An Availability Model for DSS and OLTP Applications in Virtualized Environments

Matheus Torquato*†, Charles F. Gonçalves*‡, Marco Vieira*
*University of Coimbra, CISUC, DEI - Coimbra, Portugal
†Federal Institute of Alagoas (IFAL), Campus Arapiraca - Arapiraca, Brazil
‡Information Governance Secretariat, CEFET-MG, Brazil
matheus.torquato@ifal.edu.br†, charles@cefetmg.br‡, {mdmelo, charles, mvieira}@dei.uc.pt*

***Abstract*—Decision support systems (DSS) and online transaction processing applications (OLTP) are crucial for several organizations and frequently require high levels of availability. Many organizations moved their systems to the virtualized environment aiming at improving system availability. Despite the flexibility and manageability features provided by virtualization, a question arises on what policies to apply in order to achieve high availability. Usual approaches highlight redundancy as a strategy for high availability. Still, a concern persists on what components we should consider for redundancy. This paper proposes a hierarchical availability model for evaluating different redundancy allocations for DSS and OLTP systems in virtualized environments. We present three case studies investigating only-VM redundancy and physical machine redundancy strategies. The results provide an overview of the availability impact due to each strategy. We noticed that the physical machine failure rate limits the maximum availability obtained from only-VM redundancy. We exercise our model with a genetic algorithm to find alternatives for high availability. The presented models and results may bring insights when designing availability policies.**

***Index Terms*—Availability, Hierarchical Models, Cloud Computing, OLTP, Decision Support Systems**

I. Introduction

Decision support systems (DSS) and Online Transaction Processing (OLTP) are crucial to managing data for decision making in environments such as online banking and e-commerce applications. The main similarity is that both DSS and OLTP rely on databases as the critical component to support functionalities [1]. With the increasing demands of corporate information systems, both DSS and OLTP systems must provide high availability [2].

Depending on the business model, a single organization may use both DSS and OLTP applications (henceforth, DSS-OLTP systems). An example is a brokerage firm that has to deal with the market and its customers, like what is considered in the TPCx-V benchmark[1] [3]. Moreover, many companies are moving their applications to virtualized environments [4]. That raises the problem of how to assure high availability levels in DSS-OLTP hosted in virtualized environments.

A common approach for achieving high availability is through component redundancy [5] [2]. However, an important question is what components should have replicas. Another question is, what is the availability impact of component redundancy. This problem is known as *Redundancy Allocation Problem* (RAP) [6].

Many works in the literature deal with RAP. Most of them try to explore optimization and artificial intelligence techniques to find the best RAP solutions [7] [8]. We can also highlight our previous work [9], which proposes an approach of RAP solving using models and genetic algorithms.

The current literature already provides a comprehensive cloud availability modeling framework. However, a key problem with much of the literature is the lack of application-aware models. In an attempt to contribute to the cloud availability modeling research, we developed a hierarchical model considering the specific aspects of DSS-OLTP systems (e.g., heterogeneous workload submitted to a virtualized environment with multiple VM pools).

This paper proposes a hierarchical model for solving RAP for DSS-OLTP systems hosted in virtualized servers. The proposed model is built using Reliability Block Diagrams (RBD) and Stochastic Reward Nets (SRN) [10]. We consider two redundancy approaches: i) only-VM redundancy - VM replicas on a single physical machine; ii) physical machine redundancy aiming at *high availability* (i.e., expected annual downtime of five minutes) [11]. In summary, we aim to answer the following research questions:

- **RQ$_1$** What are the DSS-OLTP systems availability levels when using only-VM redundancy?
- **RQ$_2$** What are the alternatives of physical machine redundancy to achieve *high availability* for DSS-OLTP systems?

For this study, we have considered the system architecture proposed in the TPCx-V benchmark (more details in Section II). Since the TPC benchmarks usually consider widely adopted systems, our hierarchical model may be relevant for many system managers.

We divided our results into three case studies to answer these questions. The obtained results showed that the underlying physical machine limits the only-VM redundancy availability. Even using triple redundancy in all VMs, the expected annual downtime is above 9 hours. Physical machine redundancy achieved a better result. A simple redundancy of $2N$ (physical machine and one replica) produces *high*

[1]http://www.tpc.org/tpcx-v/

availability levels (regardless of the applied VM redundancy). We also present a case study with the comparison of the approaches, i.e., only-VM redundancy and physical machine redundancy. Our results also cover the expected percentage of loss transactions in each scenario.

The main contributions of this paper are:

- The adopted hierarchical approach favors further model improvement or modification;
- An application-aware availability model for DSS-OLTP systems in virtualized environments;
- Detailed case studies exploring redundancy alternatives as only-VM redundancy and physical machine redundancy.

To the best of our knowledge, this is the first attempt to evaluate the availability of a virtualized system with DSS and OLTP applications. The results presented help us to depict a general behavior of the availability impact due to different redundancy allocation approaches. Our case studies and proposed approach may help system managers in the decision making or availability policies design.

The paper is organized as follows. Section II presents the system architecture considered and its characteristics. Section III overviews the availability models. Section IV presents our results and analysis for the three case studies. Section V discusses related work. Conclusions and future works are in Section VI.

