



D7.1 First Report on Metrics of Success Against State-of-the-Art

Contract number	688540
Project website	http://www.uniserver2020.eu
Contractual deadline	Project Month 18 (M18): 31 st July 2017
Actual Delivery Date	14 August 2017
Dissemination level	Public
Report Version	1.0
Main Authors	Peter Lawthers (APM)
Contributors	Georgios Karakonstantis (QUB), Dimitris Gizopoulos (UoA), Athanasios Chatzidimitriou (UoA), Manolis Kaliorakis (UoA), George Papadimitriou (UoA), Kostas Katrinis (IBM), Marios Kleanthous (Meritorius), Panagiota Nikolaou (UCY), Arnau Prat (Sparsity), Christos Kalogirou (UTH), Panos Koutsovasilis (UTH), Manolis Maroudas (UTH), Christos D. Antonopoulos (UTH), Spyros Lalis (UTH), Mustafa Rafique (IBM), Alejandro Lampropulos (WSE)
Reviewers	Arnau Prat (Sparsity), Spyros Lalis (UTH), Charles Gillan (QUB), Panagiota Nikolaou (UCY)
Keywords	Metrics, TCO, Application, SLA, QoS

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.

© 2017. UniServer Consortium Partners. All rights reserved

Disclaimer

This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement Nr 688540. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the project and to the extent foreseen in such agreements.

Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. UniServer is a 36-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>.

Confidentiality Note

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the UniServer Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

Change Log

Version	Description of change
1.0	Initial version delivered to EC.

Table of Contents

1. INTRODUCTION	9
2. SYSTEM OVERVIEW	10
3. APPROACH	12
3.1 OVERVIEW	12
3.2 METRICS	12
3.3 RELATED STATE OF THE ART WORK	16
4. HARDWARE METRICS	18
4.1 ERROR TYPES	18
4.2 SEVERITY FUNCTION	18
4.3 LONG-TERM WEAR	19
4.4 GRACEFUL PERFORMANCE DEGRADATION	19
5. SYSTEM SOFTWARE METRICS	20
5.1 HYPERVISOR	20
5.2 OPENSTACK	21
5.2.1 <i>Energy Efficiency</i>	21
5.2.2 <i>Proactive fault tolerance</i>	21
5.2.3 <i>Reactive fault tolerance</i>	21
6. APPLICATION	23
6.1 WSE WIRELESS JAMMER DETECTOR	23
6.1.1 <i>Availability</i>	24
6.1.2 <i>Latency</i>	24
6.1.3 <i>Accuracy</i>	25
6.2 MERITORIOUS TRADE CONFIRMATION	25
6.3 SPA: SOCIALCRM/SOCIALTV	26
6.3.1 <i>The social network server component</i>	26
6.3.2 <i>The social analytics component</i>	27
6.3.3 <i>The web app component</i>	28
7. CONCLUSIONS AND FUTURE WORK	29
8. REFERENCES	30

Index of Figures

Figure 1: Detailed Uniserver Layering	10
Figure 2: Normalized TCO to ChipkillIDC for WebSearch and Floreon+ applications [33]	14
Figure 3: Pfail, energy, and TCO estimation	15
Figure 4: Hypervisor and OpenStack relationship	20
Figure 5: DoS Sensing Architecture	23
Figure 6: Trade confirmation architecture	25

Index of Tables

Table 1: Table of Terms.....	7
Table 2: Uniserver levels	10
Table 3: Processor comparison	11
Table 4: Levels of Metrics.....	13
Table 5: Trade-off of two DRAM ECC mechanisms (ChipkillDC and ChipkillSC) [33].....	14
Table 6: TCO factors	17
Table 7: Error types due to hardware malfunction	18
Table 8: UniServer target applications	23
Table 9: Application Metrics.	23
Table 10: WSE jammer detector metrics.....	24
Table 11: Meritorius trade confirmation metrics	26
Table 12 - Social Network Server SLA metrics	27
Table 13 - SLA for the Social Analytics Component	28
Table 14: SLA for the Web App.....	28

Terminology

Term	Definition
DoS	Denial of Service
EOP	Extended Operating Points
HEI	Hardware Exposure Interface
KVM	Kernel Virtual Machine
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Recover
PMD	Processor Module
QOS	Quality of Service
RAS	Reliability, Availability, Scalability
SDC	Silent Data Corruption
SLA	Service Level Agreement
SoC	System-on-Chip
TCO	Total Cost of Ownership
TDP	Thermal Design Power
VFS	Voltage and Frequency Scaling
VM	Virtual Machine

Table 1: Table of Terms

Executive Summary

UniServer seeks to improve the performance and energy efficiency in servers by automatically discovering the capability of the underlying hardware components to function beyond nominal operating points. By taking advantage of the extended margins inherent in processors and memories, it is possible to improve the power efficiency of ARM-based micro-servers running in the cloud or edge. Understanding how to take advantage of these margins and successfully exploit them requires detailed metrics throughout the system stack. These metrics track and quantify operation at each system level, and are used to evaluate system and application behavior as compared to a conventional system that does not attempt to go beyond nominal operating points.

This deliverable describes the metrics that will be used at various levels within the UniServer framework. Metrics are categorized into three levels: hardware, system and application. The metrics at the different layers are strongly related, with metrics at a particular level being associated with the metrics at the previous lower levels.

At the hardware level the metrics are mainly related to the reliability of processor and DRAM components. The severity metric and long-term wear metric are proposed at this level. The severity metric takes into account the different types of errors that have been observed, including silent data corruption, correctable errors, uncorrectable errors, application crash/hang and system crash/hang. The long-term wear metric is related to the long-term effects on functionality and performance of the voltage and frequency scaling on the three components (DRAM, cache, and core).

The system level metrics concern the Hypervisor and Open Stack, including metrics such as expected energy consumption and Service Level Agreement (SLA) violation penalties affecting the Total Cost of Ownership (TCO).

Finally, application metrics are related to the corresponding SLA, which specifies the application's quality of service (QoS) requirements such as processing/response latency, availability and result accuracy. The ultimate goal of UniServer is to meet these QoS requirements but at a reduced energy footprint and TCO compared to a system that does not attempt to exploit the extended margins of hardware components.

1. Introduction

This document describes the initial report on the metrics that will be used to measure the success of UniServer, i.e., to evaluate the operation of the UniServer platform compared to a conventional system that does not attempt to operate beyond nominal (voltage and frequency) settings and memory refresh rates. The respective work is done as part of Task T7.5 “End-to-end TCO Analysis and Metrics of Success” under Work Package 7 (WP7 – “Cross-Layer Secure System Integration and Evaluation”), and depends on all other components of the system: the DRAM and Cache characterization from WP3, the HEI, StressLog and HealthLog developed in WP4, the Hypervisor enhancements in WP5, the OpenStack extensions discussed in WP6, and finally the applications themselves in WP7.

From the perspective of a customer, that is an end-user or the application owner, the success of the UniServer approach depends ultimately on whether the application running on the server meets the Service Level Agreement (SLA) while running under the UniServer constraints. If the SLA is met, and the cost associated with the recovery mechanisms is low compared to the achieved energy savings and thus is acceptable under the Total Cost of Ownership (TCO), then the approach can be deemed a success. The purpose of an SLA is to define, in a concrete way, the kind and quality of service the customer should receive. SLAs do not define how the service itself is provided/delivered, thus the metrics that apply to the internal functioning of the UniServer platform are not in general directly visible to the customer, even though they can impact on the delivery of the service.

Given this context and for convenience, in UniServer metrics of success are categorized into various levels, depending on that part of the system stack where the metric is used: hardware, system and application. Hardware is constantly monitored and metrics quantifying its operation are reported to the software level, which correlates the observed metric values with those of its own level. The metrics are subsequently used to take actions (e.g. move a VM from one host to another) and decide the operating margins, all based on the SLAs of the applications.

The rest of this document is structured as follows. Section 2 gives an overview of the system structure and different layers of the UniServer system stack. Section 3 discusses the overall approach and categorization of the metrics in each level of the system. Section 4 describes the metrics for the hardware components. Section 5 discusses the Hypervisor and Open Stack (cloud) related metrics. Section 6 expands on the specific metrics for each of the three target applications that are being used to demonstrate the UniServer approach. Section 7 summarizes the conclusions.

