uniserver

D6.5 – OpenStack Resilience on Extended Margins Micro-Servers

Contract number	688540
Project website	http://www.UniServer2020.eu
Contractual deadline	Project Month 36 (M36): 31st January 2019
Actual Delivery Date	1st February 2019
Dissemination level	Public
Report Version	1.0
Main Authors	Christian Pinto (IBM), Srikumar Venugopal (IBM), Christos Kalogirou (UTH), Panos Koutsovasilis (UTH)
Contributors	Christos Antonopoulos (UTH), Nikolaos Bellas (UTH), Spyros Lalis (UTH), Emmanouil Maroudas (UTH), Panagiotis Vlastaridis (UTH), Evagelia Malami, Lev Mukhanov (QUB)
Reviewers	Denis Guilhot (WSE), Christos Antonopoulos (UTH), Georgios Karakonstantis (QUB) Charles Gillan (QUB)
Keywords	Fault tolerance, extended margins, state machine, migration

Notice: The research leading to these results has received funding from the European Community's Horizon 2020 Programme for Research and Technical development under grant agreement no. 688540

Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. UniServer is a 42-month project that started on February 1st, 2016 and is funded by the European Commission. The partners in the project are:

The Queen's University of Belfast (QUB)

The University of Cyprus (UCY)

The University of Athens (UoA)

Applied Micro Circuits Corporation Deutschland Gmbh (APM)

ARM Holdings UK (ARM)

IBM Ireland Limited (IBM)

University of Thessaly (UTH)

WorldSensing (WSE)

Meritorious Audit Limited (MER)

Sparsity (SPA)

More information

Public UniServer reports and other information pertaining to the project are available through the UniServer public Web site under http://www.uniserver2020.eu.

Change Log

Version	Description of change
0.1	Outline of the deliverable
0.2	Background and motivation completed
0.3	Described injecting errors (section 5.1)
0.4	Described the Failure Lifecycle and Impl.
0.5	Populated Section 3.1 (Compute Failure)
0.6	Described the forecasting model
0.7	Described experiments and the workload
0.8	Review of the document (WSE, QUB)
0.9	Review of the document (UTH)
1.0	Final draft for submission

Table of Contents

INDEX OF FIGURES

INDEX OF TABLES

Executive Summary

This document describes the fault tolerance support for UniServer as developed in Task 6.3 within the Work Package 6 (WP6) of the UniServer Project Description of Action (DoA). This is in fulfillment of Deliverable D6.5, OpenStack Resilience on Extended Margins Micro-Servers.

The UniServer Project seeks to exploit the Voltage-Frequency-Refresh Rate (VFR) settings of processors and memory, outside of the pessimistic values imposed by the manufacturer, in order to improve overall energy efficiency and performance. One of the target deployments for UniServer is in cloud data centers where any efficiency gains aggregated over thousands of machines can vastly improve a provider's Total Cost of Ownership (TCO). However, applying extended margins also causes the probabilities of errors and failures to increase.

In this deliverable, we introduce a new proactive fault tolerance mechanism that uses predictions over system error data in order to predict possible server failure and then takes actions to conserve high value workloads from disruption. Our mechanism is decentralised and runs at the level of individual servers but is integrated with the centralised scheduling and resource management components introduced in previous deliverables. Our evaluations show that the proposed mechanism is able to accept more high-value VMs (16% higher than priority scheduling without fault tolerance) while still keeping 50% of the data center in the more energy efficient (extended margin) configuration.

1. Introduction

The UniServer project has the goal of exposing and exploiting extended margins of the underlying machine to improve its energy efficiency and performance. The extended margins refer to voltage and frequency states outside of the nominal guardbands for safe and correct operation imposed on processors and memory chips by manufacturers. While this reduces the processor power consumption, it may increase the probability of errors and failures. The UniServer project seeks to minimize the impact of these failures at all levels of the system stack on the one hand by avoiding them – through a thorough characterization of hardware and hardware / software interaction – and on the other hand by anticipating them and implementing mechanisms to avoid disruptions.

Work Package 6 (WP6) aims at providing enhancements and specialized resource management policies for OpenStack running on 64-bit ARM based micro-servers. WP6 devises techniques that improve the management of virtual machines (VMs) on hosts with heterogeneous power and performance settings. This heterogeneity introduces both advantages and tradeoffs that should be incorporated in managing the running VMs. In a previous deliverable, D6.2, this work package introduced a new scheduling algorithm that assigns high-priority, revenue generating VMs to the most reliable server machines running within guard bands while allocating other VMs to servers running in extended margins to save on overall power consumption. Subsequently, in deliverable D6.3, we discussed implementation of this algorithm in the OpenStack Resource Manager to be deployed in a real datacenter.

In this deliverable, we continue this focus on server operation in datacenters. Specifically, our concern here is to improve the resilience of the VMs to possible errors and faults, however marginal, caused by the use of UniServer extensions to server hardware. Our response has been to design a fault tolerance mechanism that proactively detects possible server failures by predicting over the system data obtained from the APIs provided by the hypervisor, and takes action to conserve high-value VMs running on it. To improve the speed of response, this mechanism is decentralised by design wherein the fault tolerance is executed at the level of individual servers, rather than at the central controller level as it is done in present-day OpenStack.

In the next few sections, we will describe the background of our research (Section 2) and the resilience requirements of running UniServer in the data center (Section 3). Then, we detail the design and implementation of the proactive fault tolerance system (Section 4) and distinguish it from the reactive fault tolerance that can be achieved in current OpenStack (Section 5). Finally, we evaluate the mechanism in a data center setting (Section 6), discuss the results and conclude the document.

2. Background and Motivation

Dealing with hardware failures is part of the daily routine of Cloud data centre management and has direct impact on the dependability and service level delivered to the final users. Failure can impact users' workloads in various manners ranging from total service disruption to reduced perceived user experience. As of today, the most common source of failures in data centres are hard disks, followed by memory chips (~3%), and only a negligible percentage (<< 1%) is due to CPUs [1]. This is also confirmed by the available literature that mostly focuses on disks (both spinning and SSDs) [2] [3] [4], DRAM chips [5] [6] and GPUs [7].

