Transcript of Keynote Speech
Recorded by Dr. Minas Dasygenis
Transcript by Antonios Ventouris
Edited by Dr. Minas Dasygenis
http://arch.icte.uowm.gr

http://pci2015.teiath.gr/keynote-speakers/
19th Panhellenic Conference on Informatics
1-3 October 2015

Keynote Speech 2: "Fifty years of evolution in virtualization technologies: From the first IBM machines to the modern hyperconverged infrastructures"

Prof. Nectarios Koziris, National Technical University of Athens

Abstract

The evolution of virtualization is an exciting history of enabling technologies that have offered the ability to organize and utilize the hardware resources more efficiently. We will start from the origins of the first IBM machines back in 1960's, where the notion of virtualization was originally introduced to partition the hardware resources more efficiently. This was well before the onset of the majority of the operating systems in the early 1970's, which were initially introduced as a software alternative to partition and expose hardware resources to multiple users. In the late 1980's and 1990's the software hypervisors allowed the multiplexing of PC hardware and enabled the use of virtual machines, which in the 2000's became a commodity that needed to be efficiently orchestrated and provisioned in bulk. The IaaS cloud era had arrived, bringing more interesting challenges for the dynamic, thin provisioning of hardware and software resources to applications. Nowadays, we are living in the times of "software defined everything". We have hyper-convergence in all our datacenter infrastructures, where all compute, network and storage resources are controlled by software. The talk will present this exciting long technology journey that started more than 50 years ago and still continues, always driven by the notion of virtualizing expensive hardware resources to multiple users.

Good afternoon to everybody. We all thank you very much for the introduction and for the invitation. Thank you very much. It's a great honor and privilege to be here in the opening of the Pan-Hellenic Conference of Informatics. Thank you very much the organizers for giving me the opportunity to present my vision and, first of all, the history of virtualization and my vision *(***)* and to the younger ones in the audience, I would like to say that we have to know very well the past in order to plan for the future. So it's very educative to follow the slides. Although there are many slides, as Mr. Karanikolas said, but I will try to focus on the most important things.

I was impressed, let me say, too much impressed, I was impressed by the numbers, the people, the organization and, having followed the PCI conference from the early years, I must say that it is all these little details that sum up and make the conference a successful one and please allow me to express my gratitude and my congratulations to all the organizers because it's a lot work and from the little greeting details of the map and the places to go around and the schedule and keeping the time, it's all very well organized and I hope that everything goes well up to the end. Can you see all the windows? I hope so. Okay.

So before starting this long journey to the… not so long, it's 50 years old, but for the young ones, it may seem to come from the ancient times, let me say a personal story first. I feel old, because, 20 years before, in my very early years as a young researcher, okay, I'm still young, I had a publication, presentation in a similar event, in the fifth Pan-Hellenic Conference of Informatics. This was done in 1995. It was the fifth Pan-Hellenic conference of informatics. I think that it was not called such way, it was called the MP conference, Yanni, correct me if I'm wrong, but okay it was then that it matured and it became the Pan-Hellenic Conference of Informatics. And I tried to find my path, my piece for the old presentation and these were the paper and the slides which were in transparences and not on a *(***)* computer. We had not projector. So if we have to remember two things from my talk, we have to remember two terms. The first one is "multiplexing", πολυπλεξία as we say in Greek, and the second one is "abstraction", αφαίρεση. So it's the only two Greek words I had in my slides, so for the non-Greek ones, don't be frightened, the rest of the talk and the rest of the slides are in English. So these two words, multiplexing and abstraction, are very important in all domains of science and, of course, in our science. And I will try to persuade you that abstraction and multiplexing led us to virtualization and led us to all this fancy stuff we have with the clouds and with *(***)* where I will end my talk. So let us go a century back, in 1919, where the American telephone and telegraph company, AT&T as we know it, was trying to convince the people to use the telephones and they were bringing a very special technology there, a technology that could allow up to five people to talk together one-to-one. Amazing. Five people to talk together through the same circuit, through the same call. And how AT&T do this? It would do this through the well-known time-division multiplexing. In telecoms, we had this notion of multiplexing for a hundred years now. So AT&T was advertising multiplexing in a way to allow many people for currently use the same medium, the same access access medium and share themselves as they were alone. Where else do we have multiplexing apart from our science? We have in the transport area. We know the big highways will have a lot of lanes, high speed lanes, low speed lanes, we all are multiplexed our cars in the lanes, and we feel that we are alone in the road, not so alone, because there

http://arch.icte.uowm.gr

are the traffic lights, there are signs, there are policemen, but okay we are all multiplexed on the same street, on the same highway and we try to be served by the same medium, which is the highway.