2. System Overview

The UniServer system can be logically divided into three parts:

Level	Description
Hardware	X-Gene2 based “Tigershark” systems or X-Gene3 based “Osprey” systems
System Software	Linux OS, virtualization extensions such as KVM/Qemu, and OpenStack cloud infrastructure
Application	Applications running in the context of a VM

Table 2: Uniserver levels

In particular, starting from the low layers we develop techniques that aim at revealing new Extended Operating Points (EOP) for each hardware component based on the component’s true capabilities (which can be well beyond its nominal and typically quite conservative operating points). This is achieved by stress-testing the hardware components during a pre-deployment phase under different points using various stress kernels. During deployment, the HealthLog daemon is monitoring the health status of the hardware under any used voltage/frequency/refresh rate (V-F-R) point and informs the system software (i.e. Hypervisor) by propagating information vectors about the performance, power, temperature, and any incurred errors.

Moreover, another Linux daemon, the StressLog, is responsible for periodic offline, on-demand stress testing of the hardware components and for producing an output vector containing the new safe system V-F-R margins that will be suggested to the software (i.e. Hypervisor) for future usage. It also produces log files recording errors (correctable or uncorrectable), system configuration values, sensor readings and performance counters. Using the information provided by the HealthLog and StressLog, the Predictor develops probability failure models and tries to predict the hardware behavior under any operating point and eventually helping the system software to decide on the optimum configuration.

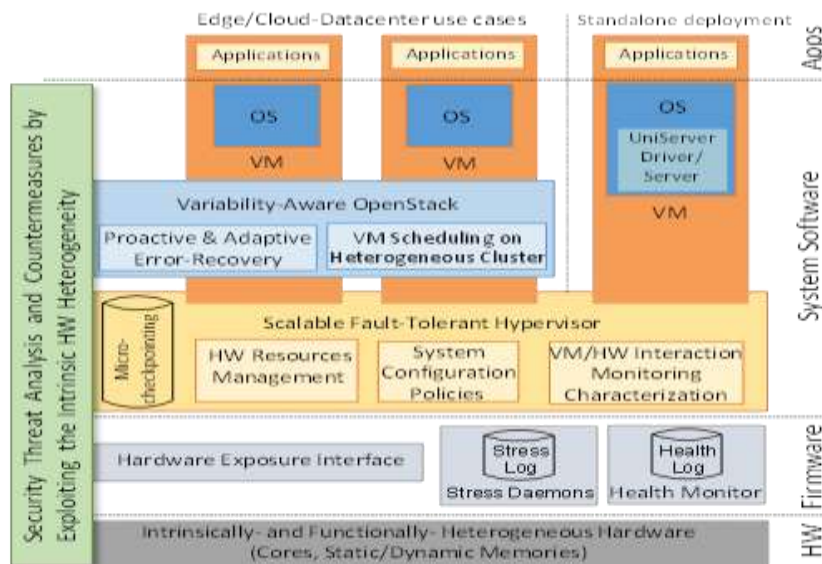


Figure 1: Detailed Uniserver Layering

The UniServer paradigm addresses a wide range of use cases, ranging from edge-based deployments in remote locations close to the end users to deployments in cloud data-centers. To facilitate such diverse use

cases, the UniServer platform must be equipped with a complete software stack that can efficiently manage compute and storage resources by offering easy installation, migration and replication of tasks, either at the node or server-rack level. To this end, state-of-the-art software packages for virtualization (Hypervisor) and resource management (OpenStack) are being adopted. Such packages, apart from managing the virtual machines (VMs) at the node level (Hypervisor) and the resources at a rack/data-center level (OpenStack), are also enhanced for optimizing the system operation by fine tuning the extended V-F-R points of the computing and storage resources, while at the same time tolerating and managing the increased failure rate due to operation beyond nominal points.

In particular, the hypervisor aims at limiting the effects of the potential faults to higher software layers by reconfiguring the system to operate within safe margins and isolating problematic processing and memory resources that affect the VMs. This is achieved by utilizing the information delivered by the HealthLog/StressLog/Predictor daemons and developing a new set of configuration properties. The optimization of operations at the EOP in UniServer is guided by the system requirements of the end-user for each VM, which are typically communicated to the Cloud provider through Service Level Agreements (SLAs). These workload-specific requirements reflect the key metrics of interest based on which OpenStack manages the nodes of the cloud/data-centre. Note that in the UniServer paradigm an additional metric, namely node reliability, is added to the traditional metrics of interest which are node availability, utilization and energy usage. Altogether, these metrics will help in system energy and performance optimization.

At the lowest level, the APM X-Gene processor family powers the system. As described in [31], the latest X-Gene 3 processor that will be used as the final chassis of UniServer compares favourably to the current state-of-the-art with respect to its competitors.

	APM X-Gene3	Intel Xeon E5-2680v4	Cavium ThunderX CP	Intel Xeon D-1540
CPU Core	Potenza++	Broadwell	Thunder	Broadwell
Max Sockets	1S	2S	2S	1S
Cores, Threads	32C/32T	14C/28T	48C/48T	8C/16T
Max Integer IPC	4 IPC	5 IPC	2 IPC	5 IPC
Base Clock Speed	3.0 GHz	2.4 GHz	2.5 GHz	2.0 GHz
Total Cache	32MB	35MB	16MB	12MB
Memory Bandwidth	170.7GB/sec	76.8GB/sec	76.8GB/sec	32.1GB/sec
Memory Capacity	1,024GB	1,536GB	512GB	128GB
PCIe Bandwidth	82.8GB/sec	78.9GB/sec	31.5GB/sec	55.3GB/sec
SPECint_rate	>500	527	350	238
Power (TDP)	125W	120W	95W	45W
IC Process	16nm FF+	14nm HYP	28nm HKMG	14nm HP

Table 3: Processor comparison

The X-Gene3 processor delivers a high per-thread performance peak as well as being power efficient [32]. The processor was designed with the data center in mind, with a full range of reliability, availability, and serviceability features. However, in the UniServer environment when subjected to intentional undervolting and frequency scaling, the processor may operate outside of its design parameters. This requires a number of trade-offs between operational correctness, power consumption, and performance. The metrics at each layer of the system help to track the tradeoffs, allowing the application to meet its targeted SLA.

3. Approach

The goal of the UniServer project is to improve the energy efficiency of ARM-based micro-servers, for both edge and cloud computing deployments. Reducing the power envelope or extracting more work from the existing power envelope is a key concern in today's environment [1]. By reducing the power consumption, edge-based computing infrastructures and data centers can reduce their TCO without compromising their service quality.

3.1 Overview

The primary aim of UniServer is to limit the pessimistic guardbands that are being adopted today in commercial servers to account for the expected performance degradation of transistors and potential functionality failures due to the increased transistor variability in nanometer technologies. While such guardbands (in terms of increased voltage margins and circuit redundancy) have successfully ensured reliable operation up to date, their effectiveness in detecting and correcting and accounting for all possible errors is being doubted by researchers, as geometries and supply voltages are being scaled down and circuits become more vulnerable to failures.

As an example, the voltage guardbands (i.e. voltage up-scaling) currently adopted against a variety of issues are already significant, ranging from at least 5% for die-to-die variations to 20% for addressing voltage droops (as was also mentioned in D3.2). Indicatively, recent measurements in ARM processors indicated more than 30% timing and voltage margins in 28nm. Recent studies have also revealed that the refresh-rate adopted in dynamic memories (DRAMs) is extremely pessimistic, and can be relaxed beyond the conservative 64ms that is currently adopted in DDR3 technologies (see D3.3). Such voltage, frequency and refresh-rate margins are becoming more prominent with the use of more cores per chip, the increased voltage droops, reliability issues at low voltages (V_{min}), and core to core variations.

The main target of UniServer is to limit such guardbands in commodity ARM based servers, thus allowing operation at reduced voltage and refresh-rate in CPUs and memories. This improves the energy efficiency, which is one of the primary concerns today due to the emergence of issues such as Dark Silicon. By limiting such guardbands the CPUs could also be operated at a higher frequency, thus allowing acceleration of program execution.

However, by limiting or even eliminating those voltage, timing and refresh-rate guardbands we are putting at risk the correct functionality of the CPUs and DRAMs due to the potential failures that may occur at lower voltages and dynamically changing operating/environmental conditions (e.g., temperature). Such timing and memory failures may disrupt the operation of the server and/or directly affect the expected QoS, which can be quantified in terms of throughput and quality-of-results (e.g. in terms of Bit-Error-Rate). As a consequence such failures will affect the SLAs in terms of availability, latency, accuracy, and throughput as agreed at the higher level between the service user and the service provider.

Therefore, in UniServer we are substantially enhancing all layers of the system stack with new capabilities for revealing the safe operating limits for each core and DIMM, while enhancing the Hypervisor and OpenStack layers for optimizing the energy end performance by managing the operation at the revealed operating limits while ensuring the desired availability, latency, accuracy, and throughput.