2.1. Fault Tolerance in Data Centers

Failures can be treated mainly in two ways: reactively and proactively. In the first case, both the user (owner of the applications) and the cloud infrastructure owner can apply checkpoint/restart policies. Checkpointing for fault tolerance at the datacentre infrastructure level is not practical because it consumes precious resources in a cluster such as storage and network bandwidth. In addition, when checkpointing at the VM level the cloud provider does not have visibility of the application and a checkpoint could cause the application to be resumed in a corrupted state. As an example, if one node in a distributed deployment crashes, having a snapshot of that node does not guarantee a smooth continuation of operations as the application might not be able to handle the loss of one of the nodes. In the Cloud domain it is more common to rely on replication of applications/services across failure domains, while checkpointing at the application level is more common in the HPC domain.

In this deliverable, the attention is more focused on pro-active fault tolerance, that is the ability of a system to predict a potential future failure and immediately react to avoid disruption in users' applications. One example action is migration of virtual machines [8] to a trustworthy node. There are various previous works on predictions of failures in datacenters [9] [10] [11] [12] [13] that employ techniques such as Bayesian models, classic machine learning theories and reinforcement learning. One common point of most of the works in literature is relying on centralized schemes based on the analysis of clusters system traces or operating system logs, trying to correlate events with potential nodes failure. Centralized pro-active fault tolerance involves one single entity in charge of scanning the entire cluster logs looking for patterns that indicate a potential failure. Understanding how such an approach is not likely to scale on production-scale datacentres with hundred thousand of nodes is immediate. Traces need to be routed to a single location in the cluster, potentially creating a bottleneck in the network. Also, the single entity in charge of failures prediction might not be fast enough in preventing the failure of a node because busy in analysing the cluster's logs.

2.2. The UniServer Hardware Reliability Model

In Uniserver, hardware errors are directly exposed to the various software layers running on it via a dedicated interface. Having direct access to hardware counters is a unique opportunity for the cloud management software to perform informed decision based on the current status of each node. The work described in this deliverable moves the fault prediction process from being performed by the centralized cloud software to a single compute node, as an extension of OpenStack Nova compute. Each Nova compute agent is in fact capable of periodically monitoring the status of the hardware and independently taking decisions that aim at minimizing the impact of a potential failure on the running workloads. The extended nova agent periodically queries hypervisor interface and, using the UniServer CPU and Memory fault model, predicts node failures. The CPU and Memory fault models, and the node state changes process, are described in the next sections. To the best of our knowledge, this is the first attempt implementing such distributed pro-active fault tolerance in OpenStack.

Table 1: Target hardware platforms

Parameter	Intel platform	ARM platform
Architecture	x86-64 (Skylake)	ARMv8 (Ampere X-Gene 3)
#Cores	4	32
TDP(W)	80	125
Technology	14nm	16nm
Max. Freq.	3.3 GHz	3.0 GHz
Cache Size	8 MB	32 MB
Memory	32 GB	128 GB

2.3. UniServer System Model

The UniServer project envisions a system model where both hardware and software present features that are not available in today's production datacentres. From the hardware standpoint, each server will expose extended margins of the underlying hardware to improve energy efficiency. The extended margins refer to voltage and frequency states outside of the nominal guardbands for safe and correct operation imposed on processors and memory chips by manufacturers. Higher energy efficiency comes however at the price of increased probability of errors or failures. To leverage such new capabilities, the software running on top of a UniServer machine is modified accordingly. At the lower level, the Hypervisor (KVM in the case of UniServer) is extended to expose switching the hardware from nominal operating conditions to extended margins, and vice-versa. In addition, an extensive set of probes is exposed as well to monitor for symptoms of errors in both CPU and memory DIMMs.

As described in Deliverable 6.3, we have introduced a set of extensions to the OpenStack cloud management framework to benefit from the unique UniServer hardware features. A novel Resource Manager was developed to be coupled with the OpenStack Scheduler and enable scheduling decisions according to the operating point

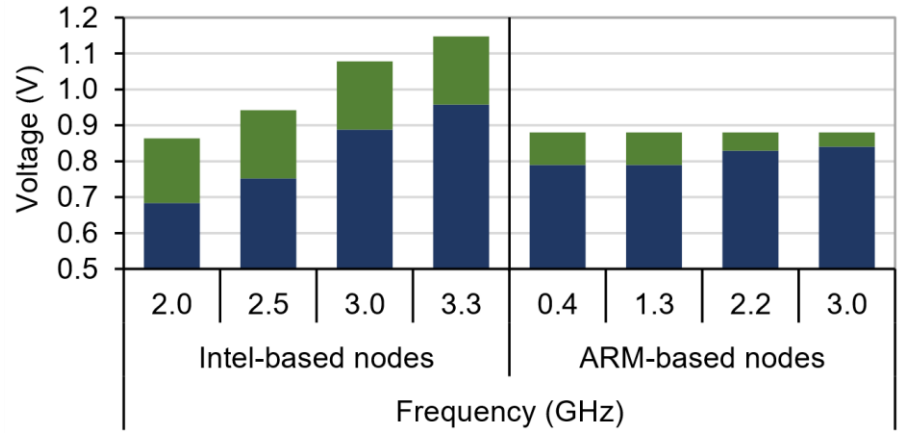

Figure 1: Voltage margins characterization for the Skylake (left) and the X-Gene 3 CPUs (right). The top (green) area of the bars denotes the range of sub-nominal voltages that did not result into any failure. The lower (blue) area of the bars are the voltages that lead to failures for some or all applications.

of each node in the datacentre. The modified OpenStack scheduler is able to distinguish between virtual machines with different priorities, assigning high priority ones only to nodes operating in nominal conditions. Low priority VMs can be assigned to any node regardless of its operating conditions. This strategy is a first strategy to cope with a higher probability of errors when running in extended margins mode. The main idea is that high priority VMs generate more revenue and thus should not be subject to the possibility of errors or failures. The Scheduler coupled with the Resource Manager is able to dynamically change the operating mode of a node to obtain the best trade-off between minimising energy consumption and minimising the number of high priority VMs being rejected at scheduling time.

In this deliverable, we are taking a step forward into further exploiting the energy efficiency gains of a UniServer enabled machine, by using the hardware probes exposing hardware monitoring information. Virtual machines are scheduled on any node regardless of its operating mode, placing the bet of scheduling high priority virtual machines also on nodes with higher probability of failure. During execution, each node is constantly monitoring the errors detected with the goal of predicting future failures. If a failure is predicted, a corrective action is taken by the node itself with the goal of keeping the high priority VMs always functional. If a node is unreliable because a fault has been predicted, it will immediately inform the central scheduler that will not consider it for further scheduling events of high priority VMs.