II. System architecture

Figure 1 displays the system architecture considered. The architecture consists of a single physical machine (*Main Node*), which hosts three Virtual Machine (VM) pools. Besides the VMs behavior, our availability model also covers some *Main Node* internal components. These components are: i) Virtual Machine Monitor (VMM) - middleware between VMs and Operating System (OS); ii) OS - in charge of communication of hardware and software; and iii) Hardware components (HW) - we consider a specific set of hardware components, as highlighted in Section III.

The VMs in the *VM2* and *VM3* pools are responsible for database processing. Therefore, they all have attached virtual disks (VD). In this paper, we consider that all attached virtual disks are in a consistent state. Thus, the system is capable of maintaining a consistent state among all the VDs in the same pool.

This architecture is a simplification of the TPCx-V benchmark design [3]. TPCx-V is used to assess the performance of virtualized environments that support database applications. The workflow is as follows. VMs from *VM1* pool receive the user's requests, which can be: i) DSS queries or ii) OLTP transactions. After initial processing, the VMs from *VM1* pool forward the requests to the *VM2* or *VM3* pools accordingly to their type: DSS queries are sent to VMs in *VM2* pool, while OLTP transactions are sent to VMs in *VM3* pool.

Differently from the TPCx-V architecture, we do not have many brokerage firms, where each one is represented by 3 VMs (1,2,3). Mapping to the TPCx-V concept, we can describe our architecture as a single scalable, brokerage firm.

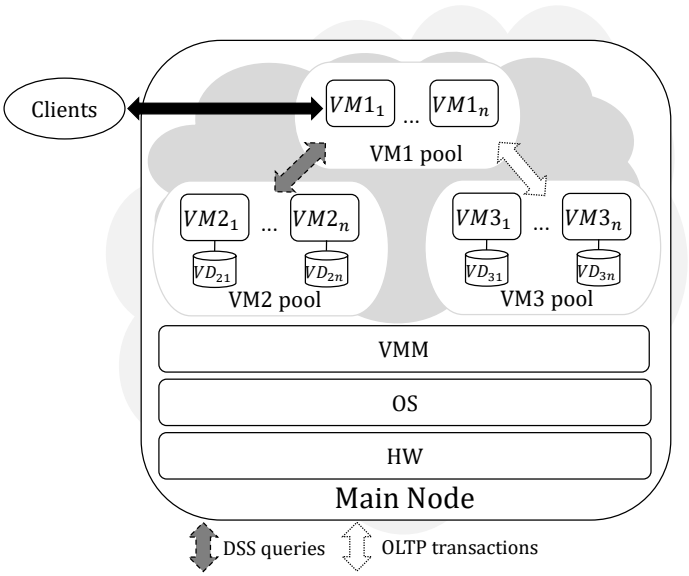

Fig. 1. System architecture

Table I presents the workload distribution based on the TPCx-V specification. As expected, we have more requests related to OLTP applications that usually receive heavier traffic. TPCx-V benchmark has eleven different transactions handled between market/client/brokers, which characterizes the workload specification. We simplified the details in our model since it only matters the generic types of transactions (DSS and OLTP).

TABLE I
Workload Distribution

Workload type	Target	Proportion
DSS	*VM2* pool	10%
OLTP	*VM3* pool	90%

As we have a heterogeneous workload, we propose two metrics for availability calculation: $Availability_{\mathrm{VM2}}$ - system availability to process DSS queries (i.e., requests for *VM2* pool), and $Availability_{\mathrm{VM3}}$ - system availability to handle OLTP transactions (i.e., requests for *VM3* pool).

The system is available for responding DSS queries ($Availability_{\mathrm{VM2}}$) when we have at least one VM running in *VM1* pool and one VM running in the *VM2* pool. Similarly, the system is available for OLTP transactions ($Availability_{\mathrm{VM3}}$) when we have at least one VM running in *VM1* pool and one VM running in the *VM3*. VMs depend on the *Main Node* to run. Therefore, the system is available only when the *Main Node* is running. Thus, the system becomes unavailable after a *Main Node* failure. The *Main Node* repair comprises the restart of all VM pools.

VMs in the *VM1* pool are stateless. Thus, instead of repairing them after a failure, the recovery consists of a new VM instantiating. As VMs in *VM2* and *VM3* pools are dealing with database processing, we should repair them after a failure. Note that, some relevant components for availability evaluation

(e.g., uninterruptible power source (UPS) and network devices) are outside of the scope of our work. We decided to neglect such components to turn our models more tractable. However, they are subject to our future works.

III. Availability Models

The proposed availability model has three submodels: **M1**, **M2**, and **M3**. The models' hierarchy (see Figure 2) is as follows: **M1** offers inputs for **M2**; **M2** offers inputs for **M3**; **M3** provides the availability results.

M1 is a RBD model for the hardware components of the *Main Node*. **M1** offers the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) of the hardware components as inputs for **M2**. **M2** is also a RBD model and provides the MTTF and MTTR of the *Main Node*. However, besides the hardware components, **M2** considers also the Operating System (OS) and the Virtual Machine Monitor (VMM) components. **M2** ignores the VMs hosted in the *Main Node*. **M3** is a SRN model of the *Main Node* considering its hosted VMs. We use SRN in this modeling to capture the dependency between VMs and the *Main Node*. **M3** model provides the availability results.

