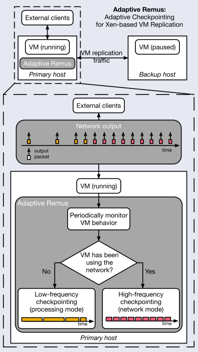
INTERNATIONAL JOURNAL OF PARALLEL, EMERGENT AND DISTRIBUTED SYSTEMS, 2016
http://dx.doi.org/10.1080/17445760.2016.1162302

Adaptive Remus: adaptive checkpointing for Xen-based virtual machine replication

Marcelo Pereira da Silva, Rafael Rodrigues Obelheiro and Guilherme Piegas Koslovski

Graduate Program in Applied Computing, Department of Computer Science, Santa Catarina State University, Joinville, SC, Brazil

ABSTRACT
With the ever increasing dependence on computers and networks, many systems are required to be continuously available in order to fulfil their mission. Virtualization technology enables high availability to be offered in a convenient, cost-effective manner: with the encapsulation provided by virtual machines (VMs), entire systems can be replicated transparently in software, obviating the need for expensive fault-tolerant hardware. Remus is a VM replication mechanism for the Xen hypervisor that provides high availability despite crash failures. Replication is performed by checkpointing the VM at fixed intervals. However, there is an antagonism between processing and communication regarding the optimal checkpoint interval: while longer intervals benefit processor-intensive applications, shorter intervals favour network-intensive applications. Thus, any chosen interval may not always be suitable for the hosted applications, limiting Remus usage in many scenarios. This work introduces Adaptive Remus, a proposal for adaptive checkpointing in Remus that dynamically adjusts the replication frequency according to the characteristics of running applications. Experimental results indicate that our proposal improves performance for applications that require both processing and communication, without harming applications that use only one type of resource.



Adaptive Remus quantifies VM metrics to infer the current hosted application load. With this information, the mechanism adjusts the checkpointing frequency between two modes. (I) networking mode: increases the checkpointing frequency whenever output traffic is detected on the VM interface; and (II) processing mode: when there is no output traffic in the VM interface, the mechanism reduces the checkpointing frequency, increasing the VM execution time. This approach improves application performance by dynamically adapting the checkpoint interval.

ARTICLE HISTORY
Received 29 January 2016
Accepted 2 March 2016

KEYWORDS
Replication; Xen; virtual machines; primary-backup; checkpointing; Remus

CONTACT Guilherme Piegas Koslovski guilherme.koslovski@udesc.br

1. Introduction

Applications can have their execution partially or totally compromised by the occurrence of faults in processing, communication or storage resources they use. Recent reports indicate that network datacentre outages cost US$5600 on average [29], lasting an average of 107 min [28]. In some cases outages can last several hours [24,31]. To overcome this situation, critical applications rely on high availability algorithms, protocols, and infrastructures to maintain correct execution in spite of faults. Usually, specialised infrastructures require a high upfront investment while application-level approaches complicate development and affect the execution time.

In parallel, network datacentres have explored the virtualization of computing resources for server consolidation and easier management of the distributed substrate topology. In a virtualized environment, users run their applications on a set of virtual machines (VMs), which are provisioned by the introduction of an abstraction layer between the physical hardware and the hosted operating system. The abstraction layer, called the virtual machine monitor (VMM) or hypervisor, has access to the entire execution state of a VM [15]. This technology can be exploited to introduce transparent fault tolerance to hosted applications. Motivated by this opportunity, fault tolerance virtualized tools have emerged for different VMMs, such as KVM Kemari [36], VMware vSphere Fault Tolerance [18] and High Availability [19], Marathon everRun Xen [23], and Remus [8].

Specifically, Remus is an open source primary-backup replication mechanism [4] for Xen-based [2] virtual machines. Remus periodically replicates an entire VM (CPU state, memory, disks) at very high frequency, saving dozens of checkpoints per second. All checkpoints saved in the primary host are asynchronously transferred to a second host (the backup). In this scenario, users typically interact with the VM hosted in the primary node, but when a crash failure occurs in this host, the backup VM is activated and starts to respond in a few milliseconds, resuming execution from the last checkpoint. Checkpointing is incremental: each checkpoint contains only the changes in state since the previous checkpoint. Such use of replication provides high-frequency synchronisation between primary and backup replicas, which results in fast failover [13].

Remus provides good fault tolerance in a transparent manner [21], but may significantly degrade application performance [14]. Since checkpointing disrupts VM execution, CPU-intensive applications perform worse as the checkpoint interval is shortened. On the other hand, longer checkpoint intervals are detrimental to applications that are sensitive to network latency. To compound the problem, the checkpoint interval used by Remus is fixed and preconfigured. Even though the characteristics of hosted applications may be known in advance (regarding the use of networking or processing resources), the choice of an ideal checkpoint interval is not trivial. Indeed, applications with mixed use of resources increase the complexity. A high frequency of checkpointing favours the networking code, however it induces an overhead in the processing step. In this scenario, an average value is not a good option, since the application will never experience an optimal frequency in either the communicating or processing parts of the algorithm.

Added latency is a common issue in primary-backup replication, since in many protocols the primary only sends responses to clients after the corresponding state updates have been acknowledged by the backup [4]. In Remus, as a new checkpoint will only be saved after the previous one has been completely stored in the backup, the predetermined checkpoint interval may be exceeded depending on the VM processing load. During this period, the VM keeps running as long as necessary, increasing the user-perceived communication latency.

Considering the current limitations of Remus, this paper introduces a mechanism that dynamically adapts the checkpointing frequency based on outgoing VM network flow. The mechanism, named *Adaptive Remus*, alternates between two modes: one with high checkpointing frequency to benefit networking applications, and a second one with low frequency to increase VM run time (benefiting CPU-bound applications). The experimental results indicate a promising use of adaptive VM replication, especially for applications with mixed use of CPU, memory and network resources. In the evaluated scenarios our adaptive variant outperformed the original Remus. For instance, in network-bound applications, *Adaptive Remus* reduced transfer time by approximately 33%. For applications which alternate between network, memory and CPU resources, performance was 29% higher compared to Remus. For CPU-bound applications, the performance was similar to the original Remus under a low checkpointing frequency. In summary, the main contributions of our work are twofold:

- A comprehensive measurement study of the performance impact of Remus on hosted applications;
- The introduction of an adaptive checkpointing algorithm in Remus to reduce replication overhead.

The remaining of this paper is structured as follows. Section 2 details Remus, the VM replication mechanism discussed in this paper. The replication overhead of native Remus mechanism is quantified in Section 3. Section 4 presents *Adaptive Remus*, a proposal to dynamically adapt the replication frequency. Experimental analysis is discussed in Section 5. Related work is reviewed in Section 6, while Section 7 concludes the work pointing perspectives to future work.

2. Remus: a VM replication mechanism

The virtualization of computing resources allows the complete encapsulation of a VM (including operating system and applications). By exploiting this opportunity, Remus provides fault tolerance for hosted applications in a transparent and cost-effective manner [8], as the mechanism is focused on commodity hardware and requires a low upfront cost deployment. Moreover, Remus is completely agnostic to the operating system running in the VM, which runs without any additional configuration.

Remus saves dozens of checkpoints per second that are transferred to a backup host. For each checkpoint the mechanism performs a stop-and-copy procedure similar to the one used in Xen VM live migration [6]. Figure 1 illustrates the execution scenario of Remus. Each host has two network interfaces: one for external access, to interact with users, and another dedicated to replication traffic. In this scenario, the VM disks are synchronised in both hosts before the replication process is started. With both disks synchronised, a paused VM is created in the backup host and a full copy of the running VM memory is transferred to the backup.

After the initial disk synchronisation, the checkpointing period begins. Periodically, the VM is paused and a new checkpoint is saved to a local buffer. At this stage only the memory content that has changed since the last checkpoint is saved. Remus resumes the VM execution while the buffer data is transferred to the backup host. Once the buffer is backed up, a confirmation is sent back to primary host, which

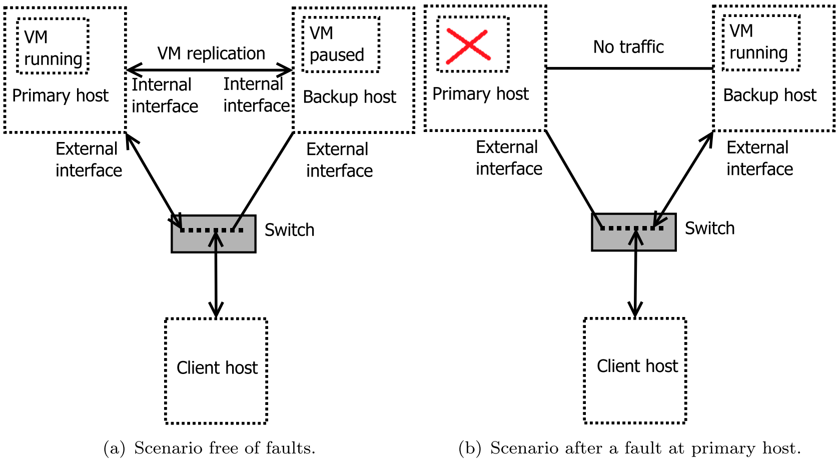


Figure 1. Remus execution scenario to replicate a VM. A running VM is positioned at primary host while the backup host keeps a paused VM.

may or may not immediately save a new checkpoint. During the period after VM execution is resumed until the next pause to save a checkpoint (and until this new state is completely transferred) there is no guarantee about what has been or is being processed in the VM. The data processed in the VM during this period will be lost at the occurrence of any fault at the primary host, that is, the backup resumes the VM execution considering the last checkpoint completely received and acknowledged. In summary, VM execution is constantly in speculative mode.