So in the area of telecoms, the time-division multiplexing was the 101 of telecommunications and the public switched telephone network works this way. They try to divide the time in slots and they allocate slots to different users. The second term that I asked you to remember is the term "abstraction". The term "abstraction" is not so clear as the term "sales". It's abstract. It means that I can leave omitted details and take the important stuff and focus on the important stuff.

So in our area, if we have an instruction X that wants… Sorry, if we have instruction X that wants to map to instruction Y, instead of going the straight line, we have direct mapping between X and Y, we have an indirect mapping between translation to X1 and then ABC and then Y1 up to Y, which means that we can have as many as we like in the intermediate form of translating things, but the end goal is reaching the Y format. Okay, it seems abstract but we will see how we used it with virtualization. Virtualization in fact is abstraction. It means translating the instruction to something intermediate, an intermediate format, intermediate binary format so we have a virtual machine instead of the real machine. So I tried to write the timeline of evolution in virtualization and I would say that we had a lot of activity in the '60s whereas we didn't have any activity in the '80s and in the '90s. And I will try to explain why for my own *(***)*. And why we are so hot in the era of 2010 and 2020 as we proceed now. So the first years the advent of hardware virtualization was done practically in 1964, which is a very old machine. I even hasn't seen it in front of me, in a museum only *(***)* in Silicon Valley outside San Francisco. There is a very good museum there, a computer history museum, where they have *(bad audio)*

So in 1964, the IBM announced a very big computer, 360, which aimed to address the needs of the scientists and the researchers which at these times they were just submitting jobs in their program. There was only one user at a time to these machines. So they were striving to find some hours to use the machine and the people were striving to… were waiting to get *(***)* because the machine was working *(***)*. So this means that only one user could use the machine and he had to wait up to the end so the next user could come. Although it is a very naïve way of computing, over one thousand machines were sold in the first 30 days, which showed how thirsty people were for computing power. So the people at this time were very thirsty for computing power, but there were two different approaches. The first approach was by IBM who was trying to make a medium that would serve all the people and have a portable way of moving the programs from one 360 to another 360, even with different architectural characteristics but with the same batch mode, even faster than before.

And the same approach came from MIT from work, sending out work by Professor Corbato *(***)* I will go to this slide in a while *(***)* …they were working in a time-sharing operating system. *(***)* …and they were trying to make an operating system as they were calling that would time, that would beam the timeshare between users. The objective of MIT was to have the proper project, called *(***)*, so that they make users to concurrently use the same computer and believe that the computer is owns… they own the computer. So the first way of doing this was a timesharing system like the one Corbato was… Corbato, as we will see, was working with an IBM machine. But the IBM people had a different perspective on this and the people from New York and the IBM headquarters addressed the bid of MIT not with

a timesharing system as the MIT worked but with a different system that was faster in terms of batch processing. So IBM responded with a bad computer in this speed, the speed was successful for GE and laps. And what we know as UNIX and his answers *(***)* was the operating system that MIT started and this was the operating system that succeeded in this speed. So we have a failure for IBM at this particular bid and, as we all know, very good things start with failures. So IBM started using the beam and then went and *(***)* for a similar bid for another timesharing system, then IBM took the work of Professor Corbato and the group that was in cables near MIT and they addressed, they created the first virtualised machine, which is the CP-40. CP40 was not a commercial machine. CP-65 was a commercial machine. And why we call it CP? Because it had a control program that controlled the hardware and could expose the hardware to many users allocating different portions of a machine as different virtual machines. So we had the control program, the ancestor hypervisor, and we had the operating system which was a CP user operating system but nevertheless it was an operating system, CMS, so we had a machine with a control program and then CMS.

