LOw Power Heterogeneous Architecture
for NExt Generation of SmaRt Infrastructure and Platforms
in Industrial and Societal Applications

Power-aware Cloud Model and Management Intermediate Report

Co-funded by the Horizon 2020
Framework Programme of the European Union

DELIVERABLE NUMBER	D5.2
DELIVERABLE TITLE	Power-aware Cloud Model and Management - Intermediate Report
RESPONSIBLE AUTHOR	IBM

GRANT AGREEMENT N.	688386
PROJECT REF. NO	H2020- 688386
PROJECT ACRONYM	OPERA
PROJECT FULL NAME	LOw Power Heterogeneous Architecture for Next Generation of SmaRt Infrastructure and Platform in Industrial and Societal Applications
STARTING DATE (DUR.)	01/12/2015
ENDING DATE	30/11/2018
PROJECT WEBSITE	www.operaproject.eu

WORKPACKAGE N. | TITLE	WP5 | Optimized Workload Management on Heterogeneous Architecture
WORKPACKAGE LEADER	IBM
DELIVERABLE N. | TITLE	D5.2 | Power-aware Cloud Model and Management - Intermediate Report
RESPONSIBLE AUTHOR	J.Nider (IBM)
DATE OF DELIVERY (CONTRACTUAL)	31/05/2017
DATE OF DELIVERY (SUBMITTED)	01/06/2017
VERSION | STATUS	V1.0 | Final
NATURE	R(Report)
DISSEMINATION LEVEL	PU(Public)
AUTHORS (PARTNER)	NALL, TECH, ISMB, CSI

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First Draft	01/03/2017	J.Nider (IBM)
0.2	Second Draft	20/05/2017	J.Nider (IBM)
0.3	Integrate changes from reviewers	31/05/2017	J.Nider (IBM), P.Ruiu, A.Scionti (ISMB), G.Renaud (HPE)
1.0	Final Release	01/06/2017	J.Nider (IBM)

PARTICIPANTS		CONTACT
STMICROELECTRONICS SRL		Giulio Urlini Email: Giulio.urlini@st.com
IBM ISRAEL SCIENCE AND TECHNOLOGY LTD		Joel Nider Email: joeln@il.ibm.com
HEWLETT PACKARD CENTRE DE COMPETENCES (FRANCE)		Gallig Renaud Email: gallig.renaud@hpe.com
NALLATECH LTD		Craig Petrie Email: c.petrie@nallatech.com
ISTITUTO SUPERIORE MARIO BOELLA		Olivier Terzo Email: terzo@ismb.it
TECHNION ISRAEL INSTITUTE OF TECHNOLOGY		Dan Tsafrir Email: dan@cs.technion.ac.il
CSI PIEMONTE		Vittorio Vallero Email: Vittorio.vallero@csi.it
NEAVIA TECHNOLOGIES		Stéphane Gervais Email: s.gervais@lacroix.fr
CERIOS GREEN BV		Frank Verhagen Email: frank.verhagen@certios.nl
TESEO SPA		Stefano Serra Email: s.serra@teseo.clemessy.com
DEPARTEMENT DE L'ISERE		Olivier Latouille Email: olivier.latouille@isere.fr

ACRONYMS LIST

ABI	application binary interface
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
IaaS	Infrastructure-as-a-Service
ISA	Instruction Set Architecture
VM	Virtual machine

LIST OF FIGURES

Figure 1 - NAND flash memory transistors in silicon....11
Figure 2 - High level diagram of the heterogeneous setup in the testbed 15
Figure 3 - OpenStack and its modules managing a set of heterogeneous compute nodes with a mix of containers and VMs....16

EXECUTIVE SUMMARY

Work Package 5 (WP5) aims at providing a mechanism to efficiently deploy applications on the data centre resources. The "efficiency" expected from the proposed mechanism strictly concerns the energy consumed by the data centre hardware resources and the relative performance of the various tasks composing applications running on the data center. Providing this mechanism requires to take into account several elements that influence the runtime execution of such tasks. For instance, a model to evaluate the best mapping of tasks with specific host architectures, a mechanism to select the best host among various with the same architecture where to run the tasks, and a way of optimizing the whole data center workload should be integrated.

T5.2 aims to show specifically how to take advantage of a heterogeneous ISA data center architecture to save power while keeping the same level of performance delivered today. By using techniques such as container migration and workload monitoring, we can keep the data center operating at top efficiency even in the face of changing applications and workloads.

Position of the deliverable in the whole project context

D5.2 reports on the efforts of T5.2 – power aware cloud model and management. Here we benchmark a system built with the various improvements in cloud hardware and software developed under OPERA. Thus T5.2 requires the input from other tasks in WP5, such as the workload decomposition (containerization) developed in T5.1, and the scheduler/orchestrator changes (including the monitoring module) that are developed in T5.4 and T5.5.