As shown in Figure 1(b), the backup activates the VM replica when a fault is detected. Indeed, a timeout is used to identify the absence of the primary VM. When the backup resumes the VM, an unsolicited ARP reply is sent on the local network to inform that the MAC addresses related to VM IP has changed [34]. In summary, the fault model implemented by Remus follows three properties: (i) tolerance to a single crash failure at the primary host; (ii) when a simultaneous fault occurs at the primary and backup hosts, the VM remains in the previous stable state; and (iii) no network requests will be answered until the VM state is completely saved on the backup host.

2.1. Replication of VM resources

Remus is based on the Xen VM live migration mechanism [6]. To perform a live migration, Xen suspends the VM and transfers its memory contents to a destination host. The execution is resumed on the new host and the VM on the original host is destroyed. Remus adapted this mechanism by periodically suspending the VM to save the changed data to a local buffer, and later resuming VM execution on the primary host, while the buffer is transferred to the backup; this shortens the downtime when compared to a migration. This procedure is periodically executed, characterising the checkpointing process. When a new checkpoint is saved the VM disk resident on the primary host is synchronised with the backup image. This synchronisation is performed by an updated version of *Distributed Replicated Block Device* (DRBD) [32].

Some modifications were performed to reduce the overhead introduced by the replication mechanism [30]. In Xen-based implementations, most of the time spent saving the state to the local buffer is due to an excessive number of calls between the VM and the hypervisor. An event channel (or virtual interruption) was implemented for paravirtualized scenarios, reducing the number of calls to suspend and resume the execution of a protected VM. In addition, Remus identifies the memory pages to be saved in the buffer, ignoring those not used since the last checkpoint. Finally, checkpoints are compressed to reduce the transfer time of the local buffer to the backup host.

2.2. Linearizability

The primary-backup replication model implemented by Remus enforces a consistent view of the system in the occurrence of faults. In Remus, during the replication process, users only interact with the primary VM without knowledge of the replication mechanism. To ensure such consistency – a linearizability [16] property – service replies are only sent to users after the backup acknowledges reception of the corresponding checkpoint [13].

For implementing such property, Remus regulates the outgoing network traffic of the protected VM. The VM receives and processes incoming network traffic without any restriction. However, outgoing traffic is held in a buffer at the primary, called the network buffer. Packets in this buffer are released only after primary and backup have synchronised their states. In Remus, the network buffer is needed to ensure that the status of network connections is stable [8]. Figure 2 illustrates network buffer operation. Outgoing network traffic (TX) from the VM is blocked until the backup acknowledges the last checkpoint. Incoming traffic (RX) is not influenced by this buffer, that is, the VM is able to receive data at any time.

2.3. Checkpointing frequency

Remus uses a fixed and predefined interval to implement checkpointing. The checkpoint interval is set before starting the VM replication and has an indirectly influence on checkpointing duration (CD),

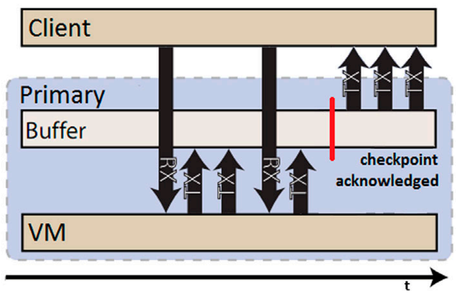


Figure 2. Outgoing network traffic from the VM is released after checkpoint synchronisation between primary and backup hosts. Meanwhile, packets are stored into a temporary buffer at the primary host.

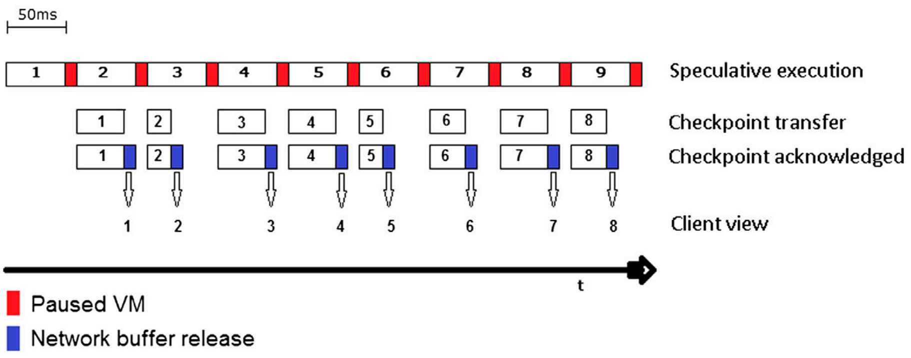


Figure 3. The native Remus mechanism.

which encompasses the time needed to pause the VM, save its state to the local buffer and transfer the contents of this buffer to the backup host. CD depends on the VM workload and affects the checkpoint copying and transfer times. As the network buffer holds outgoing packets for a CD, the latency perceived by networking applications is increased.

Figure 3 shows the replication mechanism with a 50 ms predefined interval. Taking as an example period 3, the VM run time before saving a new checkpoint was exactly 50 ms. The interval was respected because the checkpoint transfer time for period 2 was lower than the default value (50 ms). In this case, the mechanism waits for the stipulated interval be reached, ensuring 50 ms run time, and starts saving a new checkpoint. After confirming the state in the backup, an ACK is sent to the primary host and the network buffer is finally released. The vertical arrows indicate the user's view of the corresponding numbered periods, which shows that a user has a delayed perception of VM processing.

In the previous example, the checkpoint interval was constantly respected. However, this situation is not always perceived: when CD is greater than the predetermined interval, the next interval will be equal to CD. This condition leads to a delay in the next checkpoint, increasing the latency in communicating applications since the network buffer depends on the checkpoint ACK to be released [25]. Figure 4 exemplifies this scenario. The 50 ms predetermined interval was exceeded in periods 2, 3 and 6. The second period was increased to 200 ms because the transfer of the previous checkpoint (period 1) lasted for exactly 200 ms. The higher the VM run time, the more likely the next checkpointing will exceed the predetermined interval, since the amount of data to be replicated will probably increase.

Depending on when an VM output packet is generated, latency can be higher [25]. If a packet is generated near the pause to save a new checkpoint, the latency is the sum of pause and transfer times. On the other hand, if the package is generated immediately after a new saved checkpoint (at the beginning of a period in speculative mode), the latency will be the sum of the execution time until the next pause, pause and transfer times. This can be categorised as the worst case latency perceived

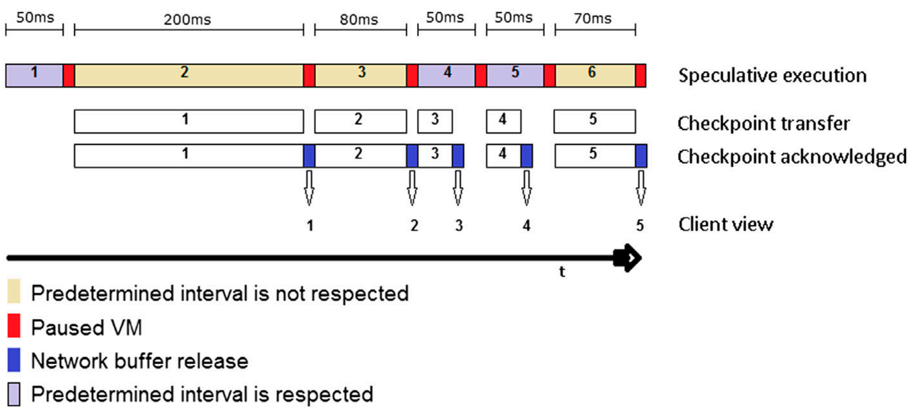


Figure 4. The run time in speculative mode varies with the replication time when the defined interval is exceeded.

by a user. Indeed, it is difficult to predict the latency an outbound packet may suffer, since the release of the network buffer is related to the value of CD, which natively on Remus is variable: the standard replication process does not always follow the predetermined checkpoint interval, as depicted in Figure 4.

