uniserver

D7.4 - Evaluation of the Prototype and Comparison to State-of-the-Art in terms of Energy, Security and Determined Metrics of Success

Contract number	688540
Project website	http://www.UniServer2020.eu
Contractual deadline	Project Month 24 (M24): 31th January 2018
Actual Delivery Date	31th January 2018
Dissemination level	Public
Report Version	1.0
Main Authors	Alejandro Lampropulos (WSE)
Contributors	ManolisKaliorakis (UoA), George Papadimitriou (UoA), Athanasios Chatzidimitriou (UoA), Dimitris Gizopoulos (UoA), Arnau Prat (SPA), PanagiotaNikolau (UCY), MariosKleanthous (MER), Philip Hodgers (QUB), Panos Koutsovasilis (UTH), Christos D. Antonopoulos (UTH), GeorgiosKarakonstantis (QUB), Konstantinos Tovletoglou (QUB)
Reviewers	Arnau Prat (SPA), Denis Guilhot (WSE)
Keywords	Microserver, Demonstrator, Energy, Reliability, Evaluation

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.

© 2018. 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.

Change Log

Version	Description of change
0.1	Initial draft – Outline
0.2	Inputs UOA, MER, QUB, UCY, SPA
0.3	Update WSE and UOA
0.4	Inclusion ToC, Tables, Figures
0.5	Initial write up of State of Art section (UTH)
0.6	Security. Initial write up of Prototype section. State of Art DRAM
0.7	Introduction and conclusions
0.8	Formatting, figures, tables
0.9	Reviewing
1.0	Final Formatting. Version released for submission to EC

Table of contents

Index of figures

Index of tables

Executive summary

This document describes the first prototype and its software modules. The platform comparison to the state of the art is also provided, as well as the metrics of success. The applications included in the use cases are described and the power savings characterised for Worldsensing's Jammer Detector. Finally, we analyse the security aspects related to the platform and include the end-to-end TCO analysis.

1. Introduction

In previous deliverables we have discussed the Uniserver platform in detail, in terms of hardware and software stack. Building the complete system is a highly complex process that is currently taking place. Some of its modules are already completely developed, tested and integrated, while others are still under development. Nevertheless, at this point we have a very stable first prototype, which can provide the first quantitative results, showing actual advantages of using the Uniserver platform with real life use cases.
In this document, we will firstly describe how the first prototype is composed and which software modules are part of it. We also provide the platform comparison to the state of the art and metrics of success. Then we explain the applications that will be utilized as use cases in order to test the prototype's advantages and will find the power savings that we get for one of them (Worldsensing's Jammer Detector). Finally, we analyse the security aspects related to the platform and include the end-to-end TCO analysis.

2. Description of the first prototype

The UniServer prototype is implemented on the APM X-Gene 2 and through its capabilities, we investigate the limits of a state-of-the-art microprocessor architecture beyond the nominal conditions.

2.1. UniServer Hardware Prototype

Figure 1 shows the block diagram of all the components of X-Gene 2. The X-Gene 2 Processor Complex consists of four Processor Modules (PMDs). Each PMD contains two high-performance X-Gene 2 cores, each of which has its own 32 KB L1 I-cache, 32 KB L1 D-cache, and Floating-Point Unit (FPU). The pair of X-Gene 2 cores in a PMD shares a 256 KB L2 cache unit which interfaces to Central Switch (CSW) interconnect. All four PMDs share an L3 cache (8 MB), which is also attached to the CSW. The X-Gene 2 Processor Complex features include:

- Eight X-Gene 2 cores operating at up to 2.4 GHz; each core contains:
  - an FPU / SIMD Unit,
  - 32 KB L1 Data Cache 8-way set-associative, write-through to L2 cache, parity-protected,
  - 32 KB L1 Instruction Cache 8-way set-associative, parity-protected.
- 256 KB L2 cache per pair of processors inclusive of L1 write-through data caches, 32-way set associative, ECC-protected. Since L3 is essentially a victim cache
- Shared 8 MB L3 cache (attached to CSW) protected with SECDED ECC
- Hardware cache coherency
- Cache snooping and invalidation of I/O accesses
- Up to four DDR3 Memory Controller Units (MCU)
- Supports up-to DDR3-1866 (933 MHz DDR interface @ 1.5 V, 1.35 V)
- One channel of DDR3 memory per MCU
- Up to 128 GB of DRAM memory per MCU
- Memory protection with 8-bit SECDED ECC

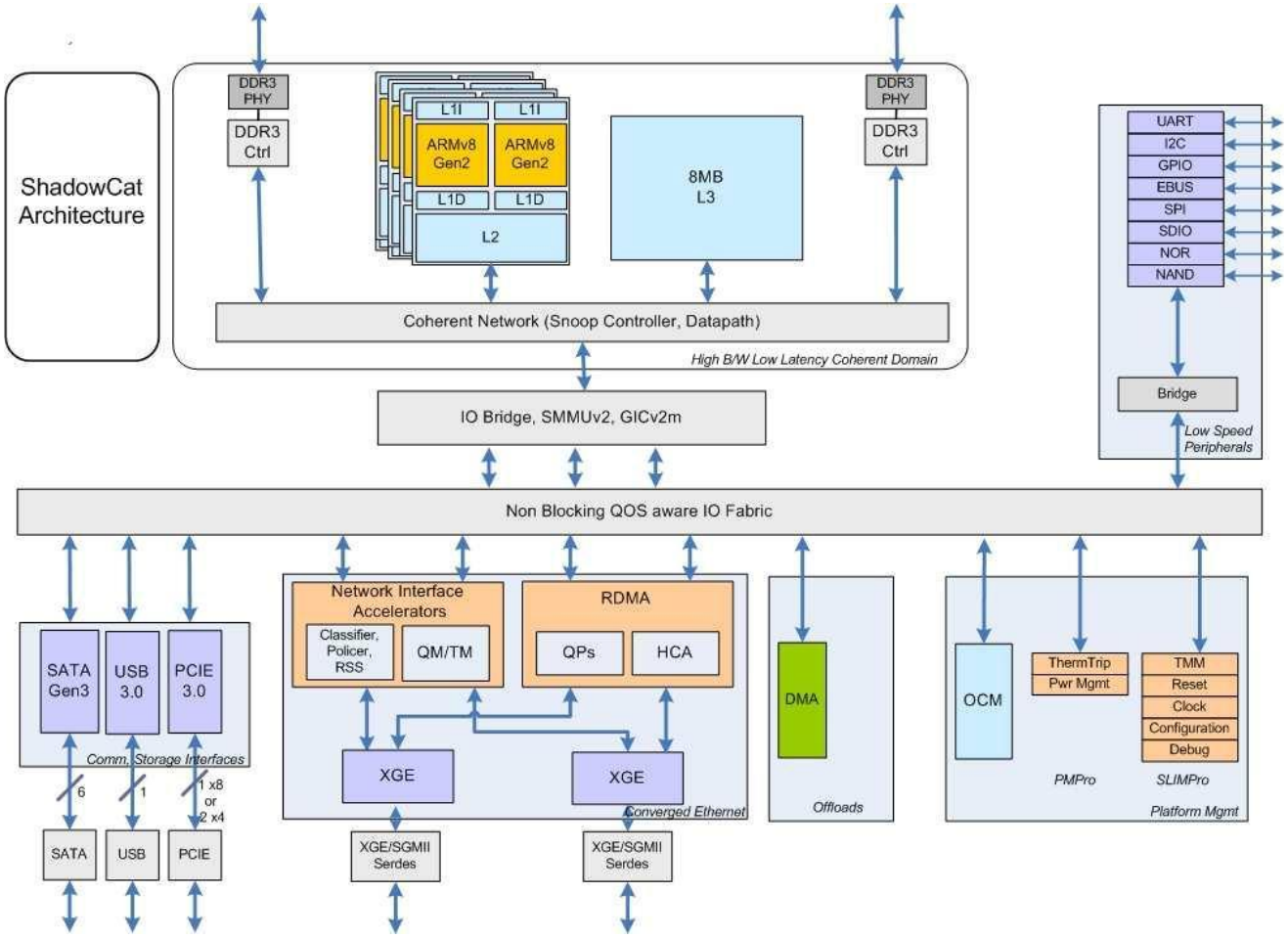

Figure 1: APM X-Gene 2 Block Diagram

The X-Gene 2 architecture offers high-end processing performance and capabilities. For example, the X-Gene 2 subsystem features the Power Management processor (PMpro) and Scalable Lightweight Intelligent Management processor (SLIMpro) to enable breakthrough flexibility in power management, resiliency, and end-to-end security for a wide range of applications. The PMpro, a 32-bit dedicated processor provides advanced power management capabilities such as multiple power planes and clock gating, thermal protection circuits, Advanced Configuration Power Interface (ACPI) power management states and external power throttling support. The SLIMpro, a 32-bit dedicated processor, monitors system sensors, configure system attributes (e.g. regulate supply voltage, change DRAM refresh rate etc.) and access all error reporting infrastructure, using an integrated I2C controller as the instrumentation interface between the X-Gene 2 Cores and this dedicated processor. SLIMpro can be accessed by the system's running Linux Kernel.

Table 1 summarizes the most important architectural and microarchitectural parameters of the APM X-Gene 2 micro-server that is used in our study. More details can be found in D3.1 - Definition of the UniServer Board, and in official APM's X-Gene 2 User's Manuals and Hardware Specification Sheets.

Parameter	Configuration
ISA	ARMv8 (AArch64, AArch32, Thumb)
Pipeline	64-bit OoO (4-issue)
CPU	8 Cores, 2.4GHz
L1 Instruction Cache	32KB per core (Parity Protected) no-write allocate, PIPT
L1 Data Cache	32KB per core (Parity Protected) write-though, no-write allocate, PIPT
L2 cache	256KB per PMD (SECDED Protected) write-back, write allocate(no-write allocate for whole cache line), PIPT
L3 cache	8MB (SECDED Protected)
Dynamic Memory	4 memory channels, DDR3-1866, SECDED ECC

Table 1: Basic parameters of the prototype.

2.2. UniServer Cross-Layer System Architecture

Figure 2 shows the overall software system stack of the UniServer framework, which includes OpenStack, Libvirt, HealthLog, StressLog, hypervisor and Predictor. The HealthLog, StressLog and Predictor components are installed as distinct systemd services under each UniServer node.

Openstack is a widely used open source middleware for cloud setups that pairs well with the popular enterprise and open source technologies. Our extended version of OpenStack includes support for monitoring VMs and determining their dynamically changing characteristics and virtual resource utilization at a finer granularity than the existing state-of-the-art. In particular, the Ceilometer component of OpenStack gathers various data about the health and performance of the underlying physical and virtual resources in the data centre with the help of Hypervisor that gathers the requested information through the StressLog and HealthLog daemons.

OpenStack Nova has the responsibility to manage the resources of the physical hosts, to map and deploy incoming VMs to available nodes, and to maintain the 'good health' of all running VMs. In the context of UniServer, Nova is extended to configure nodes using more power-efficient voltage-frequency settings. This could involve running a node at the extended margins which could lead to increased probability of faults affecting the applications running inside the VMs. Therefore, the VM scheduler within Nova has been extended to consider the sensitivity of applications to system errors before mapping VMs to nodes running in different configurations as specified in D6.2.