3. Resilience requirements for UniServer Machines

As mentioned previously, UniServer's unique approach elevates the probability of errors appearing in the hardware, and thus requires specific fault tolerance mechanisms. In the following subsections, the risk of failure from the processor due to running in extended margins is quantified.

The probability of failure when a processor is running in the extended margins is illustrated using a case-study with real hardware platforms, including commercially available CPUs used in modern servers. This analysis is targeting x86 (Intel Skylake family) and ARM64 (APM XGene3) server machines. Table 1 summarizes the target systems used for this characterization. In the following, we explain how we determine the parameters that capture the behaviour of a node and, specifically, the extended margins operating points and the failure probability when operating at such points.

3.1. Voltage margin characterization

In the Skylake processor, the voltage-frequency operating point is dynamically controlled by the P-State manager / DVFS governor. For this evaluation, four frequency points are selected at 3.3, 3.0, 2.5 and 2.0 GHz,

which cover the range of the most common frequencies applied in the balanced mode of the governor. The average nominal supply voltages for these frequencies are 1147, 1075, 922 and 850 mV, respectively.

The X-Gene 3 processor, targeting micro-servers, does not support DVFS. In nominal operation it allows only frequency scaling. We choose four frequency points, at 3.0, 2.2, 1.3 and 0.4 GHz, covering the entire range that is supported by this CPU. In all cases, the nominal supply voltage equals 880 mV.

For each nominal operating point (V, f), we experimentally determine a corresponding extended margins operating point (V_x, f). We use 24 benchmark applications from the SPECCPU 2006 suite [14] that stress different CPU components. An experiment consists of running each application for a continuous period of 10 minutes (if needed, we run the application several consecutive times). We perform a series of experiments, where we keep the CPU frequency fixed to f, and gradually reduce the CPU voltage in steps of 10 mV, starting from the nominal voltage V. We stop when one of the experiments leads to any type of abnormal behaviour (failure) and identify the immediately preceding voltage level (that did not result into a failure) as V_x for the frequency f.

Figure 1 illustrates the degree of undervolting that can be applied to each nominal operating point of the Skylake and X-Gene 3 CPUs. For the extended operating point at each frequency, we choose the lowest voltage within the green area, i.e., the lowest sub-nominal voltage that did not lead to any failure. For the Skylake CPU, the sub-nominal voltages for the four frequency points of 3.3, 3.0, 2.5 and 2.0 GHz are 929, 865, 741 and 666 mV, respectively. For the X-Gene 3 CPU, the sub-nominal voltages for the four frequency points of 3.0, 2.2, 1.3 and 0.4 GHz are 840, 830, 790 and 790 mV, respectively. The characterization process took about 32 hours for both machines. While this amount of time is not negligible, such a characterization needs to be performed once, when adding a new node to the datacentre, and then only very sporadically to capture aging effects.

3.2. Failure probability

When modern CPUs operate at nominal voltage-frequency points, hardware failures are extremely rare [15]. Failures are expected to occur more frequently if the CPU operates at a non-nominal point.

In the above experiments, when we operate the CPU at extended margins we choose the largest sub-nominal voltage that does not lead to any failures. Afterwards, we validate the safety of the identified sub-nominal voltages by executing multiple experimental campaigns of random workloads at each extended operating point for 6 consecutive days each.

Being unrealistically pessimistic, we assume that the 6 days of execution time without any errors is the mean time to failure (MTTF) for all tested extended margin points. This corresponds to a failure rate of one failure every 518,400 seconds, which -- again pessimistically -- we assume to be fatal, ignoring the possibility of correctable errors [16]. This failure rate is then used to estimate the probability of failure $Pfail_{V_x,f}$ within a scheduling period for a node that operates at any of the extended margin points (V_x, f). For example, assuming a scheduling period of 300 seconds, if a node is configured to operate at extended margins, the failure probability is 300/518,400 = 0.000579. Even with this relatively high failure probability, educated undervolting can provide a significant increase of the profit margin for the infrastructure provider.

4. Proactive Fault Tolerance

In this section, we describe how, starting from a known failure model (e.g., the UniServer failure model described in the previous section), the OpenStack Ocata cloud management software has been modified to proactively detect upcoming failures and start rescue actions for high value workloads.

A new component has been introduced in OpenStack to track each compute node's errors rate and, by exploiting the known error model, identify potential failures and act accordingly. Each UniServer compute node exports the count of errors identified by the hypervisor via the libvirt API (Deliverable 5.3). In addition, an error injection module based on a pre-defined and configurable errors distribution was developed to allow testing at scale and model failures tracking on non UniServer machines.

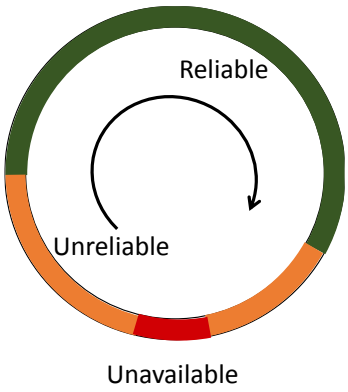

Figure 2: Proposed server failure lifecycle

4.1. Failure Lifecycle

In previous deliverables, we focused on the datacentre scheduler which resides in the central controller. Failure tolerance can be implemented at the central controller. However, as we discussed previously, centralised solutions do not scale well for datacentres with thousands of nodes. Therefore, we shift our focus in this deliverable to implement fault tolerance at the level of individual nodes. This brings us the following advantages:

1. Each node can monitor its own state and take proactive measures anticipating failure without involving the central controller. This reduces the latency of actions and improves resilience of the infrastructure.
2. We can reduce the amount of monitoring data that needs to be transferred from the nodes and stored in the central controller.

Hence, we have decided to implement our fault tolerance at the level of individual compute machines in the datacentre. Figure 2 conceptually shows the fault-driven lifecycle of a server machine in a cloud datacentre. Current datacentre resource management systems consider a node that is communicating with the central node to be in the *available* state, while a node that cannot be contacted due to a system crash or a network partition is categorised as *unavailable*. In this research, we have introduced the notion of a *reliable* state, in which the server is working as intended, and an *unreliable* state, in which the server is communicating and able to instantiate VMs but there is a high probability of system failure in a specific time horizon.