3. A measurement study of Remus replication overhead

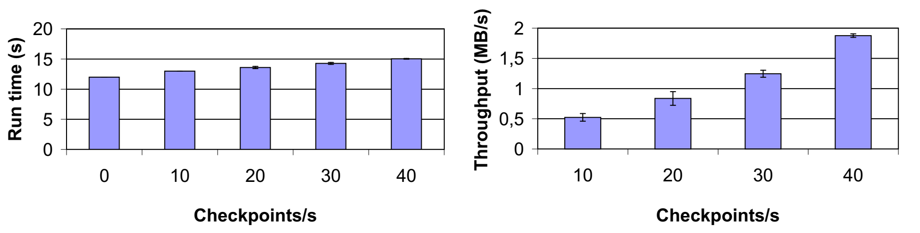
In the original experimental evaluation of Remus [8], benchmarks were used to quantify the impact caused by the replication mechanism in the execution of hosted applications, considering the variation of network latency in the physical topology. In their initial evaluation, the authors did not investigate the appropriate checkpointing frequency for communicating applications, and suggested that the overhead could be meaningful, without actually quantifying it. To complement their work, we conducted an analysis of the overhead imposed by Remus considering two scenarios: networking and non-networking applications. For all tests the frequency was set at 10, 20, 30 or 40 checkpoints/s following previous scenarios [8,30]. A zero (0) label was used to indicate tests performed on an unprotected VM. Each experiment was repeated 30 times, and the results show sample means with 95% confidence intervals (given by error bars).

The testbed consisted of three commodity computers with AMD Phenom II X4 2.8 GHz processors, 8 GB RAM, running Ubuntu 12.10 (kernel 3.5.0-17-generic) and interconnected in a LAN as shown in Figure 1(a). Primary and backup hosts have two network interfaces each, one for interacting with clients (100 Mbps) and the other dedicated to VM replication (1 Gbps using a crossover cable). Each host is virtualized with Xen 4.2.1 and protected by the native Remus mechanism, running DRBD version 8.3.11-remus. The replicated VM consists of two VCPUs, 20 GB disk and 1 GB RAM running OpenSUSE 12.2 (x86_64). The metric used for assessing application performance was benchmark run time. In addition, the throughput between primary and backup hosts was measured.

The average replication throughput between the hosts with an idle VM varied between 0.48±0.006 MB/s (10 checkpoints/s) and 1.76±0.059 MB/s (40 checkpoints/s). Indeed, results indicate that the larger the interval, the lower the average throughput, showing that checkpoint compression effectively reduces the volume of data to be replicated [8].

3.1. Non-networking applications

In this scenario, the overhead on hosted applications is only due to the need to periodically stop and resume VM execution to save a new checkpoint.



(a) Average run time. (b) Average primary-backup throughput.

Figure 5. Run time and throughput for CPU benchmark.

Table 1. Amount of memory allocated in the memory benchmark.

VM memory size	Memory allocated
1 GB	200,000 pages/781.25 MB
2 GB	400,000 pages/1,562.5 MB
4 GB	800,000 pages/3,125 MB

3.1.1. CPU-bound applications

To investigate the performance of CPU-bound applications we used the sysbench benchmark,[1] specifically the module to calculate all possible prime numbers smaller than 10,000. Figure 5(a) shows the run time for each checkpointing frequency. The run time without replication was approximately 12 s, while a 10 checkpoints/s configuration resulted in 8% performance overhead. In the worst case, with 40 checkpoints/s, the overhead was 25%. These results are in accordance with original Remus analysis: a higher replication frequency induces greater overhead on CPU-bound applications [8]. Figure 5(b) depicts the average replication throughput, which is similar to the idle VM scenario.

3.1.2. Memory-bound applications

Specifically for memory-bound applications, we varied virtual machine size to represent different memory configurations. VMs were provisioned and individually analysed with 1 GB, 2 GB, and 4 GB of RAM. An application was developed to allocate a predefined number of 4096-byte pages on protected VMs. The amount of memory allocated was set to roughly 80% of VM capacity, as shown in Table 1.

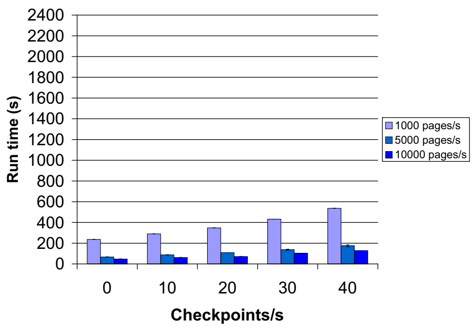
Random data were written on every page, at different rates (1000, 5000, and 10,000 pages/s), representing possible usage scenarios. To assess the impact of varying these parameters, we measured the application run time, which encompasses allocating the pages and sequentially writing on all of them. We also measured the primary-backup throughput, to see whether the replication link could be a bottleneck. Figure 6 depicts the results.

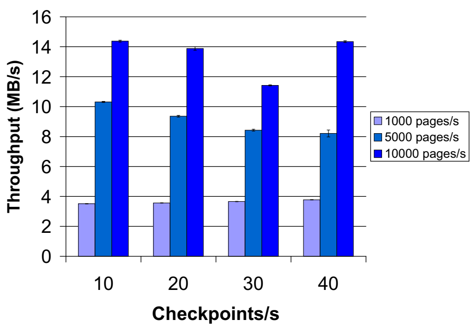
The results show that run time grows with VM size and replication frequency, and is inversely proportional to the writing rate, as expected (Figure 6(a), (c) and (e)). Less obviously, the overhead compared to an unprotected VM varies as well, increasing with memory size, replication frequency, and writing rate, from 22% for a 1 GB VM at 10 checkpoints/s and 1000 pages/s (the best case) to 282% for a 4 GB VM at 40 checkpoints/s and 10,000 pages/s (the worst case).

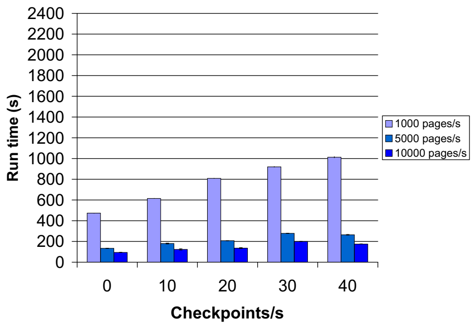
The average primary-backup throughput (Figure 6(b), (d), and (f)) exhibits small variations (less than 20%) for different VM size and replication frequencies, at a given writing rate. In general, throughput is directly proportional to writing rate, and inversely proportional to VM size and replication frequency. Although the throughput increases from 30 to 40 checkpoints/s on a 1 GB VM, overall the results highlight that the average throughput between primary and backup hosts, regardless of VM memory size, was negligible when compared to the replication link capacity.

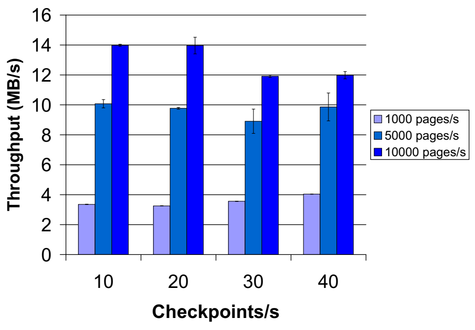
3.1.3. I/O-bound applications

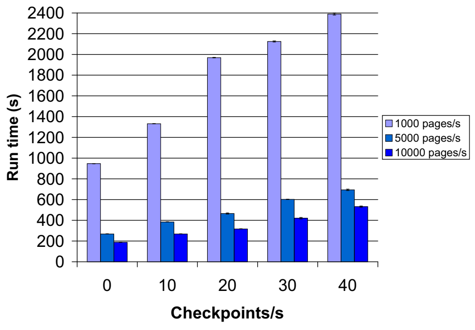
The sysbench benchmark I/O module performs random read and write operations (3 GB total) on the local VM disk. As observed in the previous scenarios, a low checkpointing frequency gives better VM


(a) Average run time for 1 GB VM.


(b) Average primary-backup throughput for 1 GB VM.


(c) Average run time for 2 GB VM.


(d) Average primary-backup throughput for 2 GB VM.


(e) Average run time for 4 GB VM.

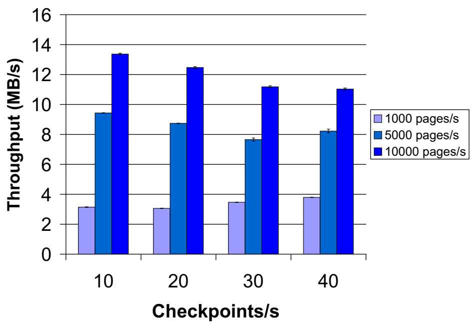
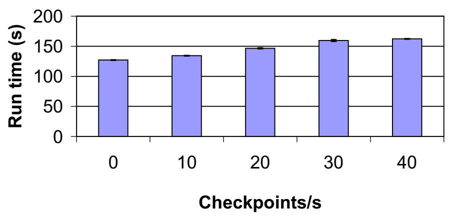

(f) Average primary-backup throughput for 4 GB VM.