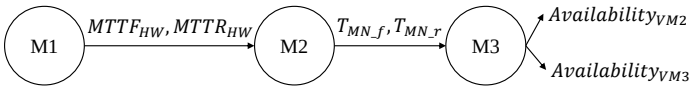

Fig. 2. Models' hierarchy

Figure 3 presents the **M1** model. **M1** considers the following *Main Node* hardware components: CPU (`CPU` block), memory (`MEM` block), cooler (`COOL` block), network interface card (`NIC` block) and power module (`PW` block). This is the same set of hardware components presented in [12] [9]. As in those previous studies, we also decided to simplify the modeling using an RBD model. RBD models assume that the failure and repairs of the components (i.e., blocks) are independent. Besides that, all the blocks are connected in series. This way, the operational mode of the *Main Node* depends on each one of these components. For the sake of simplicity, we prefer to omit the *Main Node* hard disk and consider only the VMs' virtual disks. Since the mean time to failure of hard disks is usually very high (1000000 hours) [13], the impact of such a component is negligible in our scenario.

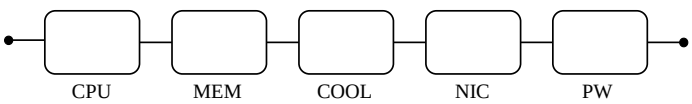

Fig. 3. RBD model for hardware components of Main Node (Model **M1**)

Figure 4 displays the **M2** model, which has three blocks: i) `HW`, a hierarchical block (highlighted in gray) that receives the hardware MTTF and MTTR from **M1**; ii) `OS` - representing the Operating System; and iii) `VMM` - representing the virtual machine monitor. All the blocks are arranged in series. Figure 5 shows the SRN model used for the availability calculation. Initially, we have a token in the place `MN_UP` representing that the *Main Node* is running. The transition `MN_f` represents the time delay associated with a *Main Node* failure. The firing of transition `MN_f` removes a token from place `MN_UP` and puts another on `MN_DW`. The place `MN_DW` with tokens represents that the *Main Node* is non-operational. As all the VMs depend on the *Main Node* to run, all the tokens related to them are removed as a token arrives in the place `MN_DW`. The immediate transitions `MNfail(1,2,3,4,5)` have an embedded guard function[2] to perform this removal (see Table II).

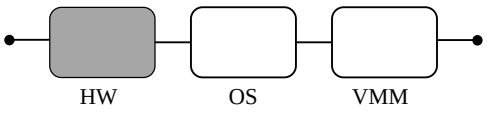

Fig. 4. RBD model for hardware and software components of Main Node (Model **M2**). Hierarchical composition block is highlighted in gray.

Transition `MN_r` represents the time delay associated with the *Main Node* repair. The firing of `MN_r` moves the token from place `MN_DW` to place `MN_UP`. As explained before, the *Main Node* repair also comprises the restart of the VMs pools. Therefore, the transition `MN_r` firing returns the original number of tokens: `NVM1`, `NVM2`, and `NVM3`, to the correspondent places: `VMs1_UP`, `VMs2_UP`, and `VMs3_UP`, respectively.

TABLE II
Guard functions

Guard	Enabling function
Gf	$(\#MN\_DW > 0)$

The bottom part of the model represents the failure and repair behaviors of the VMs pools. The number of tokens in the place `VMs1_UP` represents the number of VMs running in the *VM1* pool. The same idea applies to the place `VMs2_UP` and `VMs3_UP` for the *VM2* and *VM3* pools, respectively.

The transitions `VM1_f`, `VM2_f` and `VM3_f` represent the time delay for a VM failure in the *VM1*, *VM2* and *VM3* pools, respectively. Thus, the firing of `VM1_f` moves tokens from place `VMs1_UP` to place `VMs1_DW`; `VM2_f` firing removes tokens from place `VMs2_UP` and puts tokens in the place `VMs2_DW`; and `VM1_f` firing collects tokens from place `VMs3_UP` and deposits tokens in the place `VMs3_DW`. These transitions apply the infinite-server semantics [14]. In this context, infinite server semantics mean that the time for VMs failure in each pool is counted concurrently.

After a failure of a virtual machine in the *VM1* pool, we rather delete the failed VM and create a new one instead of repairing the failed one (transition `VM1_c`). However, in the case of a VM failure in *VM2* or *VM3* pools, we should apply the proper repair (transitions `VM2_r` and `VM3_r`), because they have attached virtual disks with meaningful state.

[2]Guard functions are Boolean expressions evaluated based on the net current marking. They disable the associated transition when the boolean expression returns *false*