3.2 Metrics

Evaluating the success of UniServer requires the development of specialized metrics, since UniServer approaches scaling in an entirely new fashion as described above. Application metrics have traditionally been based around concepts such as defects counts, performance metrics such as throughput or latency, etc. While such metrics can be considered as successful given their popularity, their limitations are well known [29, 30]. Existing metrics do not accurately describe whether a system or component is "successful"

D7.1: First Report on Metrics of Success Against State of the Art

when subjected to intentional frequency changes and undervolting. For instance, using a metric such as performance would not take into consideration the savings in energy, but also the increased probability of errors, resulting from lowering margins. Thus, a more specific set of metrics that take into account these aspects must be established, such as the number of memory errors per time quanta, or the voltage threshold for undetected memory errors [2].

The metrics for UniServer can be divided as follows:

Level	Description
Hardware	Measure, track, and quantify the operating parameters of the processor and memory.
System Software	Based on the hardware metrics, adjust the deployment and operating parameters of the OS, Hypervisor, and Cloud deployment.
Application	The metrics related to each application that quantify the success or failure of execution such as availability, latency, accuracy, and throughput.

Table 4: Levels of Metrics

The rationale for the different layers is that the notion of “successful operation” is different at each layer. Because of UniServers exploitation of the operating margins of processors and memory, errors will occur. These errors are tracked and monitored, but do not necessarily cause a failure in the overall operation of the system. For example, an undetected memory error at the hardware layer would normally be cause for a system shutdown; however, if the upper layers have indicated their ability to tolerate such errors, then the operation of the system can continue. Alternatively, if an upper layer cannot tolerate such errors, it might decide to change the operating parameters of a lower layer, in an attempt to reduce the error rate and meet the expectations of the upper layers.

Total Cost of Ownership (TCO) is a key optimization metric for the design of a system because it includes all the other metrics that will be investigated in the UniServer project. TCO can capture the implications of many parameters including performance, power and mean time to failure [33], [34]. More specifically TCO allows a quick exploration of the implications of reliability design decisions. On the contrary, the lack of a TCO model most likely leads to reliability techniques and design decisions with local scope, which consequently miss the global implications (such as cost and energy) of such decisions.

The UniServer project will provide an end-to-end TCO tool. The end-to-end TCO, which is a new concept, aims to estimate the entire eco-system lifetime capital and operating expenses including the costs of data source nodes (i.e. IoT nodes), the servers used for edge and cloud nodes and the internet network latency. The tool will identify the potential benefits of a deployment that uses the UniServer framework as compared to one without it. The tool will also be capable of assessing the TCO of future large-scale deployments and will also help investigate cost-benefit analysis for alternative UniServer configurations.

Such a tool will help to access the trade-offs between energy, throughput and the costs being paid for addressing any potential failures by operating at marginal voltage and refresh rate. For example, by reducing the voltage the failure probability will increase. Hardware ECC along with mechanisms at the Hypervisor and OpenStack such as VM or node restart will then need to be activated, which will cost energy and throughput. Considering such costs along with the energy savings under the different configurations and the various applications will help determine the preferred EOP under different conditions.

To provide an example, the trade-offs of such an analysis can be seen in the following table in case of two different mitigation mechanisms. In terms of hardware ECC, ChipkillSC is the weaker ECC mechanism than ChipkillDC [33].









	ChipkillSC	ChipkillDC
Reliability	 Cannot detect all the errors in 2 devices Corrects all the errors in 1 device	 Detect all the errors in 2 devices Corrects all the errors in 1 device
Bandwidth	 Access one DIMM	 Access two DIMMs
Latency	 Codeword in two bursts	 Codeword in one burst
Power	 Access one DIMM	 Access two DIMMs

Table 5: Trade-off of two DRAM ECC mechanisms (ChipkillDC and ChipkillSC) [33]

As the table shows, there are several trade-offs according to reliability, bandwidth, latency and power. So, it is not trivial to choose the appropriate ECC mechanism between the two. To accomplish this, an incorporation of all the parameters as well as the application characteristics in the TCO model is needed.

The following figure shows the TCO of ChipkillDC and ChipkillSC when running two applications with different characteristics. The first application is the Web Search, a highly constrained application with strict QoS requirements and a working set size of 6GB. The second application, referred to as Floreon+, is a flood prediction application, and is compute intensive, multithreaded with high QoS constraints but a small working set (44KB). As can be seen, according to the TCO, a different ECC mechanism is preferable for each application: ChipkillDC for Web Search and ChipkillSC for Floreon+.

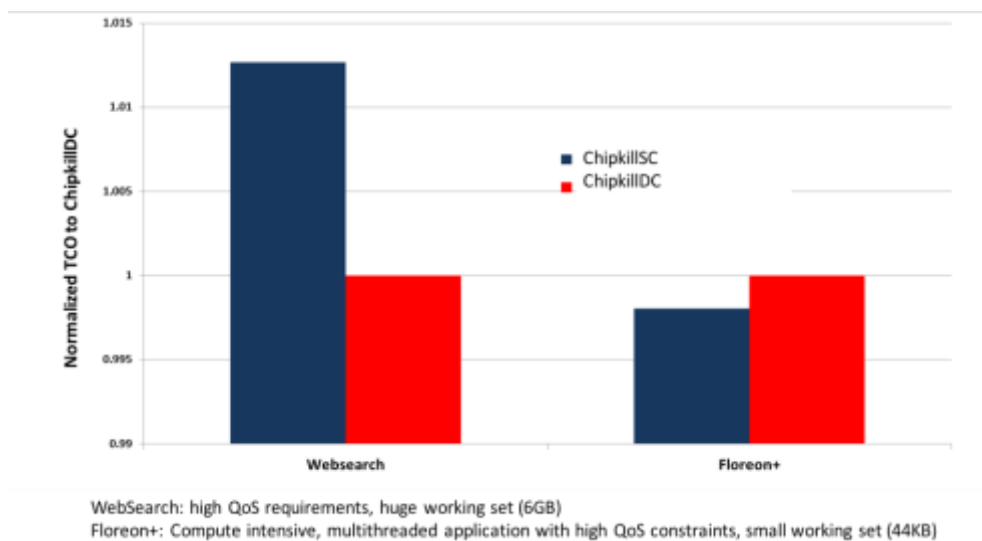


Figure 2: Normalized TCO to ChipkillDC for WebSearch and Floreon+ applications [33]

Similarly, trade-offs will be explored taking into account the three applications of the Uniserver project and also including some other parameters such as the network latency. At the end, we will decide in which location to run a specific application (edge or cloud) and also find the V-F-R that provides the maximum savings in the TCO.

The success of the project ideas depends on showing energy end/or throughput improvements when the server is operated at the marginal V-F-R levels. A *conceptual* example of the trade-offs between energy savings and mitigation costs of the massive failures can be seen in the following figure.