We also rely on the methodology set out in WP4, specifically T4.3 that is to be released in M28.

Description of the deliverable

The deliverable describes cloud data center architectures, and how their design influences power consumption. We show how performance is closely related to power consumption, with efficiency at various levels being the major factor in determining how much computation can be performed for a given amount of power.

By carefully choosing the hardware to match the problem, we can use existing hardware that has better efficiency for a particular piece of software and workload, thus improving efficiency.

List of actions and roles

LIST OF ACTIONS					
ACTIVITIES LIST AND PARTNERS ROLES	IBM	ISMB	NALL	CSI	HPE
Summary of the initial requirements	P	P	I		P
Planning the testbed	I	P	R	P	I
Selection of cloud applications	P	P	R	P	I
Writing of the deliverable	P	P	R		I

- P = Participating (includes I & R)
- I = Input delivery (Includes R)
- R = review

IBM leads the work package and the task and led the writing of the deliverable. IBM's work is focused on the cross-ISA container migration mechanism, which is composed of a cross-ISA compiler, and post-copy migration mechanism in Linux for containers (part of the CRIU project).

ISMB is responsible for the orchestration software and any modifications required to the scheduling of the various applications. They aid in the coordination and planning of the test bed, and contributed to the writing of the deliverable, as well as reviewing it for correctness.

NALL provides input in to the requirements of building the test bed. They also review the various stages of the project, and help guide the planning of the experiments.

CSI is responsible for the design and construction of the test bed. They will also help perform the testing once the development is complete.

HPE reviewed the deliverable, and gave valuable comments throughout the process. They are also responsible for the Moonshot components of the test bed, and are instrumental in coordinating with CSI and NALL.

TABLE OF CONTENTS

1 INTRODUCTION 10
2 POWER VS PERFORMANCE 11
2.1 HARDWARE 11
2.2 SOFTWARE 12
2.3 EFFICIENCY 12
3 CLOUD DATA CENTER ARCHITECTURES 13
3.1 HOMOGENOUS 13
3.2 HETEROGENEOUS 13
3.3 CONSTRUCTION & MAINTENANCE 13
3.4 EFFECTS 13
3.5 RESOLUTION 14
4 MEASUREMENTS 15
4.1 TESTBED 15
4.2 ORCHESTRATION SOFTWARE 16
4.3 METHODOLOGY 16
5 CROSS-ISA MIGRATION 17
5.1 CONTEXT MIGRATION 17
5.1.1 Virtual Machine Migration 17
5.1.2 Migration to Acclerators 17
5.1.3 Container Migration 18
5.2 CROSS-ISA COMPILER 18
5.2.1 Code Section 19
5.2.2 Optimizations 19
5.2.3 Equivalence Points 19
5.3 FUTURE WORK 19
6 CONCLUSIONS 21
7 REFERENCES 22

1 INTRODUCTION

This deliverable reports on the progress and planning related to task T5.2. The aim of task T5.2 is to investigate if it is more efficient to use cloud services in a homogeneous or heterogenous data center. It is known that cloud services provide more efficient solutions in terms of energy consumption over the more traditional client/server model. However, we don't know if a distributed SaaS application can take advantage of heterogeneous compute resources in a way that reduces energy consumption. Simulation of a set of cloud applications will be performed in order to compare the different models. We will compare our results measured on our Proof-of-concept heterogeneous cluster to published numbers from 2013.

Data centers are commonly built with thousands of machines, sometimes as large as 100,000 machines. This number of computers along with the related equipment (networking, air conditioning, lighting, etc) draws a huge amount of electricity. Not only is the electricity a large operating cost, but people also have concerns about the environmental impact of generating such large amounts, and the methods used to do so.

So for several reasons, it is a worthwhile goal to determine the best way for us to run this expensive equipment. If we can improve efficiency, we may be able to save costs, and even reduce environmental impact while still maintaining the level of service and compute power being demanded by a growing population.

Our proof-of-concept is to be performed on a much smaller scale. It is not practical to build a large scale heterogeneous data center only for the purposes of testing, so we will implement only a small portion of a data center, with the workload scaled accordingly to match the amount of compute resources available. The study aims to determine if there is a benefit from using heterogeneous resources, and if so, what kind of benefit we can expect if scaled up to the size of a common data center.

2 POWER VS PERFORMANCE

2.1 HARDWARE

When designing and analyzing computer hardware, power and performance are equivalent concepts (if we put efficiency aside for a moment). This is an important fact to realize in order to reason about how to reduce power consumption, while still being able to provide necessary services. It is necessary to understand the entire system (at least at a rudimentary level) including hardware and software in order to correctly determine which variables are important, and the weights that should be given to each variable. The following is a high-level explanation of the concept, without rigorous proofs, to give the correct intuition as to how to approach understanding the problem of analyzing energy efficiency with the hopes to improve it.