Figure 6. Run time and throughput for memory benchmark.


(a) Average run time.

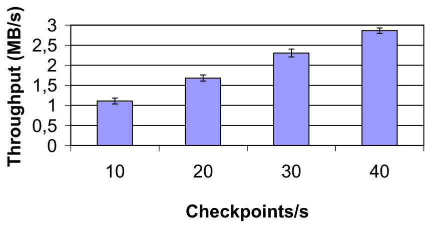

(b) Average primary-backup throughput.

Figure 7. Run time and throughput for I/O benchmark.

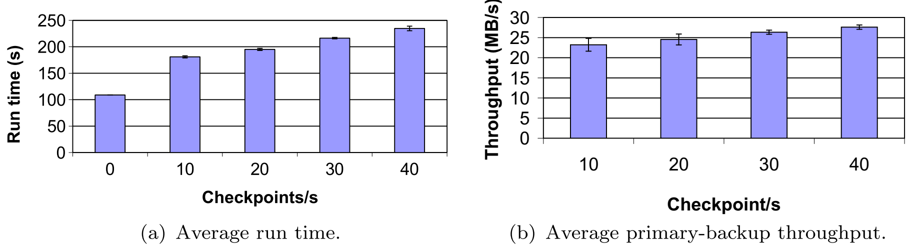


Figure 8. Run time and throughput for BIND compilation.

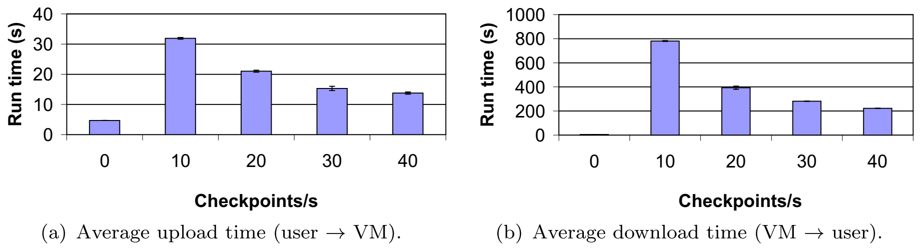


Figure 9. Upload and download transfer times of a 50 MB file.

performance (Figure 7). The worst-case overhead was seen with 40 checkpoints/s (27% overhead). The slight variation between 30 and 40 checkpoints/s follows the initial Remus experiments [8]. Compared with baseline results (no Remus) the overhead of I/O operations is tolerable for this application. In addition, the throughput (Figure 7(b)) follows the same pattern of the other experiments.

3.1.4. Source code compilation

To analyse an application with combined use of CPU, memory and I/O resources, we measured the compilation time for BIND[2] 9.9.4 (roughly 400 k lines of code), as shown in Figure 8(a). The replication overhead was 66% in the best case (with 10 checkpoints/s). For this type of application, a lower frequency (longer checkpoint intervals) is preferable. In this scenario the throughput was higher compared to previous tests (Figure 8(b)), as the diversified memory contents decreases the efficiency of checkpoint compression.

3.2. Networking applications

The performance of network-sensitive applications is impacted by the network buffer activity. This set of scenarios evaluates the overhead of TCP-based applications under different checkpointing frequencies.

3.2.1. Data transfer: download and upload

This experiment analyses the communication latency perceived by a user that interacts with a protected VM. We measured the transfer time of a 50 MB file between the user and the VM under different checkpointing frequencies and directions (download and upload) using SCP.[3]

Figure 9(a) shows the results for the upload scenario, where the file is transferred from the user to the protected VM. The download scenario (VM → user) is presented in Figure 9(b). For the upload scenario a configuration with 40 checkpoints/s has a 218% overhead compared to an unprotected VM. Analyzing the download with the same configuration, the lowest overhead is approximately 4900% (4.4 s without Remus against 222 s). In the worst case (10 checkpoints/s) the overhead is approximately 18,000%. Indeed, for download scenarios the replication overhead may be prohibitive. The difference in download and upload values stems from the interference of the network buffer. Even in speculative

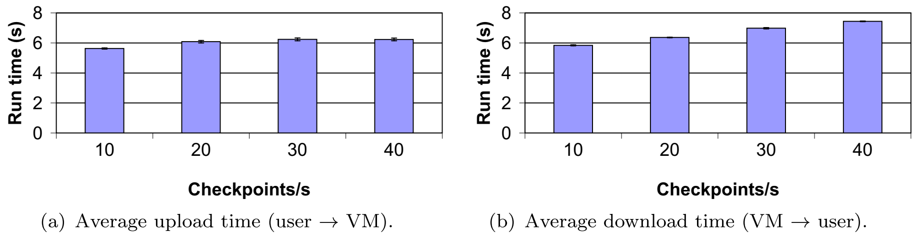


Figure 10. Upload and download times of a 50 MB file without network buffer.

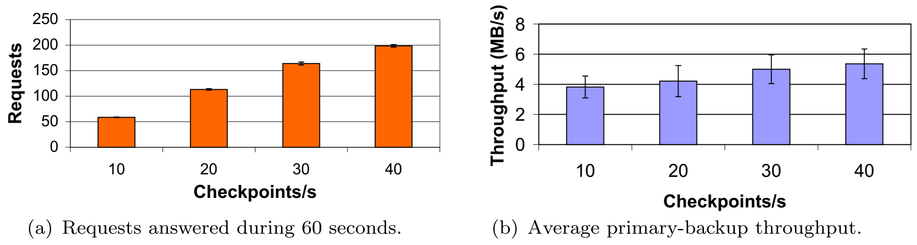


Figure 11. Number of requests answered during 60 s and average primary-backup throughput.

mode the VM never stops receiving requests, but outgoing packets are held until the checkpoint is completely saved in the backup host.

For performance analysis, the network buffer may be disabled, which does not guarantee linearizability (Section 2.2). Figure 10(a) and (b) show the upload and download transfer times, respectively, without using the network buffer. While for a 40 checkpoints/s configuration with the network buffer enabled the upload time was 13.7 s, this value decreased to 6.2 s without buffering. Considering the download time (40 checkpoints/s configuration), the transfer was performed in 222 and 7.4 s with and without buffering, respectively. We emphasise that the network buffer is not optional in Remus, being essential to provide consistency in the presence of faults.

3.2.2. *ab – Apache HTTP server benchmarking tool*

This experiment uses the ab benchmark[4] to measure the performance of a virtualized Apache 2.2.22 web server (using the default OpenSUSE configuration). The application is serving 100 simultaneous connections during 60 s. Each connection fetches a 72,950-byte HTML page generated by PHP using *phpinfo()*. Figure 11(a) shows the number of requests answered. The 10 checkpoints/s configuration is the worst case, as the web server successfully processed just 59 requests. With 40 checkpoints/s 199 requests were processed; comparing to the unprotected scenario (9770 requests), there is an 4,800% overhead. The average primary-backup throughput (Figure 11(b)) followed the pattern of the previous tests.

3.3. *Discussion*

The experimental results indicate that Remus introduces non-negligible overhead on hosted applications running in a protected VM. Specifically, memory-bound applications suffer a greater impact due to the nature of the replication mechanism: all memory contents changed since the previous checkpoint must be transferred to the backup host. For CPU-bound and I/O-intensive applications, a higher checkpointing frequency increases the number of VM-pause events and consequently the hosted application overhead. In summary, for non-networking applications a lower replication frequency is preferred. The opposite was observed for network-sensitive applications. A high checkpointing frequency results in small checkpoints and consequently a faster network buffer release.

Regardless of the checkpointing frequency, Remus prioritises the VM processing time. Speculative mode allows for immediate execution after checkpointing, independently of the transfer time to the backup host. However, the outgoing network traffic is retained in a buffer until the checkpoint is saved in the backup host. These characteristics are necessary to guarantee linearizability and unfortunately increase latency for networking applications.

The average primary-backup throughput proved to be irrelevant when compared to the total capacity of the dedicated replication link, independently of virtual machine memory size. For example, the compilation test obtained the highest average, approximately 25 MB/s for all checkpointing configurations. In geographically distributed environments (with the primary and backup nodes hosted in different datacentres), throughput can be a limiting factor. In such case, the links interconnecting the datacentres usually have lower bandwidth and higher latency than a local network. We leave this investigation to future work.

4. *Adaptive Remus*

While longer periods in speculative mode benefit CPU-, I/O- and memory-bound applications, checkpointing at higher frequencies favours networking applications. This antagonism complicates the choice of an appropriate checkpointing configuration, especially when the VM is subject to a combined use of CPU, memory, I/O and network resources. While the native Remus mechanism prioritises VM processing time to the detriment of networking applications, we present a proposal for dynamically adapting the checkpointing frequency based on the outgoing network flow of the protected VM.