We use Libvirt to bridge the gap between OpenStack and Hypervisor, which is a hypervisor-independent virtualization API. Libvirt is used to run VMs and transfers all information required by OpenStack from hypervisor. We extend the Libvirt interface, so OpenStack can request each node to operate at extended margins and collect reliability-related information, such as the number of ECC errors reported by hardware.

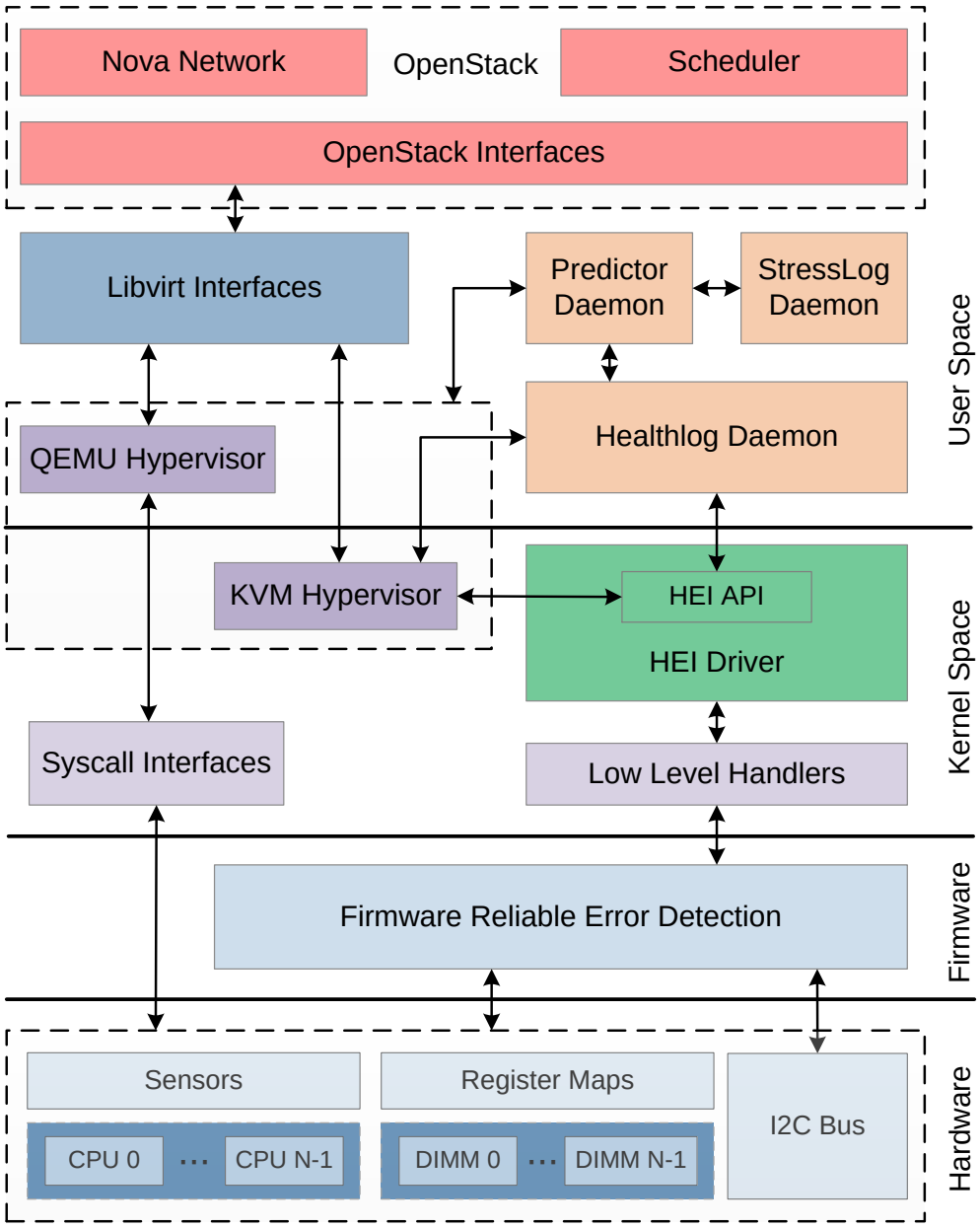

Figure 2: System software layers of the UniServer architecture.

The next layer in our architecture is the hypervisor which manages a node and interact with all system software and hardware layers within the node. It is responsible for creating an appropriate execution environment for Virtual Machines (VMs) by manipulating the power/performance/reliability trade-offs in an educated and safe manner. Specifically, it sets the system at a just-right configuration, which reduces the power footprint of each node by eliminating unnecessary hardware guard-bands, without introducing negative effects on the services running within the VMs. UniServer follows a hypervisor-based approach based on QEMU / KVM ported to the ARM64 architecture, to leverage all benefits of virtualization, such as easier deployment, administration, replication and migration, which are necessary for the targeted data centres at the Edge of Cloud.

To enable reliability facilities for hardware operated at the marginal operating hardware settings, hypervisor interacts with HealthLog and Predictor.

Operating outside the nominal hardware settings may introduce transient hardware errors during the system's lifetime. We have extended the error reporting capabilities of existing mechanisms with system configuration values, sensor readings and performance counters. We call this mechanism the HealthLog monitor that records runtime system metrics in the form of an information vector, stored in a system logfile. The HealthLog monitor interacts and exchanges information with higher system layers (e.g. Predictor and hypervisor). The HealthLog monitor provides two types of services: (a) Event-driven services, where it will collect information based on event occurrences in the system (e.g. errors) and (b) On-demand services, where the monitor will respond to requests from higher layers for specific information.

The StressLog monitor is spawned either periodically during a machine lifetime or is triggered by higher system layers (Predictor) in the case of anomalous machine behaviour. In this case, the machine being tested will be taken offline and as soon as the monitor receives the input stress target parameters from the higher system layers, it will initiate the stress test scenarios. The StressLog monitor also includes a workload suite, consisting of different benchmarks and kernels that either represent real-life applications or are hand-coded to stress specific components of the system. During a stress test, the HealthLog monitor executes in parallel to record system events (errors, system values, sensors and performance counters). The StressLog monitor takes the output of HealthLog and wraps all information into a vector to be passed to the higher layers.
Predictor is a software that utilizes online data and offline characterization data to predict the probability of failure for non-nominal voltage frequency states and DRAM refresh rates. Particularly, given availability constrains and desired number of cores and operating frequencies; predictor estimates most energy efficient voltages and DRAM refresh states that don't violate the given constraints set by OpenStack. In the UniServer framework, Predictor communicates with HealthLog to monitor a node collecting all metrics discussed in D5.1 and StressLog to run the stress-testing when it is required.
At the bottom level of our architecture there is a Hardware Exposure Interface (HEI) module which provides access to the APM firmware through I2C bus. The firmware exposes a set of hardware sensors and registers that allow software to closely monitor and control the processor. The HEI module provides a demand notification mechanism whereby software can register for a series of events that will be triggered when certain conditions are met. For example, HealthLog can invoke Low Level Handler (Deliverable D4.5) to get the logs when the specified event occurs.
Note that such a system stack can be deployed at classical centralized on the Cloud as well as new emerging de-centralized data centres at the Edge of the Internet. The same software stack without the OpenStack could also be deployed on small data centres where there is no need for specialized resource management modules as well as on individual nodes. The interfaces introduced in this deliverable will be used for any possible deployment within the centralized and de-centralized data centres as well as bare-metal deployments on individual nodes.

2.3. UniServer First Prototype System Architecture

It is important to mention that the first prototype used for the following tests and hardware characterization consists of the lower layers and the HEI. The applications run on Bare Metal and all test and characterization scripts utilize the HEI API directly in order to lower voltages, modify refresh rates, etc. On the other hand, the HealthLog is present and its deamon would be called whenever there are errors to be logged, which will allow system monitoring. Even when the rest of the modules that compose the stack are developed between 60% to 90% and the integration tests are progressing as expected, the integrated stack is not mature enough to be included in this very first prototype.
Figure 3 shows the first prototype stack. On top of it, the applications can be executed with no need of KVM, Openstack or Virtual machines.

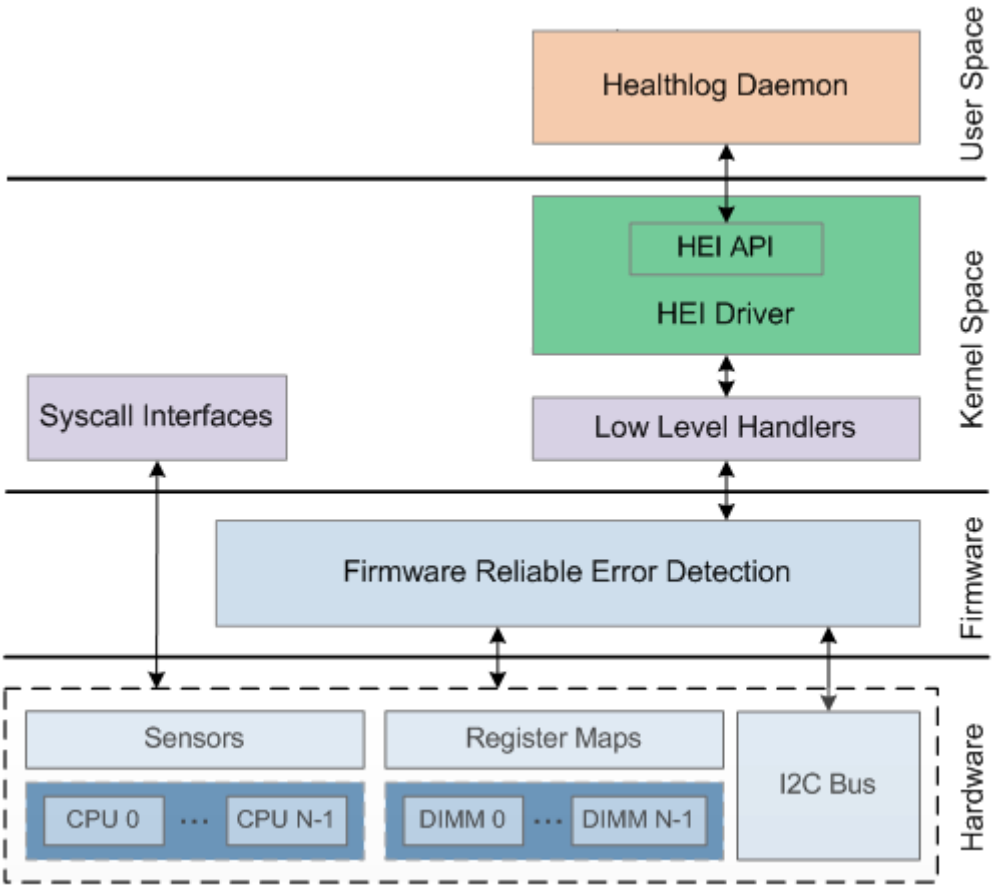

Figure 3: System software layers of the UniServer first prototype.