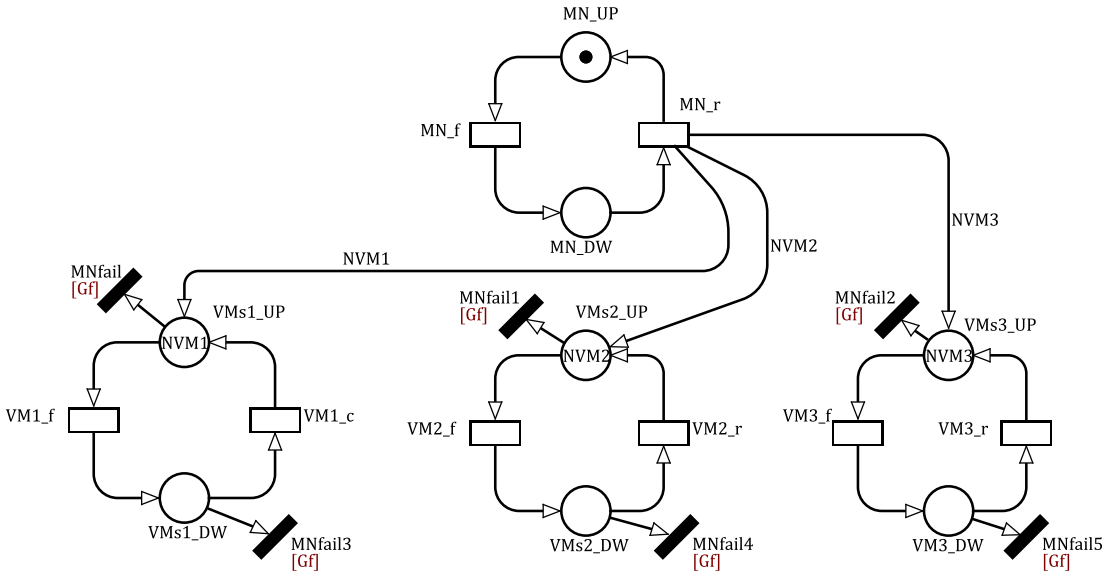

Fig. 5. SRN model of system including the Virtual Machines (Model **M3**)

Metrics computation: Consider $P\{(\#L > 0)\}$ as the steady-state probability of tokens existence in place L. Then, we compute $Availabity_{\text{VM2}}$ using the following expression: $P\{(\#VMs1\_UP > 0)AND(\#VMs2\_UP > 0)\}$. Similarly, $Availabity_{\text{VM3}} = P\{(\#VMs1\_UP > 0)AND(\#VMs3\_UP > 0)\}$.

IV. Case studies

We conducted the RBD models evaluation using the SHARPE tool [15], and the SRN models evaluation using the TimeNET tool [16]. Table III presents the values of the input parameters. All the values were retrieved from previous papers [12] [5].

The following three subsections will present: *(A)* the steady-state availability results for different scenarios of only-VM redundancy; *(B)* the *Main Node* redundancy alternatives for achieving high availability; and *(C)* comparison of *(A)* and *(B)*.

A. Case study #1 (CS1) - VM redundancy

Applying VM redundancy may be the most straightforward approach to improve system availability considering our system architecture. As an illustrative example, this case study aims to evaluate the availability improvement in different scenarios of VM redundancy allocation. We propose 27 scenarios, varying the number of VMs in each pool from 1 to 3 (scenarios definition on Table IV).

1) CS1 - Availability: Figure 6 displays the availability results for each scenario. The black bar refers to the $Availability_{\text{VM2}}$ results, and the gray bar corresponds to the $Availability_{\text{VM3}}$ results. Figure 7 shows the percentage of loss of each transaction type in the considered scenarios.

As expected, $Availability_{\text{VM2}}$ and $Availability_{\text{VM3}}$ assume the same value when adopting the same redundancy allocation. Thus, we can summarize the availability results as in Table V. We also added a column with the expected annual system downtime in hours per year.

Table V shows that the VM redundancy allocation does not affect availability when the number of VMs in any pool is higher than 2. These results suggest the availability limit

TABLE III
Parameters used in the models

Parameter	Description	Value
$MTTF_{CPU}$	Mean time for CPU failure	2500000 hours
$MTTR_{CPU}$	Mean time to repair CPU	30 minutes
$MTTF_{MEM}$	Mean time for memory failure	480000 hours
$MTTR_{MEM}$	Mean time to repair memory	30 minutes
$MTTF_{COOL}$	Mean time for cooler failure	3100000 hours
$MTTR_{COOL}$	Mean time to repair cooler	30 minutes
$MTTF_{NIC}$	Mean time for NIC failure	120000 hours
$MTTR_{NIC}$	Mean time to repair NIC	30 minutes
$MTTF_{PW}$	Mean time for power failure	670000 hours
$MTTR_{PW}$	Mean time to repair power module	30 minutes
$MTTF_{OS}$	Mean time for OS failure	1440 hours
$MTTR_{OS}$	Mean time to repair OS	1 hour
$MTTF_{VMM}$	Mean time for VMM failure	2880 hours
$MTTR_{VMM}$	Mean time to repair VMM	1 hour
T_{VM1_f}, T_{VM2_f}, T_{VM3_f}	Mean time for a VM failure	2880 hours
T_{VM2_r}, T_{VM3_r}	Mean time to repair a VM	30 minutes
T_{VM1_c}	Mean time to create a new VM (*VM1* repair)	5 minutes
$NVM1$, $NVM2$, $NVM3$	Number of VMs in *VM1*, *VM2* and *VM3* pools, respectively	Depends on the scenario

TABLE IV
SCENARIOS DEFINITION

Scn	NVM1	NVM2	NVM3
0	1	1	1
1	1	1	2
2	1	1	3
3	1	2	1
4	1	2	2
5	1	2	3
6	1	3	1
7	1	3	2
8	1	3	3
9	2	1	1
a	2	1	2
b	2	1	3
c	2	2	1
d	2	2	2
e	2	2	3
f	2	3	1
g	2	3	2
h	2	3	3
i	3	1	1
j	3	1	2
k	3	1	3
m	3	2	1
n	3	2	2
p	3	2	3
q	3	3	1
r	3	3	2
u	3	3	3