The adaptation mechanism quantifies two VM metrics (processing load and network output flow) to infer the current hosted application load. The VM processing load is quantified by analysing the checkpointing duration (CD). For example, a checkpoint which takes 80 ms to be saved and transferred to the backup host indicates the VM is under a higher load compared to other CD which lasts only 30 ms. Complementary, the VM network output flow (NOF) is quantified by observing the number of bytes sent by the VM since the last checkpoint. With this information, the mechanism, named *Adaptive Remus*, can deduce the current application load and adjust the checkpointing frequency. The mechanism operates in two modes, networking and processing:

- Networking mode: this mode increases the checkpointing frequency whenever output traffic is detected on the VM interface.
- Processing mode: when there is no output traffic in the VM interface, the mechanism reduces the checkpointing frequency, increasing the VM execution time. This is the default mode.

While several low-level optimizations to improve checkpointing performance have already been added to Remus [30], *Adaptive Remus* takes a different approach, aiming to improve application performance by dynamically adapting the checkpoint interval.

Adaptive Remus quantifies the NOF by monitoring the virtual network interface on the primary host. NOF is measured by collecting the number of network output bytes on the VM interface at the end of each checkpointing in processing mode. When NOF increases, the mechanism adjusts to networking mode. This mode runs for a predetermined number of rounds (NCN – number of checkpoints in networking mode), and then NOF is measured again. It is not trivial to determine the value of NCN, as Remus has no knowledge about the characteristics of running applications. Measuring NOF at every checkpoint induces management overhead, increasing CD and consequently delaying the start of the next checkpoint. We opted for only measuring NOF after a NCN and at each interval in processing mode.

In networking mode a new checkpoint will be saved immediately after receiving an ACK from the backup. In this mode, CD is close to the checkpoint transfer time, which means that the network buffer is released as soon as possible. This condition speedup the network buffer release. The checkpointing frequency used in processing mode should prioritise the VM execution time, reducing disruptions to hosted applications. However, depending on the VM processing load, very low frequencies can

Table 2. Variables used in the *Adaptive Remus* flowchart of Figure 12.

Variable	Description
PMF	Default processing mode frequency
NCN	Number of checkpoints in networking mode
x	A counter to verify if NCN has been reached
t	A discrete time instant
NOF	Network outgoing flow
T_{start}	Checkpointing start time
T_{end}	Checkpointing end time

Table 3. Experimental scenarios, goals and metrics.

Scenario	Main resource	Goals and metrics
DaCapo Tomcat, BIND compilation	Combined CPU, memory and I/O	Analyse run time
File transfer	Network	Analyse the impact of the network buffer on latency-sensitive applications
NAS MG, RUBiS	Combined CPU and network	Analyse the run time of networking applications
CPU utilisation	Primary host CPU	Quantify the overhead of *Adaptive Remus* on primary host

greatly increase the time needed to identify the data to be saved to the local buffer. This delay can be interpreted by the backup node as a failure of the primary, prompting activation of the backup replica. Based on experimental results we selected a default 100 ms checkpointing frequency (or 10 checkpoints/s) for the processing mode. It should be noted that, in processing mode, whenever CD exceeds the predefined checkpoint interval the mechanism loses its adaptive capacity; in this case, the checkpoint interval becomes the CD value. For example, when a 120 ms transfer time occurs with a 100 ms checkpoint interval, the next checkpoint is delayed by 20 ms. This is the standard operating mode of Remus (as exemplified in Figure 4).

The flowchart presented in Figure 12 depicts the adaptive algorithm. The variables and parameters used to express the algorithm are summarised in Table 2. In processing mode (enabled when $\text{NOF}_{(t-1)} == 0$), the algorithm behaves similarly to the original Remus. PMF determines the checkpointing frequency that ensures a minimum VM execution time. CD represents the total checkpointing time (CD = (T_{end} - T_{start})) and is compared to PMF. When it is greater than PMF a new checkpointing is immediately started. Otherwise, the VM remains executing in speculative mode until the execution time reaches PMF (sleep (PMF−CD)). NOF is measured at each checkpoint in processing mode (measure $\text{NOF}_{(t)}$) and compared to the previous one. A difference greater than zero indicates the mechanism should switch to networking mode. To avoid delaying a new checkpoint, NOF is measured only after a minimum NCN is reached (while $x <$ NCN). Depending on NOF, the mechanism remains in networking mode or returns to processing mode.

5. Experimental analysis

The experimental analysis of *Adaptive Remus* comprises CPU, memory, and network benchmarks. As discussed in Section 3, replication throughput is not considered a limiting performance factor for the LAN scenarios analysed in this work. The average run time was used as a metric for assessing the performance of hosted applications. We also measured CPU utilisation on the primary host to quantify the overhead induced by *Adaptive Remus*. Table 3 summarises the experiments. Except for the NAS experiment, which required the addition of three computers, the testbed is equal to that used in Section 3.

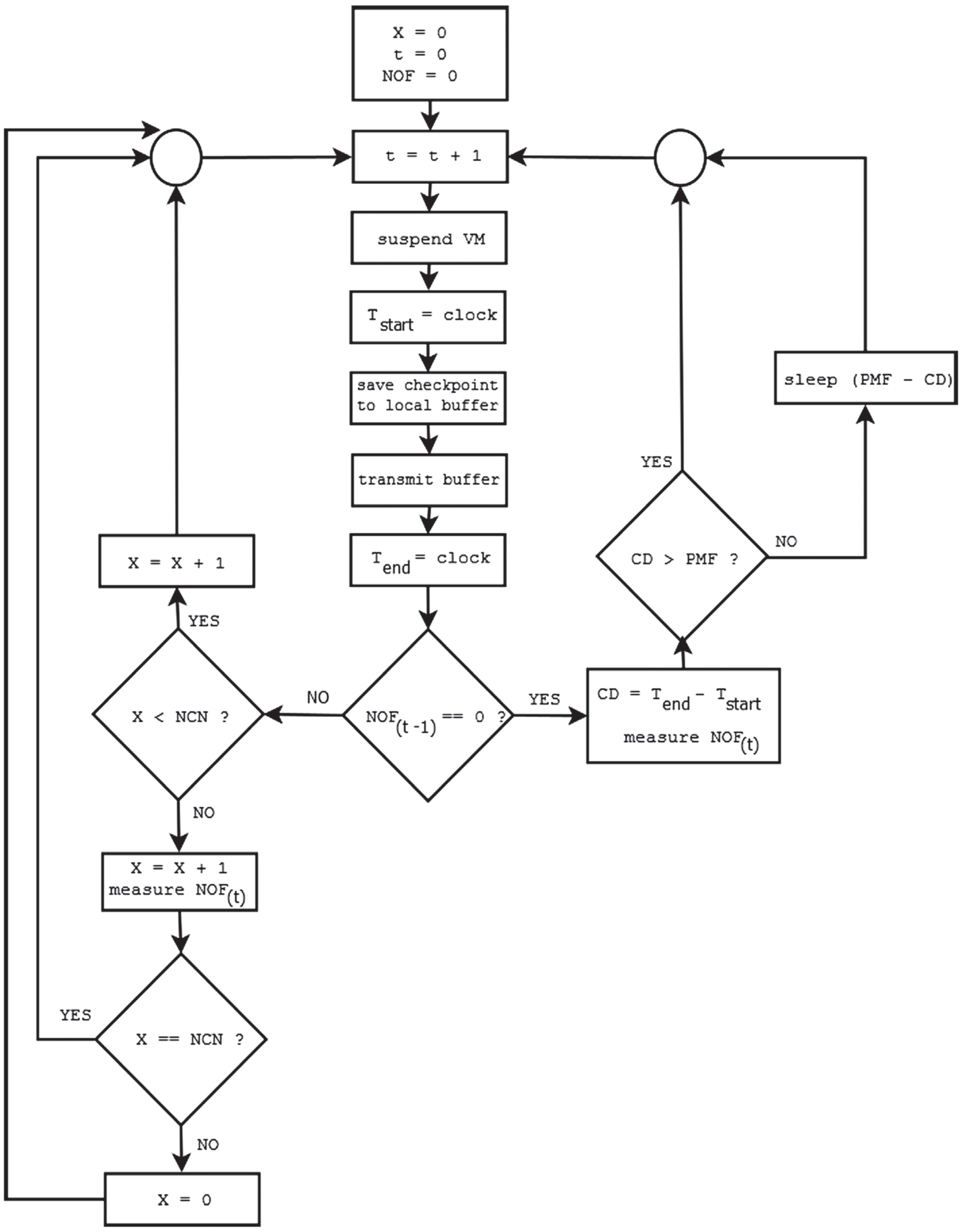
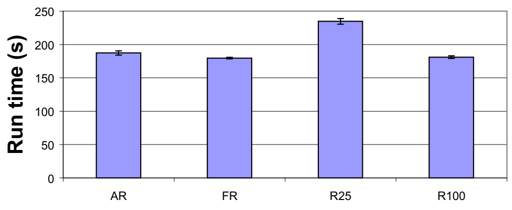


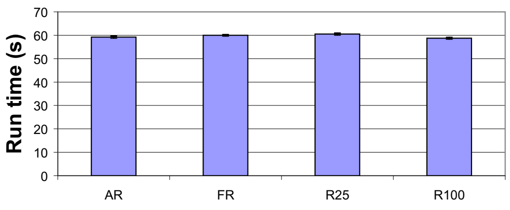
Figure 12. Flowchart proposed for the adaptive checkpointing in Remus.