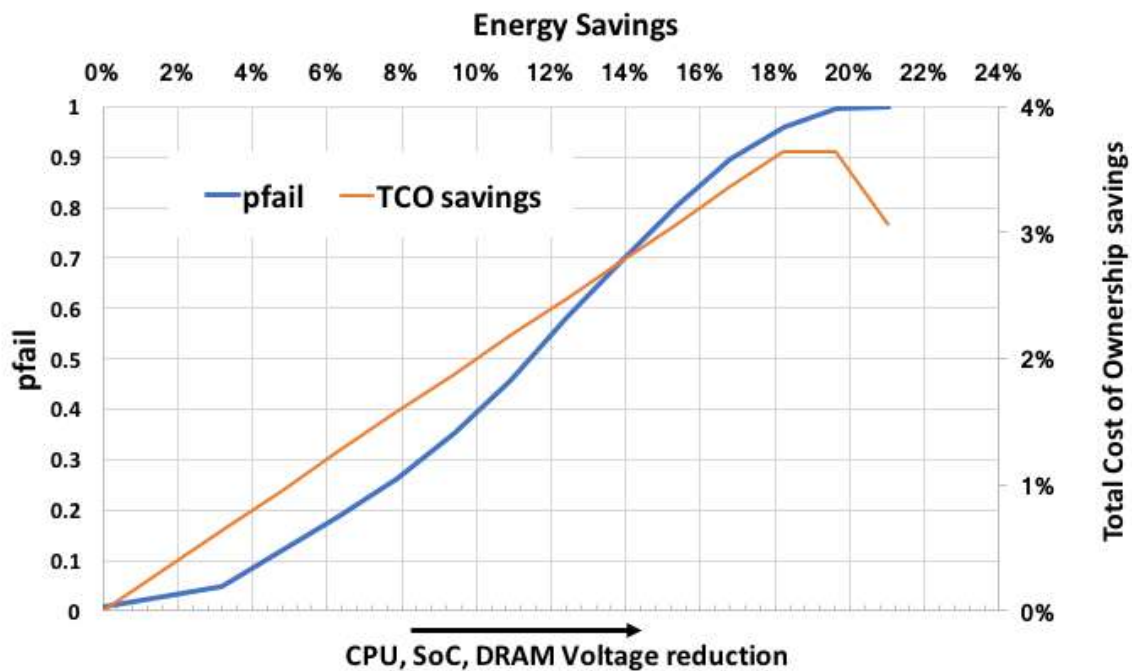


Figure 3: Pfail, energy, and TCO estimation

This graph shows a “success” scenario with the correlation between the reduction of CPU, SoC and DRAM voltage (x-down axis), the probability to have a failure (y-left axis), application or system crash, and the energy (x-up axis) and TCO savings (y-right axis). The graph shows the savings in the TCO (y-right axis) as the voltage in different components is reduced. As the graph shows, the TCO savings are increasing and reach the 3.5% point related to the nominal voltage setting (first point of the voltage setting in the x-down axis). After that point, TCO savings are decreasing due to the higher probability of failure. Higher probability of failure causes more over-provisioning in order to not violate the availability requirement of an application.

As the figure shows, even though energy savings are increasing monotonically, TCO does not have the same trend. This happens due to the extra cold spares that are needed due to the availability requirement violation of a specific application. Cold spares are server or component (DRAM, processor) modules needed for replacement when active servers or components fail. The fault rate of a server can be determined by the MTTF of its components and the Mean Time to Repair (MTTR). The cold spares are not active and they are only used when a server is down due to a failure. These spares are only accounted for in the TCO with their capital expenses and not their operational expenses (such as power). So, the point where the TCO savings are decreasing is the point that the cost of all the number of cold spares that are needed is overlapping the energy savings at a specific voltage setting.

When adopting the Uniserver framework and reducing the CPU or/and SoC voltage or/and DRAM voltage while keeping a stable workload and CPU frequency, the probability of failure is moving closer to one, whereas energy savings are increasing. The lower the voltage, the less energy consumption; however, there may be more reliability issues. So, it depends also on the application requirements to find the best voltage setting in which to run the application.

Each application has its own requirements related to performance, power and reliability/availability, as described in Section 6. TCO can encapsulate all the metrics such as reliability issues (pfail) and energy in each voltage setting and can show the best voltage setting in which to run a specific application.

The above graph highlights the complexities of analyzing the TCO factors and the need for a TCO tool to find the best configuration to run an application. It also shows that the lack of such tool can lead to wrong choice of a voltage setting.

3.3 Related State of the Art Work

Recently, the goal for improving the energy efficiency of microprocessors by reducing their supply voltage has become a main concern of many scientific studies. Gopireddy et al. [19] presents a core that is designed for voltage scalability and that can work in high-performance mode at nominal V_{dd} , and in a very energy-efficient mode at low V_{dd} . The authors evaluate their proposal in terms of energy consumption and energy-delay product, which is the consumed energy multiplied with the delay between the input and the output. The authors assume that there are no uncorrectable errors and thus metrics taking into account the reliability are not considered.

In order to help testing the effects of voltage droops, Ketkar et al. [21], and Kim et al. [22,23] propose testing frameworks to maximize voltage droops in single core and multicore chips in order to investigate their worst case behavior due to the generated voltage noise effects. These papers focus on reliably under realistic noises and worst-case scenarios, but do not study the effects of such generated noises and thus they do not propose metrics to evaluate such effects.

Studies of Gupta et al. [24] and Reddi et al. [15] focus on the prediction of critical parts of benchmarks, in which large voltage noise glitches are likely to occur, leading to system malfunctions. By predicting these parts, the system can operate at tighter frequency and voltage margins, and adapt to a more conservative configuration whenever a potential error-prone part is executed. The notion of “error probability” is adopted in the UniServer project in order to determine the reliability of a system given a set of configuration parameters and workloads. Overall, we observe that although there is a large interest in developing energy efficient systems and also techniques to deal with voltage noise effectively under DVFS settings, there is little work on exploiting unreliable configurations that can tolerate a certain degree of errors.

At the system software level, the studies in [7, 11] introduce models that involve several QoS metrics, used in the SLAs between the consumers and providers through a complete environment for cloud applications. The resulting models focus on maximizing the provider profit while taking into consideration SLA violation penalties. These works miss the opportunity of further extending the infrastructure provider profit by exploiting either VFS or voltage overscaling. UniServer adopts both voltage and frequency overscaling and proposes the respective models to estimate (and subsequently reduce) the cost to the cloud and edge operator. Moreover, UniServer can take into account the potential SLA violation penalties and highlight the optimal operating point (tradeoff between energy reduction and QoS) that increases operator profit.

Dynamic voltage and frequency scaling (DVFS) in cloud deployments has the potential to reduce power. The authors in [12] adopt dynamic voltage scaling to minimize energy consumption on high-performance computing systems. The model described in [25] proposes techniques that lower the supply voltage below nominal values, but also introduces mechanisms that allow the system to recover from the resulting timing errors and return to a stable state. Both these works try to maximize the energy gains but do not consider the contractual QoS agreement (SLA) that the cloud/edge provider must adhere to. Eventually, due to timing errors, the profit of energy saving can be reduced by the SLA violation penalties, consumption and increase profit [10]. DeMarco [26] and separately Weinberg [27] have amplified this in the sense of software system as a product delivering value for the customer. Furthermore, ISO/IEC 20000, the international service management standard for IT systems, gives strong weight to the concept of the voice of the customer.

During the last few years, datacenters have increased in numbers, size and uses [35]. In an effort to reduce costs and meet specific needs several configurations have come to market including micro-servers for I/O intensive workloads [39], [40] and blade-servers for space and power constrained environments. With these different systems comes a set of design decisions which affect the total cost of ownership. Consequently, to deliver a cost-efficient datacenter, designers should be aware of how different decisions affect the Total Cost of Ownership (TCO) of a datacenter. Several TCO models have been proposed for guiding datacenters design [33], [34], [36], [37] and [38] that mainly depend on the following factors:

TCO Type	Description
Datacenter Infrastructure Cost	The cost of acquisition of the datacenter building (real estate and development of building) and the power distribution and cooling equipment acquisition cost
Server Cost Expenses	The cost of acquiring the servers, which depreciates within 3-4 years
Intra-Datacenter Networking Equipment Cost	The cost of acquiring the networking equipment
Datacenter Operating Expenses	The cost of electricity for servers, networking equipment and cooling
Maintenance and Staff Expenses	The cost for repairs and the salaries of the personnel

Table 6: TCO factors

While the goal of datacenter designers is to minimize the TCO, another major concern is the energy consumption and the resulting environmental substantial fraction of the TCO.

4. Hardware Metrics

The foundations of the metric hierarchy are the metrics and parameters for components such as the CPU, caches, and DRAM. Through undervolting, frequency scaling, and altering the DIMM refresh rates, the operating parameters of the components are altered.

4.1 Error Types

The Hardware Exposure Interface (HEI) as described in *D4.1 “Hardware Exposure Interface (HEI) and Error Handlers Specification”* provides the ability to monitor the processor and notify other components of the system about the occurrence of an error. In the undervolting and frequency experiments [1, 2], several types of errors have been observed.

Error Type	Description
Silent data corruption	A “flip” of one or more bits without notification from the HEI, meaning the error was undetected by the processor.
Correctable error	A notification from the HEI that the processor has discovered and corrected an error.
Uncorrectable error	A notification from the HEI that the processor has discovered an error that cannot be corrected. Two bitflips in an ECC protected data word would be described as uncorrectable.
Application crash/hang	An application crash or hang with or without notification from the HEI about a processor detected error.
System crash/hang	A system crash or hang with or without notification from the HEI about a processor detected error.

Table 7: Error types due to hardware malfunction

4.2 Severity Function

As a means to aggregate the criticality or severity of such errors while also taking into account how likely it is for them to occur, we introduce [2] the Severity Function S_v , where v is the voltage, as follows:

$$S_v = W_{SDC} \cdot \frac{SDC}{N} + W_{CE} \cdot \frac{CE}{N} + W_{UE} \cdot \frac{UE}{N} + W_{AC} \cdot \frac{AC}{N} + W_{SC} \cdot \frac{SC}{N}$$

The parameters SDC , CE , UE , AC and SC denote the occurrence of SDC, CE, UE, AC and SC errors observed in N runs for the same voltage level V . Parameters W_{SDC} , W_{CE} , W_{UE} , W_{AC} and W_{SC} represent “weights” that can be flexibly set to characterize the severity of each type of error.