Figure 3 shows the transition between the states. Initially, when a server is brought up, it is in the available and reliable state. When an error is detected and the probability of failure is predicted to be higher, then the server shifts into the unreliable yet available state. When the server crashes or is not communicating, then it is considered to be unavailable and offline. When the server is brought back online, it is initially considered unreliable, due to its history. After a short maintenance period, if no errors are detected, then it is switched to a reliable state.

This conceptual lifecycle is useful for managing the server's lifecycle without involving the central controller as has been the case until now. This enables managing the state of the application VMs that are running on top of the server and safeguarding them from sudden changes in server state.

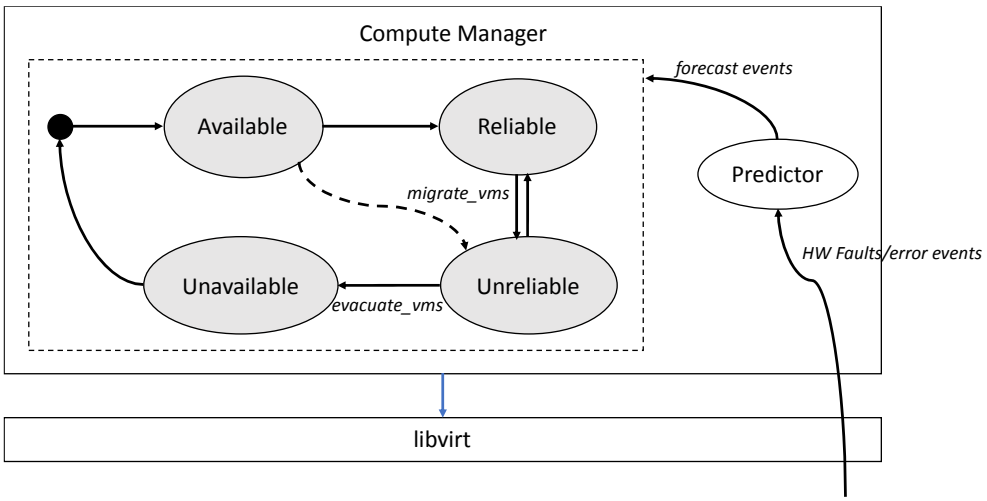

Figure 3: State Machine inside the OpenStack Compute Node

4.2. OpenStack Implementation of the Server Lifecycle

Figure 3 shows the implementation of the Server Failure Lifecycle in the OpenStack Compute manager. The Compute Manager keeps track of a server's resources such as CPUs, memory, hard disk space and network bandwidth, and also manages the hypervisor (e.g. KVM) that instantiates and operates the virtual machine instances on the server. When a VM is instantiated or destroyed, the Compute Manager reduces or augments the available resources by the amount requested by the VM and updates the database to reflect its current resource levels. The Compute Manager also creates and manages the virtual networks to which the VMs are connected. It communicates with the central controller to update the status of the physical server, and to initiate any VM migrations or evacuations.

Therefore, given the Compute Manager's centrality to managing the server resources, we decided to implement the Server Failure Lifecycle state machine in this component. Figure 3 illustrates the implementation and the control flow enabled by the state machine.

When a server is initialised (either from a cold start or a reboot), it is assumed to be Available. In case of a reboot or restart, if the server had switched to Unreliable/Unavailable prior to crashing, then its state is changed to Unreliable. Otherwise, the server is considered as Reliable.

During the server's operation, the Compute Manager constantly polls the system through the hypervisor for any new errors reported in the CPU, memory or hard disks. If so, these events are fed into the Predictor, which is a new component that we have introduced in the Compute Manager. The Predictor takes in a series of errors and provides a forecast with the probability of system failure in a specific time horizon. If this probability is higher than a (configurable) threshold, the server is transitioned into the Unreliable state. If the errors continue and the Predictor revises the failure probability to be higher than another (configurable) threshold, then the server is transitioned into Unavailable state and does not communicate with the central controller.

Our Compute Manager is able to follow different strategies for each transition. In the current implementation, the transition from Reliable to Unreliable triggers a process that examines the priority of the VMs currently running on the server. The high-priority VMs are queued to be migrated off to other, reliable nodes. This migration can be done either live or offline. We have chosen live migration to preserve the state of high-priority VMs. Current OpenStack implementation of migration requires that the Compute Manager sends a request to the central scheduler which replies back with a target host for the VM. The Compute Manager then initiates the migration process.

If errors continue to happen while the server is in the Unreliable state, failure becomes a real possibility and evasive actions need to be taken. When the Predictor forecasts a high probability of error, then the server is transitioned into the Unavailable state. This prevents the central controller from allocating any further VM instance requests to the server. The low priority VMs running on the server are shut down and marked as deleted. In the Unreliable state, there are no high-priority VMs running on the server.

There is a probability that server failure might occur at any time, even when it is in the Reliable state although it is more likely that this would happen in the Unreliable state, since errors have already started being apparent

at the hardware level. In case of abrupt failure, current OpenStack failure mechanisms are applied, wherein a server (or its Compute service) is marked as disabled if it does not report to heartbeat messages from the central controller. Our implementation of the Unavailable state provides a more graceful method of managing this disruption and provides up-to-date information to the central controller.

When the server is brought back to Available state, after a period of maintenance (for example), it is set into Unreliable due to its previous history. This implies that no high-priority VMs are scheduled on to the server unless it is monitored through a (configurable) probationary period and reported as not producing any errors. According to the bathtub curve, errors are more prevalent at the beginning and the end of the server lifecycle. Hence, this method manages the risk of server failure when it is brought back from stasis after a crash.

4.3. Forecasting Failures

The Predictor provides a short-term forecast on the probability of system failure. However, at the hardware level, we obtain data only on the number of errors emanating from the hardware. Therefore, the challenge is to construct a statistical pattern out of this data that can aid us in determining the probability of system failure.

While several machine learning methods have been proposed for predicting server failures, these are not suitable for deployment in the unique environment in which UniServer operates. Firstly, these have been mostly developed for modelling server crashes due to hard disk failures. The UniServer model envisages errors emanating from the processor and memory due to operating servers at voltages outside of the manufacturer recommended guardbands. This renders models based on analysis of hard disks' SMART data to be inapplicable to our context.

Secondly, the extended voltage margins outside of the guardbands are unique for each processor. This runs counter to central machine learning models that assume that the data obtained from each server are uniform and can be coalesced into a single model. Hence, each individual server must have its own custom parameters for the prediction model.

