolerance solution like BLCR, possibly making it more expensive when considering software migration costs [26]. Our experiments showed that although spot instances with low eviction rates are almost guaranteed to be cost-effective for big workloads, caution must be taken when dimensioning virtual clusters for smaller tasks.

Estimating costs empirically has its own embedded costs, making it a possibly expensive solution. However, testing our workload at the same infrastructure that will be used for a production execution brings the best possible results, specially at cloud environments. There are unknown variables that could play a significant role performance-wise when using virtualized (or even bare-metal) cloud infrastructures, such as distance between physical machines running each virtualized node, which is basically random every time a new cluster is spawned. This means that empirical test results will be as reliable as the provider infrastructure, making advisable to run the production application at the same spawned infrastructure as soon as good empirical test results are reached.

To assist the migration of legacy HPC applications, we proposed a software platform with modules for automated virtual environment configuration based on sample experiments, which predicted our test case application execution time with up to 94% accuracy, accruing minimal extra costs for large workloads and a relatively small execution time overhead (up to 2% increase in total execution time). Finally, despite our focus on a case study application, our proposed fault tolerance strategies and job cost estimator can be generalized to a wide range of tightly-coupled MPI applications.

As future works, we intend to apply different ULFM-based strategies, specially dynamic approaches that do not require the full MPI world restoration, allowing applications to continue executing with a reduced number of nodes while balancing the workload. Furthermore, we plan to apply container migration technologies as an alternative for BLCR-based rollback restart, expecting easier workload migration and faster cluster setup time when executing workloads.

References

[1] S. Coghlan, and K. Yelick, "The Magellan final report on cloud computing," 2011

[2] J. Emeras, S. Varrette, and P. Bouvry, "Amazon Elastic Compute Cloud (EC2) vs. In-House HPC Platform: A Cost Analysis," 9th IEEE International Conference on Cloud Computing (CLOUD), 2016

[3] H. Richter, "About the Suitability of Clouds in High-Performance Computing," Academy & Industry Research Collaboration Center (AIRCC), 2016

[4] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," ACM Computing Surveys, 2019.

[5] N. Losada, P. González, M. J. Martín, G. Bolsica, A. Boutellier, and K. Teranishi, "Fault tolerance of MPI applications in exascale systems: The ULFM solution," Future Generation Computer Systems, 2020.

[6] H. Huang, and A. S. Usmani, "Finite Element Method," Finite element analysis for heat transfer: Theory and software, Springer, 1994.

[7] J. Kafle, and L. P. Bagale, "Numerical solution of parabolic partial differential equation by using finite difference method," Journal of Nepal Physical Society, 2020.

[8] F. B. Kjolstad, and M. Snir, "Ghost cell pattern," In Proceedings of the 2010 Workshop on Parallel Programming Patterns (ParaPLoP), Association for Computing Machinery, 2010.

[9] Amazon official press release, available at: https://press.aboutamazon.com/news-releases/news-release-details/amazon-web-services-announces-spot-instances-amazon-ec2/.

[10] P. H. Hargrove, J. C. Duell, "Berkeley lab checkpoint/restart (BLCR) for linux clusters," Journal of Physics: Conference Series, 2006.

[11] K. Dhabaleswar, K. Tomko, S. Karl, and M. Amitava, "The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI Library for HPC," 2013.

[12] Amazon Elastic Fabric Adapter, available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html#efa-basics

[13] MPI Forum ULFM discussion, available at: https://github.com/mpi-forum/mpi-issues/issues/20/.

[14] I. Laguna, D. Richards, T. Gamblin, M. Schulz, and B. Supinski, "Evaluating user-level fault tolerance for MPI applications," 21st European MPI Users' Group Meeting (EuroMPI/ASIA '14). Association for Computing Machinery, 2014.

[15] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bolsica, and J. Dongarra, "An evaluation of user-level failure mitigation support in MPI," Recent Advances in the Message Passing Interface, Springer, 2012.

[16] B. A. Azeem and M. Helal, "Performance evaluation of checkpoint/restart techniques: For MPI applications on Amazon cloud," 9th International Conference on Informatics and Systems, 2014.

[17] C. Qu, R. N. Calheiros, and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," Journal of Network and Computer Applications, 2016.

[18] L. Teylo, L. Arantes, P. Sens, and L. Drummond, "Scheduling bag-of-tasks in clouds using spot and burstable virtual machines," IEEE Transactions on Cloud Computing, 2021.

[19] L. Zheng, H. Zhang, L. O'Brien, S. Jiang, Y. Zhou, M. Kihl, and R. Ranjan, "Spot pricing in the cloud ecosystem: A comparative investigation," Journal of Systems and Software, 2015.

[20] Y. Gong, B. He, and A. C. Zhou, "Monetary cost optimizations for MPI-based HPC applications on amazon clouds: Checkpoints and replicated execution," IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2015.

[21] W. Voorsluys and R. Buyya, "Reliable Provisioning of Spot Instances for Compute-intensive Applications," IEEE 26th International Conference on Advanced Information Networking and Applications, 2012.

[22] Amazon official press release, available at: https://aws.amazon.com/about-aws/whats-new/2018/01/amazon-ec2-spot-two-minute-warning-is-now-available-via-amazon-cloudwatch-events/

[23] I. Jangjaimon and N. Tzeng, "Effective Cost Reduction for Elastic Clouds under Spot Instance Pricing Through Adaptive Checkpointing," IEEE Transactions on Computers, 2015.

[24] S. Di, Y. Robert, F. Vivien, D. Kondo, C. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). Association for Computing Machinery, 2013

[25] S. Yi, D. Kondo, and A. Andrzejak, "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud,", In HAL-00464222, 2010.

[26] M. A. dos Santos and G. G. H. Cavalheiro, "Cloud infrastructure for HPC investment analysis", RITA, 2020.

Authorized licensed use limited to: University Library Utrecht. Downloaded on November 22,2023 at 23:29:41 UTC from IEEE Xplore. Restrictions apply.