For each individual run, SDC, CE, UE, AC and SC take binary values, i.e., they can be either 1 (the respective error occurred, one or more times in this run) or 0 (such an error did not occur in this run). Note that more than one type of errors may occur in the same run, e.g., one can observe SDCs together with CEs or even UEs, in which case SDC and CE and CE would all be equal to 1. However, there are certain combinations that cannot occur, e.g., it is not possible in the same run to observe a crash (AC/SC) and an SDC. This is because SDCs are determined and logged only after running the application to completion.

As mentioned above, the weights can be set in an open way to reflect the criticality/severity of such errors. For example, one can set $W_{CE}=1$, $W_{UE}=2$, $W_{SDC}=4$, $W_{AC}=8$ and $W_{SC}=16$ (where CEs are considered far less

critical than SCs which are considered to be the most severe errors). Then, for a single run, S_v can take a value from 0 to 19, since the system can crash ($SC=1$) after having detected and logged one or more correctable and uncorrectable errors ($CE=1$ and $UE=1$). Other runs can result in different values between 0 and 19; but note that not all values are feasible, e.g., it is not possible for S_v to be 12 because it is not possible to experience both an AC and to detect/log an SDC.

Aggregating over N runs, S_v gives the average of the individual single runs, reflecting the criticality/severity of these different types of types of errors while also taking into account the likelihood of their appearance when operating the system at voltage level V . If a particular type of behavior appears much more often than others, this will be reflected in the value of S_v . For example, assuming the weights given above, if when under-volting the system at 900mV for a large number of N the value of S_v is close to 5, this means that the program at this voltage level most likely leads to SDCs while also having some CEs at the same time ($SDC/N \approx 1$ and $CE/N \approx 1$, while $UE/N \approx 0$ and $AC/N \approx 0$ and $SC/N \approx 0$).

It is important to stress that both the error occurrences and weights can be strongly application-specific, leading to significantly different values for different applications. As example, when operating the system at 900mV, and after performing a large number of runs N for two different applications A_x and A_y , one could get $S_v(A_x) = 12.5$ and $S_v(A_y) = 4.5$. What this effectively means is that A_y is much more tolerant to under-volting at 900mV compared to A_x . Notably, a similar observation can be made when comparing two different cores, say C_x and C_y , for the same application A . In this case, $S_v(C_x) = 12.5$ and $S_v(C_y) = 4.5$ would mean that core C_y is more robust to under-volting than core C_x when the system is running application A .

The severity metric, its calculation, and the corresponding viruses are described in detail in *D3.3, "First Analysis of On-Chip Caches and Dynamic Memories"*, and in [2].

4.3 Long-Term Wear

One aspect of the UniServer approach that requires additional time to explore is the long-term effect of voltage and frequency scaling on the components. This level of investigation cannot be carried out until the parts have been stressed for a sufficiently long period of time. Upon the delivery of the latest X-Gene3 based systems, the existing X-Gene2 systems which are being used as the current chassis of the UniServer platform will be converted into long-term test beds. In that way, it will be possible to re-run the experiments to determine how aging affects the severity metric.

4.4 Graceful Performance Degradation

Performance can be affected due to disabling of resources that do not operate reliably due to aging or operating at small margins. It is useful, therefore, to assess the trade-off between performance and TCO of future servers supporting deconfiguration of processor resources such as cache sections to facilitate more energy efficient operation.

To achieve this, all the characterization results for the CPUs and DRAMs under different configuration parameters will be fed to the developed TCO tool. The tool will also take into consideration any energy and performance penalty due to the recovery mechanisms (i.e. ECC, task and node restart) that may be needed due to the potentially increased detected error count when the node will be operating at scaled voltages and refresh rates. Finding the operating regions where the system SLAs can be met without any frequent disruptions and the energy savings due to scaled V-F-R are larger than the incurred fault mitigation costs will be a major target of the TCO tool. It will also be a predictor that will determine the success of the project ideas. Note that in order to achieve the above we must ensure that the system degrades gracefully, rather than abruptly, with massive failures due to the potentially increased failures under scaled V-F-R. This will help to keep the mitigation cost low and the error count controllable.

5. System Software Metrics

The system software stack of the UniServer framework comprises the operating system, hypervisor and VM/Cloud management software (OpenStack). Since the system software stack must be able to handle the instability inherent in undervolting and frequency scaling, UniServer focuses on enhancing the hypervisor and OpenStack components to tolerate and even compensate for this instability.

The relationship between the hypervisor and OpenStack is shown in the following diagram:

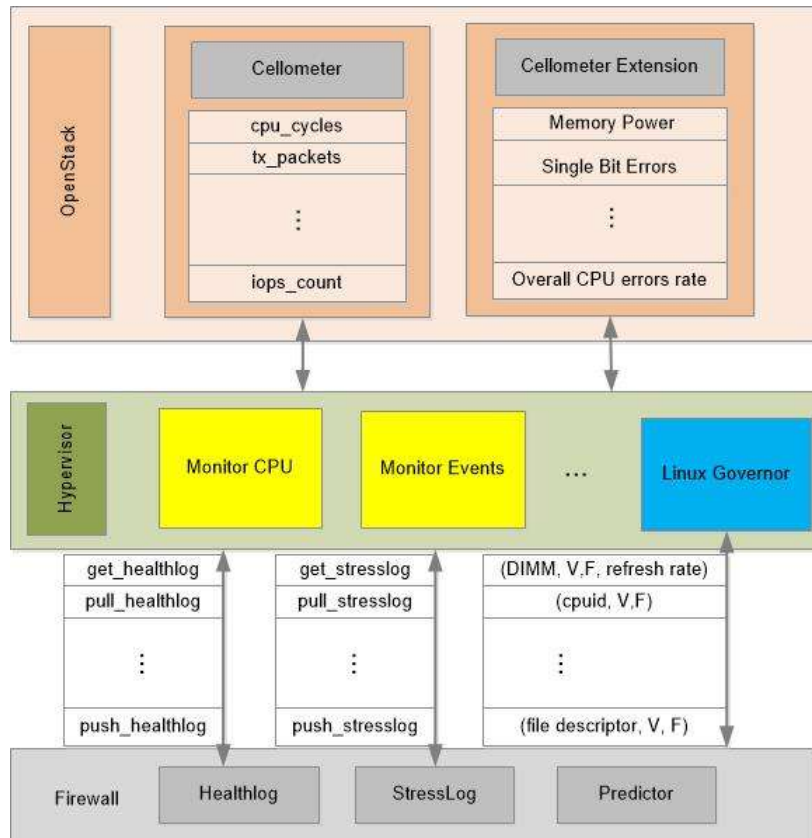


Figure 4: Hypervisor and OpenStack relationship

5.1 Hypervisor

At the hypervisor layer, hardware metrics related mainly to power and performance (such as CPU and memory utilization, cache misses etc.) are already monitored by an existing application programming interface (API). In UniServer, we are enhancing this API to enable the collection and monitoring of information related to reliability. This is done through UniServer-specific daemons, which will be collecting all detected CPU and DRAM errors as described in *D4.2 “HealthLog Specifications and Interface”* and *D4.3 “StressLog Specifications and Interface”*.

The Hypervisor relies on the HealthLog and other components to track the current reliability and stability of the system. For example, if a core or set of cores is deemed unreliable, then key processes can be migrated to reliable cores. Memory can also be partitioned into reliable and unreliable domains. The hypervisor itself will be restricted to running on reliable cores, and allocating memory only from the reliable domain.

The application and the operating environment inherently drive the metrics that will influence the choice of operating parameters. These metrics are described in detail in [1] and illustrate the relationship between expected energy consumption at the Hypervisor level and the SLA/TCO of OpenStack.

5.2 OpenStack

OpenStack provides the framework for managing the Virtual Machines (VMs) and the underlying platforms they run on. WP6 aims at improving the management of virtual machines VMs that are running on nodes with heterogeneous power, and performance settings. This heterogeneity in power and performance adds an additional parameter, i.e. reliability, that should be incorporated in managing the running VMs. This requires developing techniques for improving the resilience of cloud infrastructure and the running workloads. The exposed extended margins are available to OpenStack through the libvirt [3] interfaces. The detailed enhancements and specialized resource management policies for OpenStack are described in *D6.1: OpenStack support for UniServer*.