Lastly, most of these models require data to be gathered from the servers and analysed in the central controller. These models are heavyweight and require significant computational resources to produce predictions. Our design requires that each server predict its own state. Hence, the models at individual servers must be lightweight so that the capacity of the server to host VMs is not impacted.

Given the above requirements, we opted for a simple statistical combination of moving averages and linear regression to construct a prediction model to forecast server failure. Moving averages with overlapping windows dampen the effect of sudden spikes (such as a large number of errors). Linear regression (Ordinary Least Squares) is a standard statistical method to predict the trend in a series of observations.

Therefore, given a list of n sequential error observations, $E = \{o_0, o_1, \ldots o_{n-1}\}$, the array of moving averages over a window of size w $(w < n)$ is:

$E_m = \{a_0, a_1, \ldots, a_{n-w+1}\}$, wherein $a_k = \frac{1}{w}\sum_k^{k+w-1} o_i$

We then compute the slope of the simple linear regression line passing through the moving averages by using the standard formula:

$slope = \frac{m\sum E_m.X - \sum E_m.\sum X}{m\sum X^2 - (\sum X)^2}$, where $m = n - w + 1$, and $X = \{0, 1, \cdots m\}$

Briefly, a positive slope implies that the number of errors and their magnitude are increasing, which indicates that the system is unreliable. A slope that is zero or close to it indicates one of three scenarios – zero errors (stable, reliable operation), intermittent errors but no apparent trend (unstable equilibrium, unreliable operation), or consistent production of errors (imminent failure, soon to be unavailable). We use the cumulative sum of errors to distinguish between these three situations. A negative slope indicates that the system has left a period of unreliable operation and is now reliable.

In our implementation, we observe and gather the errors emanating from the hardware and their timestamps into an array that we input into the model which returns the value of the slope. This is used to drive the state change, if needed.

4.4. Scheduling Algorithm

Deliverable D6.2 introduced a priority-based scheduling algorithm that we subsequently implemented into OpenStack (deliverable D6.3). Briefly, the scheduler aims to allocate high-priority or high value VM requests (these are considered equivalent) to the most reliable hosts so that these are never disrupted.

In our previous work, we configured a portion of the servers in a datacentre in extended margins (outside of the guardbands) while the rest of the servers were left operating in the conservative nominal (within guardbands) settings. The former are more power efficient while being considered less reliable than the latter. Therefore, the scheduler would place high-priority VMs on to the hosts running in nominal mode. When there are no more servers running in nominal mode, the scheduler flips selected servers running in the extended mode to nominal and evicts low-priority VMs from these nodes to satisfy high-priority requests. The downside of this flipping process is that it may require servers to be restarted to safeguard against latent errors, and disrupts the normal operation of these servers.

By introducing the reliable and unreliable states along with a proactive Predictor whose forecasts drive the transitions between the states, we have eliminated the requirement for servers to be locked into a certain configuration until resource requirements drive the change. When high-priority VMs arrive, the scheduler now searches for servers that are in the Reliable state, regardless of whether they are running in the nominal or extended mode. If there is no capacity in existing Reliable servers, then the scheduler evicts low-priority VMs from selected Reliable servers until enough resources are freed.

When a server transitions from Reliable to Unreliable due to errors, the scheduler receives requests for migrating high-priority VMs that are hosted on the server. The scheduler then identifies a target host using the mechanism described previously. The actual migration is handled by the OpenStack Compute agents running on the servers.

5. Reactive Fault Tolerance

Reactive fault tolerance is the ability of recovering the VM previously executing on a failed node. This strategy enters into play if there was no pro-active fault tolerance policy in the datacentre or if that was not effective enough to migrate high value VMs before the failure happens.

In the context of UniServer, reactive fault tolerance is implemented by re-using the priority-based scheduler introduced in Deliverable 6.3. Upon the failure of a node, it is possible to access the list of VM instances previously running on that node. The policy to react to the failure is to re-submit all the VMs to the priority scheduler respecting the same priority levels originally assigned by the user. Re-scheduled VMs are treated as new ones and the scheduling decision will be subject to both the current availability of resources as well as the current operating conditions of all the nodes.

Understanding if a node is available is a trivial task in OpenStack as each Nova Compute service periodically sends a heartbeat signal to the Nova Conductor. This information is in turn updated into the OpenStack Nova database. A periodic task scans the compute services in the Nova database to identify failed nodes. When a failure is detected all the VMs are re-submitted to the priority scheduler using the Nova evacuate functionality that automatically re-deploys all the VMs formerly running on the failed node.

6. Evaluation

In this section, we describe the experimental evaluation of the fault tolerance mechanisms introduced in this deliverable.

6.1. Experimental Setup

The proactive fault tolerance mechanism introduced in this document requires a full datacentre with shared storage to be set up in order to demonstrate its utility. While the UniServer project is based on Ampere's XGene servers with ARM CPUs, the evaluation of the OpenStack fault tolerance mechanisms required more machines than the XGene servers currently available to all the project partners. Hence, we have decided to use virtual

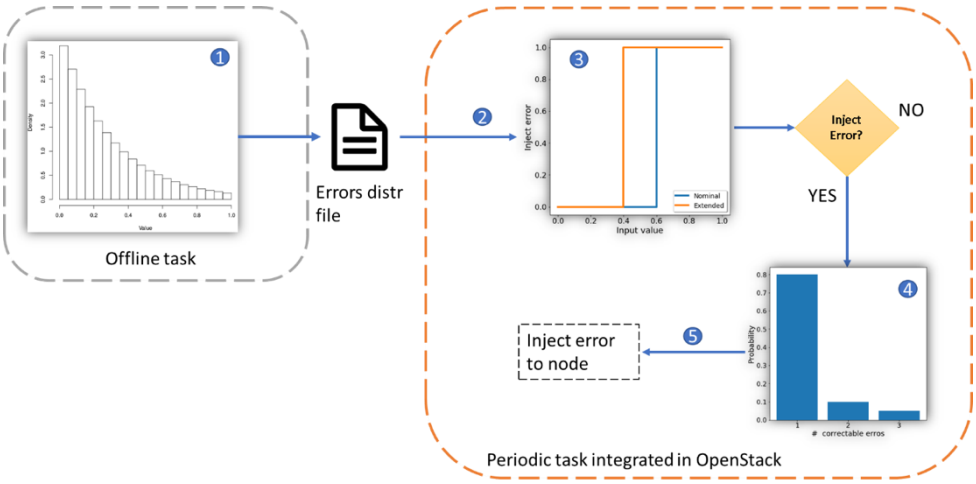

Figure 4: Error Injection Process

clusters to emulate the operation of a datacentre with the fault tolerance mechanisms applied. Our changes to OpenStack have been verified to run on top of the two XGene servers currently installed in IBM Research – Ireland. Therefore, the results of this evaluation would be consistent if it was conducted on a similarly-sized XGene cluster

We have used the nested virtualization [17] feature of KVM to generate a virtual cluster on a powerful compute server. The physical machine is a 64-bit IBM Power8E server with 192 cores and 1TB of RAM. We have created a cluster of 21 VMs, each with 12 cores and 40GB RAM, on which we installed and configured OpenStack modified with the fault tolerant mechanism presented previously. One of the VMs is configured as the controller node, hosting components such as the scheduler, conductor, network manager and disk image service. The other 20 VMs are used to emulate compute nodes configured with the State Machine described previously. We can create VMs on top of these to emulate the operation of a set of physical machines in a cluster.