Computer logic hardware is built from transistors in a silicon die, which consume energy to perform various functions (such as logical operations like *or*, *and*, *xor*, etc.). These transistors have an associated energy consumption (known as dissipation) which includes the energy required to make the calculation as well as some energy lost to leakage (including heat). Each transistor takes up a known amount of silicon in the die, which is known as its *footprint*. The dissipation and footprint are related to the silicon process, which indicates the physical characteristics of the transistors, including size, switching speed, and many more. More advanced processes generally produce hardware that consumes less energy, allows for faster switching speeds, and smaller transistors (higher density) on the die. The advancement of silicon manufacturing processes is the main contributor to the increase of transistor density and decrease in power consumption per transistor in subsequent generations of chips. This has been studied extensively, and is well documented [1] [2].

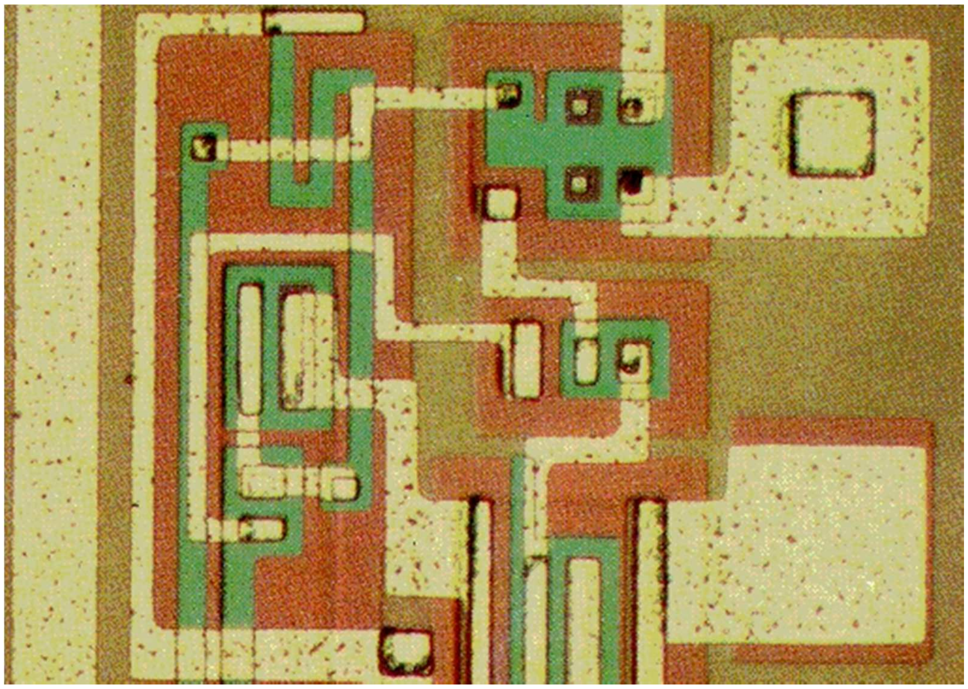
Figure 1 - NAND flash memory transistors in silicon

In order to operate as part of a system, each logic block receives a clock signal to help synchronize the various stages of computation. The clock is generated many times per second (generally hundreds of millions, or billions of times per second), and is known as the clock frequency. The effects of changing clock frequencies on power consumption have also been studied extensively, and are well documented [3]. Thus in order to gain more performance from a given piece of hardware, there are two general directions that can be taken: Increase the frequency, or increase the parallelism by adding more logical blocks (and therefore more transistors). It should be clear that in both cases, additional energy is required to obtain the performance improvements.

2.2 SOFTWARE

After this realization, we may now turn our attention to higher layers in the stack, namely the software. In terms of workload analysis, we may classify software into two broad categories: *batch software* - in which the software runs largely independent of the user, and *interactive software* - which waits for input from the user before it may act. In cloud computing, both types of workloads are present, and must be accounted for. In many ways, these different workloads may be viewed as complementary, and the two types may be scheduled together to increase efficiency. Since batch software does not rely on user input, it generally runs as fast as it can within the limits of a particular machine. Batch software can thus be further divided into *CPU-bound* and *memory-bound* workloads, according to the resource that limits the performance of the software.

2.3 EFFICIENCY

The unavailability of a particular resource while the demand is high is known as the limiting factor. Once a program has reached the limit on a resource, but the system still has an abundance of other resources is what create inefficiency when running computers, and this is where the opportunities lie for improving efficiency. Efficiency can be loosely defined as the amount of useful work being done divided by the amount of energy taken. If the amount of useful work is small relative to the amount of energy, the efficiency is low. It is possible that a computer system is built with inherent inefficiency (such as transistors with a high level of leakage) and it is possible that even very inherently efficient systems are used in an inefficient way. Let us look at the converse situation for an example. If we suppose that a certain batch job were to require all of the computer's resources - exactly 100% of the CPU, 100% of the memory, 100% of the disk and 100% of the network while running, we can consider this computer and workload to be perfectly matched, and to have no inefficiency (as long as the job is running). Unfortunately, this ideal situation rarely occurs. The fact is, even if a piece of software were to be perfectly tuned to a particular machine, it is likely that the software will pass through several phases during the execution, each requiring a different level of resources. That means that even a single piece of software can have changing requirements, while the hardware of a machine remains relatively static during its lifetime.

