D7.5 Second Vertical, Full, System Integration and Validation Report

Contract number	688540
Project website	http://www.uniserver2020.eu
Contractual deadline	Project Month 36 (M36): 31st January 2019
Actual Delivery Date	30 January 2019
Dissemination level	Public
Report Version	1.0
Main Authors	George Papadimitriou (UOA), Athanasios Chatzidimitriou (UOA), Dimitris Gizopoulos (UOA)
Contributors	Marios Kleanthous (MER), Christian Pinto (IBM), Denis Guilhot (WSE), Manolis Maroudas (UTH), Christos Kalogirou (UTH), Panos Koutsovasilis (UTH), Panagiotis Vlastaridis (UTH), Evagelia Malami (UTH), Lev Mukhanov (QUB), Zacharias Hadjilambrou (UCY), Arnau Prat (SPA)
Reviewers	Zacharias Hadjilambrou (UCY), Arnau Prat (SPA), Georgios Karakonstantis (QUB), Charles Gillan (QUB)
Keywords	Integration, StressLog, HealthLog, Predictor, Applications, Hypervisor

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.

© 2019. 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.

© 2019. UniServer Consortium Partners. All rights reserved

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.

© 2019. UniServer Consortium Partners. All rights reserved

Change Log

Version	Description of change
0.1	Draft
0.2	Review from partners
1.0	Version for delivery to EC

© 2019. UniServer Consortium Partners. All rights reserved

Table of Contents

1. INTRODUCTION 9
2. SYSTEM OVERVIEW 10
3. HARDWARE/FIRMWARE LAYER 12
3.1 STATUS 12
3.2 HEI 12
3.2.1 Status 12
3.3 HEALTHLOG MONITOR 12
3.3.1 Status 12
3.4 STRESSLOG MONITOR 13
3.4.1 Status 13
4. SYSTEM SOFTWARE LAYER 14
4.1 LINUX PLATFORM 14
4.1.1 Status 14
4.2 PREDICTOR 14
4.2.1 Status 14
4.3 HYPERVISOR 14
4.3.1 Status 15
4.4 OPENSTACK 17
4.4.1 Status 19
5. APPLICATION LAYER 20
5.1 WSE WIRELESS JAMMER DETECTOR 20
5.1.1 Status 20
5.2 MER: POLARIS REGULATORY REPORTING PLATFORM 20
5.2.1 Status 21
5.3 SPA: SOCIAL NETWORK SERVER SOCIALCRM/SOCIALTV 21
5.3.1 The social network server component 21
5.3.2 The social analytics component 21
5.3.3 The web app component 22
5.3.4 Status 22
6. CONCLUSIONS AND FUTURE WORK 23
7. REFERENCES 24

© 2019. UniServer Consortium Partners. All rights reserved

Index of Figures

Figure 1: Detailed UniServer Layering 11

Figure 2: Module Layering 11

Figure 3: Hypervisor: looking downwards the software stack 15

Figure 4: Hypervisor - looking upwards the software stack 15

Figure 5. Overview of memory allocation for application running on heterogeneous-reliability memory framework 16

Figure 6. Performance overhead manifested for memory with disabled interleaving (Non-interleaved system) and memory with enabled software-level interleaving (Shimmer) 17

Figure 7: Hypervisor and OpenStack relationship 18

© 2019. UniServer Consortium Partners. All rights reserved

Index of Tables

Table 1: Table of Terms.. 7
Table 2: UniServer levels.. 10
Table 3: UniServer target applications 20

© 2019. UniServer Consortium Partners. All rights reserved

Terminology

Term	Definition
ACPI	Advanced Configuration and Power Interface
APEI	ACPI Platform Error Interface
HEI	Hardware Exposure Interface
KVM	Kernel Virtual Machine
PMD	Processor Module
QEMU	Quick emulator
TCO	Total Cost of Ownership
SoC	System-on-Chip
VM	Virtual Machine

Table 1: Table of Terms

© 2019. UniServer Consortium Partners. All rights reserved

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, the goal is to improve the power efficiency of ARM-based micro-servers running in the cloud or edge.

This report deliverable describes the spiral and incremental integration status as of project month M36 (January 2019) of the hardware and software components of the UniServer system. The tasks T7.1 - *Vertical, full system integration* and T7.4 - *Pointing of small-scale application and evaluation* contribute to this deliverable.

This is a status report only; it does not describe the interfaces in detail. For additional information, the reader is referred to the documents listed in the References section.

© 2019. UniServer Consortium Partners. All rights reserved

1. Introduction

The UniServer project targets the development of a unique methodology and infrastructure for exposing the pessimistic design margins in commercial servers and exploiting them through intelligent power, performance and reliability management schemes at the software and hardware layers. This includes the hardware, firmware, system software, virtualization, and cloud management layers.

This report describes the current status of the integration efforts of the UniServer modules. A brief description of each module will accompany the description.

The remainder of this document is structured as follows: section 2 provides an overview of the UniServer system, section 3 gives a summary of the hardware/firmware layer, section 4 describes the important UniServer modules in the system, section 5 discusses the application layer, and section 6 presents the conclusions and directions for future work.