3. State of the art

In this section, we compare the Uniserver platform with the state of the art by analysing and contrasting different components, such as CPU, GPU, memory, etc.

Operation of commercially available CPUs at extended margins

Many research approaches have emerged in the last few years that relax conservative guard bands to improve energy efficiency. Prior work focusing on commercially available chips includes [1], [2], [3], [4].
The heuristics presented in [1] and [2] that dynamically reduce voltage margins while always preserving safe operation, are based on the error correction ECC hardware built on modern processors such as the server-class Intel Itanium 9560. A key observation there is that as V_{dd} is lowered, ECC detectable and correctable errors appear before detectable uncorrectable errors, SDCs and CPU crashes. The rate of ECC correctable errors is used as an indicator on how to adjust the Vdd voltage. In the Uniserver platform (XGene2), errors reported by the ECC mechanism appear very rarely. In addition, SDCs and crashes typically occur before any indication is received through ECC monitoring. Since the underlying hardware platform does not provide indications on the safety of operation under extended margins, such proxies cannot be applied to Uniserver.
Authors in [4]exploit the large margins available when only one core in a server-class 8-core Power7+ processor is utilized, turning under-utilized margin into power and performance benefits. Similar to the Uniserver characterization, this paper finds that as more cores are progressively utilized by multi-threaded applications, they cause larger IR drops across the chip and the benefits of an adaptive margin scheme begin to diminish. In the context of Uniserver, however, we plan to mitigate such effects through an educated selection of co-scheduled threads within each CPU.

Operation of commercially available GPUs at extended margins

The authors in [3]present a study of the voltage margins on several Kepler and Fermi GPUs. They first characterize the impact of process, temperature and voltage variations on Vmin, and then predict safe values of Vmin by deploying a linear regression and a neural network model. They show that high energy margins can be achieved by shaving conservative guard bands in modern GPUs. This approach is compatible with the approach followed by Uniserver, however the Uniserver ecosystem is vastly more complex than the GPU ecosystem. On the one hand, CPUs are architecturally much more complex than GPUs. On the other hand, the software stack on top of a CPU, and particularly the software stack in the context of real applications in the focus of Uniserver, is multi-layered, very deep with dependencies and interactions between different levels and the hardware.

Simulation-based CPU studies

In [5]the authors identify microarchitectural events (such as branch mispredictions and cache misses) that flush and stall the CPU pipeline and cause large voltage droops. They then propose a voltage emergency predictor that learns the signatures of such voltage emergencies and triggers a CPU throttling mechanism (e.g. increase voltage or decrease frequency) to prevent their recurrence. This work is based on CPU modelling on the Simplescalar simulator. In a subsequent paper, the authors measure run time voltage emergencies on a Core 2 Duo processor and attribute them to pipeline stalls and flushes [6]. Based on these experimental observations, they propose that a mechanism that uses more aggressive margins and a recovery (checkpoint-based) scheme may be better than a conservative static margin. Moreover, they propose a program scheduling mechanism so that the combined voltage droop is cancelled-out as much as possible. Based on these experimental observations, they perform a parametric study (with respect to the potential voltage margin elimination and potential cost of error recovery mechanisms) to identify areas in the parameter space in which the aggressive elimination of margins combined with a recovery mechanism to account for potential errors would be preferable over statically using conservative guard bands. Moreover, they propose a scheduler which tries to co-schedule threads in a way that cancels-out voltage droops. Common to Uniserver, these approaches identify the potential energy benefit of eliminating voltage margins to improve energy efficiency. However, contrary to Uniserver, they neither suggest nor experimentally evaluate on a real system mechanism that exploit these observations to actually improve the energy efficiency of complex multilayer software systems.

Circuit- / microarchitectural-level techniques

A number of proposed techniques present design approaches at the circuit or micro-architectural level that trade reliability for lower voltage, by attempting to reduce voltage down to the point that produces the highest tolerable error rate without causing catastrophic failures [7]. Several approaches propose methods which ensure correct operation of caches under undervolted conditions at the microarchitectural level [8], [9], [10]. Architectural techniques are presented to eliminate data corruption, and by extension enable cache operation at scaled voltage settings. The Razor processor is designed with built-in support for dynamic detection and correction of timing failures of the critical paths [11]. EVAL is a framework for dynamic adaptation of supply voltage, processor frequency and body bias using a machine learning algorithm [12]. Similar ideas include dynamic pipeline adaptation transferring the time slack of faster pipeline stages to the slower ones (ReCycle) [13], and using variable latency techniques to mitigate the impact of variations on the register file and execution units in a microprocessor [14]. Although all aforementioned approaches provide solid guidelines for future hardware designs, Uniserver targets commercially available chips and the proposed mechanisms are limited to the software, without requiring hardware modifications.

Operation of memories at extended margins

Recent works on DRAM have tried to save the DRAM power by limiting the refresh rate [15], [16], [17], [18] and/or lowering the supply voltage [19]. The majority of those rely on offline DRAM cell characterization, which is then be used either to group cells according to their retention time and adjust the refresh rate for each group separately [16], [17] or to design a tailored error mitigation scheme [15], [18]. Retention time is considered only based on an offline DRAM cell characterization that is being performed on FPGA setups without any OS, while using a number of micro-benchmarks consisting of data- and access- patterns that were found to stress the DRAM cells under scaled refresh or voltage [20], [19]. Authors in [19] exploit the variable access latency on DRAM cells when the voltage is lowered and introduce a new dynamic voltage and frequency scaling scheme on DRAM. In [21],the authors try to operate at aggressively scaled refresh by omitting the refresh entirely under specific circumstances. Khan at el [22] have indicted recently that the retention time vary dynamically and thus the majority of existing techniques may not be effective.

In UniServer, we advance the state of the art by characterizing the energy-reliability trade-offs under reduced refresh rate and voltage in DRAM chips within commercial servers with fully fledged OS. In our characterization we take into consideration a variety of factors that may affect such trade-offs aiming at identifying the most dominant ones and consequently tailoring the HW 'health' monitoring around those.

4. Metrics of success

The success of the UniServer platform depends on whether the application meets Service Level Agreement (SLA) while running under the UniServer constraints. SLAs define the kind and quality of service that the user asks and receives. Thus, application SLAs determine the application metrics. Except the application SLAs, there are also some other metrics that apply to the internal functioning of the UniServer platform and are not directly visible to the user.
According to this, UniServer metrics of success are categorized into three categories, related on the system level that is used. The three categories are: hardware metrics, system metrics and application metrics.

At the hardware level, the metrics are mainly related to the reliability of processor and DRAM components. Such metrics are: correctable errors, uncorrectable errors, application crashes, system crashes and silent data corruption errors. Some of the errors can be provided by the Hardware Exposure Interface, which has the ability to monitor the processor.
In addition, the severity metric is a metric that aggregates the criticality of different types of errors and 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, Service Level Agreement (SLA) violation penalties and Total Cost of Ownership (TCO). 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. The lack of a TCO model most likely leads to design decisions with local scope, which consequently miss the global implications (such as cost, energy and reliability) of such decisions.

Finally, application metrics are related to the application's 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.
The three UniServer applications mostly share a common set of metrics such as availability, latency, accuracy and throughput. Table 2 shows the three UniServer application metrics. For more details about the specific requirements of each application please consult D7.1.

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 2: UniServer Application Metrics

5. Applications

The best way to validate that the prototype meets the different metrics of success is to utilize challenging real-world applications. Verifying the proper behaviour of such software will provide significant evaluation of the

performance of the prototype and its limits. Next, we describe the three applications that are used for hardware characterization and validation purposes.

5.1. Jammer Detector

The Jammer detector (DoSSensing) tool from WorldSensing has been developed to respond to one of the most critical and impactful threats in critical infrastructures' wireless networks, the Denial of Service, which is commonly seen in the form of jamming at the lowest level, the physical layer. DoSSensing is a stand-alone solution that monitors the whole wireless spectrum to detect anomalies derived from a Denial of Service attack, leaving all the wireless devices useless. This approach is made without needing to be integrated internally in the wireless network, representing a great advantage from the competitors offering a wide and an easy-to-deploy solution for the most heterogeneous and challenging critical infrastructure wireless environments. To that end, the tool firstly performs a detailed analysis of the radio frequency spectrum, and later processes the acquired data to identify potential anomalies, giving rise to alarms and warning messages.

The main architecture of the solution is as follows:

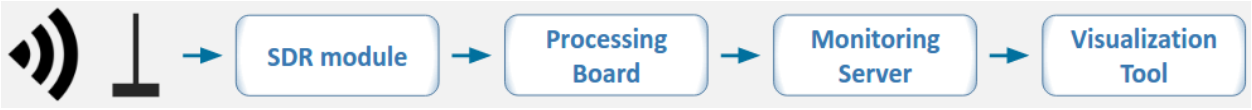

Figure 4: Main architecture of the solution

Basically, the DoSSensing solution consists of a sensor with an antenna connected to a SDR module which digitalizes the radio spectrum to a binary stream and transmits this to a Processing Board. The board will process in real time the incoming data and apply filters and algorithms to match the signals found to 4 types of well-known jamming signals: Pulsed Jammer, Wide Band Jammer, Continuous Wave Jammer and LFM Chirp Jammer. If one of those is detected, this incident will be communicated by the sensor to the Monitoring Server running on a separate machine. Finally, the Visualization tool will periodically get detection events from the Monitoring Server so that it can present them in real time.

Next figure shows how the system fits on an edge-based environment.

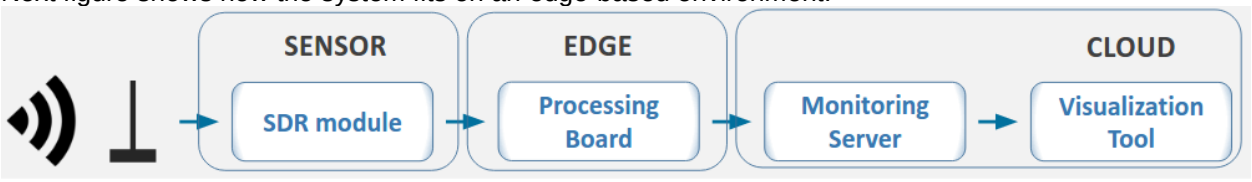

Figure 5: System's fit on an edge-based environment

Powerful edge devices such as XGene would allow the sensor to be simple and low cost, as it can concentrate the main processing of high amounts of radio frequency spectrum data on the edge. With such an approach, more than one instance of the processing application can be executed on the processing board (edge device) by connecting various sensors to it. The detection results can be transmitted to the cloud for storage and visualization.

The following figure shows the visualization tool in action. Filtered for Pulsed jammer attacks only, the tool is composed of a real time JNR (jammer to noise ratio) viewer, a frequency alert widget, an attack counter and a speedometer, which calculates the jammer detection speed (time between subsequent positive detections) in real time.