If we consider the case of interactive jobs, the situation is clearly worse. The purpose of interactive software is to serve the user on demand. If the user does not have a request ready, the software must sit idle waiting for a command from the user, but it must still be responsive as soon as a command is sent. The variation in demand on system resources in the case of interactive jobs is much larger than in the case of batch jobs. The load is harder to predict, since we can never know when a user will submit a request, and therefore utilization has large fluctuations (spikey).

To handle unpredictable loads, data centers are generally built to handle *peak load* - the largest load expected at any particular period in the lifetime of the data center. The problem is that the data center rarely reaches peak load, and the majority of equipment spends time idle, waiting to serve incoming requests. So if we compare average load to peak load, we can see that the larger the difference, the larger the inefficiency of the data center. Common data centers have reported numbers in the 20%-30% range of efficiency. Some specialized data centers (like at Google) may be considerably higher (reaching above 80%) but there are many smaller data centers that commonly run at only 10%-15% efficiency.

The scheduler plays an important role in load balancing, and this can have a profound effect on power consumption. This concept is explained further in D5.4.

3 CLOUD DATA CENTER ARCHITECTURES

We propose that the architecture of the data center is one of the sources of inefficiency. Not all workloads run well on all types of processors, and not all processors should necessarily be of the same size and strength. The following is an overview of data center architecture, and some description of the problems being faced today.

3.1 HOMOGENOUS

A homogeneous data center is one in which all compute nodes are identical. From the point of view of the scheduler (or orchestrator), all nodes are to be treated equally, and there are no significant differences that should be taken into account when making placement decisions. This is a goal that many data center operators strive to achieve. In section 3.3 we will provide an argument that explains why most data centers fail to achieve this ideal, and what we should do about it.

3.2 HETEROGENEOUS

A heterogeneous data center is exactly the opposite of a homogeneous one. It means the compute resources are different in some significant way, and implies that those differences should be taken into account during placement or scheduling. Sometimes the differences are minor, such as the generations of the processors (i.e. mixing Intel architecture Haswell and Broadwell based machines), or amount of physical memory. Sometimes the differences are more pronounced, such as the inclusion of FPGA or GPGPU compute resources, or the mixture of different ISAs (such as x86 and ARM). In the OPERA project, we are more interested in the both definitions (the more diverse the compute resources, the better chance of running workloads optimally).

3.3 CONSTRUCTION & MAINTENANCE

The composition of a cloud data center is dependent on several factors. Generally, an IT department develops a procedure for provisioning a new machine in the data center, which may include steps such as upgrading BIOS firmware, tagging and cataloging physical resources for inventory tracking purposes. These procedures must be documented and learned by the IT staff to be able to add machines to the data center as it grows. Any changes to this procedure costs time in documenting the change, retraining the staff, and errors that occur due to the change. Therefore, to minimize costs of building and maintaining the data center, it makes sense to keep a homogeneous class of machines. Unfortunately, this is not always possible. Assume for the moment that during the initial construction, and even for a year or two after the construction, the same physical machines are used for every compute node in the data center. After some time, some machines will fail and will need to be replaced. As time progresses, the likelihood of not being able to replace the machine with an identical model increases. After only a few years, the likelihood of finding the identical machine is practically zero. At this point, a very similar machine is substituted for the original. It is still a different machine, with different characteristics. Perhaps it has more memory, or more cores in the CPU, or the CPU speed is slightly faster. The data center has at this point become a heterogeneous data center, completely unintentionally.

3.4 EFFECTS

The world of HPC and supercomputing have struggled for years with the issue of how to guarantee homogeneous performance of a parallel job. We are facing a very similar problem in cloud data centers today when running Big Data or analytics jobs. Even though these are inherently batch jobs, we still face the issue of large inefficiencies, which are proving quite difficult to solve. If there are parallel jobs running in a heterogeneous data center (such as with Spark or Hadoop), each node completes its work in a different amount of time. Despite the hardware being identical (at least as close as practically possible) unavoidable and unpredictable external activity introduces entropy into the system. Such activity includes interrupts grabbing the CPU, and background tasks inside disks which cause some accesses to take longer. This leads to reduced efficiency as certain nodes will complete their processing faster, and be forced to wait for slower nodes. As a second example, sometimes heterogeneity is simply ignored, and VMs running on two different physical machines are sold as identical compute resources. If the data center is leasing compute resources (like an IaaS cloud) customers may notice that sometimes they get a “fast” virtual machine and sometimes they get a “slow” virtual machine, despite the technical specifications being identical (i.e. both being ‘large’ instances of a VM on Amazon EC2). In both of these cases, heterogeneity is being perceived as a drawback by both the cloud operator and the customer (even if the customer is internal).