In the slide, you can see the original book, the seminal book of MultiCS. MultiCS was the operating system that the project MAC was doing and it was its spine. This TSS, timesharing system, was the answers of MultiCS and then MultiCS was the answers of *(***)* we see in the next slide. So I just put this slide to show to you that an industrial vendor, IBM very very big at this time, they were proposing something different, but MIT university in their labs, another research centre, worked something else. In their labs at MIT were using the technology that was done in-house, in MIT, from professor Corbato and his team with an IBM machine. So it's very important and I take the opportunity to say that it's very important that, if there are any guys from the industry, it's very important to have a close collaboration with the industry, and this example here shows us that IBM lost a bid, but could follow the next one and create a virtualized hardware CPCMS because they have a very good collaboration and a team of people collaborating with MIT inside MIT. So it's very important that we have a good collaboration between the industry and academia, and we have numerous examples in our science, and this example is very vivid. It shows what *(***)*. So in the 1960s, in the late 1960s, we had two things to fight each other. Batch processing versus timesharing. So the people that they were pro the operating system, they will say that we will just need a timesharing system, and the IBMers, the people that come from the big hardware companies, they would say we want batch processing.

The people that they were proposing a virtualization, they would propose a virtualization for things like isolation, because, if we are many users and we use the same … each one of us has its own VM, we are isolated so, whenever I'm doing something wrong, I don't spoil, I don't block, I don't install the *(***)* of the others, whereas, in operating system, I can block, I can break the whole system by doing a malicious code. The people that were pro virtual machines, they were arguing for ease of system programming, because, with a virtual machine to expose a hardware to the programming, so the system programming can develop very good work with the control of the hardware, even if it is virtualized, and the portability of the code *(***)*. So things were maturing and, in 1974, there was a seminal paper by Popek and Goldberg that was describing the model of hardware and the VMM, as we see, the virtual machine monitor, a hypervisor *(***)*, which is a piece of software that controls the hardware, and the VMs that interact only with the VMM, interact only with a hypervisor, so we can have plenty of VMs that talk only with a hypervisor, and the hypervisor controls the hardware and the control program that IBM initially, originally

http://arch.icte.uowm.gr

produced. And the machines that we all know was the CP-67, the IBM CP-67, it's a well-known portable *(***)*, we see the first virtualized machine in a CP-67, and, then, the machine 370 was the next machine. So this seminal paper had two types of hypervisors as we know them. The type 1 hypervisor which is the one we say the bare-metal hypervisor means a hypervisor that runs on top of the hardware, controls all the hardware and exposes the hardware or abstracts the hardware to many virtual machines to many operating systems *(***)* operating systems.

And, on the other hand, we have the type 2 VMM, the type 2 hypervisor, where we have the hardware, we have a time operating system that runs together with the hypervisor which is for the host operating system, and the guest operating system runs on top of the hypervisor. For the younger ones in the audience, you can see that the things that they were said in the 1974 are the things that we have now. So we can say we have not so much, because the most of the things were said 40 years before. The modern hypervisors that we have, I'm sure that you all know VMware work, Oracle Xen Virtual Machine's work, XenServer, are type 1 hypervisors, and type 2 hypervisors were VirtualBox and VMware Workstation. So the seminal paper of 1974 introduced the two types of hypervisors, bare-metal ones or hosted ones. Before somebody can ask me, what can we say that KVM will launch. KVM, as we know, is a kernel module Linux host operating system, so it can be considered like type 1 hypervisor.

So ending up the IBM story, things matured in the 1979, this machine was capable of hosting many many VMs through a hypervisor which is called control program there, the control program controls the hardware, and we have several operating systems here, CMS was an operating system, a CP using operating system at this time. We have DOS, *(***)* another VS1, another operating system. So, this machine was a virtualized hardware machine well sold back in late 1970s to allow concurrent execution of many operating systems where many users have their own machine each one of them and they would test develop etc. So the one thread that was led by IBM okay had been influenced by time shared systems and what Corbato in MIT was doing, follow the path of hardware virtualization of splitting the hardware into resources and allocating resources to different virtual machines.

The other path that we know mostly is the path of the Unix model, the operating system that was matured in the 1970s and with the timeshare, the hardware, the computer resources to many users, as we know. It's a difficult operating system. We have a team process and we have many users. Each of them they can run their own codes but all of us have *(***)* to the same kernel, and this may be bad in terms of isolation. But it's very good in terms of taking, of exploiting all the capabilities dynamically of the hardware. So the Unix timeline in the '70s was this one. It started in 1969 and we all know the evolution then. It's important to understand that they were two different perspectives. Timesharing the hardware or splitting in the hardware and dividing it to virtual machines or coexisting both. And history showed us that we have to coexist both. So let me proceed with the 1980s and the advent of the personal computer.