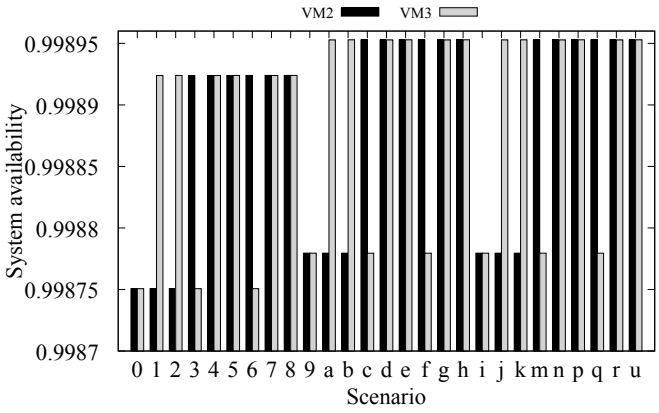

Fig. 6. Availability results

TABLE V
SUMMARY OF AVAILABILITY RESULTS. (X CAN BE 2 OR 3)

NVM1	NVMX	Availability (VMX)	Downtime (hr/yr)
1	1	0.998751	10.94
1	2	0.998924	9.43
1	3	0.998924	9.43
2	1	0.998780	10.69
2	2	0.998953	9.17
2	3	0.998953	9.17
3	1	0.998780	10.69
3	2	0.998953	9.17
3	3	0.998953	9.17

for VM redundancy. We tested scenarios with more VMs and obtained the same results.

Initially, we expected the system availability to improve when VM redundancy is increased. However, our results contradict those initial expectations, suggesting that we need to analyze our results with more caution. Note that, the system availability depends on the *Main Node* availability. Therefore, the system failure rate is always determined by the *Main Node* failure rate (i.e., transition `MN_f`).

We verified the availability of *Main Node* without VMs running ($Availability_{\mathrm{MN}}$). The result is $Availability_{\mathrm{MN}} = 0.998953$, the availability limit observed in Table V, thus confirming our claim that the *Main Node* is limiting the overall system availability.

Despite the availability limit reported above, it is important to develop approaches to improve system availability further. As shown in Table V, the best availability scenario reaches a downtime of more than 9 hours per year. These results are far from the desired *high availability* levels with about 5 minutes of downtime per year [11]. In Section IV-B, we exercise our models to find alternatives to reach *high availability* levels.

2) CS1 - Percentage of loss transactions: Figure 7 shows the steady-state percentage of loss transactions for each scenario. These results are based on the expected amount of each request type, as presented in Section II, and in the system availability.

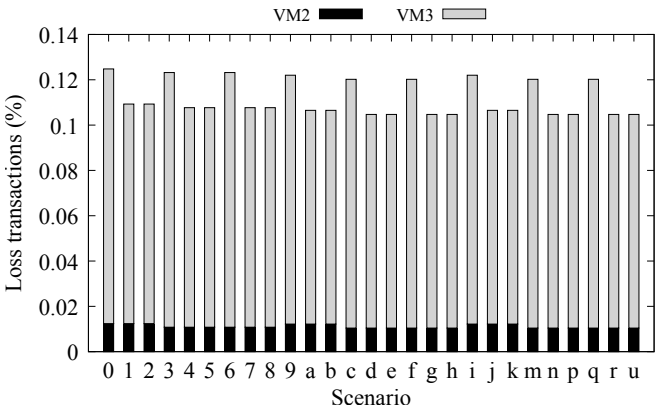

Fig. 7. Percentage of Loss transactions

The average loss transactions is of 0.1115%. As expected, the percentage of loss transactions varies accordingly to the VM redundancy allocation strategy. For example, in scenarios with fewer VMs in the *VM2* and *VM3* pools than in *VM1* pool, we noticed higher percentage of loss transactions. Besides that, as more transactions are targeted for VMs in the *VM3* pool, the *VM3* redundancy provides better results than redundancy on other pools. Table VI presents a summary of the percentage of loss transactions (LT%) results. The columns **LT% *VM2*** and **LT% *VM3*** refer to the percentage of loss transactions for *VM2* and *VM3* pools, respectively. The column **LT% sys** provides the total percentage of loss transactions.

TABLE VI
SUMMARY OF LOSS TRANSACTIONS (%) RESULTS

NVM1	NVM2	NVM3	LT% *VM2*	LT% *VM3*	LT% sys
1	1	1	0.0124	0.1124	0.1248
1	1	2 or 3	0.0124	0.0969	0.1093
1	2 or 3	1	0.0108	0.1124	0.1232
1	2 or 3	2 or 3	0.0108	0.0969	0.1077
2 or 3	1	1	0.0122	0.1098	0.1220
2 or 3	1	2 or 3	0.0122	0.0943	0.1065
2 or 3	2 or 3	1	0.0104	0.1098	0.1202
2 or 3	2 or 3	2 or 3	0.0104	0.0943	0.1047

OLTP downtime may harm company reputation. Moreover, system unavailability for DSS queries may jeopardize timely decision making. Therefore, the system managers should decide upon the alternatives which one suits their needs.

B. Case study #2 (CS2) - High availability through Main Node redundancy

As reported in the previous case study, the use of an only-VM redundancy policy is not enough to achieve *high availability*. There are some approaches to deal with this problem. The first is to apply fault tolerance techniques in the *Main Node*. The use of redundant hardware components or software fault tolerance techniques may improve the overall system availability, as both techniques may increase the *Main Node* MTTF.