Many of the advanced features enabled by UniServer for the XGenes, such as the extended mode, were not available to us in the virtual cluster. Also, there would be no errors generated in the Power processor as envisaged by the UniServer. Even if we were able to run on an equivalent number of XGenes, we would have to wait for a long time before an error is generated in the extended mode. Hence, for our experiments, we had to create a synthetic process to inject errors to trigger the fault tolerance processes. The next section describes this process in more detail.

6.2. Injecting errors

For the testing at scale, the pro-active fault tolerance OpenStack components were evaluated on a set of machines that do not provide facilities for accessing the errors counts at the CPU and memory level. Instead, each node generates errors according to a well-known distribution, using an ad-hoc software component that injects errors into the OpenStack Nova libvirt driver. To validate the actual benefit of the new proactive fault tolerance component developed for OpenStack, a fault injection system has been developed as well, modelling the arrival of correctable errors on each compute node. Such component is directly interfaced to the UniServer libvirt API by providing a custom implementation of the `getErrorsUniserver()` function. Each time the function is invoked, errors are injected and generated from the Weibull distribution that is renown of being representative of hardware components failures over time [18].

The process of failure generation is composed of 5 phases, depicted in Figure 4. In phase 1, the Weibull distribution is used to generate a pre-defined number of values between 0 and 1. This is an offline operation that is performed before starting OpenStack. The parameters of the distribution are configurable during file creation. The file generated during this first phase is then fed to the failure injection task integrated within OpenStack Nova. At instantiation of OpenStack, a dedicated thread is created to take care of the injection of faults to the node. This thread periodically reads a value (phase 2) from the file described above and applies to it a filter (phase 3) implemented as a step function, to decide whether an error is to be injected or not. As visible in Figure 4, the step function uses two different thresholds to model nominal and extended margins operating conditions. In other words, the threshold is used to model a higher likelihood of error when the node

is operating in extended margins. Errors are injected when the value read from the file is greater than or equal to the threshold. The model developed for this deliverable envisions the possibility to inject one, two or three errors at a time. The number of errors is selected using a pre-defined static probability (phase 4). For the sake of simplicity, the injection task generates only one class of errors (e.g., CPU) as opposed to the various error classes tracked by hypervisor.

After the error injection task computes the number of errors to be injected, the value is stored inside the libvirt driver module of OpenStack Nova (phase 5). Upon a call to the `getErrorsUniserver`, errors are returned from the local value stored in the libvirt driver and not from the hypervisor.

Via the OpenStack Nova configuration, it is possible to set the interval at which errors are injected into the node, the probabilities used to choose the number of errors to be injected, the error threshold for both nominal and extended margin operation, and the path to the file holding the values generated from the Weibull distribution.

6.3. Workload and Experiment Design

Our aim here is to highlight the changes that are brought by the fault-tolerance approach described previously. We have used the same workload model used in the previous deliverable (D6.3) that was used to evaluate the OpenStack Resource Management and Scheduling under UniServer conditions. We are repeating the workload description here for sake of completeness.

The workload was devised to be a stream of VM requests arriving at the resource manager during a particular interval. We created a workload trace of 256 VM requests with an inter-arrival time of 30 secs. We assigned priorities randomly to these requests. Table 2 shows the distribution of the priorities as well as the resource requirements for VM belonging to each priority class.

Table 2: Workload Mix for the Experiments

Priority	Flavour (resource req.)	Share of Total
0	Small – 1 CPU, 1GB RAM	40%
1	Medium – 2 CPU, 2 GB RAM	30%
2	Large – 4 CPU, 4 GB RAM	20%
3	XLarge – 8 CPU, 8 GB RAM	10%

The workload mix that we have chosen reflects the standard workload for any cloud provider wherein there are lots of requests to create small VMs with low priority, while the larger, high-priority, and more profitable VMs are rarer. We have also defined a lifetime for VMs with priority 0 and 1 where the former were alive for 3 minutes while priority 1 VMs would be killed after a duration randomly drawn between 3-6 minutes. This reflects the scenario in which VMs are initialised to execute a single analytics or a monitoring task and are then shutdown.

We have created custom metrics to reflect the measures that we consider to be of interest to the fault-tolerant approach. These are:

1. Number of active VMs over time
2. Number of VM requests rejected by the scheduler
3. Number of servers in nominal and extended margins over experiment duration

Metrics 1, and 2 reflect the effect of the fault tolerance mechanisms on the workload, especially on the revenue-generating high-priority VMs while metric 3 focuses on the power efficiency of the physical infrastructure. These metrics were computed using custom scripts that mined the log files in the different servers and database entries created over the course of the experiment.