We all know that IBM in 1981 opened the architecture, so anybody could make referrals *(***)* etc. that could interface with the personal computer. This open architecture create fire to a whole ecosystem of operating systems. And it was very important because, ten years later, we ended up with many many different operating systems. So in the '60s, we had the model of a main frame which was very expensive and the problem was how to share it to many users. But, in the '80s, there not so many many frames, and each one of us had a

http://arch.icte.uowm.gr

personal computer. So initially, the need for virtual machines was not so big. That's why we couldn't say that the progress in virtual machines was not done in the '80s. Although IBM was selling products that were virtualized in terms of hardware in the '80s and the '90s and so forth. But, the problem is that the personal computer gave us a lot of computation, computing power that we had in our offices, and the focus was on doing a lot of operating systems. So VM study became *(***)*, the operating systems were multitasking and we had no serious need for VMs. The problem is that things go around. Opening the PC architecture, having a lot of PCs, writing a lot of codes, brought the need for many, for different operating systems. And we had an abundance of operating systems and we reached at a point that these operating systems should coexist in the same house. That's where the need for virtualization came back again.

So in the '90s, in the late '90s, we had too many types of operating systems, desktop ones. In the 2000, we have the mobile ones, too many flavours *(***)*, too many applications, and there came the need to host them together into the same house. So this is a source, this is a comic that says that, in the '90s, we have three operating systems in our house, now we have four and then the rest is comic. But it shows that we have a lot of operating systems and we have the need to coexist together. At the same time, where we have this boom in operating systems and the need for doing hypervisors for the PC, some people were improving things in the programming language industry. And it's true that the CNC++ were not very easy in terms of porting them to a different machine. So in the… there was a *(***)* in some microsystems, which was *(***)* and, then, in 1994, JAVA, which came out of a C++ project they had inside Sun and they were trying to make things easier for the software engineer to have programs that were portable and could run to different machines. So I'm sure that you all know the JAVA model which was pioneering at these times, then many other language followed, was the idea of a language virtual machine, which says that we have a source code in JAVA, we take it out with a JAVA profiler and we get a bytecode and, then, this bytecode runs on any JAVA virtual machine. So we can have different operating systems as long as they have a JVM, as long as they have a JAVA virtual machine, they can run our bytecode.

And of course, in order to improve things, the JVM *(***)* just in time compiler to compile dynamically the instructions here and run more efficiently in the hardware lab. So what did the model say? We have an abstraction again, we have *(***)* abstraction, that's why we have to remember the word "abstraction", we have the source code which is compiled in an intermediate form named the JAVA bytecode, and, then, the bytecode can run *(***)* on any hardware as long as we have the virtual machine. But this time it's a language virtual machine, not hardware virtual machine. So virtualization came already to the language domain, *(***)* application domain. For the people that they are aware of a just in time compiler, a just in time compiler is just a way of *(***)* by compiling the bytecode to the target architecture. So we have virtualization on the hardware level, we have virtualization on the language level, but it's still not where it's covering virtualization in all areas. There was a seminar that a magazine, a computer magazine had in 2004, a special issue for virtualization technologies. It was explaining why we need virtualization. Because, in terms isolation, nothing harm, nothing bad I do should influence the VM of the others. In terms of movability, I can take my VM anywhere, I can run in another hardware and I can migrate the whole system and all this fancy stuff. So a modern system looks like this. We have the hardware, we have the hypervisor and we have many *(***)* operating systems with our application. And we can take this VM and we can put it in another machine. So the *(***)*,

http://arch.icte.uowm.gr

opening the architecture of the PC and bringing many operating systems led to the boom in the hypervisor area. So in the 2000, we had… 1998 VMware, we had the *(***)* virtual PCs and VirtualBox, all these hypervisors that run natively in a host OS on top of an x86 machine.

We can see the different characteristics here. Some of them are full virtualized, some of them are para-virtualized. I'm about to explain a bit later what's the difference between them. So there is a *(***)* system of hypervisors for the common hardware we have, the interpart. So these are the types of virtualization in terms of hardware, hardware virtualization, we have today. The machine there *(***)* we were explaining. We have a lot of operating systems supported altogether. And the old one, the VM/370 was the original machine with hypervisor and, now, we have the VMware, ESX, Xen etc. The operating system level it's another technical we haven't discussed, it was not so well developed in the '80s, but it came back now. I'm not sure who of you have heard Solaris zones or freebsd jails. Jails were the answers *(***)* container *(***)*. So, we have one way of virtualizing a whole hardware and hosting the operating system there and our applications. We have the linking to have on the same operating system different containers that pack only the necessary stuff and our application. So this is the container of the operating system level virtualization we have now today, and, of course, the language which is the well-known JAVA platform we have in our laptops today. So to sum up, it depends on the layer of instruction that we want to have. So we have the instruction, separate architecture abstraction, which is the hardware abstraction, so we have hardware virtualization.