The second alternative is to replicate the *Main Node*. Thus, the system may run in an active-active redundancy scheme [17]. Active-active redundancy allows both the system and its replicas to run the same service for the users. The main advantages of the active-active redundancy are the availability and performance improvement. For this case study, we decided to verify only the availability impact of active-active redundancy schemes. However, we highlight that active-active redundancy may have relevant power consumption and cost impacts. These impacts may be taken into account in the redundancy policy design.

The main goal of this case study is to verify active-active redundancy policies on *Main Node*, aiming at achieving *high availability*. To solve the RAP, we used a genetic algorithm (RENATA)[3] [9] to perform a search in the solution space. We used our model as input for RENATA, and selected the availability goal of 0.99999.

RENATA output suggests that a redundancy allocation of $2N$ (i.e., just a *Main Node* and one replica) is enough to reach *high availability* in all scenarios. We present the detailed results next.

1) CS2 - Availability: Figure 8 presents the availability results when applying $2N$ *Main Node* for the scenarios mentioned earlier. We also summarized the availability results in Table VII. However, this time we also include a column for the annual downtime in seconds.

These results show that the *Main Node*'s $2N$ redundancy leads to a significantly higher availability than the previous

[3]RENATA is open-source, and its latest version is available at the link: https://github.com/matheustor4/RENATA/.

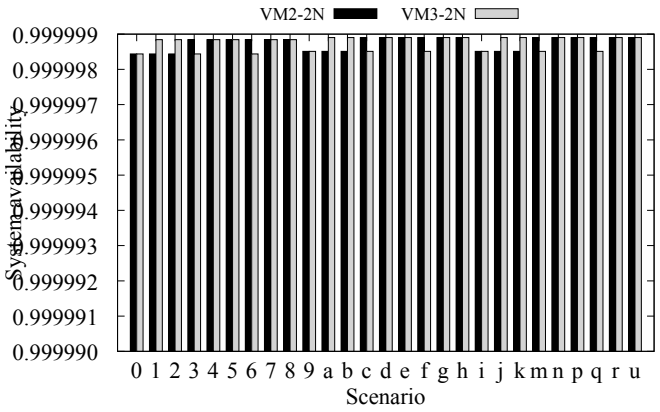

Fig. 8. Availability results - *Main Node* $2N$ redundancy

TABLE VII
SUMMARY OF AVAILABILITY RESULTS - *Main Node* $2N$ REDUNDANCY (Y CAN BE 2 OR 3)

NVM1	NVMY	Availability (VMY)	Downtime (hr/yr)	Downtime (sec/yr)
1	1	0.9999984391	0.013673	49.22
1	2	0.9999988420	0.010144	36.52
1	3	0.9999988421	0.010143	36.51
2	1	0.9999985105	0.013048	46.97
2	2	0.9999989033	0.009606	34.58
2	3	0.9999989033	0.009606	34.58
3	1	0.9999985105	0.013048	46.97
3	2	0.9999989035	0.009606	34.58
3	3	0.9999989035	0.009606	34.58

case study (Section IV-A1). In every considered scenario, the expected downtime is less than one minute per year. Still, it presents a pattern similar to the previous case study where availability results are the same when the number of VMs in each pool is equal or greater than 2.

2) CS2 - Percentage of loss transactions: Figure 9 presents the percentage of loss transactions when applying *Main Node* $2N$ redundancy. Note that the plot in Figure 9 uses a scale different from the plot in Figure 7. In the scenario with *Main Node* $2N$ redundancy, the percentage of loss transactions is low. In the worst case, without VM redundancy on the physical machines, the expected percentage of loss transactions is only 0.0001561%. Therefore, for the sake of brevity, we omit the table with a summary of the results in this case.

Due to the *high availability*, the expected loss transaction reached negligible levels. However, it is essential to pinpoint some relevant conclusions from this case study. Firstly, note that this availability modeling focuses only on specific components of the environment. Thus, there are other components as network devices and UPS that may also affect system availability. These components are neglected in this study. Nevertheless, these case studies reveal that, in some cases, VM redundancy is not enough to reach *high availability*. Besides that, they also try to quantify the availability improvement in terms of system downtime.

Secondly, we highlight that the MTTR of the *Main Node*

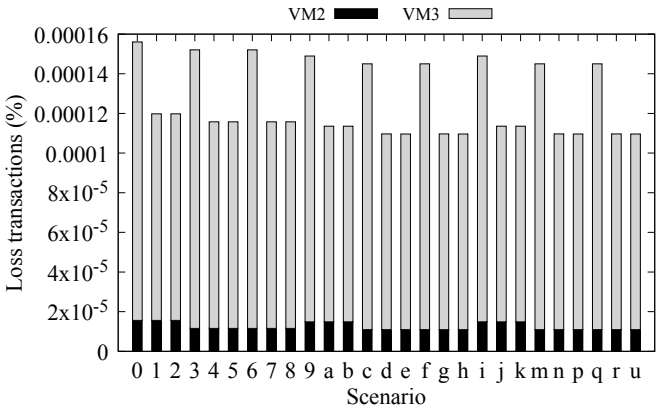

Fig. 9. Percentage of Loss transactions - *Main Node* $2N$ redundancy

is about one hour. Besides that, as the *Main Node* MTTF is of about 950 hours, the probability of simultaneous failure of the *Main Node* and its replica is low. However, if such an event occurs, the system will experience about one hour of downtime (i.e., time to repair a failed *Main Node*).