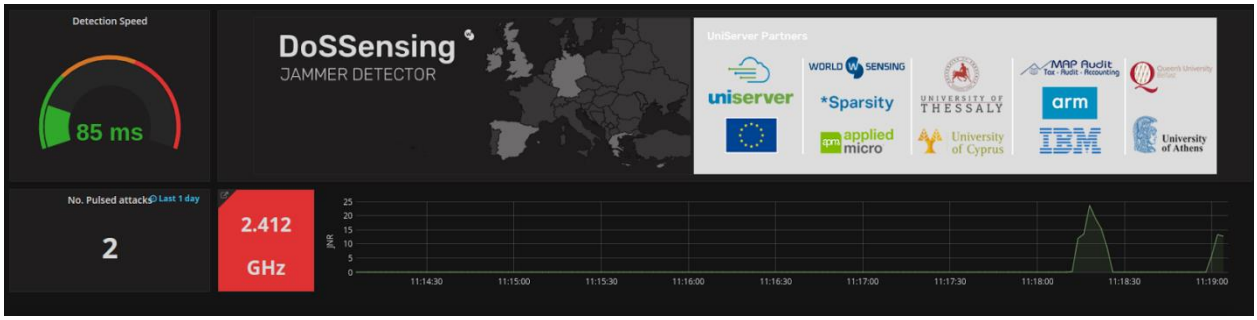

Figure 6: DoSSensing Jammer Detector GUI

5.1.1. Requirements

The DoS Sensing solution requirements will depend on the different use cases, i.e. client needs. Some of the basic requirements that could be used as a reference have been defined and are summarized in the table below, as well as described in detail in the following subsections.

Requirement	Description
Availability	High: 99%. Moderate: 75%. Low: 50%
Latency (5 MHz band)	Low: 100 ms per decision
Accuracy	97%; 3 undetected SDCs in 100 decisions
Data rate	Max: 305 Mbps. Min: 30.5 Mbps

Table 3: DoS Sensing solution basic requirements

5.1.1.1 Availability

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

5.1.1.1.1 High availability

Some of the most demanding use cases will require 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 affect the proper behaviour of a wireless network. Thus, the attack is considered a threat if it is continuously detected for at least 5 seconds. If the solution is unavailable 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.
This means that the maximum time that the solution can be unavailable is 5 seconds. In this case, we can afford 1% unavailability. One of the use cases where this type of solution is needed will be a hospital, for example.

5.1.1.1.2 Moderate availability

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

5.1.1.1.3 Low availability

The solution may still be interesting in cases where connections are not critical or there are not many connections to monitor, and availability of 50% or less could be optimum. 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.

5.1.1.2 Latency

The latency (in this case measured by the detection time) will always depend on the width of the band that we are analysing. The detection time that we have achieved on a regular Processing board was **100ms** to make a decision (jammer present or not) on a **5 MHz** band. Thus, we can calculate the time to analyse the whole WiFi band (2.4 GHz to 2.5 GHz) to be 2 seconds. Clearly, if the Processing board is powerful enough, this latency can be lowered but this will result in a more expensive solution.

This is a requirement that could also be loosened, depending on the use case needs. For example, installing the solution in a non-critical environment, such as a city where it does not need to be reactive instantly, could bear less strict detection time requirements.

5.1.1.3 Accuracy

The accuracy is a requirement that will vary a lot depending on the conditions in which 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 analysed band, the detection accuracy will decrease. For this reason, 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**.

5.1.1.4 Data rate

The data rate needs to be measured between the different modules of the solution. The highest data rates exist between the SDR module and the Processing Board. In this case, the maximum rate is **305 Mbps** (5 Msps) and the lower rate is **30.5 Mbps** (0.5 Msps). Higher data rates ensure better results because the data has more granularity but lower rates can also be used successfully.

Data rate between the Processing board and the Monitoring server will be much lower because the only information that is transferred is related to the detection events themselves. Only the type of jammer, frequency, jammer power and timestamp are transferred. This will result in about 100 bytes of payload per packet and minimum 40 packets per second (one packet per decision per algorithm).

5.1.2. Software adaptation

In order to perform a successful characterization of any software, a few conditions have to be followed. Firstly, the application and its dependencies have to be ported and tested on the host platform (in this case the X-Gene2). Secondly, whatever result the software generates as an output has to be deterministic and repeatable. This is mandatory because in order to detect SDCs (Silent Data Corruption) among different runs, the output needs to be compared and verified. Lastly, it is possible that the application processing or memory requirements are not sufficient to stress the hardware to the necessary extent. This is why, if it makes sense in the use case context, multiple instances should run on the same machine and this must be supported.

The first step was to port the application and its dependencies to the X-Gene platform. The main difficulty here was that the Jammer Detector is based on an SDR framework called GNU Radio, which was not available for the corresponding platform. The main effort was to make those dependencies work correctly and share them to the rest of the consortium.

In addition, the software must support the use of input data from files or some input sets that are repeatable for the tests so this feature and the generation of significant input sets was done too. On the other hand, GNU Radio framework actually uses its own scheduler to distribute input data flow (samples) through the application and this is done depending on the processing load at each thread. This is not helpful if we want to control the exact data that is taken as an input in order to generate a deterministic and repeatable output. For this purpose, a new version of the Jammer Detector was developed, which would slightly reduce its performance but would ensure a deterministic output at the same time.

Last but not least, the software had to be adapted to support several instances running in parallel on the same context. This was needed because a single instance was not providing enough CPU utilization. Moreover, several scripts were developed to launch the application processes in a simple and straightforward way.

5.1.3. Characterization process

To characterize the Jammer Detector, we used the framework that was described in detail in D3.3 “First Analysis of On-Chip Caches and Dynamic Memories” that gives us the opportunity to change the PMD voltage, the SoC Voltage, the DRAM Voltage and the DRAM refresh rate of the X-Gene 2 chips. Apart from the errors, the SDCs and the Crashes we also extended the framework in order to check the QoS requirements of the application.

To ensure the statistical significance of the delivered results, we launched 100 iterations of the characterization campaign in order to detect the safe operations margins. Moreover, we repeated the characterization

experiments of the application in three different TTT chips of the three partners that are involved in the characterization of the X-Gene 2 (UoA, UCY, QUB).

We separated the characterization of the Jammer Detector in four different steps:

- Identification of the safe operation margins when we reduce the voltage of the PMDs.
- Identification of the safe operation margins when we reduce the voltage of the SoC.
- Identification of the safe operation margins when we reduce the voltage and we increase the refresh rate of the DRAM.
- Combination of the identified operation margins from the three previous steps concerning the PMDs and the DRAM in order to measure the maximum power savings.

For all the aforementioned steps, we ensure the correct operation and the QoS of the application.

5.1.4. Tests

After identifying all the safe margins concerning the PMDs and the DRAM then we measured the power savings in off-nominal conditions compared to the power that is consumed using the nominal values.

5.1.5. Results

The following table presents the nominal values and the safe operations limits (best cases observed for100 iterations of the experiment) that were identified for all the parameters and the three chips that were used for the evaluation of the Jammer Detector application.

Parameter	Nominal	UoA chip	UCY chip	QUB chip
Vmin – PMD	980 mV	930 mV	910 mV	930 mV
Vmin – SoC	950 mV	920 mV	870 mV	900 mV
Vmin – DRAM	1500 mV	1428 mV	1428 mV	1428 mV
Refresh Rate min – DRAM	64 ms	2279 ms	2279 ms	2279 ms

Table 4: Safe operation limits (best cases observed for 100 iterations of the experiment) for the UoA, UCY and QUB chips.

Next, we will present the average power savings obtained with the three chips running on the safe operation limits found for UoA hardware, which is the most conservative one.

Figure 77 illustrates the average power savings that were measured for the chip located in UoA for 100 iterations of the experiments at Vmin PMD = 930 mV, Vmin SoC = 920 mV, Vmin DRAM = 1428 mV amd RR min DRAM = 2279 ms. We can observe 20.2% power savings in total compared to the nominal values.

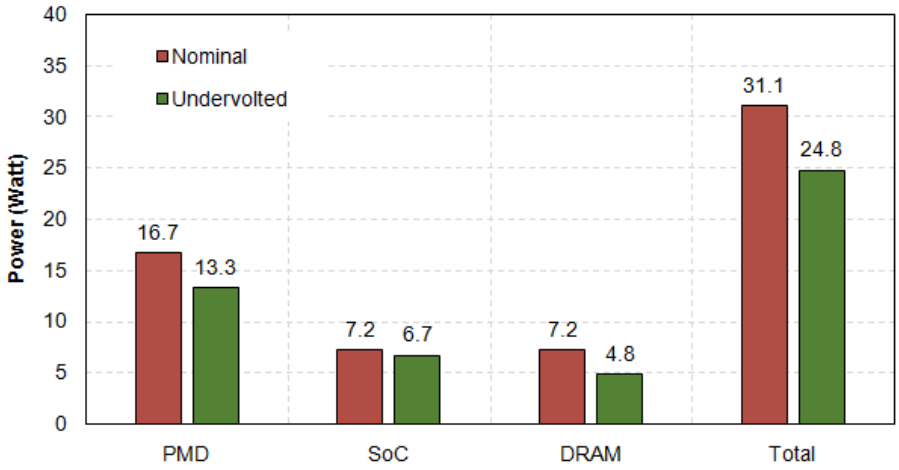

Figure 7: Average power savings for 100 iterations of the experiment in UoA chip.

Figure 8 presents the total power savings that were measured for the chip located in QUB at Vmin PMD = 930 mV, Vmin SoC = 920 mV, Vmin DRAM = 1428 mV amd RR min DRAM = 2279 ms. We can observe 12.3% power savings in total.

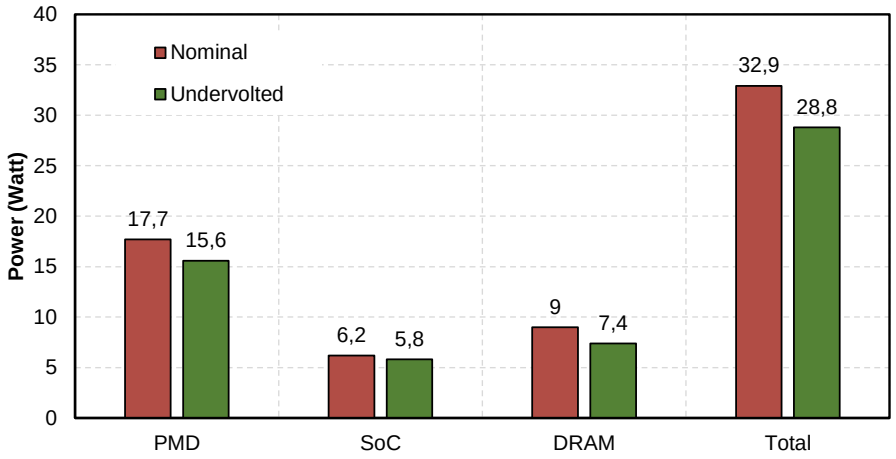

Figure 8: Average power savings for 100 iterations of the experiment in QUB chip.

Finally, Figure 9 illustrates the total power savings that were measured for the chip located in UCY at Vmin PMD = 930 mV, Vmin SoC = 920 mV, Vmin DRAM = 1428 mV amd RR min DRAM = 2279 ms. We can observe 12.5% power savings in total.