If we go to the application *(***)* interface, we have the operating system virtualization which means different containers. You can see it in the blue light there. And if we go to the application, domain and language etc., then go to the application programming which is the top. So we have three levels *(***)* which level is more suitable for our case or we can have all of them together. And this is the situation today. Some stuff regarding the hardware virtualization, I don't know how much time do I have. Do I have time? Do I have ten minutes? Okay. So these are the three types of virtualization, the full, the partial and para-virtualization, I'm not going to detail for them. I'm sure that you can have slides and you can find more details elsewhere. The OS-level virtualization which, as I said, is the container stuff we are hearing today, we are running on top of the same OS. We have multiple isolated but user space to our containers. Some will say it is secure enough. So how much can support *(***)* on different kind of containers onto the same operating system. Again, I can have different operating systems on the same hardware with hardware virtualization. I can have different applications, different containers, containerized applications with my data on top of the same operating system. This is very convenient, but how secure is this? And okay, it is secure, but you understand that, as long as you go on top of the operating system, or go to the user space, things become a little more tricky, a little more not isolated, a little more insecure, because these containers have access to the same kernel. And the big issue is what about the data of those containers and how we can have a container move from one host to another. And of course, at a very high level, at the level that people in language are working, we can have a language virtual machine, we have the abstraction machine definition, we have an intermediate code, a bytecode, we translate the source code into a bytecode and then, as long as this bytecode finds a VM to run, we can run it in any machine, so we have portability of the code. We're starting with JAVA *(***)*, but we have now many scripting languages with every virtual machine, Ruby, Python *(***)* and we use there this kind of just in time compilers and machines for them. So let me sum up how we are now. We have a plethora of operating systems, different operating systems for different needs for the same

http://arch.icte.uowm.gr

hardware, x86 is the dominant hardware, we have a diversity of workloads, that's the situation we have today, and, since we have a diversity of workloads, we have to find a way to load them rapidly and to port them from one container to the other. When I say container, I mean from one machine to the other.

Now, hypervisors is a mature technology, hardware hypervisors *(***)* are very mature. And the problem is that, with this kind of multiplexing, this kind of exploiting the real hardware in many virtual resources, I mean, I have a machine with six doors or eight doors and I can virtualize it, I can split it into tens of machines, so I can now have tens or thousands of VMs. I mean, I can have a rack of servers and the rack of servers can contain thousands of VMs. So how can I manage thousands of virtual machines which have their own OS? So there comes the cloud. So the cloud is a way to orchestrate all these virtual machines that we have in a very convenient way, taking the hustle out of the user and putting it into the VMs. I'm going to speak about the IaaS, the Infrastructure as a Service. We can call them simply software stacks to manage our VMs, our hardware VMs. We have the well-known Amazon cloud which is based on a closed source solution. And we have open-source ones, like the OpenStack, the OpenNebula, the CloudStack and the Synnefo, which was a software stack that we did in Greece while I was with the Greek Research and Education Network. The idea in an IaaS cloud is that the user can *(***)* a number of VMs out of different flavours, and you will see that the cloud, an IaaS cloud, typically has hundreds of different flavours, with different characteristics, different memory size, different number of *(***)*, different number of discs, of size of disc, kind of disc etc.