3.5 RESOLUTION

So from all of the aforementioned reasons, it seems that heterogeneity is something that data center operators should fear, and avoid at all costs. The initial reaction is then to abstract away the differences in the management software, and try to make all of the nodes look the same. We believe that is not possible to achieve, and in fact the differences should be embraced by including knowledge in the orchestration software. It is clear that we need a new approach to managing the data center hardware not only so that we are aware of the differences and can deal with them, but so that we can actually take advantage of the differences, and gain from them. Heterogeneous compute resources do incur a cost (installation procedures, more complicated scheduling decisions) but cloud operators can also benefit from them with improved energy efficiency and improved resource utilization. The rest of the deliverable deals with the technical issues of how to modify the cloud orchestration software to be aware of the heterogeneous nodes, to be aware of the requirements of the workloads, and to correctly match them to improve efficiency.

4 MEASUREMENTS

In order to test our hypothesis, we must build 3 systems that can be compared. We have chosen to reuse the VDI testbed for this purpose, since it matches our system requirements, and is already being built. We want to test our hypothesis on 3 systems:

1. "legacy" system - composed of a set of older homogeneous machines from ca. 2013
2. "modern" system - composed of a set of modern homogeneous machines from ca. 2016
3. "hetero" system - composed of a mix of modern heterogeneous machines

4.1 TESTBED

The testbed is described in detail in deliverable D2.3 section 3.1. For convenience, here is a summary of the testbed environment in place at CSI.

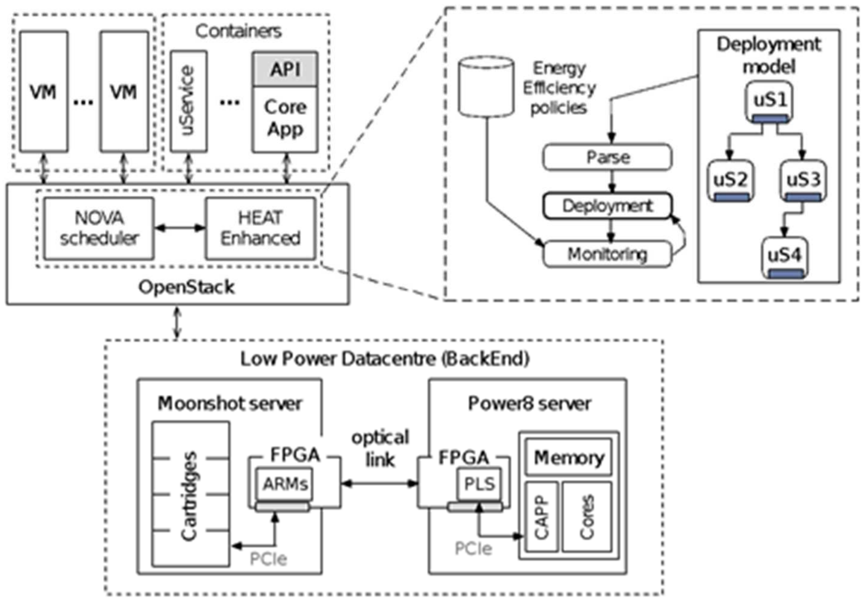

Figure 2 - High level diagram of the heterogeneous setup in the testbed

The "hetero" backend is composed of a set of heterogeneous servers, namely HPE Moonshot and OpenPOWER. The Moonshot gives us the opportunity to test two different architectures (x86_64 and ARM64), while the OpenPOWER provides the third architecture (POWER ISA). These machines are connected over a standard 1Gb Ethernet link, and communicate over TCP/IP. For testing the container migration, we will prefer testing with a 10Gb Ethernet link, which is more in line with modern server hardware.

We will benchmark the "hetero" system described above against a second system built from only one ISA (the "modern" system). These include servers of a type which has been deployed recently. This is meant to represent the state-of-the-art in data centers, and will help us determine what progress has been made in server hardware since the "legacy" system was deployed.

The "legacy" system will be used to show how well modern workloads (PaaS applications) would run on older machines, and we use this to show the culmination of changes in hardware by moving from a previous-generation system (which is still deployed, but known to be inefficient) to an experimental system (which has not yet been deployed).

4.2 ORCHESTRATION SOFTWARE

The orchestration software we have chosen is OpenStack. This is inline with the decisions made in D5.1 for integrating the modified TOSCA descriptor. These decisions are described in detail in D5.1, but a summary is shown here for completeness.