5.1. Remus configurations and variants

Adaptive Remus (AR for short) was compared to two baselines, original Remus with fixed checkpoint intervals of 25 ms (R25) and 100 ms (R100). We also implemented another adaptive approach, guided only by VM processing time without considering the network activity. This version – Floating Remus (FR) – toggles between two checkpointing frequencies, 25 and 100 ms. The alternation is defined by



(a) BIND compilation.



(b) DaCapo Tomcat.

Figure 13. Run time for CPU-, memory-, and I/O-intensive applications.

CD, as follows: if CD is less than 25 ms, the interval is set to 25 ms; if CD is equal to or greater than 25 ms, the interval is set to 100 ms. In FR, CD is measured after each checkpoint, and is the factor responsible for adjusting the replication frequency (unlike AR, which relies on network traffic). Thresholds and parameters (NCN $= 30$) were based on the performance tests presented in Section 3.

5.2. Applications with combined use of CPU, memory, and I/O resources

Run time is used as a metric to compare the performance of applications sensitive to CPU, memory, and I/O operations on the four Remus variants.

5.2.1. Source code compilation

In this experiment we measured the time needed to compile BIND on a protected VM. This process involves use of CPU and memory without network communication. There is also significant I/O activity, as dozens of new files are created and saved on the local VM disk. Figure 13(a) shows the results of the four variants.

CPU-bound applications perform better with a lower checkpointing frequency, as longer intervals minimise the interference in VM execution. FR and AR self-adapted to the nature of the experiment obtaining similar results to R100, which had the best performance. When comparing AR to R25, the former has an average time 25% lower than the latter. For R25, whenever CD is greater than 25 ms, the interval is extended to CD. This workload gets better performance from longer checkpoint intervals, and AR was nearly as good as the best variant (R100).

5.2.2. DaCapo Tomcat

The DaCapo Tomcat benchmark [3] simulates a set of queries submitted to a Tomcat server.[5] This test does not involve the network, being totally dependent on the VM processing time. Figure 13(b) shows the results for the four Remus variants. All versions have a similar performance due to the constant processing load imposed by the benchmark (CD always exceeds 100 ms). Measuring NOF on every checkpoint led CD to be slightly longer in AR; primary host overhead is discussed in more detail in Section 5.5.

5.3. Networking applications

A 50 MB file transfer was carried out to evaluate the performance of networking applications. The transfer time is approximately 4.5 s when Remus is not running and is used as base for comparisons. Figure 14 summarises the results. A 100 ms checkpoint interval is better for CPU-bound applications but the same does not apply to communicating applications (R100 becomes impractical in this scenario). R25 and FR have similar results (both versions have a default 25 ms checkpoint interval). Transfer time for AR was approximately 93% lower than for R25 and FR, as AR immediately starts saving a new checkpoint when the previous one is finished. Consequently, the network buffer is released as often as possible.

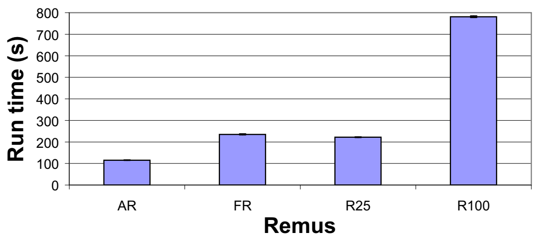


Figure 14. Average transfer time of a 50 MB file (VM → user).

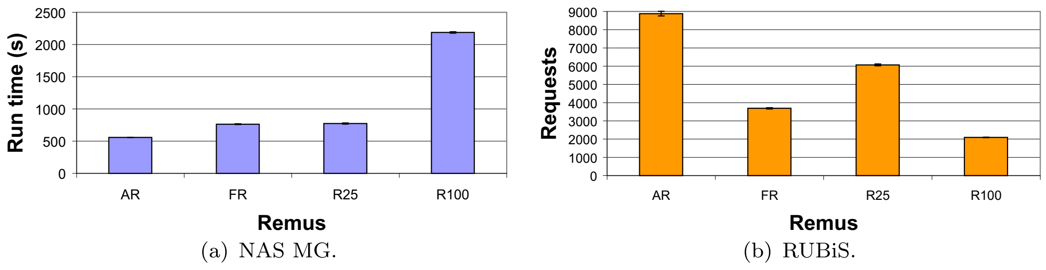


Figure 15. Results for applications with combined CPU and network use.

5.4. Applications with combined Use of CPU and network resources

Two applications sensitive to CPU time and network resources are evaluated in this experiment. Run time is adopted as the comparison metric.

5.4.1. NAS benchmark

The NAS benchmark [1] comprehends a set of applications designed to assess performance in parallel computing architectures. For this experiment, we selected the class B configuration with 4 VMs (only one was protected). Among the available benchmark applications on NAS NPB3.3-MPI version, the MG (multi-grid) was selected in this experiment. MG has a combined use of CPU and network resources with a stable memory consumption; in a previous study [35], NAS MG had 59.6% of CPU utilisation, generated 19 Mbps of network traffic and used 117 MB of memory (average values).

As depicted in Figure 15(a), AR has the lowest run time while FR and R25 showed similar results. For networking applications is preferable a checkpointing frequency regulated by NOF (as implemented by AR) and not a mechanism guided by fixed configurations, such as FR, R25 and R100.

5.4.2. RUBiS

RUBiS is an auction site modelled after eBay.com which requires processing and network resources in different proportions.[6] This application is used to evaluate application design patterns and application server performance and scalability. In this experiment, the testbed to execute RUBiS was composed of (i) a server to simulate the basic functionality of an auction site, such as navigation, search, purchase and sale negotiations; and (ii) a user emulator, which generates a sequence of interactions with the site. Only the VM hosting the server was replicated. The metric chosen to represent the performance of RUBiS was the number of operations served by the site. Each operation performs a research (by category and region) and displays the selected items. The scenario simulated 300 parallel users interacting with the site.

Figure 15(b) shows the number of successful operations. We highlight the overhead of delayed network buffer release in R100 and FR versions. Also, AR completed approximately 350% more operations than R100. FR, which uses a 100 ms checkpoint interval, had the second worst performance. The number of completed operations for R25 was 31% less than for AR.

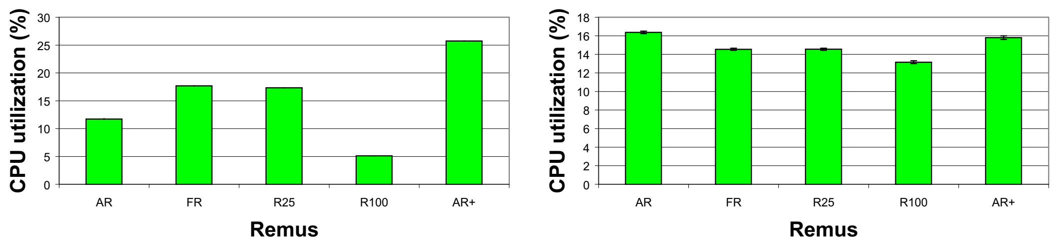


Figure 16. Average CPU utilisation on the primary host.

5.5. CPU overhead on the primary host

The applicability of *Adaptive Remus* in production scenarios is affected by the overhead imposed by the dynamic checkpointing mechanism. As the Remus decision algorithm runs only on the primary host, the average CPU utilisation on this host was collected in two scenarios: (i) with an idle VM (Figure 16(a)) and during the execution of DaCapo Tomcat (Figure 16(b)), which represents the worst case CPU overhead observed in our experiments. For comparison purposes we implemented a version of *Adaptive Remus* (identified as AR+) that always executes in network mode, and corresponds to the worst-case execution profile in terms of CPU utilisation.

Figure 16(a) shows CPU usage increasing with checkpointing frequency. Except for AR+, average utilisation did not reach 20% in any version. R25 and FR show a similar percentage due to its standard 25 ms checkpoint interval. Considering R100 and AR, the difference is induced by the AR operations to measure the VM network flow every checkpoint. AR+ represents the worst case for this scenario: almost no data is replicated when the VM is idle, causing a extremely short CD. Even in this case, CPU usage remained below 30%. The performance overhead is reduced when the VM has a constant processing load as shown in Figure 16(b). In this scenario, the 100 ms interval is always exceeded and therefore all versions have the same checkpointing frequency.