© 2019. UniServer Consortium Partners. All rights reserved

2. System Overview

The UniServer system can be logically divided into four parts:

Level	Description
Hardware	X-Gene2 based “Tigershark” systems or X-Gene3 based “Osprey” systems
Firmware	HEI, and UniServer system components (HealthLog, StressLog and Predictor)
System Software	Linux OS, virtualization extensions such as KVM/Qemu, and OpenStack cloud infrastructure
Application	Applications running on bare metal hardware or in the context of a VM

Table 2: UniServer levels

The newly developed UniServer components are the *HealthLog Monitor* [4] [8], the *StressLog Monitor,* and *the Predictor*. These components do not exist in standard system software distributions, and become part of the Firmware and System Software levels in the above table. In addition, there has been extensive work focused on the virtualization layer, which is part of System Software.

The hardware components are stress tested during a pre-deployment phase using various stress methods, consisting of a combination of micro-viruses (small execution time validation programs) [11] [12] [9] and real application benchmarks. The *HealthLog Monitor* continuously monitors the health status of the hardware under a particular voltage/frequency/refresh rate (V-F-R) point and provides the information to other layers in the system via vectors describing the performance, power, temperature, and any incurred errors.

The *StressLog Monitor* [5] is responsible for testing the hardware components, producing a log file (the StressLog) containing a summary of the execution, including observed errors (read from the HealthLog Monitor log), observed Silent Data Corruptions (SDCs), application crashes, etc. Typically, the StressLog Monitor is called by the Predictor using different V/F configuration parameters, different micro-viruses/benchmarks, etc.

The Predictor consumes the logs produced by the StressLog Monitor and builds models to estimate safe voltage/refresh-rate margins for different operating points. At runtime the upper layers (i.e. the Hypervisor), can ask the Predictor for optimal operating points.

The UniServer layering is shown in Figure 1.

The Hypervisor attempts to limit the effects of potential faults in 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 Monitor and the Predictor models and developing a new set of configuration properties. Finally, when aging effects are being observed by the Hypervisor via the HealthLog Monitor, the Hypervisor can ask the Predictor to rebuild the models which in turn, will make the Predictor to call the StressLog Monitor and restart the training cycle.

The virtual machines are all controlled by the cloud management software, OpenStack. OpenStack controls multiple nodes and tries to minimize the energy footprint of the system, while at the same time respecting Service Level Agreements. It queries the Hypervisor to identify the trade-offs between energy, performance and reliability, and sets nodes at appropriate configurations and schedules VMs.

© 2019. UniServer Consortium Partners. All rights reserved

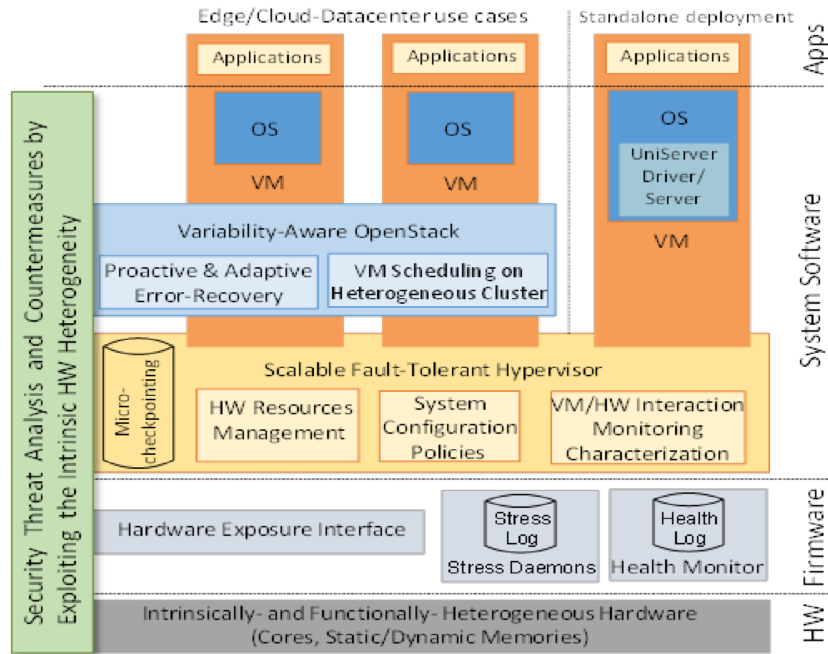

Figure 1: Detailed UniServer Layering

An overview of the modules in the system software layers is shown in the following diagram (Figure 2).