OpenStack is used to deploy the PaaS application, and manage its microservice components at runtime.

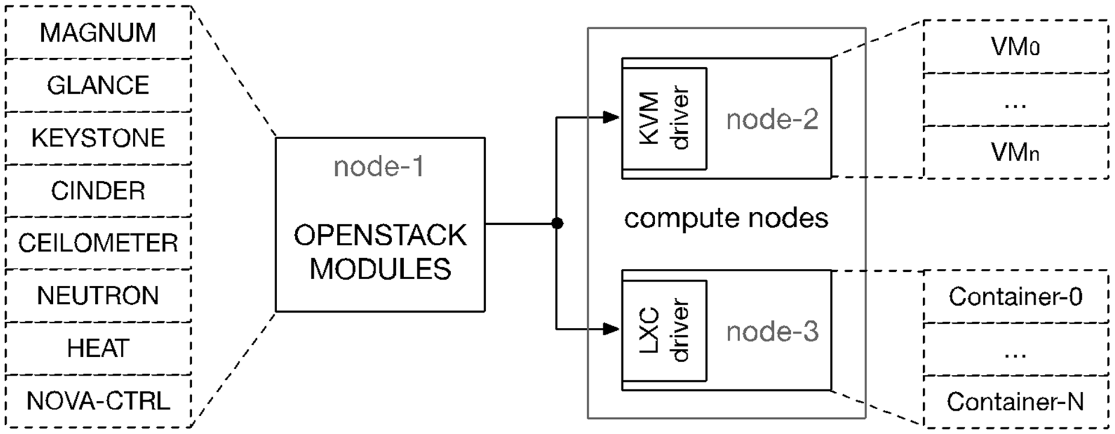

Figure 3 - OpenStack and its modules managing a set of heterogeneous compute nodes with a mix of containers and VMs

4.3 METHODOLOGY

The goal is to compare the use of cloud services in a homogeneous vs a heterogeneous data center. Our hypothesis is that we can take advantage of the variation in server architectures to reduce costs by lowering energy consumption, and improving efficiency. This is accomplished both through the use of modern hardware and modern software architectures. We aim to pinpoint exactly how much benefit each contributes to the overall effect of energy savings.

We will run a selected subset of cloud applications on all of the architectures, and take performance and power measurements. By analysing the results, we will be able to determine the improvements made by using various generations of hardware, and by moving to a heterogeneous setup.

By taking advantage of both software and hardware improvements, we will show the most marked improvement in overall power consumption vs performance.

5 CROSS-ISA MIGRATION

The orchestration layer (in our case, OpenStack) will not always make the correct decision during initial placement. Even if the decision where to place a new VM or container is correct, the system is dynamic and conditions may change so as to cause inefficiency (under-utilized machines, machines with not enough compute power, etc). The best policy is to constantly monitor the health of the system (by watching the efficiency of individual nodes) and to make modifications as needed from time to time to bring the system back into balance. In order to make these modifications to balance the system, several different techniques may be used depending on the technologies that have been deployed.

If microservices are in use (containerized application components), and the microservices have been designed to be scalable and resilient, it is possible that scaling up (creating more instances) or scaling down (destroying instances) of a given microservice is the best way to balance the load. In other cases, creating or destroying containers is not possible, and we must resort to migration. Virtual machines are are more heavy-weight example of an execution context, and the price for instantiating new VMs may be considerably higher than migrating an existing one.

5.1 CONTEXT MIGRATION

In the OPERA project, we look at how to implement migration in a heterogeneous data center. First, let us list the options that don’t make sense, and explain why they don’t make sense. We can then deal with the remaining options as viable, and look at which ones we should focus on first.

5.1.1 Virtual Machine Migration

In the case of machines with different ISAs (but still general purpose CPUs) VM migration between two machines may have merit. This however is a hard problem that we are not attempting to address in the OPERA project. Today, it is possible to migrate a VM to a machine with a different ISA by employing binary translation (such as QEMU [4]) which can emulate the target ISA and translate from the source ISA through software functions. This kind of emulation is at least an order of magnitude slower than running native code, which means it should not be used in performance-critical situations. Due to the amount of software running, it also consumes more energy than the native equivalent, and is therefore not efficient from several perspectives. This problem may be a target of future work, but it is currently beyond the scope of the OPERA project.

5.1.2 Migration to Acclerators

Migrating workloads to accelerators such as GPGPUs is a possible avenue that may show results in the future, but is currently out-of-scope for the OPERA project. As we explained in section 2, our definition of ‘heterogeneous’ is focused on multiple ISAs and different types of compute resources. Since virtualization is not supported on many types of compute resources (such as GPGPUs and in most cases, FPGAs) it does not make sense to migrate a virtual machine to such resources. It is likely that much more efficient methods of moving a workload to such compute resources exist, and should be preferred over VM migration in this case. Currently, we view these accelerators as dedicated resources. In WP3 for example, the video processing of Micmac is being offloaded to an FPGA, but this is static offloading. In that particular design, the FPGA cannot be reused for another workload, and cannot be shared among multiple applications, even if they were to require the same service. This approach shows promise, as the FPGA can be tuned to be far more efficient than the CPU for this type of processing. However, in WP5 we are focused on exploring the general case of programs that run on the CPU, and how to best utilize the various CPU resources available in the system at a particular time.