In the context of UniServer project, the following are the metrics of that can be used to evaluate the success of the cloud infrastructure in a data center:

5.2.1 Energy Efficiency

One of the objectives of the UniServer project is to improve the energy efficiency of the data centers. To this end, the enhanced OpenStack for UniServer gathers fine grain resource utilization measurements of the workloads (i.e., VMs) running in the cloud data center, and resource utilization measurements of the physical resource of the data center. Based on the observed resource utilization measurements, OpenStack may ask the underlying system software stack, through the enhanced libvirt interface for UniServer, to adjust the voltage and frequency settings of the physical server which is hosting a particular VM such that the performance requirements of the running application are not violated.

Therefore, one of the metrics of success is to see how much energy efficiency is achieved by the cloud service provider using the enhanced OpenStack for UniServer as compared to the standard DVFS settings, where the operating system governor is set to default *ondemand* setting. We also plan to compare the achieved energy efficiency with other governor settings, i.e., *conservative*, *powersave*, and *performance*, but we anticipate that switching to these settings may not be the correct choice because of the diverse nature of workloads running in a cloud datacenter. We expect to achieve more energy efficiency in UniServer using the enhanced system software and cloud middleware stack as compared to the standard DFVS settings.

5.2.2 Proactive fault tolerance

In order to meet the objectives of the UniServer project in terms of improving the availability of the running workloads in the face of system failures, techniques will be developed in the enhanced OpenStack for UniServer to proactively avoid failures. One of the metrics of success will be to see the effectiveness of the developed techniques and compare them with the ability of standard OpenStack in avoiding failures. In this context, we will consider those failures that may result in the unavailability or disruption of the running workload.

The current state of the art OpenStack does not employ any specific technique to proactively avoid failures that may lead to the unavailability of the running workload. We plan to measure the number of failures that may be avoided using proactive fault tolerance techniques as compared to the total number of faults that are experienced by the system. The accuracy and the effectiveness of proactive fault tolerance techniques would be determined by the number of failures that are prevented by the enhanced OpenStack as compared to the standard OpenStack framework.

5.2.3 Reactive fault tolerance

UniServer aims at developing proactive techniques at the cloud middleware level to handle system-level failures. However, it is anticipated that not all failures would be handled by the proactive fault tolerance techniques. To handle those failures, the enhanced OpenStack for UniServer will employ reactive techniques to mitigate the impact of a failure and to make the workload available again as soon as possible, e.g., by restarting the workload on a different node. One of the metrics of success is to measure the effectiveness of

D7.1: First Report on Metrics of Success Against State of the Art

the reactive techniques, and to see how mean-time-to-recover (MTTR) a workload improves in the enhanced OpenStack as compared to the standard open-source OpenStack available to the cloud providers.

A general technique used in the data centers to improve the availability of workloads during failures is to replicate the workload by running duplicate VMs; however, this approach may not be effective for UniServer where the aim is to not only improve the energy efficiency but also to reduce the total cost of ownership (TCO). Therefore, we can measure the effectiveness of the reactive fault tolerance technique without running replicated VMs, and then compare the MTTR of the enhanced versus vanilla OpenStack framework.

6. Application

The metrics at the application level are the final arbiter of success or failure of the UniServer approach. There are three target applications, listed in the table below.

Application	Provider
Wireless jamming detection	WSE
Trade confirmation	Meritorious
Social CRM and Social TV	SPA

Table 8: UniServer target applications

These applications mostly share a common set of quality of service (QoS) metrics, which are in turn used to express the individual application requirements in the form of respective Service Level Agreements (SLAs):

Metric	Description
Availability	Percentage of uptime
Latency	Delay in providing results
Accuracy	Precision of output
Throughput	Data Rate, application specific. Not all applications need have a Data Rate metric

Table 9: Application Metrics.

Of course, each application has different QoS requirements, which depends on its nature. For instance, one application may have very high accuracy requirements but more relaxed availability requirements compared to another one. The QoS requirements of each of the UniServer applications are discussed in more detail in the following.

6.1 WSE Wireless Jammer Detector

The SDR Jammer Detector is a wireless security component of the Denial of Service (DoS) Sensing solution. The detector identifies jamming signal threats that aim to generate DoS attacks by interfering with wireless network communications. For this purpose, this solution implements a smart sensor that detects threats and communicates detection events to visualization software so that users can easily identify the type of jamming signal generating the attack.

The following figure summarizes the overall architecture of the jamming detection solution

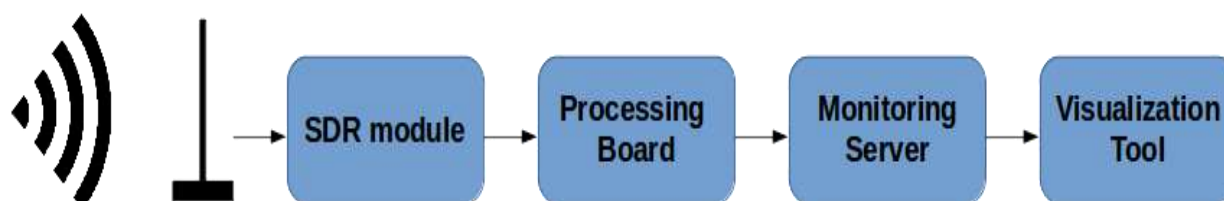


Figure 5: DoS Sensing Architecture

The Jammer Detector consists of a sensor that has an antenna connected to an SDR module, which digitalizes the radio spectrum to a binary stream and transmits this to a Processing Board (running the

D7.1: First Report on Metrics of Success Against State of the Art

Jammer Detector application). The board processes the incoming data and applies filters and algorithms to match the signals found to 4 types of well-known jamming signals. If one of those is detected, this incidence is communicated to the Monitoring Server, running on a separate machine. Finally, the Visualization tool periodically gets detection events from the Monitoring Server so that it can present them in real time.

The QoS requirements of the Jammer Detector are summarized in the table below, and are described in more detail in the following subsections.

Metric	Description
Latency	Each decision in 100ms
Availability	High: 99%. Moderate: 75%. Low: 50%
Accuracy	97%; 3 undetected SDCs in 100 decisions

Table 10: WSE jammer detector metrics

6.1.1 Availability

Availability depends on the type of installation where the solution is deployed. There are cases where the availability will be high, moderate or low.

High availability: Some of the use cases will demand that detection is available 99% of the time. It is important to emphasize that a jamming attack should be held through time in order to be effective and really compromise the proper behavior of a wireless network. Thus, the attack is considered a threat if it continuously detected for at least 5 seconds. If the solution is not available for 5 seconds every 500 or 600 seconds (10 minutes), this is acceptable, as a worst case scenario will give us a 10 second interval to detect a threat. Thus, the maximum time that the solution can afford to be unavailable is 5 seconds. In this case, the application can afford 1% unavailability. One of the use cases where this type of solution is needed will be a hospital, for example.

Moderate availability: There are cases, such as smart construction monitoring, where the availability can be lower due to the criticality of data and the amount of data transferred, which would be a few bytes per hour or less. In this case, the reliable transmission of packets is important, but it is not critical if a few of them are lost. In this type of scenario, a 75% availability can be accepted. Actually, in a smarter solution, the jammer detector would only be needed if the system detects that there is packet loss, which should be quite infrequent.

Low availability: In cases where connections are not critical or there are not many connections to monitor but the solution may still be interesting, only 50% availability or even less could be an option. This would be the case in shopping malls or train stations, where the connections would be performed by clients trying to access the network for non-critical purposes, such as recreational activities.

6.1.2 Latency

It is important that jamming detection is fast, i.e., for the application to have a relatively small latency. The latency will always depend on the width of the band that the application is analyzing. The maximum latency that has been experienced on a regular processing board was **100ms** to make a decision (jammer present or not) on a **5 MHz** band. Thus, the time needed to analyze the whole WiFi band (2.4 GHz to 2.5 GHz) is about 2 seconds. Clearly, if the processing board is powerful enough, this latency can be lowered but this will result in a more expensive solution.

6.1.3 Accuracy

The accuracy requirement varies depending on the conditions where the jammer detector is used, the power of the jammers used for the attacks, etc. In case the environment is very noisy and there are several other signals being transferred on the analyzed band, detection accuracy will decrease. For this, there are several parameters that need to be adjusted to the environment where the detector will be installed.

The calculated accuracy is obtained through simulation where the environment is friendly and the noise is controlled. In this case, the false positive or true negative rates are almost 3%, giving a best case scenario of 97% accuracy.

6.2 Meritorious Trade Confirmation

European Markets Infrastructure Regulation ([EMIR](#)) came into force on 16 August 2012, and introduced requirements aimed at improving the transparency of Over-The-Counter (OTC) derivatives markets and to reduce the risks associated with those markets. In order to achieve this, EMIR requires that OTC derivatives meeting certain requirements be subject to the clearing obligation and for all OTC derivatives that are not centrally cleared that risk mitigation techniques apply. In addition, all derivatives transactions need to be reported to Trading Repositories (TRs).