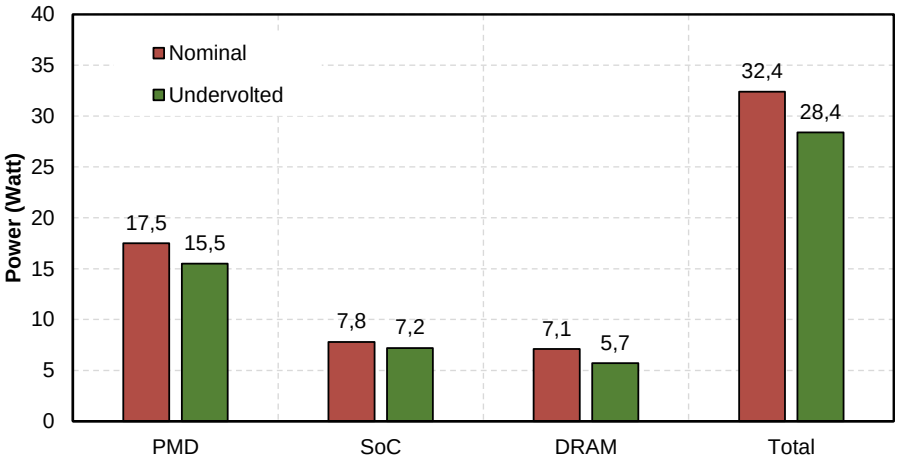

Figure 9: Average power savings for 100 iterations of the experiment in UCY chip.

As it can be seen from the figures, power savings can vary significantly in relation with the hardware. Different chips can throw different savings while running at the same margins, as they all have their own unique characteristics.

5.2. Social CRM

The Social CRM is built after the following architecture, which consists of several components.

1. The social network server component which runs on a large data centre using a distributed graph database (JanusGraph) and sends periodically the requested updates to the Social CRM WebApp Backend.
2. The Social CRM WebApp Backend, which consists of a MongoDB instance used to store all the data used by the WebApp.
3. The Social CRM Analytics component which process data from the Social CRM WebApp Backend and injects the results back into the storage.
4. The Social CRM WebApp Frontend, which runs an apache Tomcat WebServer that consults the data found in the Social CRM WebApp Backend for exploration

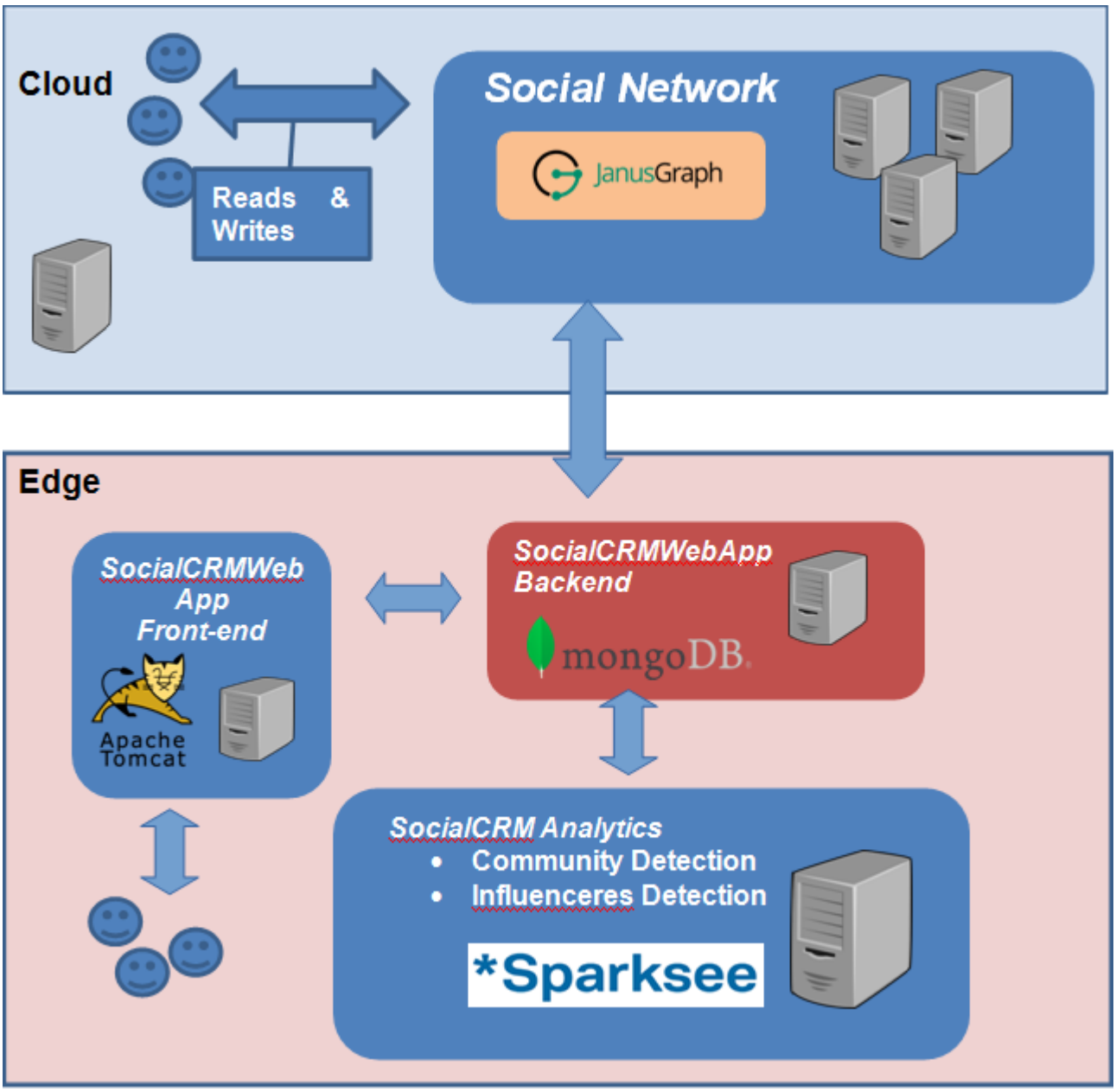

Figure 10: SocialCRM Architecture

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.

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.

Finally, from time to time, the Social Network sends new batches of data to the registered clients and they insert them into the Social CRM WebApp Backend.

The Social CRM WebApp

The Social CRM WebApp is split into two sub-components. The first is the FrontEnd, which consists of a web application running on an Apache Tomcat web server. The WebApp is used to explore the data stored in the SocialCRMWebApp Backend, which contains the influencers and communities found by the SocialCRM Analytics components. Figure 11 shows two screenshots of the SocialCRMWebApp Frontend application.

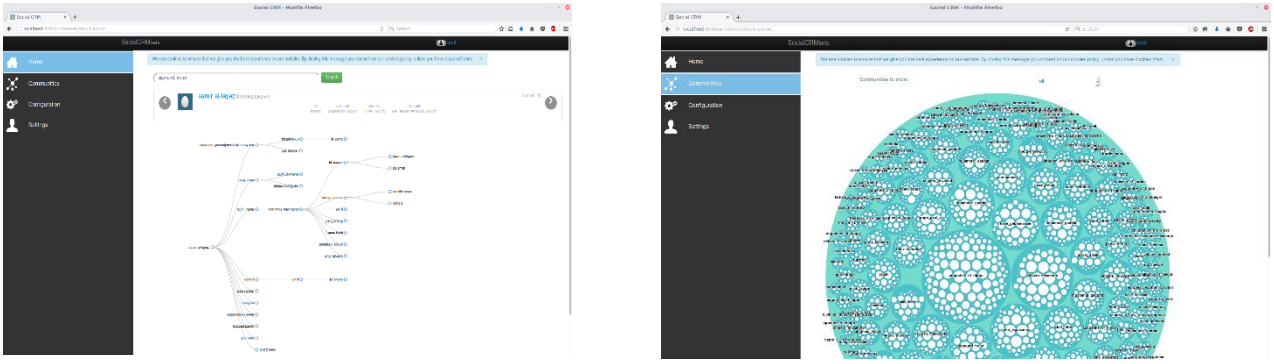

Figure 11: SocialCRMWebApp

The Social CRM Backend stores both the data received from the Social Network server and the results processed by the Social CRM Analytics component. From time to time, the Social CRM Analytics downloads from the Social CRM WebApp Backend the latest data and stores back the results of its analysis, to be consumed by the end users using the Social CRM WebApp Frontend.

The Social CRM Analytics component

The Social CRM Analytics component downloads, periodically, the latest data from the Social CRM WebApp Backend and performs complex analytics over these data. These analytics include detecting communities of people talking about specific topics, and also detecting the main influencers about these topics.

5.2.1. Requirements

The social network server component

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%
Accuracy	100%
Throughput	95% queries <= 2 seconds from their issue time. Queries/second fixed by the scale factor used

Table 5: Social Network Server SLA metrics

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

The Social CRM WebApp

In order to simplify the testing, we will only test the performance/SLA of the Social CRM WebApp Backend. The SLA 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.For the backend, we must guarantee that this is interactive and available. This is similar to the social network server, but more relaxed since it is much less used.

Metric	Description
Latency	<=2 seconds
Availability	90%
Accuracy	90%
Throughput	10 queries / second

Table 6: SLA for the Web App

The Social CRM Analytics component

Ideally, the client would like to have always the most up-to-date data pre-processed and available for interactive inspection by the Social CRM 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 7: 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 deadline required by the application.

5.2.2. Software adaptation

We have ported the Sparksee Graph Database to the ARM64 architecture. Additionally, we required compiling the MongoDB for the CentOS Linux distribution used by the X-Gene platform. Last but not least, we have had to adapt existing components to build the Social Analytics and WebApp components from scratch. In the case of the Social Network Server, we have ported the initial version that was using Sparksee as a backend, to JanusGraph in order to be used as a distributed backend (Sparksee is single node). So far, the development status of the SocialCRM components is the following:

- Social Network Server: 50%
- SocialCRM Analytics: 100%
- SocialCRMWebAppFrontEnd: 100%
- SocialCRMWebAppBackEnd: 100%

The characterization of the application must be done in a per component basis, since different SLAs are required for each of the components. Thus, we are initially focusing on testing and characterising the SocialCRM Analytics component. In order to help in this characterization process, we have developed a script to detect SDCs for the SocialCRM Analytics component, as well as provided sample datasets to be tested independently of the WebAppFrontEnd/Backend and the Social Network Server. Finally, we have set up VMsand Containers to streamline the overall process but also to make the environment more similar to a real setting.

5.3. Financial trading

MER's Financial Reporting Platform (Polaris) is responsible for validating and logging the lifecycle of Financial Institutions EMIR reports. It was developed as a third-party service for Investment Firms (IFs) to comply with the European Markets Infrastructure Regulation (EMIR) that came into force on 16 August 2012. The Regulation requires that all derivatives transactions need to be reported to Trading Repositories (TRs). The reporting of a derivative transaction should take place at T+1, where T is the transaction date, and it involves any daily modifications/updates of the transaction until the termination of the derivative contract. The key features of the platform are (a) the extensive pattern matching and validation of EMIR report fields to comply with the regulation standards using a Validation Kernel and (b) the logging of the reports and their status based on Trade Repository's feedback provided by the Feedback Processor. Figure 12 shows the full EMIR reporting lifecycle and all the components related to Polaris.