Even if we are to consider the case of “high level synthesis” languages such as OpenCL and CUDA, there is not yet a clear path for migrating general applications (or portions thereof) to such wildly different compute resources.

5.1.3 Container Migration

Container migration (and process migration as the generalized case) is a more relevant problem for which we can develop a solution. Containers are more relevant due to the advent of microservices, in which application are decomposed, and each component is isolated from the others by way of containers. Some containers contain services that cannot easily be replicated or scaled (such as in-memory databases), and the cost to migrate such containers is likely to be less than attempting replication. Moreover, container migration is performed at the system level, which means the application does not need to be aware of how to migrate itself, nor that the migration is even taking place. Thus we may provide migration support for such containerized applications that do not contain support for scaling or replication.

Deliverable D5.4 section 3.3 contains a more detailed description of the implementation of container migration within the context of the OPERA project. That section also contains a detailed description of the post-copy migration technique, which is used to reduce migration costs.

5.2 CROSS-ISA COMPILER

The key to supporting cross-ISA container migration lies in the compiler. In a technique pioneered at UCSD [4] and later refined at Virginia Tech [5], a standard compiler is modified to be able to generate an executable image with the property of being able to be migrated across ISAs. This magic is documented in their published papers, but a short summary follows for the reader’s convenience.

The overall idea is to create an executable that can run equally well on all ISAs. In practice, the technique is limited to the ISAs targeted during the compilation process, and for which support has been added to the toolchain. Each ISA today defines its own application binary interface (ABI) which specifies details of the linked image such as default padding of data structures, enums, the size of basic types such as int, etc. The details of these ABIs are all slightly different among the ISAs, but they all share common elements. In general, the hardware of the various architectures are able to execute programs that do not strictly adhere to these ABIs, but cannot guarantee compatibility and interoperability with library code (static or dynamic) that were generated according to the ABI. So from that knowledge, we draw two conclusions: 1) we may modify the ABI as long as is it adheres to the stricter set of rules defined by the ISA, and 2) we must recompile all code to ensure compatibility across various application components such as libraries. Armed with those assumptions, we can take a closer look at how an executable is defined and generated to see what needs to be modified. Executables are built in sections, where each section defines a particular use, such as whether the contents should be loaded into memory, at what address, and what its purpose is. For example, data sections are loaded to memory, and generally have a specific alignment. Code sections are also loaded to memory, but may specify that they are executable, and may have a different alignment requirement. If we know how to deal with the data and code sections, we have won most of the battle.

In order to align the data section, a new application binary interface (ABI) is defined as the superset of all the target ISAs. That means we force the padding and alignment of variables to follow the strictest ABI of the targeted ISAs. Luckily for us, the 64-bit ISAs all look very similar, and don’t show the wide variation that was apparent in 32-bit and 16-bit ISAs. That means smaller changes are required to support 64-bit ISAs. In addition, we must guarantee that the load address of every variable is identical across all target architectures to ensure compatibility.

5.2.1 Code Section

The code section is compiled separately for each architecture, using the existing compiler support for the particular architecture. The essential difference is that each function must start at a known address, and that address must be kept constant across all architectures. In practice, this is a two-stage process. First, all code is compiled for all architectures, generating object code (.o files) and assembly (.s files). Then, the code is analyzed by comparing the equivalent .o files for each architecture to make a map of function sizes and locations. The tool compares the different versions of a particular function (one function at a time) to determine the largest version of the function. The functions of the remaining target architectures are padded (by adding no-ops in the assembly files) so the function size becomes constant across architectures. The files are then recompiled to produce new .o files, in which the function size is guaranteed to be constant across architectures. This property is necessary for calculating equivalent offsets during migration between architectures.

5.2.2 Optimizations

A very high percentage of compiler optimizations are kept, and only a few optimizations need to be turned off in order to support cross-ISA compilation. This is largely due to optimizations that cut across multiple layers in the middle-end, and are architecture-specific optimizations that are applied too early in the compilation process. According to the results published by UCSD [4], these optimizations account for only a few percent of performance, and are negligible. We believe it would be possible to reimplement these optimizations at a later stage in compilation, thus making them applicable only once the correct architecture has been selected. These changes however are not an essential part of the OPERA project, and are not in scope for the project.