The reporting of a derivative transaction involves any daily modifications/updates of the transaction until the termination of the derivative contract. This requires the processing and validation of a large amount of information and handling sensitive client's data.

CME ETR: Chicago Mercantile Exchange EMIR Trade Repository
 REGIS-TR: Iberclear and Clearstream European Trade Repository

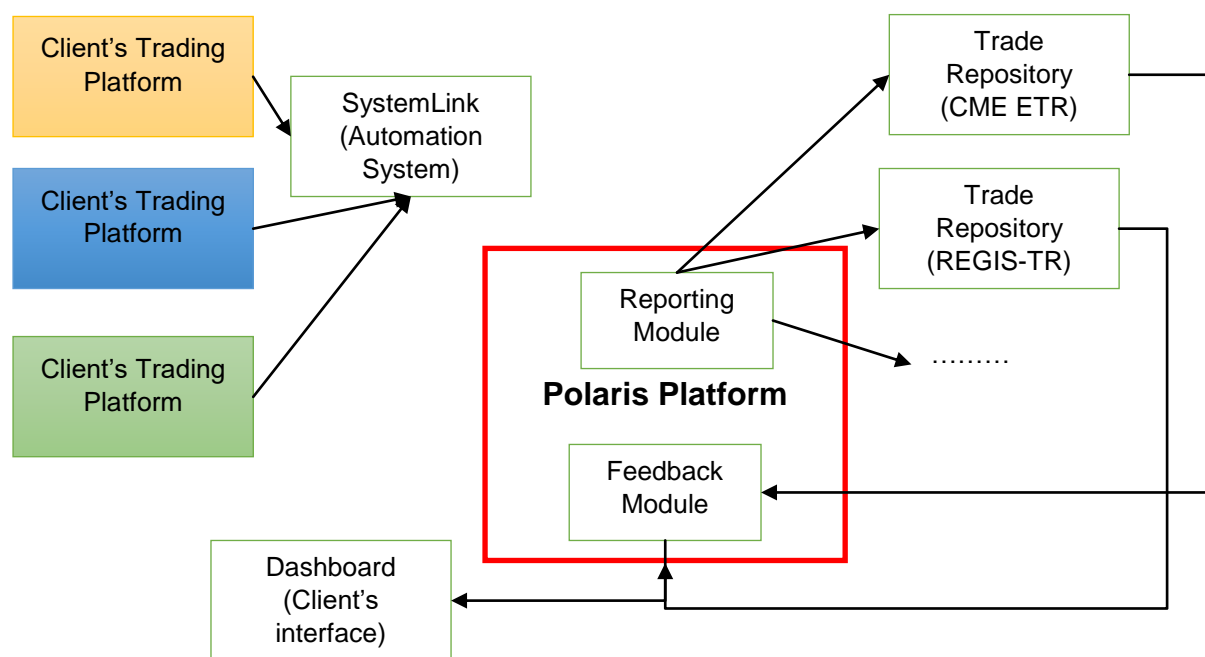


Figure 6: Trade confirmation architecture

The goal is to reduce the TCO via UniServer, which also includes power reduction. In this application, there is no difference if the cost comes from capital cost, operational cost, or maintenance.

The QoS requirements of the Meritorious trade confirmation application are summarized in the table below:

Metric	Description
Latency	3000 transactions/min
Availability	99%
Accuracy	100%

Table 11: Meritorious trade confirmation metrics

Trades must be reported on the day T+1 deadline. That is not related to the market close but when the trade was executed on any market. By execution, what is meant is the time a transaction request by an investor was accepted by the Investment Firm (the client). For example, any transactions executed from 00:00:00 to 23:59:59 on 04/07/2017 (day T) must be reported by the end of the next day that is 23:59:59 of 05/07/2017 (day T+1). Trades are coming from markets from all over the world and thus can be trades executed any time of the day on working days.

6.3 SPA: SocialCRM/SocialTV

The two SPA apps (Social CRM and Social TV) share the same three architectural components:

1. The social network server component which runs on a large data center
2. The social analytics components that run on client's premises and which process data from the server regarding specific subscribed events
3. The web app, which also runs on client's premises and that connects to a database with the results of the social analytics component to allow their exploration

The QoS requirements of each component are discussed next.

6.3.1 The social network server component

The social network server uses synthetically generated data, which simulates a real social network in time. This dataset is bulk loaded into the server to set it into a working state, and spans several years of social network activity. Additionally, the synthetic datasets also contain "update" streams, which basically consist of the update activity in the social network (new messages, new friends, etc.) in the subsequent months in the simulation that have not been bulk loaded, but are fed into the driver which is responsible for issuing these updates at runtime. Also, parameters are provided to the driver to perform read queries.

At the beginning of the execution and before starting issuing queries, the driver creates a workload of operations. In other words, it creates a mix of read and update queries given the information provided by the dataset (update streams + parameters), where each query has an associated "issue time". That is, if execution starts at point t , each query has an associated timestamp with value $t+i$, which corresponds to that time this query will be issued by the driver to the server in the future. This time $t+i$ is different for each query.

Last but not least, queries have dependencies. For instance, if at some point a user A is added to the social network, we cannot add a friendship between A and another user B until the server confirms that user A information has been committed into the database. The driver creates a pool of worker threads. Each thread picks the next query to execute in the timeline if and only if its dependencies have been resolved already and the issue time of the queries is equal or greater than the current time. This process continues until all queries have been executed.

D7.1: First Report on Metrics of Success Against State of the Art

In an ideal situation, the driver will execute the queries at their corresponding issue time. This can be seen as users issuing queries at different moments in time, and getting the answers correctly. But what if the server starts lagging at some point (queries are taking too much to execute)? That means that at some point, some queries will be delayed from their issue time. Thus, each query will have an "actual issue time", which is the time this query was actually picked by a worker thread and issued to the server given the performance provided by the server.

Since the social network server is meant to be interactive, the benchmark defines a minimum SLA in order to consider an execution valid. This SLA is defined as follows: For >95% of the queries, (actual issue time - issue time) <= 2000 ms. In other words, if this SLA is achieved, the server is assumed to be capable of correctly serving the workload. Additionally, the driver reports the latency per query time. Although the benchmark does not define a minimum latency for each query, we are free to do so. For instance, we could extend the SLA to also require each query type to have a 95% percentile of latency below 1 second.

Metric	Description
Latency	<= 2 seconds
Availability	99,9999%
Accuracy	100%
Throughput	95% queries <= 2 seconds from their issue time. Queries/second fixed by the scale factor used

Table 12 - Social Network Server SLA metrics

All the SLA metrics of the Social Network Server component are automatically measured and reported by the LDDB Social Network Benchmark Driver.

6.3.2 The social analytics component

The analytics component, which is installed in client's premises, receives periodic batches of updates on those specific events tracked. For instance, imagine a client A wants to track the activity related to Donald Trump. The social network server will buffer this activity and from time to time or when a buffer is filled it will send that to the client (the social analytics component). From time to time, the client will process the data and combine (or not) with the previous data, and update its snapshot of information with the most recent data received from the server. The type of processing is like finding influencers, detecting communities, etc.

Ideally, the client would like to have always the most up-to-date data pre-processed and available for interactive inspection by the webapp. However, this might not be feasible depending on the cost of the analysis, which in turn depends on the amount of data to process. But many applications do not require the most recent data but can work with delays that can go from hours to a few days.

Under these circumstances we have the opportunity to heavily exploit the TCO. The user should know the relation between the costs at different configurations and how fast (and thus recent) the analysis is, and let him choose the best option for her (also considering the rate the data from the server is arriving).

Note that for this component, the server does not require a high availability; it just must guarantee that the data will be processed and can be retrieved by the web app when this is available. The web app can be in another server, which will just retrieve (copy) the latest database and replace the previous one. In other words, the server could be in suspended mode 90% of the time if this fits the needs of the user.

Metric	Description
Latency	N/A
Availability	The time required to perform the computation. Flexible, depends on the TCO.
Accuracy	Flexible, depends on the TCO
Throughput	1 execution per 24h

Table 13 - SLA for the Social Analytics Component

In order to measure the SLA of the Social Analytics Component, we will track the execution time and see if it meets the 24 hour deadline required by the application.

6.3.3 The web app component

For the web app, we must guarantee that this is interactive and available. This is similar to the social network server. We could measure the latency of the requests and the user experience. Assuming a workday of 8 hours, an SLA of 99% would guarantee a non-availability of at most 5 minutes per day. This seems reasonable.