5.6. Discussion

Adaptive Remus obtained an equivalent or improved performance in all experiments, adapting the checkpointing frequency to the characteristics of application running on the protected VM. Unlike R25, R100 and FR, *Adaptive Remus* in networking mode increases as much as possible the frequency, saving a new checkpoint immediately after the preceding was saved and confirmed in the backup host. Analyzing CPU-bound applications, AR adopts an 100 ms interval when no communication is involved, resulting in a similar performance to R100 for applications that require higher longer periods of execution. For example, with R25, a 26 ms CD results in exactly 26 ms of speculative run time, while in AR a 100 ms interval is selected. Moreover, when CD exceeds the stipulated checkpoint interval in R25 and R100, the interval is extended to CD, indirectly prioritising processing time. For applications that depend on both network and processing, such as RUBiS and NAS MG, waiting 100 ms to save a new checkpoint is unacceptable. In these cases, increasing the checkpointing frequency provides better application performance.

6. Related work

Related work can be divided in two categories. First we review proposals involving adaptive checkpointing in non-virtualized environments, and then we discuss work on VM replication.

6.1. Non-virtualized environments

While innovative in Remus, mechanisms to adapt the checkpointing frequency have been explored in the literature. For example, [40] seeks to minimise the overhead of checkpointing a programme by increasing and decreasing the frequency guided by checkpoint size and cost. The proposal presented in [38] adapts the checkpointing frequency in embedded real-time systems to minimise power consumption. The proposal tolerates a preconfigured number of faults for a given task. These studies focused on controlling the application run time by minimising failover time, without worrying directly with responsiveness during failure-free periods.

In addition, checkpointing and recovery techniques have been addressed in high performance computing (HPC) scenarios [12]. The work described in [5] proposes the use of adaptive checkpointing in computational grids, adjusting the interval between checkpoints according to estimated job run time and frequency of resource failures. To reduce overhead during failure-free periods, incremental checkpointing and checkpoint compression are used [26,27], just as in Remus. However, both techniques can increase failover time, since recovery operations (uncompressing checkpoints, restoring state from increments) are more costly. In Remus, these operations are performed on the backup immediately upon reception of a checkpoint and not during recovery, which minimises failover time (the backup VM always has the last saved state) and has no impact on the performance of hosted applications.

Some high availability models for HPC are based on fault forecasting methods. The mechanism described in [22] indicates that, for a well-known application and physical topology, it is possible to infer around 80% of network and memory faults and approximately 47% of I/O faults before they actually occur. This information is used to proactively migrate a process, eliminating the checkpointing overhead.

6.2. Virtualized environments

In virtualized environments, replication mechanisms are usually based on VM live migration techniques. In live migration itself, adaptive checkpointing is not used since migration is sporadic, not continuous; the goal is to minimise service downtime when a VM needs to be moved to another physical host, e.g. for host maintenance, load balancing, or energy savings [17,37].

A performance study evaluating the quality of voice transmissions protected with Remus was presented in [20]. The delay introduced by the network buffer led to a traffic explosion at the end of a checkpointing, degrading audio quality. *Adaptive Remus* could be useful in this environment since it is able to accelerate the release of the network buffer.

SecondSite [30] explored Remus to completely replicate a site between distributed data centres. The goal is to replicate a VM in another datacentre in order to tolerate natural disasters, hardware failures, or energy interruptions. VM connectivity is preserved by BGP routing changes, with routers configured in advance. In this work, checkpoint size is the determining factor for the VM run time. SecondSite can benefit from *Adaptive Remus* to dynamically adapt the checkpointing frequency over WAN links. A Remus update to perform virtual machine checkpointing based on clients' point of view was proposed in [9]. The extension, termed COLO, considers primary and backup VMs as active replicas (requests are processed by both replicas). However, messages from backup VM are intercepted and analysed by primary node. In this scenario, a checkpoint is only performed when clients' point of view is not consistent (replicas have produced a different response for the same subset of requests). Although COLO reduces the number of checkpoints performed by Remus, it requires intervention on TCP/IP kernel module to identify response similarities. Moreover, COLO degrades the performance of I/O intensive applications and increases overhead when multiple applications are hosted on a VM.

A primary-backup solution to replicate VMs and tolerate crash failures was implemented in [36]. The mechanism was implemented without a network buffer, replicating the VM state on the occurrence of I/O events. In the same vein, a mechanism to tolerate crash failures in the KVM virtual machine monitor was proposed in [7]. Unlike Remus, the mechanism uses a shared disk without buffering network packets. These approaches reduce the overhead imposed on the VM processing time but do not guarantee linearizability. A solution to reduce the overhead of the checkpointing process is presented in [39]. The time to save a checkpoint is guided by the VMM memory access. Although the procedure can reduce the transfer time of checkpoints, dynamic adaptation of checkpointing frequency is not addressed.

Memory compression techniques have being applied in Xen-based systems to decrease the data volume saved and transferred in checkpoints [10,30], but depending on the application, they are not always effective. *Adaptive Remus* is agnostic to memory compression techniques and can be potentially combined to outperform virtual machine replication.

A solution based on VMware [33] implemented VM replication by logs that are sent to the backup immediately after instruction execution on the primary. Similar to Remus, the system uses a network buffer but automatically searches for a server with enough resources to act as backup (to implement this feature the solution uses shared disk storage). In this scenario, VM processing time varies depending on the capacity of the selected backup host. This commercial solution does not apply to commodity servers and is limited to the use of only one virtual processor per VM.

A theoretical analysis of distributed diskless checkpointing for virtual machine fault tolerance was discussed in [11]. This work attributes the replication overhead and performance degradation of hosted applications to disk synchronisation, and attempt to use RAID storage systems to rely on memory, rather than disks, to store checkpoints. When a fault is detected, the virtual machine storage disk is reconstructed from RAID system. As the authors have indicated Remus as a possible framework for implementing a prototype, the adaptive checkpointing introduced by *Adaptive Remus* is a natural candidate for being incorporated in the solution.

The work described in [14] combines two approaches: checkpointing frequency adjustment and the reduction of the transferred volume. While this proposal addressed three variables in their formulation (memory, bandwidth and interval) *Adaptive Remus* uses the native Remus optimizations (checkpoint size is reduced with compression techniques and identification of recently changed pages). The upper limit for the checkpoint interval was defined by the authors as 2 s, 20 times the limit adopted by *Adaptive Remus* (which aims at preserving VM reliability).

7. Conclusion

Computating infrastructures are subject to interruptions. Users of critical applications rely on high availability solutions, which usually demand the acquisition of specialised hardware or software. However, with the advent of virtualization, techniques for providing high availability have become economically accessible. In this context, Remus was introduced as a VM replication mechanism for the Xen hypervisor that tolerates crash failures. Based on the primary-backup replication model, Remus keeps an updated copy of the VM, encapsulating its applications and operating system. Through high-frequency synchronisation (dozens or hundreds of checkpoints per second), the replicated VM hosted by a backup node is kept paused standing ready to assume execution in the event of a fault at the primary host.

The use of Remus in production scenarios faces a limitation, as the fixed checkpoint interval is suboptimal for networking applications. In this scenario, this work proposed *Adaptive Remus*, a mechanism capable of operating in two checkpointing modes, networking and processing. Based on the VM network output and no longer on a single fixed and predetermined interval, the mechanism is able to accelerate the network buffer release, increasing the checkpointing frequency and decreasing the latency perceived by networking applications. In the absence of network flow, the mechanism returns to its default processing mode, reducing the replication frequency. Experimental results show that, compared to Remus, *Adaptive Remus* is able to significantly improve performance for networked applications while providing equivalent performance for non-networked ones.

As future work, we identified two main avenues of research: (i) Even decreasing the VM run time as much as possible there is no guarantee that the checkpoint transfer time is less than the stipulated interval. A possible approach would be to fix the beginning of a checkpoint, regardless of the previous checkpoint have already been saved on the backup or not. The current version of Remus allows this configuration in experimental mode (there is no documentation available on its use). We aim to combine *Adaptive Remus* with this features and se if we can further improve application performance. (ii) So far we have focused on LAN deployments of *Adaptive Remus*. An interesting research question is whether we can apply the same ideas in geographically distributed environments, with replicas residing in different datacentres.

Notes

1. Available at https://launchpad.net/sysbench.
2. Available at http://www.isc.org/downloads/bind/.
3. Part of OpenSSH, http://www.openssh.com.
4. Available at http://httpd.apache.org/docs/2.2/programs/ab.html.
5. Available at http://tomcat.apache.org.
6. Available at http://rubis.ow2.org/.