C. Case study #3 (CS3) - Comparison of CS1 and CS2

This last case study provides a comparison of the previous case studies. Its results comprise the reduction in annual downtime and percentage of loss transactions. This comparison tries to quantify the improvement due to *Main Node* redundancy.

Figure 10 presents the downtime reduction results. For any scenario, the downtime reduction surpasses 9 hours per year. Specifically, in the scenario with only one VM in each pool, the *Main Node* $2N$ redundancy approach results in an annual downtime reduction of 10 hours and 56 minutes. In the scenarios with three VMs in each pool, the downtime reduction due to *Main Node* redundancy is of 9 hours and 10 minutes. Such an impact may produce significant profit improvement for the DSS-OLTP business.

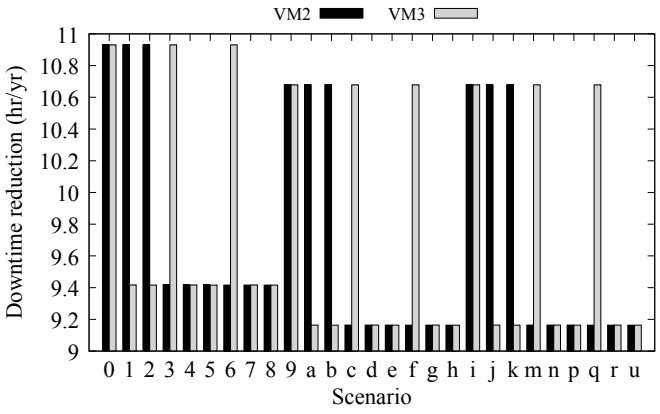

Fig. 10. Downtime reduction in hours per year

Table VIII presents the reduction of loss transactions provided by *Main Node* redundancy over the only-VM approach. A significant reduction of loss transactions can be observed in all scenarios. This result is due to the low percentage of loss transactions when there is redundancy at the *Main Node*.

TABLE VIII
SUMMARY OF REDUCTION OF LOSS TRANSACTIONS (%)

NVM1	NVM2	NVM3	Reduction LT% *VM2*	Reduction LT% *VM3*	Reduction LT% sys
1	1	1	0.0124	0.1123	0.1246
1	1	2 or 3	0.0124	0.0968	0.1092
1	2 or 3	1	0.0108	0.1123	0.1230
1	2 or 3	2 or 3	0.0108	0.0968	0.1076
2 or 3	1	1	0.0122	0.1097	0.1219
2 or 3	1	2 or 3	0.0122	0.0942	0.1064
2 or 3	2 or 3	1	0.0104	0.1097	0.1201
2 or 3	2 or 3	2 or 3	0.0104	0.0942	0.1046

V. RELATED WORK

This work is based on our previous papers [5] [9]. In these works, we explored only the generic aspects of cloud computing availability modeling. In the present paper, we try to extend our contributions considering a more specific business model as of TPCx-V. Therefore, we can observe the direct impact of redundancy allocation strategies in a DSS-OLTP system hosted in a virtualized environment.

Vieira and Madeira propose a dependability benchmark for OLTP applications [18] [19]. The authors provide the DBench-OLTP benchmark, which extends the TPC-C benchmark by adding dependability measures and faultload. Different from their approach, we focus on evaluating availability under different redundancy allocation policies (i.e., VM redundancy and *Main Node* redundancy). We also consider DSS systems.

Muppala et al [20] present a Stochastic Reward Net model for OLTP availability and performance evaluation. The considered system architecture also comprises components for receiving and forwarding requests; and components for database processing. Tang et al. [21] propose an availability model for an Application Server system based on Sun Java System Application Server (JSAS). Their insightful work comprises measurements, modeling, and statistical techniques in the availability evaluation. Unlike their works, we consider a heterogeneous system availability (i.e., $Availability_{\mathrm{VM2}}$ and $Availability_{\mathrm{VM3}}$) based on the VMs distribution in each pool. Besides that, different from the authors, we consider that the DSS-OLTP application is in a virtualized environment.

Cherkasova et al. [22] present a comprehensive modeling framework for OLTP applications. Their work uses Hierarchical Colored Petri Nets in the modeling. The architecture is based on the TPC-A benchmark. Similarly, our work is based on a TPC benchmark. However, in our case, we consider the TPCx-V benchmark. Finally, instead of focus in performance, our work focuses on availability evaluation.

Feng et al. [23] provide availability and performability evaluation approach for OLTP applications considering the Internet of Things environment. Their evaluation framework adopts a hierarchical composition using Markov models and Generalized Stochastic Petri Nets. Unlike their work, we

evaluate different scenarios of redundancy allocation for VMs and *Main Node*.

VI. Conclusion and Future Works

We presented a hierarchical model for availability evaluation of a DSS-OLTP infrastructure hosted in a virtualized environment. Our main goal was to evaluate availability when applying redundancy allocation policies on both VMs and their physical host. Finally, we also investigated approaches to reach *high availability* using a genetic algorithm to find the proper redundancy allocation solutions.

Our main conclusions are that the policy with only-VM redundancy faces an availability limit. The physical machine failure rate imposes an availability limit for the only-VM redundancy approach. This happens because, in the only-VM redundancy approach, all VMs depend on the physical machine to run. Besides that, there is a significant availability improvement when using only a $2N$ redundancy in the physical machine. As the DSS-OLTP system is usually crucial, the system managers may consider a physical machine redundancy instead of VM replication for availability improvement. We hope that system managers will find our approach useful for availability policies design.