Metric	Description
Latency	<=2 seconds
Availability	99%
Accuracy	100%
Throughput	10 queries / second

Table 14: SLA for the Web App

The SLA of the Web App will be measured by means of well-known tools such as jmeter (jmeter.apache.org), which allows testing web applications and assess their responsivity and throughput.

7. Conclusions and Future Work

This deliverable has described the metrics used by UniServer at the hardware, system, and application levels in the initial period. The detailed description of each metric has been provided, as well as how the metrics are used at each level. The relationships between the levels in the system have been explained, as well as the interactions between levels. The importance of the metrics with respect to application deployment and TCO has been addressed with specific references to three different applications.

Future work will focus on fine-tuning the interactions between the layers in the system, and on the development and deployment of the end-to-end TCO tool with the main aim of discovering the optimum V-F-R points where the obtained energy savings are larger than the incurred correction/recovery costs due to the potentially increased errors. To achieve this, the work in all layers of the UniServer framework, conducted in the individual WPs (WP3-WP6), focuses on discovering operating regions where performance scales gracefully and massive failures are avoided. Note that the metrics explained in this report are the ones that have been set in the initial period and they will be updated if needed in the future deliverables.

With the deployment of the upcoming X-Gene3 based systems, the existing X-Gene2 systems can focus on the effects of long-term wear and graceful degradation of resources. Lastly, the security implications of operating outside the processor margins will be also explored trying to safeguard the UniServer platform against any threats that may become possible due to operation at reduced V-F-R.

8. References

- [1] C. Kalogirou, P. Koutsovasilis, M. Maroudas, C.D Antonopoulos, “Edge and Cloud Provider Cost Minimization by Exploiting Extended Voltage and Frequency Margins”, Department of Electrical and Computer Engineering, University of Thessaly
- [2] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, S. Das, “Title: Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs”, IEEE/ACM International Symposium on Microarchitecture, 2017
- [3] Libvirt, “libvirt – The virtualization API”, <http://libvirt.org/index.html>
- [4] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. IEEE Journal of Solid-State Circuits, 41(4):792–804, 2006.
- [5] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. Razorii: In situ error detection and correction for pvt and ser tolerance. IEEE Journal of Solid-State Circuits, 44(1):32–48, 2009.
- [6] D. Dib, N. Parlavantzas, and C. Morin. Sla-based profit optimization in cloud bursting paas. In Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, pages 141–150. IEEE, 2014.
- [7] D. Dib, N. Parlavantzas, and C. Morin. Sla-based profit optimization in cloud bursting paas. In 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 141–150, May 2014.
- [8] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. ACM SIGCOMM computer communication review, 39(1):68–73, 2008.
- [9] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on, pages 38–43, Aug 2007.
- [10] K. H. Kim, A. Beloglazov, and R. Buyya. Power-aware provisioning of cloud resources for real-time services. In Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science, page 1. ACM, 2009.
- [11] J. Kosinski, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, and P. Niedziela. Definition and evaluation of penalty functions in sla management framework. In Networking and Services, 2008. ICNS 2008. Fourth International Conference on, pages 176–181. IEEE, 2008.
- [12] Y. C. Lee and A. Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In Cluster Computing and the Grid, 2009. CCGRID’09. 9th IEEE/ACM International Symposium on, pages 92–99. IEEE, 2009.
- [13] K. Parasyris, V. Vassiliadis, C. D. Antonopoulos, S. Lalis, and N. Bellas. Significance-aware program execution on unreliable hardware. ACM Trans. Archit. Code Optim., 14(2):12:1–12:25, Apr. 2017.
- [14] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson. Investigating the interplay between energy efficiency and resilience in high performance computing. In Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, pages 786–796. IEEE, 2015.
- [15] E. Le Sueur, and G. Heiser, “Dynamic voltage and frequency scaling: the laws of diminishing returns”, in International Conference on Power Aware Computing and Systems (HotPOWER), 2010.
- [16] V. J. Reddi, M. S. Gupta, G. H. Holloway, G.-Y. Wei, M. D. Smith, and D. M. Brooks, “Voltage emergency prediction: Using signatures to reduce operating margins”, in International Conference on High-Performance Computer Architecture (HPCA), 2009, pages 18–29.
- [17] H. Duwe, X. Jian, D. Petrisko, and R. Kumar, “Rescuing uncorrectable fault patterns in on-chip memories through error pattern transformation”, in International Symposium on Computer Architecture (ISCA), 2016, pages 634–644.
- [18] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, “Towards a software approach to mitigate voltage emergencies”, in International Symposium on Low Power Electronics and Design (ISPLED), 2007, pages 123-128.

- [19] B. Gopireddy, C. Song, J. Torellas, N. S. Kim, A. Agrawal, and A. Mishra, “ScalCore: Designing a core for voltage scalability”, in International Conference on High-Performance Computer Architecture (HPCA), 2016, pages 681–693.
- [20] P. N. Whatmough, S. Das, Z. Hadjilambrou, D. M. Bull, “An all-digital power-delivery monitor for analysis of a 28nm dual-core ARM Cortex-A57 cluster”, IEEE International Solid-State Circuits Conference (ISSCC), 2015, pages 262-264.
- [21] M. Ketkar, and E. Chiprout, “A microarchitecture-based framework for pre- and post-silicon power delivery analysis”, in International Symposium on Microarchitecture (MICRO), 2009, pages 179–188.
- [22] Y. Kim, and L. K. John, “Automated di/dt stressmark generation for microprocessor power delivery networks”, in International Symposium on Low Power Electronics and Design (ISPLED), 2011, pages 253-258.
- [23] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, “AUDIT: Stress Testing the Automatic Way”, in International Symposium on Microarchitecture (MICRO), 2012, pages 212–223.
- [24] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wai, and D. M. Brooks, “An event-guided approach to reducing voltage noise in processors”, in Design, Automation & Test in Europe Conference (DATE), 2009, pages 160-165.
- [25] A. Cavelan, Y. Robert, H. Sun, and F. Vivien. Voltage overscaling algorithms for energy-efficient workflow computations with timing errors. In Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale, pages 27–34. ACM, 2015.
- [26] DeMarco, T., Management Can Make Quality (Im)possible, Cutter IT Summit, Boston, April 1999
- [27] Weinberg, Gerald M. (1992), Quality Software Management: Volume 1, Systems Thinking, New York, NY: Dorset House Publishing, p. 7
- [28] Kai Yang, The Voice of the Customer: Capture and Analysis, pub: McGraw Hill (new York) 2008, ISBN 0-07-146544-8
- [29] Norman, E. Fenton and Martin Neil, Software metrics: successes, failures and new directions, The Journal of Systems and Software 47 (1999) 149-157
- [30] Tom Gilb, Software Metrics, pub: Winthrop Computer Systems, 1976, ISBN-13: 978-0876268551
- [31] L. Gwennap, Performance Arms X-Gene 3 for Cloud, <http://www.linleygroup.com/uploads/x-gene-3-for-cloud.pdf>
- [32] L. Gwennap, X-Gene 3 Challenges Xeon E5, http://www.linleygroup.com/cms_builder/uploads/x-gene-3-white-paper-final.pdf
- [33] Panagiota Nikolaou, Yiannakis Sazeides, Lorena Ndreu, and Marios Kleanthous. Modeling the implications of dram failures and protection techniques on datacenter tco. In Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, pages 572–584, New York, NY, USA, 2015. ACM
- [34] D. Hardy, M. Kleanthous, I. Sideris, A.G. Saidi, E. Ozer, and Y. Sazeides. An analytical framework for estimating tco and exploring data center design space. In International Symposium on Performance Analysis of Systems and Software, pages 54–63, 2013
- [35] Cisco, “Cisco global cloud index: Forecast and methodology, 20112016,” 2012
- [36] C. Patel and A. Shah, “Cost model for planning, development and operation of a data center,” HP Laboratories Palo Alto, Tech. Rep., 2005
- [37] J. Karidis, J. E. Moreira, and J. Moreno, “True value: assessing and optimizing the cost of computing at the data center level,” in Proceedings of the 6th ACM conference on Computing frontiers, ser. CF '09. New York, NY, USA: ACM, 2009, pp. 185–192.
- [38] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, “Making scheduling “cool”: temperature-aware workload placement in data centers,” in Proceedings of the annual conference on USENIX Annual Technical Conference, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 5–5.
- [39] Calxeda, “<http://www.calxeda.com/>.”
- [40] seamicro, “<http://www.seamicro.com/>.”

[END OF DOCUMENT]