At the beginning of the experiment we configure 50% of the nodes to operate in extended mode. Each compute node runs one instance of the errors generation component described in Section 6.2. The series of numbers

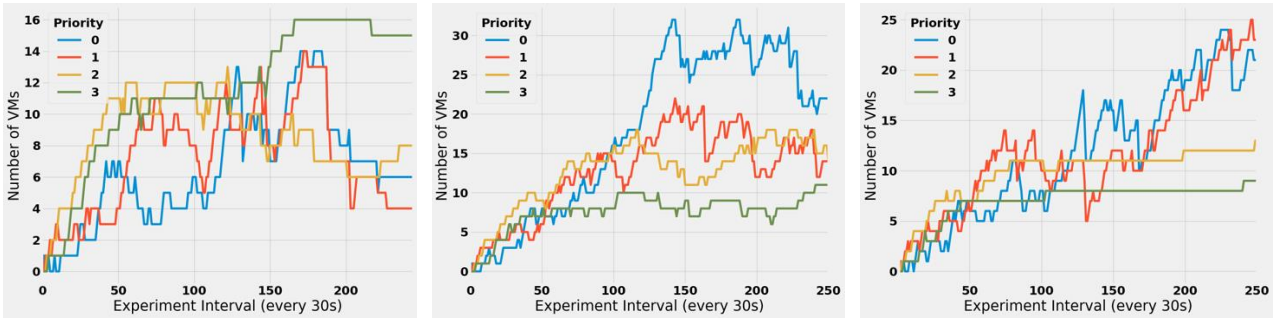

Figure 5: Active VMs over time grouped by priority level. Each chart represents a different scenario: sm+r (left), sm+nomig (center), nosm+nomig (right)

used to decide when to inject an error (Weibull) is kept constant across all scenarios for the sake of comparison of the results.

6.4. Results and Discussion

The experiment described in the previous section was run to compare the three following scenarios:

1. Node state and resiliency manager enabled (`sm+r`)
2. Node state and resiliency manager with migration disabled (`sm+nomig`)
3. Priority scheduler without state and resiliency manager (`nosm+nomig`)

In the first two scenarios, we take the bet of scheduling high value VMs on any node regardless of its operating mode. Nodes are assigned an operating mode at the beginning, and it is kept unchanged during the whole experiment. The state manager on each compute node is constantly monitoring the errors logged by the hypervisor and predicts the node state (the four in Figure 2). When a node transitions to *Unreliable* state, all the VMs with priority of 2 or more are migrated to a safer node. In case of *Unavailable* node all the workloads currently running are killed to simulate an abrupt node failure. The resiliency manager integrated in the scheduler allows high value VMs (priority 2 or more) to be scheduled only on *Available* or *Reliable* nodes.

The third scenario uses only the priority scheduler introduced in Deliverable 6.3. The priority scheduler allocates VMs on compute nodes according to their priority. VMs with priority 2 or 3 (high value) are only scheduled on nodes operating in nominal conditions. Lower priority VMs can be scheduled on nodes regardless of their current operating mode. In the case where there are no nodes in nominal state that can host the VM the priority scheduler switches a node (if available) to nominal conditions and schedules the VM on it. In this case the state manager on each node is used only to model nodes failure when transitioning to state *Unavailable*. No actions are taken for other state transitions. In all scenarios, VMs with priority of 2 or more can cause the eviction of one or more lower priority VM when all the resources in the cluster are occupied by other VMs.

The idea behind the three scenarios is to show the effectiveness of a proactive fault-tolerance mechanism coupled with the opportunities for reduced energy consumption introduced by UniServer. In addition, the first two policies differ only on the proactive action taken in case of Unreliable nodes; `sm+r` migrates VMs off unreliable nodes, while `sm+nomig` does not. Our goal here is to highlight the importance of the proactive action in guaranteeing high availability for high priority jobs.

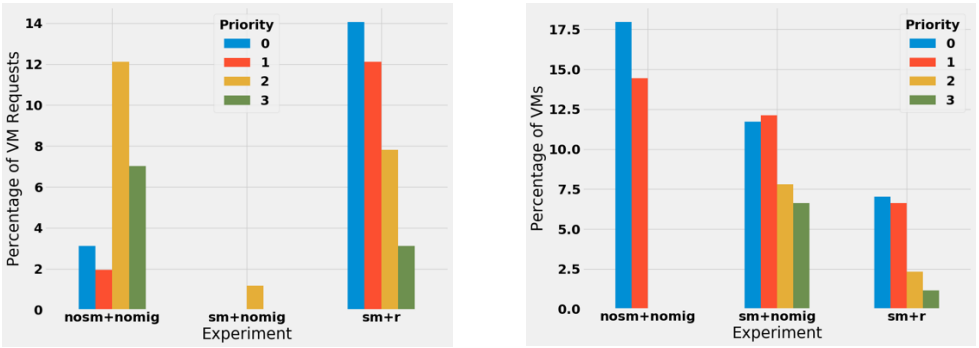

Figure 6: Percentage of VMs deleted (left) and rejected (right) grouped by priority level.

The first metric we analyse is the number of active VMs (percentage over the total number of VMs scheduled in the experiment) over time (Figure 5) grouped by instance priority. The first thing to note is that the proactive fault tolerance (`sm+r`) mechanism guarantees a higher number of high value VMs over time with a peak of 16% of the VMs running with priority 3. The priority scheduler (`nosm+nomig`) instead is not able to react to node state changes and loses precious workloads due to node failures. To validate the importance of a proactive action the `sm+nomig` scenario shows how disabling migration of workloads brings the active high priority VMs count down, with a similar trend measured with the priority scheduler. The resiliency manager alone is not able to preserve as many high value workloads as the case where resiliency manager and compute state node manager work together. The proactive migration of VMs is coping with the potential failure of nodes and avoids high priority workloads to fail.

Interesting to note is how the number of active priority 0 VMs is particularly high in the `sm+nomig` case (mid chart in the figure). This is a result of the failure of nodes that cause the destruction of high priority VMs, making room for new incoming VMs. Priority 0 VMs are the most frequently scheduled and quickly dominate the experimental cluster. When migration is enabled (left chart) the migration causes the eviction of low priority VMs in favour of higher priority ones.

The activity of VMs by itself does not give a complete understanding of how effective is the proactive fault tolerance policy in preserving high value workloads. In Figure 6, we show the percentage (over the total number of VMs submitted) of VMs deleted because of nodes failures (left) and VMs rejected because of a lack of resources (right). As expected, in the `nosm+nomig` scenario nodes failures are never killing high value workloads because those VMs are never scheduled on nodes running in extended mode. In our experiment nodes running in nominal conditions do not fail. When the full proactive policy is enabled (`sm+r`) high priority VMs are allowed to run on nodes in extended mode and thus are subject to nodes' failures. Those priority 2 and 3 VMs deleted are the ones for which the proactive policy is not acting fast enough as the node they are running on fails before the migration to a safer one is completed.