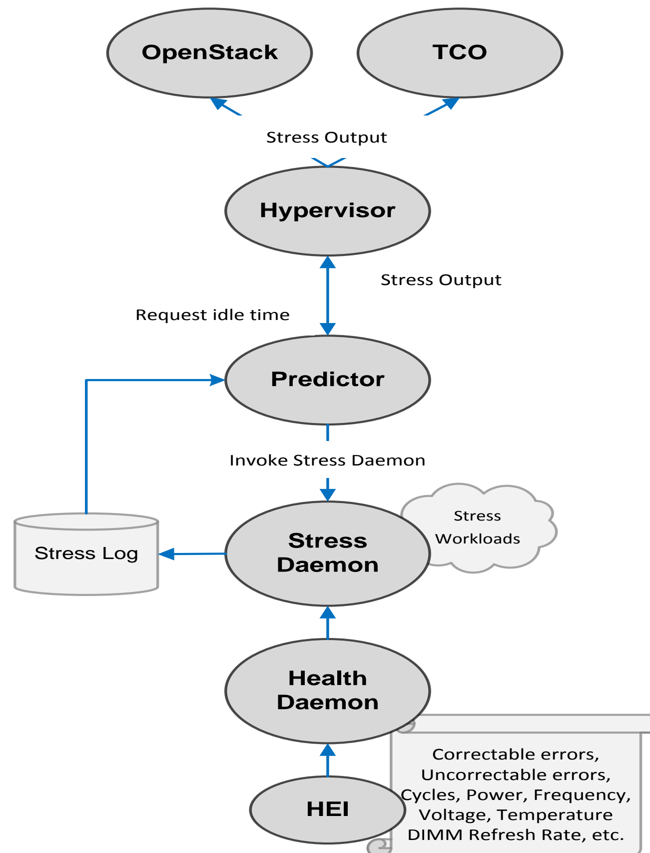

Figure 2: Module Layering

© 2019. UniServer Consortium Partners. All rights reserved

3. Hardware/Firmware Layer

The interface between the X-Gene processor and the system software (Linux) is described in detail in D4.1 [6]. The API includes a notification interface, the intent of which is to provide an indication of failures during undervolting. However, undervolting testing [7] has shown that the first indication of an error is often silent data corruption, requiring additional checks in the upper layers.

3.1 Status

The X-Gene2 “TigerShark” platforms have been deployed since project month M5; the X-Gene3 “Osprey” platforms have been delivered as of project month M20.

As of project month M31, the API is fully functional on both the X-Gene2 based “TigerShark” and X-Gene 3 based “Osprey” platforms. This is an I2C based interface running on Centos based Linux 4.11. The notification API is layered on top of the standard Linux APEI error reporting interface, leveraging the existing firmware and reducing the necessity of custom firmware builds.

The primary consumer of the HEI API is the HealthLog Monitor, and the integration between the HealthLog Monitor and hardware/firmware has been completed.

3.2 HEI

The Hardware Exposure Interface ([6]) provides two separate functions:

- Exposure of architecture-specific sensors and registers via an I2C register map
- Notification of processor error events

The I2C register map allows upper layer modules to monitor and control power and frequency for the processor and DIMMs. The event notification API provides a straightforward interface for the receipt of error events. The HEI has been implemented for both X-Gene2 and X-Gene3. The implementation is quite different, as the firmware for each chip shares almost nothing in common.

3.2.1 Status

The HEI has been implemented and integrated with the HealthLog Monitor for X-Gene2 and X-Gene3 platforms supporting all the I2C registers specified in the D4.1 document.

3.3 HealthLog Monitor

The HealthLog Monitor ([4] [8]) receives error events from the hardware/firmware via the HEI notification interface. HealthLog monitors the state of the system producing a log that is consumed by the StressLog Monitor and the Predictor. It also provides an interface for receiving notifications and trigger action upon event occurrences. The HealthLog Monitor also features an error testing interface, to ensure that the errors that can be reported are correctly parsed by the other modules in the system. In addition, the HealthLog Monitor keeps track of some statistical information regarding health-related incidents and provides these statistical data through an internal text-based API, on an "on-demand" scheme.

3.3.1 Status

As of project month 31 (M31), the HealthLog Monitor has been integrated into the system, and the interfaces to the Predictor and StressLog Monitor are being finalized, along with the "on-demand" serviced, as these are described on the D4.6. Additional interfaces for reporting high-level information on the log (by the Hypervisor) and for interacting with other monitoring services have also been integrated.

© 2019. UniServer Consortium Partners. All rights reserved

3.4 StressLog Monitor

The StressLog Monitor ([5]) can be triggered at any time to run stress tests to produce logs containing the details about the observed system's behaviour (such as aging effects). The tests can be configured in a way that combinations of different benchmarks/viruses, bound to different cpu cores, and with different V/F parameters can be executed. The module is typically called by the Predictor, which uses the logs to build the models to predict the behaviour of the system at different operating points. Alternatively, it can be triggered by the system's administrator or the Hypervisor if aging effects are being detected via the HealthLog Monitor.

3.4.1 Status

The StressLog Monitor has been integrated into the system, and the interfaces to the Predictor are being finalized. Additionally, it has been integrated with the HealthLog Monitor so it can read the errors that occur in the system during the execution of the stress tests. There is one pending integration with the HealthLog regarding the hardware counters. The StressLog is ready to support hardware counters from an interface perspective, but integration is pending in order to retrieve this data from the HealthLog, when support for hardware counters becomes available.

© 2019. UniServer Consortium Partners. All rights reserved

4. System Software Layer

4.1 Linux Platform

As for the M31 of the project, both TigerShark and Osprey platforms use the Centos 7.3 distribution, featuring a Linux 4.11 kernel.

4.1.1 Status

Along with the hardware platforms, a Linux distribution supporting the hardware and providing the HEI API has been delivered for both “TigerShark” and “Osprey” systems.

4.2 Predictor

The safe voltage (and refresh rate for DRAM) predictive mechanism will be based on the StressLog runs. The Predictor will drive the StressLog to execute specific workloads that stress the CPU, SoC and DRAM for various active core mappings (simply put the ids of active cores) and frequencies under various voltage/refresh rate levels. The end result will be a predictor state that will hold the safe voltage for various frequencies and active core mappings. Due to high number of active core mappings (255 for 8-core X-Gene2), to minimize the StressLog execution time we propose to characterize only the V_{MIN} of: a) the full utilization scenario (8 cores), b) each core alone, and c) each PMD alone fully utilized (a PMD holds two cores). Even though this approach trade-offs some energy-efficiency potential for faster V_{MIN} characterization, focusing the V_{MIN} predictions on fully utilized and isolated core executions has provided significant energy efficiency improvements on Intel CPUs as shown by partners [10].