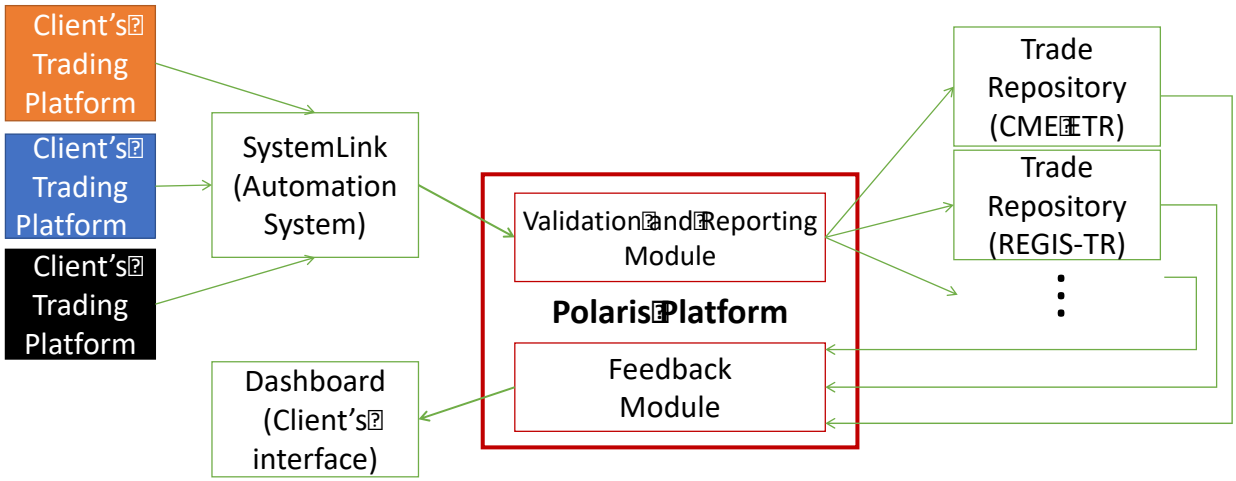

Figure 12: EMIR reporting lifecycle

Initially, the data are collected from client's trading platforms using SystemLink, a separated service, and CSV files are produced to be used as inputs to Polaris Platform. The CSV files can contain records of only one of the below 5 message types:

a) New Trade (NT) message; it's sent when a new transaction is executed

b) Modification Trade (MT) message; it modifies a previously reported NT
c) Valuation Update (VU) message; it updates the value of a previously reported NT
d) Collateral Update (CU) message; it updates the collateral of a previously reported NT
e) Termination Trade (TT) message; it terminates a previously reported NT

The files are then entered the first stage of reporting, the validation and mapping procedure, that validates the collected information and prepares the report to be submitted to Trade Repository (TR). At this point, any invalid records are reported back to the client to be corrected and resubmitted. The second stage of reporting is the process of feedback returned from the TR. Once the TR receives and validates the trades with additional rules, it returns a summary for each record in the report if it was accepted or rejected and the reason. The feedback files are processed and the rejected records are informed back to the client for resubmission.

5.3.1 Requirements

Requirement	Description
Availability	99%
Latency	Maximum of 3 hours
Accuracy	100%
Data rate/Performance	Min: 3000 records validated/minute

Table 8: Requirements table

Polaris platform requirements are mainly defined by the below three constraints:

a) The regulations T+1 deadline, where T is the date of the execution of the transaction. This suggests that the Investment Firms should report the T date transactions by 23:59 the next day, T+1
b) The Investments Firms reaction to rejected records. As the regulation reporting may return rejected trades due to client's misconfiguration during the submission, the result of the submission should be returned within the same day, for the client to take actions and resubmit the rejected transactions still in the T+1 deadline
c) The volume of each client and the total volume of all clients. This metric defines the required resources that we need to deploy based on our clients' reporting volume. It also defines the allocation of each client to a specific queue to make sure that he will get the report in time based on his volume. The volume can be estimated before starting the submissions process based on the number of transactions in the files submitted by the client.

Currently we have the below setup to satisfy our business requirements:

a) A set of clients that are estimated to have a total of ~0.5 million records are allocated on a single VM for validations and submission which will require about 2.5 hours for back to back processing (worst case scenario when they submit all at the same time and fill up the queue)
b) The agreement with all the clients to submit their trades before 21:00 gives us a maximum 3-hour timeframe before the deadline, thus 2.5 hours for processing and 30 minutes for any unexpected events
c) A failure with a MTTR of 15 minutes (time to reboot or reroute to another VM, recover to the checkpoints etc.) will still allow us to send the report of the last client before the end of the day.

Availability: We are obligated to send our feedback response before the end of the day, 23:59:59, in agreement with the client to initiate their submission before 21:00 on T+1. This is because clients' submissions may have invalids and they need to get their feedback early so they can react and resubmit to avoid missing the T+1 deadline. Given this constrain and the unpredictable behaviour of clients, regarding the submission time, we are forced to have that 99% availability; that's about 10 to 15 minutes of downtime per day that can be handle from the current setup.

A lower availability can be tolerated as our application doesn't require real-time interaction with the client and the time slot for reporting on the worst-case scenario is long enough, 3 hours, while the time to process a day's report of a single client takes 15 minutes on average except few big clients that may require up to 45 minutes. Thus, by dropping the availability and therefore reducing further the 3 hours reporting timeframe, more and

shorter queues will be required. To achieve this, we need to scale up our infrastructure to handle the total volume at the worst-case scenario and still satisfy our business requirements

Latency: Our target time to reply to the client is 3 hours considering the worst-case scenario described above. This means that from the time the client submits his data using our system we should validate his messages and return the first feedback from Polaris.

Accuracy: A 100% accuracy is required by our application as any changes in the information will lead to false report. In addition, there is no way to capture the errors during the reporting if the problem appears in a text field and doesn't affect the validation rules. For example, if the price of a transactions is reported differently but it's still a number then both Polaris and TR will accept this as a valid value but can dramatically change the end result of the reporting. Audit checks can be applied to check the validity of the reporting that this will reduce the performance of the platform significantly.

Data rate/Performance: Currently, our platform can achieve a minimum of 3000 record validations per minute when running on a high performance x64 machine with virtualization using KVM. This number was measured for the scenario where all records are valid and without any conflicts for the New Trade (NT) message that has the most validations. If all records are valid and without any conflicts it means all records will be prepared for reporting and inserted in the local database which is the case that has the longest execution time for a specific number of records.

5.3.2 Software adaptation

Polaris platform is part of a regulation reporting product that consists of several services. It requires the SystemLink service that prepares the input CSV files, a database engine that maintains the clients' previous submissions, their status and checks for conflicts and a TR that makes a final validation of the submission and returns a feedback for processing. Polaris is the most demanding service in this product, both in memory and CPU requirements and that's why it was selected as a UniServer benchmark application.

In order to provide a real evaluation of the platform, several adaptations were required to emulate the surrounding services and the full cycle of the reporting. To achieve this, Polaris code was re-written to emulate the 3 phases of the cycle:

a) the initialization phase, where production files are copied to the input folder and the database is initialized to a deterministic state
b) the validation phase, where the actual work of Polaris platform is executed and includes the record validations, conflict detection, database update, submission file preparation and file submission to TR
c) and finally, the feedback phase where we emulated the reception of the feedback files from TR, which are copied to the feedback input folder, and the processing of the records and updating of the database based on the status of each record (Accepted or Rejected and the relevant error in this case)

In addition, the code was also adapted to be run using a configuration file, especially for the UniServer project, to enable users to run the 3 phases of the software separately for better and more accurate evaluation, for example to skip phase (a) which is the initialization and then run phases (b) and (c) which are the actual work done by the real application.

Once the platform was adapted to a first version that satisfies the UniServer needs, the code was ported to ARM architecture and compiled on the XGene machine. In addition to this, a VM was prepared with the MySQL database service, and Perl installed and all the necessary Perl modules required by Polaris run, along the Polaris platform already installed. The prepared VM and an executable with instructions on how to install in a new system were distributed to the partners.

6. Security

The UniServer concept supports the deployment of a potentially large number of micro-servers in an edge computing paradigm. They will typically be constituted as single-site installations, or as low density micro-server clusters. The traditional cloud-server model, shown in Figure 13, which consists of many high-density server racks, is typically housed in a large building or complex, protected by many security features such as perimeter fencing, access controls, security guards etc. Due to the large number of servers that are being protected at the same time, the cost of implementation is averaged over the population, providing economies of scale. However, for edge deployments, where a large number of discrete installations are spread over large geographic regions, the costs associated with such security provisions are inherently uneconomic.

For many remote micro-server deployments, such as depicted in Figure 14, security measures may amount to nothing more than a simple cabinet enclosure, primarily designed for environmental protection. For the motivated attacker, this offers the potential to gain direct physical access to the micro-server. Therefore, in addition to defending against the existing range of networking and software-based attacks, further consideration needs to be given towards physical attacks, where it is assumed that the attacker has full access and thus scope to tamper with the system hardware or take measurements to reveal secret information through the leakage of side-channel information.

Figure 13: Cloud Server Infrastructure in secure warehouse complex.

Figure 14: Isolated Edge Server deployment.

6.1. Bare-Metal Deployment

The full-stack system model of Figure 15, depicts the UniServer architecture for a cloud micro-server style implementation. The current prototype implementation is constituted as a bare-metal deployment, as shown in Figure 15, suitable for running standalone applications; for example, as would be used for a single micro-

server deployment. In this model the UniServer software and applications are running under a single operating system, without the use of virtualisation.

6.1.1. Security Considerations

As shown in Figure 15, the bottom hardware layer of the stack consists of sensors and registers which report various parameters, such as temperatures, voltages and errors for relevant system components, such as the processor modules, cores and memory modules. These values are presented in a set of memory mapped registers which can be read, and/or written to. The registers are exposed via the Firmware Reliable Error Detection layer and the I2C bus. At this layer, access to the registers and their values is generally protected behind the firmware layer, however, as part of the UniServer development, there is an ability to directly read/write many of the sensitive registers via the system I2C bus. This I2C access is currently available from user space command line, as detailed in section 3.1 of deliverable 4.1, issuing i2cget and i2cset commands. For example, commands such as below that read the SOC VRD temperature, and change the ACPI state in PMD0;

```
$ i2cget –y 1 0x2f 0x11 w
$ i2cset –y 1 0x2f 0xE1 0x1 b
```

This direct control of registers would enable an attacker to issue commands that directly control the value of processor and memory voltages, generating errors, making the system state unstable and unreliable, possibly causing the system crash. In a live deployment, access to I2C registers via the command line should be restricted to those requiring essential access privileges.

The next level in the stack is the Firmware Reliable Error Detection layer, which exposes the registers up through low level handlers into the Hardware Exposure Interface (HEI) driver. The HEI driver incorporates the HEI Applications Programming Interface (API) which enables applications running from user space to register for event notifications, and for the Hypervisor and HealthLog Daemons to interact with the sensitive extended margins registers. This ability to register with the HEI API is a potential attack vector for rogue applications to monitor, or potentially change, the register values. It is therefore recommended that in live deployment, that registration to the HEI API is vetted, ensuring that only authorised applications can use the API to register or issue API calls.