5.2.3 Equivalence Points

An *equivalence point* is the name given to a location of the program at which it is legal to migrate. It is called an equivalence point because we can guarantee a translation of the machine state from one architecture to another at this point in the execution. The equivalence points include all function start addresses (it is the easiest case to migrate at the start of a function) but also include multiple points throughout the function. We can consider all basic blocks to be a potential equivalence point (any code up to a jump or branch instruction), but there are several reasons why a basic block may not constitute an equivalence point, such as architecture specific optimizations that remove some state information necessary for the transformation to another architecture. Part of the future work is how to ensure enough equivalence points are generated, and experimentally determining how many are necessary for average cloud workloads.

5.3 FUTURE WORK

Our first implementation of post-copy migration is based on the CRIU tool [4], which is the mechanism responsible for performing the migration. CRIU is platform-agnostic, meaning it does not matter on which platforms the container is migrated. This is a necessary, initial first step to support post-copy migration, which we believe is key to solving the heterogeneous migration problem. Once this mechanism is in place, we can proceed with developing the second necessary mechanism, which is compiler support for cross-ISA migration.

As mentioned above, the base compiler is taken from Virginia Tech [4], which currently supports cross-ISA compilation for x86_64 and ARM64 platforms [8]. We plan to extend the support to include POWER8 which is necessary to support one of the target architectures of the OPERA project. Once this support is in place, we can compile an application with cross-ISA support, and migrate it among the various pairs of machines using CRIU and the post-copy migration mechanism. Since the modifications to the LLVM/clang compiler were released as open-source in November 2016, Virginia Tech has been building a community of users to drive the compiler forward to commercial acceptance and usage. We have become early adopters of this new technology, and are trying to help make the compiler a more finished product. We plan to contribute changes back to this new community, (such as support for POWER8) to help grow the user base, and usability of the project.

The third and final step of the project is to test cross-ISA post-copy migration using the FPGA accelerator described in deliverable D6.5. In this phase, we will replace CRIU with the FPGA and driver to support kernel-mode remote page faults. This experimentation is not likely to be accepted in a commercial setting in the short term, which validates the need for the CRIU post-copy solution. However, we believe that in the long term, remote page faults will require the support of specialized hardware as described in D6.5, which are an order of magnitude lower latency than standard networks.

6 CONCLUSIONS

This deliverable reports on the progress and planning related to measuring the efficiency of using SaaS cloud services in a homogeneous vs heterogeneous data center. The study aims to determine if there is a benefit from using heterogeneous resources, and if so, what kind of benefit we can expect if scaled up to the size of a common data center.

We aim to show how we can improve efficiency in order to save operating costs of the data center, and even reduce environmental impact while still maintaining the service level agreements. By taking advantage of different hardware, we can make some incremental improvements. However, we believe the majority of the improvement will come from software, which is better designed to utilize the available hardware with high efficiency. By monitoring the system, the orchestrator can become aware of inefficiencies, and can reschedule components (microservices) through mechanisms such as container migration in order to increase efficiency.

Our proof-of-concept is to be performed on three different setups, in order to show the progression of hardware over time, with the baseline system being the state-of-the-art of 2013, and the target system being the state-of-the-art available today. The software is based on OpenStack, and will be tested with and without our modifications, in order to give a better understanding of how the software influences the efficiency of the cloud data center.

7 REFERENCES

[1] G. Moore, "Progress In Digital Integrated Electronics," *International Electron Devices Meeting,* pp. 11-13, 1975.

[2] C. C. Hu, "Modern Semiconductor Devices for Integrated Circuits," 2010. [Online]. Available: https://people.eecs.berkeley.edu/~hu/Book-Chapters-and-Lecture-Slides-download.html. [Accessed 2017].

[3] D. Chinnery and K. Keutzer, "Closing the Power Gap Between ASIC & Custom: Tools and Techniques for Low Power Design," 2007. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-68953-1_2. [Accessed 2017].

[4] QEMU, "QEMU - the FAST! processor emulator," QEMU, 2017. [Online]. Available: http://www.qemu.org/.

[5] M. DeVuyst, A. Venkat and D. Tullsen, "Execution Migration in a Heterogeneous-ISA Chip Multiprocessor," *ASPLOS XVII,* 2012.

[6] A. Barbalace, R. Lyerly, C. Jelesnianski, A. Carno, H. Chuang, V. Legout and R. B, "Breaking the Boundaries in Heterogeneous-ISA Datacenters," *ASPLOS XXII,* 2017.

[7] CRIU, "Checkpoint/Restore In Userspace," CRIU, 2017. [Online]. Available: https://criu.org/Main_Page.

[8] V. Tech, "Cross-ISA compiler based on LLVM," [Online]. Available: git://chronoslinux.org/popcorn-compiler-arm-x86.

[9] V. Tech, "Popcorn Linux - Heterogeneous Compilation," 2017. [Online]. Available: http://popcornlinux.org/index.php/compiler.