Okay. So let me say a few words about the original work we had done and for the Synnefo software and the Okeanos cloud, which is one of the largest, allow me to say, still the largest public cloud in Europe. It's a cloud that orchestrates virtual machines of the end users, like you. The *(***)* was written in the late 2010, 15 code experts. While I was preparing my talk for today, I was revisiting some historical notes from some people in the '60s, and they were saying that any group beyond 12 programmers cannot do substantial work. All the big projects come with less than 12 programmers. One is very few, more than 12 is very big. Take care. It doesn't mean than any group of 12 people can create a software like UNIX. So even if Unix was originated by a hundred people, it doesn't mean that any hundred people can create the Unix again. So, I'm not claiming that Okeanos is the perfect software, but I'm very proud of the work we did there. And the numbers were really astonishing. You can see thousands, thousands of users, thousands of VMs, hundreds of thousands of spawning and respawning and resources, which means that we have built an orchestrator that is really unbreakable and can bring support to many many many virtual machines. Why we did the Okeanos? Because, at this time, there were no big *(***)* and the OpenStack, which is well-known, didn't work, still doesn't work, but the idea it was that we would control our stack. For the last three years up to mid-October 2014, we had very very intense work done there and software now is very stable. And I know that some very good friends *(***)* in the infrastructure session will share some details and some applications and services that *(***)* on top of the Okeanos cloud. So Synnefo is the orchestrator of managing all these VMs. And seeing that it's open source and we have very good collaboration with Google, with their orchestrator, Ganeti, we are contributing still with Ganeti and Google. The idea was that we had virtual machines, any kind of virtual machines, we put scale up to hundreds of thousands, tens of thousands and hundreds of thousands, by adding more more and more quick. And we could start a VM in three clicks. Okay, these are some details which show technically why the approach that we followed *(***)* the abstraction approach was very

http://arch.icte.uowm.gr

good and why it leads to the hyper-convergence I'm going to speak in the last ten minutes. Okay. So the approach we followed was a level approach. The resources for the users are these: compute, network, storage and firewall, and these are for the orchestrator of the virtual machines, the virtual internet, the virtual disc and the virtual files. So with these Legos, if we put them together in a way that makes sense, we can have this one. So this is the idea originally the idea of the cloud. We were thinking of very simple bricks that could bring any IT infrastructure, the most complicated one, like this one I've showed here, 3P architecture made of these virtual bricks, virtual resources coming from the Okeanos cloud.

So we have compute, VMs, we have networks, we have discs and we have firewall, so we can bid an arbitrary morphology of IT based on these Lego stack. And the difficult part now is, if there is a failure in a physical machine that costs this here, what will happen with the whole IT infrastructure? So some of you I know that you know the… you have seen the UI. So the problem is that, from the user perspective, I don't want any failures, I want to scale it out, I want to grow, and the problem is that, in order to address all these needs for the users, I have a fragmentation of things, a fragmentation of infrastructure. Many vendors come in my data centre proposing different solutions and I have all these different solutions to glue them together. So our data centre became a messy data centre, a data centre where we had different goods from different vendors. And each vendor were proposing that his management software was doing the right stuff and the other were impractical. So the idea is that we have users that they needed available VMs, isolated, with *(***)*. This time I was asking for VMs that was relinquishing them, from the one hand. And from the other hand, I had an HP, an IBM, a Dell server, I had an *(***)* net up at storage area. So I have two different things to come together, so my server, a fat server, with many *(***),* virtualized is like a big vessel, a big ship, with a lot of workload on top. Imagine that you are in the middle of the ocean and that this vessel breaks. What will happen to your containers? And this is the idea of containers.

Okay, another vessel will come and you have to migrate all your load from this vessel to the new one. So this happens with one vessel with all this workload on top of it. And do the analogy, you have a workload in your server and you have to migrate because a fan broke, the power supply didn't work, so you have to migrate live, port the VMs from one machine to another. And your data centre is like this. Your data centre has a lot of workload, different workloads, each of these containers I'm sure it's the best, it's the telephone of the user that says my workload is the best one and I want to take care of it, it's the best of everybody. So, you have a way to manage all these workloads horizontally and give to any user the ability to use any of these frames, any of these vessels, because everybody is important. So the problem is much more difficult, much more complicated than the real world, because, in the real world, I'm sure that all these areas are split vertically and *(***)* and share between different logistic companies.

But, in our world, any container can use any crane to get this job done, and any vessel could get its… to be hosted. Okay. So we have to call somewhere else, all this messy stuff, and the solution for all this messy stuff. And the solution comes from the vendor's world is called convergence. In the last… for the last five years, vendors were coming at GRnet, at other fora and discussions we had, they were proposing their convergence solution. And the word "convergence" is a very very *(***)* Convergence means that we glue everything together. So I keep saying this stuff that, if you are a blacksmith, you see everything as nails and hammer. So if you are a hardware vendor, you think that the solution could be a big rack like this. So

http://arch.icte.uowm.gr

the hardware vendors brought the word “converged architecture” which means that we have storage, computing power, cooling, network, and the management software all bundled together in a stuff like this.