The hierarchical approach eases the modification of specific parts of the proposed model. Therefore, it can be extended for other types of applications hosted in virtualized environments. As future work, we aim to expand the current model to also cover other relevant devices as network devices and UPS. This increment may impose significant changes in the current model. Large and complex models may turn their evaluation into a computationally prohibitive task. To avoid such problems, we aim to use the *interacting models* approach, as presented in [24].

We also intend to expand the current model to cover performability and security measures. The first idea is to use the RiskScore approach [25] to evaluate security risk. For the performability measures, we can use a hierarchical composition, including queueing models.

Acknowledgments

This work has been partially supported by Portuguese funding institution FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and Ph.D. grants SFRH/BD/146181/2019 and SFRH/BD/144839/2019. This work is also funded by European Social Fund, through the Regional Operational Program Centro 2020.

References

[1] S. Chaudhuri, U. Dayal, and V. Ganti, "Database technology for decision support systems," *Computer*, vol. 34, no. 12, pp. 48–55, 2001.

[2] M. del Pilar Angeles and V. G. Castro, "V+h: Hybrid architecture for dss and oltp," *International Journal of Information Management*, vol. 33, no. 6, pp. 940 – 947, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268401213000972

[3] A. Bond, D. Johnson, G. Kopczynski, and H. R. Taheri, "Profiling the performance of virtualized databases with the tpcx-v benchmark," in *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 2015, pp. 156–172.

[4] "User Stories - OpenStack Open Source Cloud Computing Software." [Online]. Available: https://www.openstack.org/user-stories/

[5] M. Torquato, E. Guedes, P. Maciel, and M. Vieira, "A hierarchical model for virtualized data center availability evaluation," in *2019 15th European Dependable Computing Conference (EDCC)*. IEEE, 2019, pp. 103–110.

[6] M.-S. Chern, "On the computational complexity of reliability redundancy allocation in a series system," *Operations research letters*, vol. 11, no. 5, pp. 309–315, 1992.

[7] W. Liao, E. Pan, and L. Xi, "A heuristics method based on ant colony optimisation for redundancy allocation problems," *International Journal of Computer Applications in Technology*, vol. 40, no. 1-2, pp. 71–78, 2011.

[8] Z. Ouyang, Y. Liu, S.-J. Ruan, and T. Jiang, "An improved particle swarm optimization algorithm for reliability-redundancy allocation problem with mixed redundancy strategy and heterogeneous components," *Reliability Engineering & System Safety*, vol. 181, pp. 62–74, 2019.

[9] M. Torquato, L. Torquato, P. Maciel, and M. Vieira, "Iaas cloud availability planning using models and genetic algorithms," in *2019 9th Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 2019, pp. 1–10.

[10] K. S. Trivedi and A. Bobbio, *Reliability and availability engineering: modeling, analysis, and applications*. Cambridge University Press, 2017.

[11] J. Gray and D. P. Siewiorek, "High-availability computer systems," *Computer*, vol. 24, no. 9, pp. 39–48, 1991.

[12] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *Dependable Computing, 2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on*. IEEE, 2009, pp. 365–371.

[13] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, no. 1, 2007, pp. 1–16.

[14] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*. Wiley New York, 1995, vol. 292.

[15] R. A. Sahner, K. Trivedi, and A. Puliafito, *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Springer Science & Business Media, 2012.

[16] A. Zimmermann, "Modelling and performance evaluation with timenet 4.4," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2017, pp. 300–303.

[17] R. Romera, J. E. Valdés, and R. I. Zequeira, "Active-redundancy allocation in systems," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 313–318, 2004.

[18] M. Vieira and H. Madeira, "A dependability benchmark for oltp application environments," in *Proceedings 2003 VLDB Conference*. Elsevier, 2003, pp. 742–753.

[19] ——, "Benchmarking the dependability of different oltp systems." in *DSN*, 2003, pp. 305–310.

[20] J. K. Muppala and K. S. Trivedi, "Composite performance and availability analysis using a hierarchy of stochastic reward nets," *Computer Performance Evaluation, Modelling Techniques and Tools*, pp. 335–350, 1991.

[21] D. Tang, D. Kumar, S. Duvur, and O. Torbjornsen, "Availability measurement and modeling for an application server," in *International Conference on Dependable Systems and Networks, 2004*. IEEE, 2004, pp. 669–678.

[22] L. Cherkasova, V. Kotov, and T. Rokicki, *On Net Modeling of OLTP for Parallel Systems*. Hewlett-Packard Laboratories, 1993.

[23] Y. Feng, Z. Zhang, H. Liu, D.-C. Zuo, and X.-Z. Yang, "An approach for evaluating availability and performability of data processing center in the internet of things environment," in *2010 First International Conference on Pervasive Computing, Signal Processing and Applications*. IEEE, 2010, pp. 1035–1038.

[24] M. Torquato and M. Vieira, "Interacting srn models for availability evaluation of vm migration as rejuvenation on a system under varying workload," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 300–307.

[25] M. Torquato, P. Maciel, and M. Vieira, "A model for availability and security risk evaluation for systems with vmm rejuvenation enabled by vm migration scheduling," *IEEE Access*, vol. 7, pp. 138 315–138 326, 2019.