The UniServer software of HealthLog, Predictor and StressLog operate as Daemons in the user space. As user space entities, they are managed through the kernel, and are provided with their own system memory, segregated from other running applications. In the top layer of Figure 15, above the HealthLog daemon, are the user applications, highlighting that they are running on the same host OS and users space as the UniServer software. Although the applications will all be managed by the kernel with separate physical memory space, they will potentially be able to view and access the same directory structure on the hard drive enabling access to critical files such as log, policy or configuration files that are stored there. This could then provide an open target for malicious actors that are able to access the system. For this reason, it would be prudent to apply stringent access and privileges rights for users and applications running on the UniServer system.

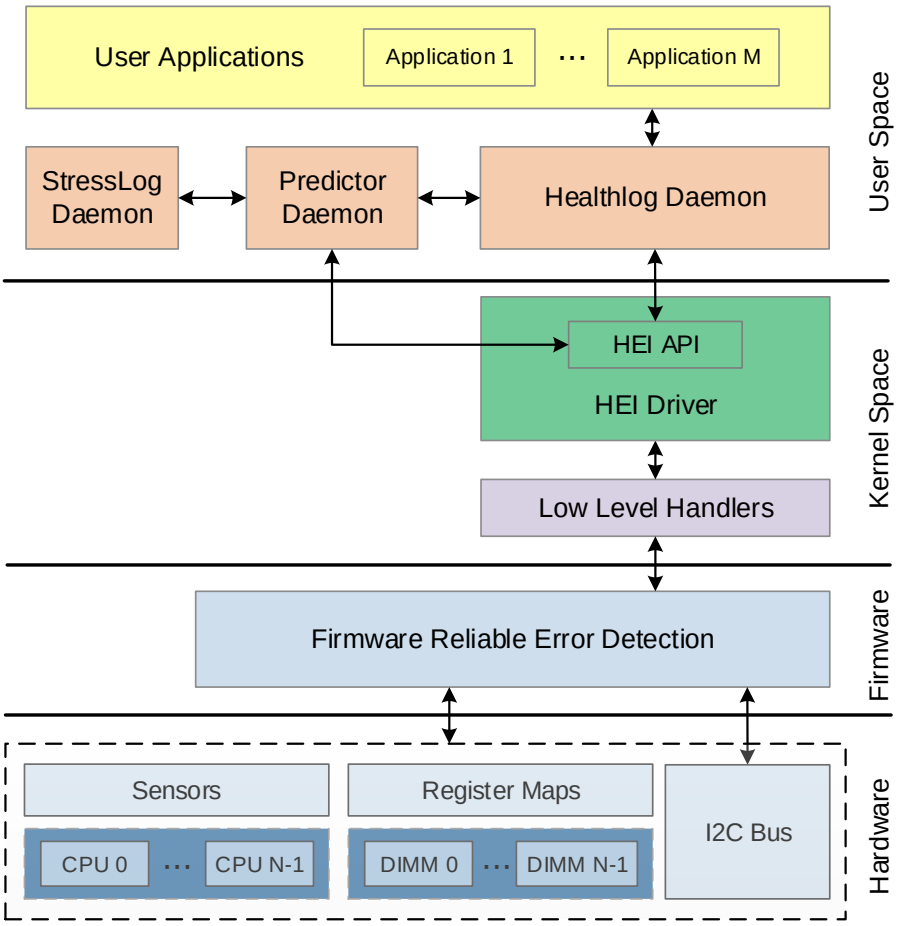

Figure 15 Bare Metal deployment of the UniServer architecture.

The central hub of the UniServer ecosystem is the HealthLog, which acts as a gateway for messages coming up through the HEI. The HealthLog operates in two primary servicing modes; firstly, it provides an event-driven service, where it collects system event notifications sent up from the firmware layer, for example system error notifications, and then also provides an on-demand service in support of the predictor and StressLog modules.

HealthLog monitoring gathers information categorised as reliability, power, thermal and performance metrics. In regular time intervals (or on demand), the HealthLog monitor will produce an output information vector containing the current state of these metrics, which will then be recorded within log files. These log files can then be queried by other elements, such as the Predictor and TCO.

In the event handling servicing mode, notifications are streaming in from the HEI interface, via the registered HEI API interface. These values are parsed from the text stream and placed into the information vector, which is then written to the log file. At the same time, this event information is compared against configuration policy to determine if current event state warrants further action, such as notifications to other system software. The on-demand query handling is used to query specific information from the system state, or to initiate any on-demand updates of register values. The on-demand process uses system /dev/ht as the interface. The HealthLog log file is now available for use by the Predictor or TCO to analyse the history of the system state to make analyses and predictions on whether system state needs to be changed, whether StressLog needs to be run, or whether cost/performance metrics are being met.

The StressLog will run periodically throughout a machine's lifetime, for example to compensate for aging effects of the hardware, or when triggered by the Predictor, in the case of anomalous machine behaviour. In that event, the machine being tested will be taken offline and run through various stress test scenarios, using the provided stress target parameters. The StressLog monitor takes the output of HealthLog and wraps all information into a vector to be passed to the higher layers.

The Predictor utilises characterisation of data to determine the probability of failure for non-nominal voltage-frequency and DRAM refresh states. The availability constraints, provided by OpenStack, define the desired

number of cores and the operating frequencies, with the predictor estimating the most energy efficient voltage and DRAM refresh states, whilst avoiding violation of system constraints.

An important consideration from a security perspective is where the log and policy/configuration files sit. Although the UniServer software Daemons run in user space, and therefore in system RAM, the log and policy files are stored on the hard drive in the clear. Since the HealthLog log file is the means by which the Predictor (and therefore StressLog) gain their information, it is important for the stability of the system that the information held in this file is not corrupted or tampered with by a malicious user.

It is not difficult to imagine that if the HealthLog information values are tampered with or corrupted, that the Predictor may then determine that the system should be taken offline for StressLog re-characterisation and subsequent restoration. This process could be repeatedly triggered, affecting availability and possibly full denial of service. It is a recommendation, therefore, that the log and policy files are stored in encrypted format, to avoid reading and manipulation by others. Additionally, consideration should be given as to whether the files should be digitally signed, to ensure their authenticity and origin from a trusted process. These recommendations would naturally have overheads in terms of real-time performance, so their implementation would need to be considered carefully in terms of system performance and operability.

Although the bare metal implementation is the simplest, both conceptually and in terms of implementation, it may be considered an advantage to deploy a hypervisor to better separate the host OS and UniServer software from 3rd party applications. Although it should be noted that the recently published Meltdown attack of 0, discussed further in deliverable 7.2, has implications for virtual environments, particularly those which are not fully virtualised. For example, approaches utilising a container paradigm, where the Kernel is shared, e.g. Docker, LXC, and OpenVZ, which have been shown to be vulnerable to Meltdown and permit attacks that leak data across container memory boundaries. Virtualisation, in-itself, is not a full solution since an attacker with physical access to a system can access all areas and could, for example, connect to system management ports such as serial and network ports, connect to hard drives ports to access the data bus directly, or mount side-channel attacks to steal sensitive information. The recommendation for use of encryption and/or signing of the sensitive files should however also assist in protecting against such attacks.

7. End-to-End TCO analysis

The end-to-end TCO analysis aims to estimate the entire eco-system lifetime capital and operational expenses considering the number and costs of the servers used for edge and cloud nodes and the internet network latency. A TCO analysis can be useful to decide in which location (edge or cloud) to run a specific application to satisfy its requirements while providing cost savings. TCO is strongly correlated with the number of nodes that are used to run the application. The more nodes per application, the higher the cost to run the application. So, it is critical to determine the minimum number of nodes that are needed to satisfy the application requirements to reduce cost.
The following example of end-to-end TCO analysis illustrates how the internet network latency and the number of nodes affect the decision whether to run at the edge or the cloud, the DoS Sensing application.
The results are obtained by running Detection of Jammer Attack Application (DoSSensing) on the X-Gene2 with ARMv8 cores. We have experimented for different number of instances per node. Also, for the network latency we have pinged an Amazon server in Ireland.
The TCO results explored with the AMPRA tool [29] and COST-ET tool [30]. The baseline configuration without UniServer runs the application at nominal settings (pmd:980, soc:950, dramvol.:1500, dramrefr.:78), whereas for the UniServer technology, the minimum safest voltage domain was selected (pmd:910, soc:870, dramvol.:1428, dramrefr.:2783).

Figure 16 shows the end-to-end latency running the DoS Sensing application either at the Cloud or at the Edge with different number of instances of the application per node. The figure shows for each case the breakdown of the end-to-end latency into compute and network. We can observe that DoS Sensing compute latency increases as the number of instances increase. In addition, we can see that edge deployment has lower network latency than cloud.
As a result of this network latency reduction, an edge deployment can accommodate more instances per node while satisfying QoS. Specifically, for 250ms 99th end-to-end latency the number of instances that can be accommodated is two for the cloud and four for the edge. This clearly illustrates the benefits of running at the Edge.
Additionally, we investigate the benefits from UniServer technology when applied both at the Edge and Cloud.

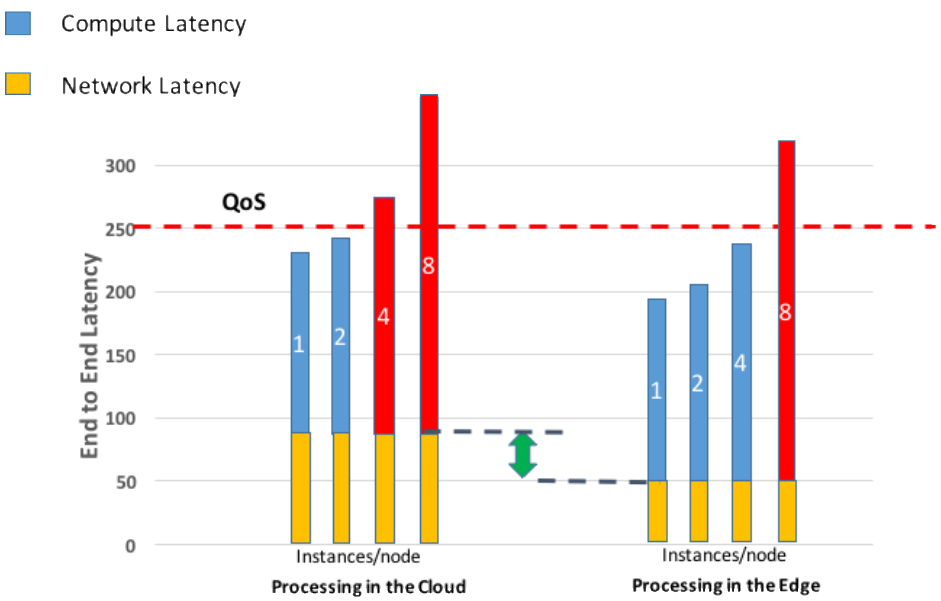

Figure 16: Compute and Network Time for Cloud and Edge Deployments

It is important to notice that in edge deployments there are some power constraints. To control power constraints, servers in the edge are power capped. Taking into account that we have a power capping of 70Watts; four instances are not applicable. Figure 17 shows that without UniServer four instances cannot run in the same node due to power constraint and power capping. On the other hand, UniServer technology can reduce power by operating at marginal voltage and refresh rate and avoid power capping. Specifically, the number of instances that can be accommodated is two for the edge without UniServer and four for the edge with UniServer. This clearly illustrates the benefits of running at the Edge with UniServer.