Overall, including the rejected VMs, the `sm+r` scenario fails in handling ~4% of priority 3 and ~9% of priority 2 VMs. This represents a reduction of respectively 50% and 30% with respect to the `nosm+nomig` case. Again, to show the importance of VMs migration in this context, the `sm+nomig` case experiences a higher number of deleted VMs due to nodes failures. VMs are scheduled on any node and the absence of a proactive action

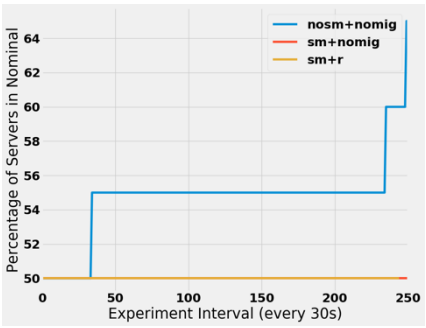

Figure 7 Percentage of servers in nominal mode over time

increases the number of failed high value VMs. Interestingly, `sm+nomig` shows an extremely low number of rejections. This is due to the node failures. Since migration is not enabled, the existing VMs on failed nodes are destroyed which creates more room for accepting new incoming VM requests.

At this point we can draw a preliminary conclusion: the full proactive policy further improves the utilization of the cluster from the point of view of the active/accepted VMs with respect to the priority-based scheduler of D6.3. However, the cherry on top appears when analyzing the experiment from the perspective of the compute nodes. The full pro-active policy is able to guarantee a better level of service compared to the priority scheduler while also keeping 50% of the compute nodes constantly operating in extended mode. Instead the `nosm+nomig` case keeps switching nodes to nominal to host new incoming high-priority VMs. This behavior is visible in Figure 7 where the scenario without resilience and node state manager approaches the end with 64% of the nodes operating in nominal conditions.

The extended pro-active fault tolerance support introduced in OpenStack increases the opportunity for improved energy efficiency at the cluster level by always operating a set of the machines always in extended mode. Without this support, the scheduler instead is not able to predict and avoid failures, and can't take the risk of placing valuable workloads on nodes running in relaxed conditions.

7. Conclusion

This deliverable presents the activities carried out in the context of fault tolerance, and more precisely regarding proactive fault tolerance. We started by presenting the motivations behind the need for such proactive mechanisms. Thanks to a characterization process based on a pre-defined set of benchmarks we showed that reducing the voltage supply of the CPU increases the probability of failure. This drawback can limit the reduced consumption benefits when undervolting is applied at large scale. Simply undervolting a high number of servers in a datacenter could increase the aggregate probability of failure to the point where the gains are not worth the risk.

The UniServer software stack introduces novel capabilities integrated in the hypervisor that expose the probability of failure of nodes at a given operating point. We have extended OpenStack with a novel distributed fault tolerance mechanism that, by accessing real-time information from the hypervisor, can take proactive action to save valuable workloads (e.g. migrate to a safer node). The distributed design breaks the monolithic management of nodes in OpenStack thereby enabling nodes to self-assess their state and proactively inform the central scheduler of the workloads that need to be preserved. The new fault tolerance mechanism is implemented in the OpenStack Nova compute manager component running on each node, and with extensions to the priority scheduler presented in the Deliverable 6.3.

Our experimental evaluation shows that when the proactive fault tolerance policies are enabled, we are able to service the same set of virtual machines with a set of nodes constantly operating in extended mode, in contrast to the implementation in Deliverable 6.3 where compute nodes are switched to normal operating mode to host high value workloads. Even though part of the nodes in our experimental setup were always operating in extended margin mode, the number of high value workloads not properly serviced (deleted for failure or rejected) is significantly lower than with scheduler based only on the priorities of VMs. This demonstrates that the combination of hardware and software innovations introduced by UniServer clearly creates opportunities for increasing the energy efficiency of a datacenter while still guaranteeing the level of service of current datacenter infrastructures.

References

[1] G. Wang, L. Zhang and W. Xu, "What Can We Learn from Four Years of Data Center Hardware Failures?," in *International Conference on Dependable Systems and Networks (DSN)*, 2017.

[2] E. Pinheiro, W.-D. Weber and A. Barroso, "Failure Trends in a Large Disk Drive Population," Usenix FAST, 2007.

[3] B. Schroeder and G. A. Bibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?," in *Usenix FAST*, 2007.

[4] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubamaniam, B. Cutler, J. Liu, B. Khessib and K. Vaid, "SSD failures in datacenters: What, when and why?," *ACM Sigmetrics Performance Evaluation Review,* vol. 44, no. 1, pp. 407-408, 2016.

[5] J. Meza, Q. Wu, S. Kumar and O. Mutlu, "evisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *International Conference on Dependable Systems and Networks (DSN)*, 2015.

[6] B. Schroeder, E. Pinheiro and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," *ACM SIGMETRICS Performance Evaluation Review,* vol. 37, no. 1, pp. 193-204, 2009.

[7] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben and P. Navaux, "Understand ing GPU errors on large-scale HPC systems and the implications for system design and operation," in *HPCA*, 2015.

[8] D. Ruprecht, D. Jones, D. Shiraev, G. Harmon, S. Maya, M. Krebs, M. Baker-Harvey and T. Sanderson, "VM Live Migration At Scale," in *International Conference on Virtual Execution Environments (VEE)*, 2018.

[9] Z. Xue, X. Dong, S. Ma and W. Dong, "A Survey on Failure Prediction of Large-Scale Server Clusters," in *International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, 2007.

[10] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi and Y. Matsumoto, "Online failure prediction in cloud datacenters by real-time message pattern learning," in *International Conference on Cloud Computing Technology and Science*, 2012.

[11] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, C. Browne and B. Barth, "Linking Resource Usage Anomalies with System Failures from Cluster Log Data," in *nternational Symposium on Reliable Distributed Systems*, 2013.

[12] Y. Watanabe and Y. Matsumoto, "Online Failure Prediction in Cloud," *Fujitsu scientific & technical journal,* vol. 50, no. 1, pp. 66-71, 2014.

[13] X. Chen, C.-D. Lu and K. Pattabiraman, "Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study," in *International Symposium on Software Reliability Engineering Workshops*, 2014.

[14] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News},* vol. 34, pp. 1--17, 2006.

[15] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, 2010, pp. 193--204.

[16] A. Bacha and R. Teodorescu, "Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[17] e. a. M. Yehuda, "The turtles project: design and implementation of nested virtualization," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010.

[18] A. C. Cohen, "Maximum likelihood estimation in the Weibull distribution based on complete and on censored samples," *Technometrics,* vol. 7, no. 4, p. 579588, 1965.

[19] C. Reiss, J. Wilkes and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," Google Inc, Mountain View, CA, USA, 2011.