If the system continuously asks for a non-characterized core mapping, this core mapping can be characterized in the next StressLog run to determine a safe voltage for that particular core mapping for improving the energy-efficiency. For determining the safe voltage of a core mapping, we will characterize the worst-case scenario. For instance, if the goal is to find the safe voltage when running 4 cores in clustered configuration (for X-Gene 2 this translates to two fully utilized PMDs and two fully idle PMDs), we will choose the two most unreliable PMDs and perform the V_{MIN} characterization on them. The discovered voltage will be used each time the hypervisor requests a voltage for any mapping with 4 clustered cores. This approach provides some safety during execution and reduces training time, but some energy-efficiency improvement opportunity is lost. With that said, for enhanced energy-efficiency savings we are currently working on implementing a linear regression model based on performance counters that will allow to lower the voltage even more depending on the active core mapping and active workload. Partners showed that such model works with success on Intel CPUs. Furthermore, we are currently investigating an anomaly detection mechanism. This mechanism will gather live monitored performance counters and other sensors data and will attempt to detect an anomaly, and if such anomaly exists the voltage/refresh rate margins would have to be reset back to nominal.

4.2.1 Status

The interfaces between the Predictor and other software components (Hypervisor, Stress Monitor) have already been defined and implemented. The first implementation of the Predictor module is finished with the Predictor gathering information from the StressLog and creating a database with virtually zero pfail (probability of failure) voltage for different active core mappings and frequencies. These values can be returned to Hypervisor when requested.

4.3 Hypervisor

The hypervisor stack layer from a top-down view, consists of Libvirt [16], QEMU, the KVM module and the rest of the Linux kernel. It is the connection point between the high level (OpenStack) and low-level components (Predictor, StressLog, HealthLog) of UniServer.

© 2019. UniServer Consortium Partners. All rights reserved

Looking upwards the software stack as Figure 4 shows, the communication entry point for the node is Libvirt.

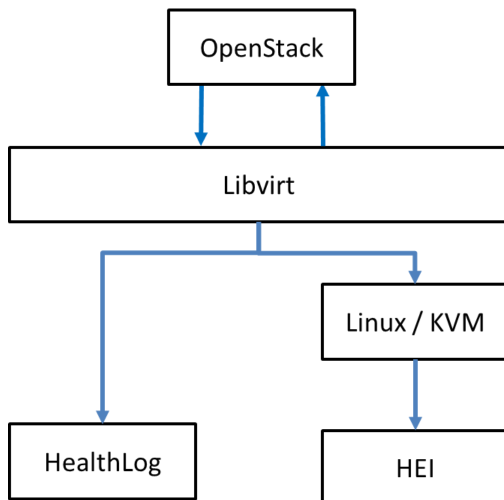

Figure 4: Hypervisor - looking upwards the software stack

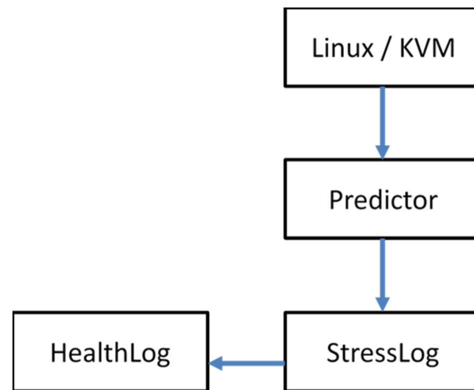

Figure 3: Hypervisor: looking downwards the software stack

Libvirt invokes QEMU, the user-space half of the hypervisor which in turn utilizes KVM, the kernel-space half responsible for the low-level interaction between the virtual machine and the host machine. KVM is part of the Linux kernel and tightly dependent on the internal structures of it. Therefore, we present KVM and Linux kernel as one software entity. All these modules are standard components of the Linux virtualization ecosystem that have been extended where needed to support the demands of UniServer.

On top of the hypervisor layer, Libvirt is a hypervisor-independent virtualization API and toolkit supporting a range of operating systems enabling virtualization. Libvirt is used to bridge the communication between hypervisor and upper software layers such as OpenStack which is discussed below. The Linux kernel harvests any useful information from either HealthLog, HEI or standard OS interfaces (sysfs, procfs).

Looking downwards the software stack as Figure 3 shows, the Linux kernel coordinates the lower level components (as discussed in previous sections) and they in turn, assist it on making decisions.

4.3.1 Status

We have extended the C and Python API of Libvirt to receive requests and propagate node related information from / to OpenStack to satisfy the requirements of UniServer as described in D5.3. Also, the integration and bidirectional information flow with Health Monitor and the Predictor has finished.

The hypervisor layer relies on the Health Monitor 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 and/or OS & Hypervisor functionality can be migrated to reliable cores. Memory can also be partitioned into reliable and unreliable domains. The hypervisor itself will be restricted to allocating memory only from the reliable domain.

Regarding the Linux kernel level of the software stack, a number of techniques and approaches have been implemented and characterized w.r.t. the CPU resource, the energy consumption and worst-case behaviour in the presence of a transient / permanent hardware fault that may disrupt the execution on the relaxed domain.