So Cisco comes… I invited Cisco here. Cisco comes, IBM comes, and sells you a unified computing infrastructure, a converged one, which says that I can host tens of thousands of VMs and hundreds of terrabytes and tetrabytes inside your cloud. And this is your cloud. Okay. If I ask you if something happens to this rack? If the kind lady that does the housework and cleans the data centre pulls out a plug? If the fan breaks? What will happen to all these tens of thousands of VMs inside this cabinet? And the idea is, okay, you can buy a second one, so you can migrate between two of them. And I’m saying, “okay, why I have to log into you”, because you have to buy from the same vendor, and they say “don’t worry, I’m compatible with the others, but the others have to try more to be compatible with me”. So the vendor perspective on how the mess of data centre can be arranged is the converged infrastructures. And the market is real big. So the idea of converging everything to a single chassis, a single cabinet where you put everything together and you virtualize all the layers, is very attractive, but does not work. And does not work because the data centre needs to scale out, needs to scale to infinity all layers, to scale to infinity.

So the reality in the data centre is that we have infrastructure, we have servers, we have storage, we have network, which are independently virtualized, so they are multiplied, the idea of multiplexing as I said. So we multiply the resources, we have tens of thousands of network forts, tens of thousands of discs, tens of thousands of VMs, but we have somehow to glue them together, to orchestrate them together. So the idea we followed in our work at the Synnefo software, but this is being followed already by all major data centre operators that they are hosting IaaS clouds, is that we try horizontally to glue together vertical partition things. I mean, a typical server would have a storage, would have not a virtualized server, would have this type of stuff, vertically access to storage from each node. When it’s virtualized, we have a hypervisor, we have the guest VMs, we have the storage here, but we have to make a glue. So all this storage is being seen as a unified one. I’m skipping some details to show you what we did.

The Archipelago approach, the Archipelago was the storage that we developed, originally developed inside the Synnefo, and this kind of software defined storage as we say, decouples the logic of the actual physical storage. So we built a distinguished storage system. So this distinguished storage system decouples the storage, the sources of storage backends. So *(***)* means you can buy discs from the Stournara street, and the Stournara street is the retailers. You can buy any disc you want. You just put a lot of servers with a lot of discs. And you glue them together with a software. And whenever a new VM wants another disc or when a new VM wants to make a snapshot, it takes blocks, obviously it’s okay, obviously it’s out of this pool. So the idea is that files, user files, images, I mean, templates for creating VMs, volumes which are the discs that the VMs have, all are considered resources. Snapshots which are I press a button and I want to keep a still image of my VM. So this is the architecture of a storage defined… of a software defined storage. The architecture says that the backend, okay, it cannot be seen very good here, it’s just a bunch of loads, a bunch of objects, sorry. And then, the user has an image file, he closed it to a disc, and the disc can start a VM and all these are maps to the same substrate. So the idea of a storage defined… of a software defined storage is that we had a substrate of objects, in our case it was the Ceph file system and the RADOS object store, and, on top of it, we

http://arch.icte.uowm.gr

orchestrate different lead based on the needs of the users. So the top virtually is like this. We have physical hosts with VMs, we have our secret software, and we glue together storage that comes from a retailer. And this can scale out to infinity. The experience we have with this was that we could allow… we could tolerate any kind of *(***)* So it means that somebody would like to take out a container from the server and change the track, and the VMs were just migrated to another track or they were just migrating to another data centre. We could even migrate VMs, workloads, from one data centre to another. And okay, you can ask me what happened with the data. I'm very happy to answer to you that the data could be also synchronized *(***)*, but this is another story.