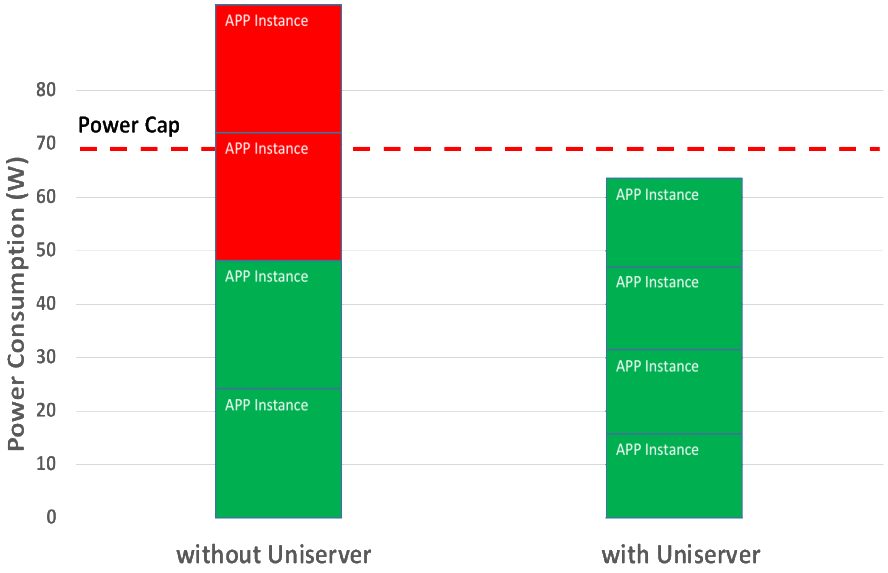

Figure 17: Power Capping with and without UniServer

The following figure shows the total TCO findings by running the DoS Sensing application in the Cloud with and without UniServer and by running the application in the Edge with and without UniServer, as well. As can be seen, according to the TCO, running the application either at the cloud or at the edge without UniServer, the TCO remains the same due to the same number of instances per node (2 instances per node). On the

other hand, having UniServer, TCO can be reduced, reaching 50% savings. The best configuration is when the application runs in the Edge with the UniServer software.
Similarly, trade-offs will be explored taking into account the three applications of the UniServer project. 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.

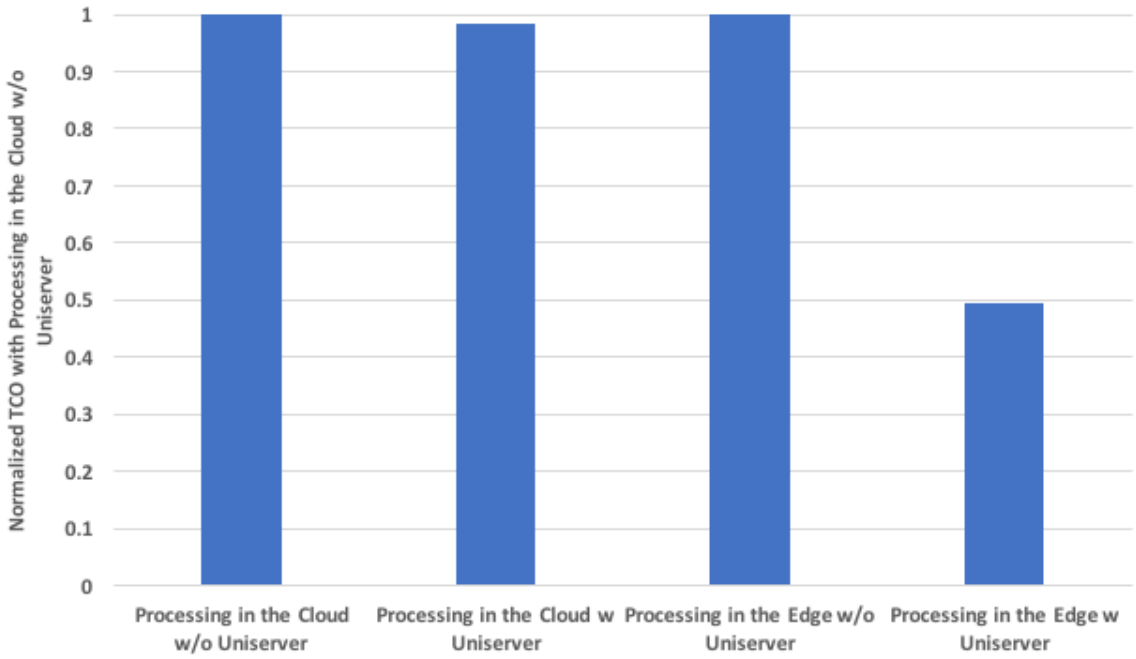

Figure 18: End to End TCO Findings with and without UniServer

8. Conclusions

The deliverable analyses the Uniserver platform's advantages regarding the state of the art and its metrics of success. It also introduces the first stable platform prototype and describes its characteristics as well as security concerns. The prototype can be used to validate the power savings that were intended at the beginning of the project.
As we can observe, the results presented regarding a real-life use case (WSE Jammer Detector edge application) show promising results, with up to 20% power savings. On the other hand, end to end TCO analysis provides up to 50% savings under particular power capping constraints. These findings give us encouraging results and a starting point to continue the complete software stack integration towards the final system software, which will provide full benefits as well as significant power savings. Next steps will be to finish the development, integration and testing of the complete software stack and perform hardware characterization with both Sparcity and Meritorious applications to validate the results already obtained in order to analyse the behaviour of the full Uniserver platform.

References

[1] 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.

[2] A. Bacha and R. Teodorescu, "Dynamic Reduction of Voltage Margins by Leveraging On-chip ECC in Itanium II Processors," *SIGARCH Comput. Archit. News*, vol. 41, pp. 297-307, 6 2013.

[3] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose and V. J. Reddi, "Safe limits on Voltage Reduction Efficiency in GPUs: A Direct Measurement Approach," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[4] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd and V. J. Reddi, "Adaptive Guardband Scheduling to Improve System-Level Efficiency of the POWER7," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[5] Y. X. L. J. e. a. Dong Y, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471-1480, 2012.

[6] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G. Y. Wei and D. Brooks, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[7] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Designing a Processor from the ground up to allow Voltage/Reliability Tradeoffs," in *16th International Conference on High-Performance Computer Architecture {(HPCA-16} 2010), 9-14 January 2010, Bangalore, India*, 2010.

[8] H. Duwe, X. Jian, D. Petrisko and R. Kumar, "Rescuing Uncorrectable Fault Patterns in On-Chip Memories through Error Pattern Transformation," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[9] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu and S.-L. Lu, "Improving Cache Lifetime Reliability at Ultra-low Voltages," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2009.

[10] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah and S. L. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," in *2008 International Symposium on Computer Architecture*, 2008.

[11] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge, "Razor: a Low-Power Pipeline based on Circuit-Level Timing Speculation," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003.

[12] S. R. Sarangi, B. Greskamp, A. Tiwari and J. Torrellas, "EVAL: Utilizing Processors with Variation-induced Timing Errors," in *41st Annual {IEEE/ACM} International Symposium on Microarchitecture, November 8-12, 2008, Lake Como, Italy*, 2008.

[13] A. Tiwari, S. R. Sarangi and J. Torrellas, "ReCycle: Pipeline Adaptation to tolerate Process Variation," in *34th International Symposium on Computer Architecture, June 9-13, 2007, San Diego, California, {USA}*, 2007.

[14] X. Liang and D. M. Brooks, "Mitigating the Impact of Process Variations on Processor Register Files and Execution Units," in *39th Annual {IEEE/ACM} International Symposium on Microarchitecture, 9-13 December 2006, Orlando, Florida, {USA}*, 2006.

[15] D.-H. K. a. M. K. Q. Prashant J. Nair, " ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates.," in *In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13).* , 2013.

[16] B. J. R. V. a. O. M. Jamie Liu, "RAIDR: Retention Aware Intelligent DRAM Refresh," in *In Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*, 2012.

[17] S. H. a. E. R.] R. K. Venkatesan, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *In The Twelfth International Symposium on High-Performance Computer Architecture*, 2006.

[18] D.-Y. S. Y.-J. C. C.-L. Y. a. C. M. W. Chung-Hsiang Lin, "SECRET: A Selective Error Correction Framework for Refresh Energy Reduction in DRAMs," in *ACM Trans. Archit. Code Optim. 12, 2*, 2015.

[19] A. G. Y. S. G. A. A. N. C. A. K. D. L. M. O. H. H. a. O. M. Kevin K. Chang, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *Proc. ACM Meas. Anal. Comput. Syst. 1, 1, Article 10 (June 2017), 4*, 2017.

[20] B. J. Y. K. C. W. a. O. M. Jamie Liu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*, 2013.

[21] É. Z. D. M. M. M. H. C. B. C. W. a. N. W. Matthias Jung, "Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs," in *In Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS '15)*, 2015.

[22] D. L. Y. K. A. R. A. C. W. a. O. M. Samira Khan, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS Perform. Eval. Rev. 42, 1*, 2014.

[23] M. C., Corporate Security Management, Elsevier Inc., 2015.

[24] S. Institute, "operating system security and secure operating systems," [Online]. Available: https://www.giac.org/paper/gsec/2776/operating-system-security-secure-operating-systems/104723. [Accessed 12 2017].

[25] IBM, "Business Intelligence Architecture and Deployment Guide - Securing the Operating System," [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSEP7J_10.2.1/com.ibm.swg.ba.cognos.crn_arch.10.2.1.doc/c_securing_the_operating_system.html. [Accessed 12 2017].

[26] Z. Wang, X. Jiang, W. Cui, W and P. Ning, "Countering kernel rootkits with lightweight hook protection.," in *In Proceedings of the 16th ACM conference on Computer and communications security (pp. 545-554).*, 2009.

[27] S. Friedl, "An Illustrated Guide to the Kaminsky DNS Vulnerability," [Online]. Available: http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html. [Accessed 12 2017].

[28] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, "Meltdown and Spectre. Bugs in modern computers leak passwords and sensitive data.," Jan 2018. [Online]. Available: https://meltdownattack.com/meltdown.pdf.

[29] Panagiota Nikolaou, YiannakisSazeides, Lorena Ndreu, and MariosKleanthous. Modeling

the implications of dram failures and protection techniques on datacentertco. In Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, pages 572–584, New York, NY, USA, 2015. ACM

[30] 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

[31] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y.Yarom and M. Hamburg, "Meltdown and Spectre. Bugs in modern computers leak passwords and sensitive data.," Jan 2018. [Online]. Available: https://meltdownattack.com/meltdown.pdf.