System call migration, page fault migration and scheduler migration are the main techniques we focused on during our research in order to proactively improve the relaxed capabilities of the Linux kernel. Apart from the target architecture and platform, namely arm64 on the TigerShark and Osprey boards, we have also tested this migration idea on the x86 architecture. Besides the academic interest, the benefits of this technique were outweighed by the performance overhead which lead to increased energy consumption in both boards and both architectures. Also, in the worst-case scenario, where a hardware fault manifests during operating at extended margins, the operating system was unresponsive without a chance for applying any reactive checkpoint / restore scheme in both the vanilla and our extended version of Linux kernel.

© 2019. UniServer Consortium Partners. All rights reserved

Interrupt pinning and source isolation are two common available approaches to mitigate critical software functionality from possible hardware errors. Both approaches are orthogonal to our developed techniques during the research phase and are mostly a matter of system configuration.

Selective point of operation w.r.t. code criticality e.g. between user-space and kernel-space is difficult to flourish due to TigerShark and Osprey board limitations such as high latency of voltage change through the existing low-level API. Considering all the trade-offs, deploying a production system, we decided to integrate the vanilla-based version of Linux kernel with its existing relaxed capabilities, leaving any migration techniques off the final release. Improving the Linux kernel as a monolithic and complex entity to reliably run on more aggressive extended margins needs further research and testing in our opinion.

In addition, we have designed and implemented two CPU governors that the hypervisor can utilize to improve energy efficiency of the system or limit the CPU power consumption under a power consumption cap.

The first governor supports only x86 architecture and targets to improve energy efficiency of the CPU at the granularity of the exciting workload. Specifically, our characterization process of various x86 chips shows that a safe underscaled voltage point (V_{min}) – no manifestation of any kind of error that leads to a system crash – depends on the exciting workload. Based on this observation, we extracted and associated key performance characteristics for a representative set of workloads with their corresponding V_{min} and thus created a model that can provide safe underscaled voltage points for any kind of workload. To this end, we deployed a governor that constantly monitors these performance counters of the system, passes them in our model and applies the new V_{min}, that our model suggests, to further improve the energy efficiency of the CPU.

The second governor supports both arm64 and x86 architectures and targets to minimize the performance penalties when the CPU operates under a tight power cap. Specifically, conventional CPU power capping mechanisms, like Intel's R.A.P.L, limit the CPU power consumption under a cap, by scaling only the frequency operating point of the CPU. Eventually, this is translated as a performance penalty to the exciting workload. Our governor exploits voltage underscaling and targets to meet the same power caps as conventional mechanisms, but with significant higher frequency operating points. This is achieved because our governor firstly reduces the voltage operating point of the CPU and if the power consumption reduction is not enough to meet the applied power cap, only then reduces the CPU operating frequency.

Furthermore, in order to protect the system from catastrophic errors from memory, we have developed a mechanism that can provide elevated memory protection to critical OS and application software data, while allowing less critical application data (e.g. heap) to be stored within memory domain(s) whose energy-efficiency and reliability could be adjusted. We call our protection mechanism, heterogeneous-reliability memory framework.

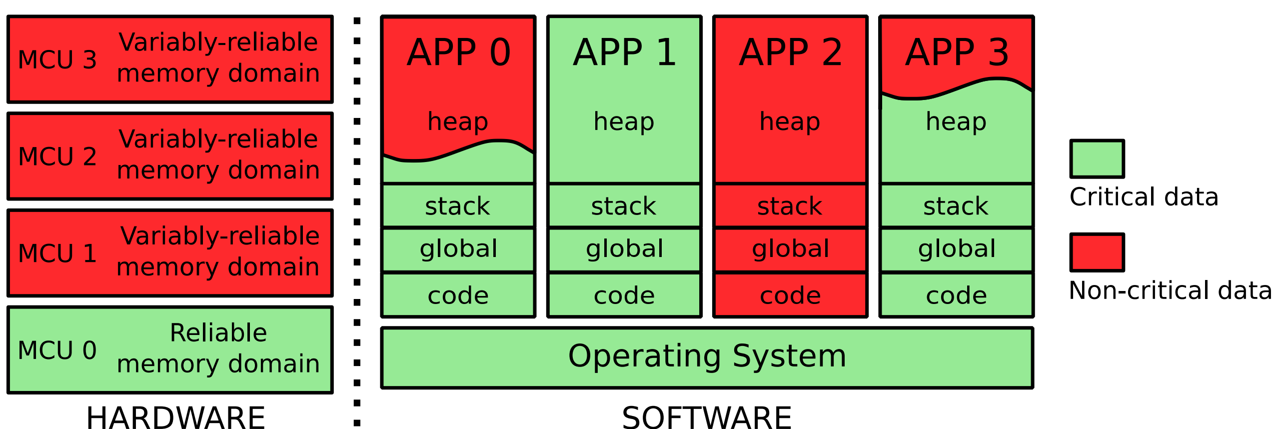

Figure 5. Overview of memory allocation for application running on heterogeneous-reliability memory framework

Figure 5 presents a heterogenous-reliability memory framework and examples of applications allocating memory in this framework. Particularly, in this example, APP 1 and APP 2 assign all the application data is to either the RelMem domain (the reliable domain) or the VarMem domain (the domain with varying reliability).