So the idea is that the only way to scale to infinity is to have software defined data centre. It means software controlling all the hardware resources and decoupling the hardware, the virtualized hardware resources from the manager. The Synnefo software is there. It's being used by several data centers in Europe and hardware installations in US and I'm sure that more details you can find in the infrastructure session. So the challenges we are having. How much time do I have? I don't have time. See, it's the typical answer I get always from the session centre. What are the challenges we are having? We had diverse workloads, every workload is important. Workloads come and go. We have to optimize scheduling. I'm sure you know schedulers that come into life now, the Mesos scheduler, the *(***)* etc., the Omega project from Google. We have to scale anything, cores, a number of cores, the size of memory, discs etc. I want… A user would say "I don't want another disc, I want just my disc and double the number of blocks". And how we control this stuff? How we control all these resources? The software defined data centre. It means that we control the provisioning of all resources through software, that the resources will be *(***)*, will be naïve. We take out all the management software, all the management goodies from the vendors and we put software. We control with our software the hardware resources and that's it. So we can do all this kind of stuff which is very important for a data centre, believe me, with applications, snapshots, clones, theme provisioning, *(***)*, synchronization, disaster recovery, you want the VM not to be destroyed, but you can take it and go to the other data centre which is a thousand kilometers away from here, this is what the Google does and the other data centers, out of commodity physical storage, out of commodity stuff. My kids love the *(incomprehensible)* story, so I have this comic. It seems that we have to forget the typical discs and we have to focus on the not flash, *(***),* the memories will become bigger and the discs will become bigger and will become based on solid state. So we have to forget the typical discs you know. And you see that the cost, this is the cost for a terabyte of a capacity disc and the cost for a terabyte of a flash disc, you see the point here. In the next year and the flash will become cheaper *(***)* So it's very important. So all of our storage will be flashed. So I'm claiming and I'm concluding, I'm claiming that converging, hyper-converging means taking all the resources, gluing them together, controlling them with software, will bring at the storage domain the end of the storage arraying. They were no cases for storage arrays inside the data centers. Okay, they were big cases for big systems, for big companies like the banks, the banks are still in the era of *(***)*, so it's okay *(***)* the storage with magnetic discs.

So to my understanding, the storage arrays are dead, software defined storage leads everything in the data centre but we storage hyper-converged ones. Cheap discs come together, virtualized and the blocks, everything for the VM. This is the ecosystem of a big companies that they are dominating the software defined storage and you can see that all of them are selling software apart from some that keep selling appliances, because it's a

http://arch.icte.uowm.gr

psychological way, I'm purchasing the equipment, I have to see it, whereas, when I'm buying software, I don't see it, so I cannot claim *(***)* used. By this I'm finishing, it's very important to understand that we are living in an era of *(***)* We have virtualized everything, from machines to networks to discs. So all the resources in a data centre are being virtualized. We have to control. How we control in a unified way? With the software. Problem is that, when we asked a typical user, he says "buy another cabinet of discs, buy another array of discs". So this quote by *(***)* reminds me what we are living nowadays. People are selling to us arrays, vertically converged, as they say, equipment but we need no *(***)*, we need something else. We need a machine, we don't need more horses. Thank you.

- Thank you very much for this insightful and fascinating presentation. We could take one or two questions if you… Yes, *(***)*

Question by Dr. Minas Dasygenis:

- Thank you for you talk and I would like to congratulate you for the Okeanos project, which I'm one of the first adopters of this technology and, in fact, I was using, with my students, I'm teaching distributed systems, open MPI, and we were constructing big clusters of it and we were using in teaching parallelization techniques. And I would like to comment on a phrase that you said a couple of times. The scale to infinity. It seems that, even though software can support scaling, economics does not support scaling. And for this reason, the Okeanos project has been paused for new users, and only the old users like me and my previous students can use it. So one of the question is about Okeanos and do you think that, first of all, it will expand or, due to stringent economics in Greece, it will shut down as some things came to close down? Thank you very much.

- Okay, I'm really delighted that you were one of the first adopters of Okeanos service and that you enjoyed the service. I'm also using this for my courses and it's a perfect way of organizing the course with hundreds of VMs for hundreds of students at one time. What the academics will say is that we design things, we are ambitious, we have the ambition to scale to infinity, as I said, but the problem is that the fact that Okeanos *(***)* now is not due to the software or not due to our design. It's due to the bids, to the bureaucracy, to the way things work when you are living a big bid in Greece. And I would like to separate these two things, the software is very good, it needed insight and a lot of design in the previous years. But now it's very mature and the fact that the users do not enjoy this type of stuff is that people at GRnet and, because I'm a member of the GRnet from last December, people at GRnet had their hands tied, they cannot buy the equipment. So at the end of the day, you need more servers, you need more storage, you need more discs. So I'm sure that they are planning and they are… at the moment, they have open bids and they will soon increase the physical resources because, in order to virtualize the resources, you need to have the physical ones first. As far as I know, they will put more iron to the data centre and increase the number of data centers. So you will have soon better news than the one you have in the last year. But the software was a very good example of how with the software we can control the hardware resources.

- Thank you very much and I think this concludes our key note.

- Thank you very much.

- I'm sorry we have to end this. *(***)* in 3:30, so you have a short time for a coffee break.

http://arch.icte.uowm.gr