Acknowledgement

The authors would like to thank UDESC and LabP2D (http://labp2d.joinville.udesc.br) for providing the resources used to implement and analyse *Adaptive Remus*.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

[1] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, D. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga, *The NAS parallel benchmarks*, Int. J. Supercomput. Appl. 5(3) (1991), pp. 63–73.

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

[3] S.M. Blackburn, R. Garner, C. Hoffman, A.M. Khan, K.S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S.Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J.E.B. Moss, A. Phansalkar, D. Stefanovič, T. VanDrunen, D. Dincklagevon, and B. Wiedermann, *The DaCapo benchmarks: Java benchmarking development and analysis*, in *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-Oriented Programing, Systems, Languages, and Applications*, ACM Press, New York, NY, USA, 2006, pp. 169–190.

[4] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg, *The primary-backup approach*, in *Distributed Systems*, 2nd ed., edited by S. Mullender, ACM Press/Addison-Wesley Publishing Co., New York, NY, 1993. pp. 199–216.

[5] M. Chtepen, B. Dhoedt, F. De Turck, P. Demeester, F. Claeys, and P. Vanrolleghem, *Adaptive checkpointing in dynamic grids for uncertain job durations*, in *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, 2009. ITI '09*, Dubrovnik, June 2009, pp. 585–590.

[6] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. JulC. LimpachI. Pratt, and A. Warfield, *Live migration of virtual machines*, in *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation, NSDI'05* Vol. 2, USENIX Association, Berkeley, CA, USA, 2005., pp. 273–286.

[7] W. Cui, D. Ma, T. Wo, and Q. Li, *Enhancing reliability for virtual machines via continual migration*, in *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems, ICPADS '09*, IEEE Computer Society, Washington, DC, USA, 2009, pp. 937–942.

[8] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, *Remus: High availability via asynchronous virtual machine replication*, in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08*, USENIX Association, Berkeley, CA, USA, 2008, pp. 161–174.

[9] Y. Dong, W. Ye, Y. Jiang, I. Pratt, S. MaJ. Li, and H. Guan, *Colo: Coarse-grained lock-stepping virtual machines for non-stop service*, in *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13*, ACM, New York, NY, USA, 2013, pp. 3:1–3:16.

[10] Y. Du, X. Shi, H. Jin, and S. Wu, *FITDOC: Fast virtual machines checkpointing with delta memory compression*, in *IEEE 17th International Conference on Computational Science and Engineering (CSE), 2014*, Chengdu, China, December 2014, pp. 291–298.

[11] B. Eckart, X. He, C. Wu, F. Aderholdt, F. Han, and S. Scott, *Distributed virtual diskless checkpointing: A highly fault tolerant scheme for virtualized clusters*, in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, Shanghai, China, May 2012, pp. 1120–1127.

[12] I.P. Egwutuoha, D. Levy, B. Selic, and S. Chen, *A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems*, J. Supercomput. 65(3) (2013), pp. 1302–1326.
[13] E.N.M. Elnozahy, L. Alvisi, Y.-M. Wang, and D.B. Johnson, *A survey of rollback-recovery protocols in message-passing systems*, ACM Comput. Surv. 34(3) (2002), pp. 375–408.
[14] B. Gerofi and Y. Ishikawa, *Workload adaptive checkpoint scheduling of virtual machine replication*, in *IEEE 17th Pacific Rim International Symposium on Dependable Computing (PRDC)*, Pasadena, CA, December 2011, pp. 204–213.
[15] R.P. Goldberg, *Architecture of virtual machines*, in *Proceedings of the Workshop on Virtual Computer Systems*, ACM, New York, NY, USA, 1973, pp. 74–112.
[16] R. Guerraoui and A. Schiper, *Software-based replication for fault tolerance*, Computer 30(4) (1997), pp. 68–74.
[17] W. Hu, A. Hicks, L. Zhang, E.M. Dow, V. SoniH. JiangR. Bull and J.N. Matthews, *A quantitative study of virtual machine live migration*, in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, CAC '13*, ACM, New York, NY, USA, 2013, pp. 11:1–11:10.
[18] V. Inc, *VMware vSphere Fault Tolerance (FT)*, 2012. Available at http://www.vmware.com/products/vsphere/features/fault-tolerance.
[19] V. Inc, *VMware vSphere High Availability (FT)*, 2012. Available at http://www.vmware.com/products/vsphere/features/high-availability.html.
[20] P. Koppol, K. Namjoshi, T. Stathopoulos, and G. Wilfong, *The inherent difficulty of timely primary-backup replication*, in *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '11*, ACM, New York, NY, USA, 2011, pp. 349–350.
[21] G. Koslovski, W.L. Yeow, C. Westphal, T. Huu, J. Montagnat, and P. VicatBlanc, *Reliability support in virtual infrastructures*, in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, IN, November 2010, pp. 49–58.
[22] Y. Liang, Y. Zhang, and M. Jette, *Bluegene/l failure analysis and prediction models*, in *International Conference on Dependable Systems and Networks (DSN)*, Philadelphia, PA, June 2006, pp. 425–434.
[23] E.R. Marathon and C.X. Server, *Marathon everRun VM: Worlds First Fault-Tolerant, High Availability Software for Server Virtualization*, 2007. Available at http://raxco.presite.be/downloads/everrun-vm-datasheet_1231851832.pdf.
[24] J. McCarthy, *The 10 Biggest Cloud Outages of 2012, CRN*, 2012. Available at http://goo.gl/0civLK.
[25] D. Petrovic and A. Schiper, *Implementing virtual machine replication: A case study using Xen and KVM*, in *Proceedings of the IEEE 26th International Conference on Advanced Information Networking and Applications, AINA '12*, IEEE Computer Society, Washington, DC, USA, 2012, pp. 73–80.
[26] J.S. Plank, M. Beck, G. Kingsley, and K. Li, *Libckpt: Transparent checkpointing under unix*, in *Proceedings of the USENIX 1995 Technical Conference, TCON'95*, USENIX Association, Berkeley, CA, USA, 1995, pp. 18–18.
[27] J. Plank and K. Li, *Ickp: A consistent checkpointer for multicomputers*, IEEE Parallel Distrib. Technol.: Syst. Appl. 2(2) (1994), pp. 62–67.
[28] Ponemon Institute, *Understanding the cost of data center downtime: An analysis of the financial impact on infrastructure vulnerability*, Tech. Rep., 2011. Available at http://goo.gl/2jM3c.
[29] Ponemon Institute, *2013 Study on Data Center Outages*, Tech. Rep, Emerson Network Power, 2013. Available at http://goo.gl/v5DBJO.
[30] S. Rajagopalan, B. Cully, R. O'Connor, and A. Warfield, *SecondSite: Disaster tolerance as a service*, SIGPLAN Not. 47(7) (2012), pp. 97–108.
[31] J.R. Raphael, *The Worst Cloud Outages of 2013 (So Far), Infoworld*, 2013. Available at http://goo.gl/6Q7oK7.
[32] P. Reisner and L. Ellenberg, *Replicated storage with shared disk semantics*, in *Proceedings of the 12th International Linux System Technology Conference (Linux-Kongress)*, Hamburg, Germany, 2005.
[33] D.J. Scales, M. Nelson, and G. Venkitachalam, *The design of a practical system for fault-tolerant virtual machines*, SIGOPS Oper. Syst. Rev. 44(4) (2010), pp. 30–39.
[34] W.R. Stevens, Tcp/ip Illustrated: The Protocols Vol. 1, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, 1993.
[35] J. Subhlok, S. Venkataramaiah, and A. Singh, *Characterizing NAS benchmark performance on shared heterogeneous networks*, in *Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02*, IEEE, Washington, DC, USA, 2002, p. 91. Available at http://dl.acm.org/citation.cfm?id=645610.661710.
[36] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, *Kemari: VM Synchronization for Fault Tolerance*, in *Proceedings USENIX Annual Technical Conference (Poster Session)*. Boston, MA, 2008.
[37] T. Wood, K.K. Ramakrishnan, P. Shenoy, and J. Merwevan der, *CloudNet: Dynamic pooling of cloud resources by live wan migration of virtual machines*, in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '11*, ACM, New York, NY, USA, 2011, pp. 121–132.
[38] Y. Zhang and K. Chakrabarty, *Energy-aware adaptive checkpointing in embedded real-time systems*, Design, Automation and Test in Europe Conference and Exhibition, Muenchen, Germany, 2003 (2003), pp. 918–923.
[39] J. Zhu, W. Dong, Z. Jiang, X. Shi, Z. Xiao, and X. Li, *Improving the performance of hypervisor-based fault tolerance*, in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium*, Atlanta, GA, (2010), pp. 1–10. doi:10.1109/IPDPS.2010.5470357.
[40] A. Ziv and J. Bruck, *An on-line algorithm for checkpoint placement*, IEEE Trans. Comput. 46(9) (1997), pp. 976–985.