© 2019. UniServer Consortium Partners. All rights reserved

To implement such a framework, we disabled memory interleaving on X-gene2 servers, which enables us to allocate data on a specific Memory Controller Unit (MCU) and thus specific memory domain (RelMem or VarMem). We are utilizing the 4 Memory Controller Units to divide the available address space into 4 separate memory domains, whose reliability could be controlled independently by adjusting the supply voltage and refresh rate depending on the criticality of the stored data. To enable the application control data allocation on RelMem and VarMem, we introduce 4 fake NUMA (Non-Uniform Memory Allocation) domains, each of which is assigned to the physical memory space that belongs to a specific MCU. We modified the standard NUMA interfaces to allow the application allocate data on RelMem or VarMem. Specifically, we allocate data for application using the numactl command; the parameter –membind can be passed to numactl to define a NUMA memory domain that will be used for data allocation.

However, by disabling memory interleaving, we introduce an average performance overhead of 49.39%, as the memory bandwidth is not fully exploited. To reduce this performance overheads, we implement a software interleaving technique that decreases the average overhead down to 7%, while providing 9% energy consumption reduction when relaxed memory parameters are applied.

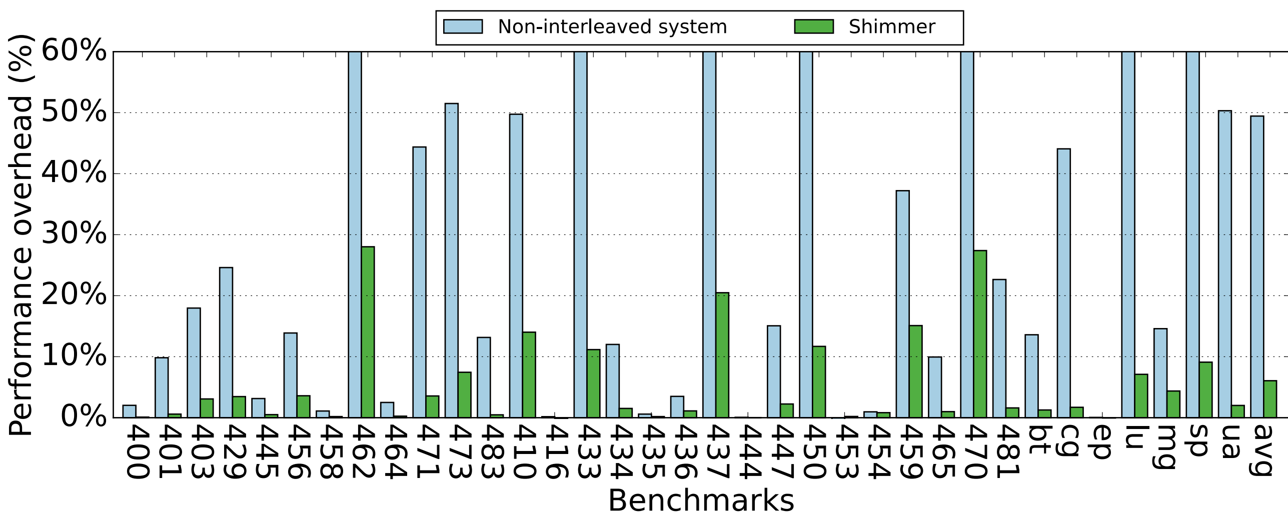

Figure 6. Performance overhead manifested for memory with disabled interleaving (Non-interleaved system) and memory with enabled software-level interleaving (Shimmer)

Figure 6 presents the performance overhead introduced when we disable interleaving (Non-interleaved system) and enable software-level interleaving (Shimmer) compared to the baseline configuration for the SPEC and NAS benchmarks. We calculate this overhead by measuring the execution time of each benchmark for the two configurations normalized to the execution times obtained for the default server configuration. Note that, in this figure, we use benchmark identification numbers instead of the whole SPEC benchmark names.

We modified KVM and QEMU to enable running VMs on heterogenous-reliability memory, allocating all the VM data on VarMem. Our modifications allow the user either to run VMs allocating all the data on VarMem, or allocate on VarMem only the data of an application running within a VM.

We had initially implemented our framework on the x86 architecture by utilizing the different sockets of a server to differentiate the memory domain. The x86 framework was mainly used for characterization of the DRAM as there were considerable performance overhead. Our modified hypervisor is extended with the heterogeneous-reliability memory framework for the arm64 architecture on the Tigershark and Osprey boards.

4.4 OpenStack

OpenStack is the cloud management framework selected for handling virtual machines within the UniServer environment. OpenStack enables the datacentre owners to control the whole lifecycle of VMs (creation, deletion, migration etc.), networking between VMs and monitoring for the whole infrastructure. The monitoring part is related to compute nodes resources availability, and performance metrics. Among all the components in OpenStack the following have been extended to benefit from the functionalities enabled by UniServer:

© 2019. UniServer Consortium Partners. All rights reserved

- Ceilometer: OpenStack telemetry system, capable of monitoring virtual and physical metrics.
- Nova: responsible for handling the lifecycle of virtual machines, from creation to destruction; it also handles the physical resources on each compute node assigning them to VMs.
- Horizon: web-based user interface used to interact with OpenStack installation. The relationship between the hypervisor and OpenStack is shown in the following diagram:

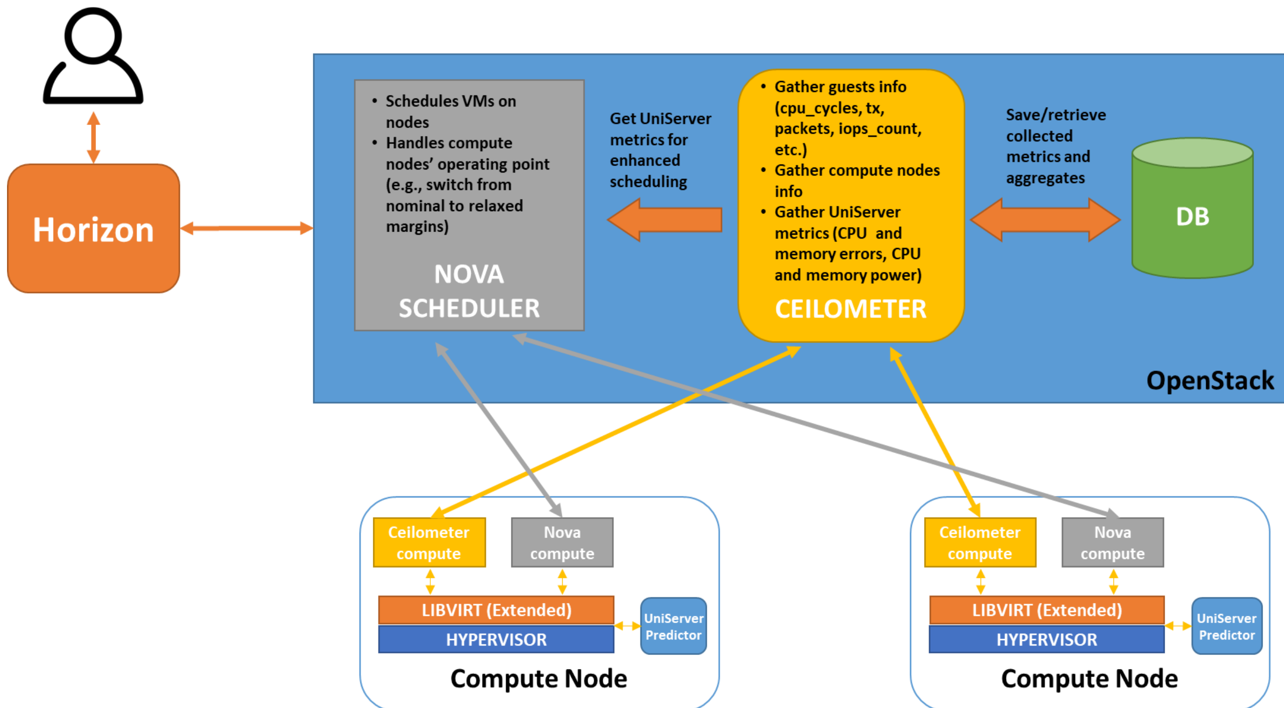

Figure 7: Hypervisor and OpenStack relationship

The integration of the extended Ceilometer component was discussed already in [13]. With respect to the previous integration report OpenStack Nova and Horizon were the focus of development activities. Figure 7 shows how the extended Nova and Horizon components interact with each other and with the rest of the UniServer software stack.

Two of the Nova sub-components were extended for UniServer: the scheduler and the compute node manager. The scheduler is in charge of accepting VM creation requests and, by observing the current status of the compute nodes, schedule VMs according to their resource requirements (e.g., memory, VCPUs, disk, etc.). The extension to the scheduler aims at reducing the power consumption of the data centre by leveraging the relaxed operating mode of UniServer nodes when scheduling VMs. According to the VM priority (translated into revenue generation) the scheduler selects the destination node, trying to maximize the number of nodes running in relaxed margins and minimize the probability of failure. The Nova compute node manager was extended to implement de-centralized pro-active fault tolerance by using the predictor component of the UniServer software stack. Each compute node is autonomously polling the predictor accessible through the hypervisor to estimate how safe is the current operating mode. If a node is self-identified as unsafe all the high-value VMs are migrated to safer nodes. This functionality enables more scheduling policies to be implemented for further power saving. The combination of the extensions to the scheduler and compute node manager aims at a further reduced energy footprint of a cloud datacentre without impacting the service level perceived by the user.

The web-based user interface, Horizon, was extended accordingly to present the user with the enhanced information available for UniServer compute nodes.

A comprehensive description of the above extended components is available in [14] and [15].

© 2019. UniServer Consortium Partners. All rights reserved

4.4.1 Status

The extended Nova and Horizon components were successfully tested on two X-Gene2 based “Merlin” boards. The extended Nova is able to schedule VMs according to the current operating mode of the two nodes. Nova is also able to switch the operating mode of a node by directly using the UniServer extended Libvirt interface. UniServer related data is exposed to the user via the Horizon dashboard. The above components are also integrated with the previously discussed Ceilometer extensions providing enhanced metering capabilities, enriched with the unique set of information provided by a UniServer system.

© 2019. UniServer Consortium Partners. All rights reserved

5. Application Layer

There are three target applications for UniServer, listed in Table 3.

Application	Provider
Wireless jamming detection	WSE
Polaris Regulatory Reporting Platform	Meritorious
Social Network Server SocialCRM/SocialTV	SPA

Table 3: UniServer target applications

The applications do not have any specific interfaces to the UniServer system; they remain agnostic about the type of platform they are executing on.

5.1 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.

5.1.1 Status

The WSE Wireless Jammer Detector application has been the first priority. It was the showcase of the first UniServer demonstration, performed during project month 22 (November 2017). A few minor adjustments were made to the application since, and it has been tested during the third year of the project. More in-depth characterisation is currently underway, and the results will be reported in the deliverables D7.8 and D7.9

5.2 MER: Polaris Regulatory Reporting Platform

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.

Polaris was developed to support our clients on the obligation, to the EMIR, for all EU counterparties to derivative transactions to report such transactions to Trade Repositories (TR) – entities licensed and regulated by the European Securities and Markets Authority (ESMA) who shall collect and maintain the records of all derivatives trade-related data.

The reporting of a derivative transaction should take place at T+1 day, where T is the date where the transaction was executed, and it involves any daily modifications/updates of the transaction until the termination of the derivative contract.

© 2019. UniServer Consortium Partners. All rights reserved

- The complete solution of Polaris platform provides validation of the reporting and speedups the process as compared to reporting directly to Trade Repositories.
- The platform provides additional compliance checks in the reported trades to ensure the validity of the reported information also from the compliance aspect.

5.2.1 Status

The Polaris Platform has been completely migrated to the ARM based UniServer platform and in addition, the Micro Polaris Benchmark was developed, as an open source approximation of the original platform, to be distributed with the StressLog application.

5.3 SPA: Social Network Server SocialCRM/SocialTV

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

The **social network server** component which runs on a large data center

The **social analytics components** that run on client's premises and which process data from the server regarding specific subscribed events

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

5.3.1 The social network server component

The social network server represents a server serving a typical social network such as Facebook. It implements queries to post content (e.g. messages or photos), retrieve friends, add interests etc. plus other more complex social network queries for analyzing user's behavior that are useful for the social network provider. The social network server is tested using the LDBC Social Network Interactive benchmark (LDBC-SNB), which is a state-of-the-art benchmark for linked data technologies, for which an implementation is provided.

LDBC-SNB 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.

The goal of the social network server component is to test and understand how the underlying technology graph database technology behaves on the UniServer platform on workloads like the ones modeled by the benchmark, which can fit both the large-scale server the smaller edge computing use cases.

5.3.2 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.

The goal of this component is to test how the underlying technology behaves in workloads that involve executing expensive graph analytics algorithms such as community detection, influencers detection, or interest

© 2019. UniServer Consortium Partners. All rights reserved

similarity for recommendation engines based on collaborative filtering. Such algorithms are typically run periodically on the latest available data, in batches.

5.3.3 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. Although these components will be delivered for visualization purposes, the goal is not to test them as these are out of the business of what SPA provides as a technology provider.

5.3.4 Status

We have provided all the necessary stuff to test the social network server, including preloaded datasets, scripts to install and run the server as well to parse and understand the results. The implementation of the social network server is provided here (https://github.com/DAMA-UPC/ldbc-sparksee), while the UniServer related stuff (DockerFiles, scripts to download data, Sparksee license etc.) can be found here (https://hpdc-gitlab.eeecs.qub.ac.uk/aprat/uniserver-ldbc-sparksee). Additionally, a preloaded VM with the application has been provided. Finally, note that an ARM compatible version has been developed for this Project.

We have developed the two social analytics components, the SocialCRM (community detection, influencers identification) and SocialTV (recommender engines based con collaborative filtering). Such components can be found in (https://hpdc-gitlab.eeecs.qub.ac.uk/Jordi.Urmeneta/uniserver-spa-socialcrm) and (https://hpdc-gitlab.eeecs.qub.ac.uk/Jordi.Urmeneta/uniserver-spa-media) respectively, which not only provide the application but also DockerFiles and scripts to run them in containers. Additionally, the SocialCRM app has been integrated within a VM to be easily distributed among partners

.

© 2019. UniServer Consortium Partners. All rights reserved

6. Conclusions and Future Work

This report deliverable has described the developments and enhancements of the UniServer system at the hardware, system, and application layers in the period between month M21 and M31. The modules at each layer have been discussed, as well as the level of integration. The remainder of the project will focus on preparing all components on the target hardware, both X-Gene2 and X-Gene3 based platforms, for the final UniServer project presentation.

© 2019. UniServer Consortium Partners. All rights reserved

7. References

[1] D5.1 1st Report on Hypervisor /System / Software Interface

[2] D5.3 2nd Report on Hypervisor / System / Software Interface

[3] D6.1 OpenStack Support for UniServer

[4] D4.2 HealthLog Specifications and Interface

[5] D4.3 StressLog Specification and Interface

[6] D4.1 Hardware Exposure Interface (HEI) and Error Handlers Specification

[7] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs", IEEE/ACM International Symposium on Microarchitecture (MICRO 2017), Cambridge, MA, USA, October 2017.

[8] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, "HealthLog Monitor: A Flexible System-Monitoring Linux Service", IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS 2018), Costa Brava, Spain, July 2018.

[9] G. Papadimitriou, A. Chatzidimitriou, M. Kaliorakis, Y. Vastakis, D. Gizopoulos, "Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs", IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2018), Belfast, Northern Ireland, United Kingdom, April 2018.

[10] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, C. Magdalinos, and D. Gizopoulos, "Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors", IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS 2017), Thessaloniki, Greece, July 2017.

[11] D3.3 1st Analysis of On-Chip Caches and Dynamic Memories

[12] D3.6 2nd Analysis of On-Chip Caches and Dynamic Memories

[13] D7.3 First Vertical, Full, System Integration and Validation Report

[14] D6.3 OpenStack Resource Manager for UniServer

[15] D6.5 OpenStack Resilience on Extended Margins Micro-Servers

[16] libvirt, "implementing a new API in libvirt," [Online]. Available: http://libvirt.org/api_extension.html.

[END OF DOCUMENT]

© 2019. UniServer Consortium Partners. All rights reserved