Toward El Dorado for Cloud Computing: Lightweight VMs, Containers, Meta-Containers and Oracles

Eugene Eberbach and Alex Reuter
Rensselaer Polytechnic Institute
Hartford, Connecticut, USA
eeberbach@gmail.com , reuter@hotmail.com

Abstract- **Cloud computing offers glimpses of the straight road toward hypothetical El Dorado for distributed and grid computing, but the state of the art is currently unclear and the road is anything but straight. The goal of this paper is to dive deep into technologies and techniques underpinning cloud computing and try to determine the viability of concepts like meta-containers, lightweight VMs, DIME network cognitive architecture, Straight Road, and Mesos DCOS. We also try to answer the question to which degree cloud computing requires meta-layers with power of Turing's Oracles.**

Keywords- cloud computing, lightweight VMs, meta-containers, DIME/DNA, Mesos, self-reference, Turing Machine, completeness, hypercomputation, Oracles

I. Introduction

Cloud computing provides services in transparent and flexible way over the Internet allowing to share computing resources in the form of software (SAAS), platforms (PAAS) or infrastructure (IAAS). Shared resources can be real or virtual. Virtual can be anything: machines, memories, i/o, appliances. Virtualization of resources provides the illusion that more resources is available than physical ones, and better replies for the peak demand, reconfiguration and load balancing, in general. All these implementation details are hidden from the clients, who are supposed to receive pure computational service without being bothered by implementation details.

In fact, this represents comeback to the old ideas of services provided to clients by computer centers, mainframes, or supercomputers, i.e., reversing the trends provided by PCs, tablets, and smartphones in the situation when client-based computing platforms lack sufficient capabilities for more elaborate computations, and then for help they can use cloud on the Internet. The cloud provides a kind of the black box, i.e., users want the results/services and not necessary the details how these results have been obtained. Exactly abstracting from unnecessary details led to a new more elegant name – "cloud" – instead to call it simply a black box. Thus **cloud computing** can be considered as a relatively new concept in computing, developed in the late 1990s and the 2000s when the right time and technologies came. However, in reality, cloud computing has roots in 1950s mainframes with dumb terminals and 1960s – when IBM developed first VMs for System360/370/390. As it is typical for computing and other sciences – the history repeats itself – perhaps using new catch phrases and new names under the old repacked concepts.

Cloud computing is also related to the old client-server services on the Internet – one of the first and main applications of computer networks – only single servers have been replaced by much more complex collections of servers or subnetworks providing computational services of potentially arbitrary complexity.

Typically cloud computing is implemented using virtualization – cloud can exist without virtualization although it will be difficult to manage and inefficient.

However, the cloud implementer pays the price for virtualization, because virtualization provides a computational overhead – smaller than using bare hardware for cloud implementation, but still an overhead. Systems built today which run on the cloud often use dynamic resource provisioning to respond to fluctuations in demand. To minimize the time it takes to activate these resources and also to be able to more finely carve the resources themselves, lightweight virtual machines in the form of containers have begun to be used, and even meta-containers. This is situation analogous to processes in concurrency, where to minimize overhead with switching of context, creation and destruction of processes, the lightweight processes – threads and fibers were introduced.

The paper is organized as follows: in section II briefly containers and meta-containers are discussed. In section III Docker, in section IV the DIME Network Architecture, and in section V Nagios and Mesos Straight Road are outlined – both implementing either containers or meta-containers techniques. In section VI we discuss whether cloud computing requires Oracles, and what potentially could be gained if it was possible to implement them. Section VII contains conclusions.

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia
© 2015 ACM. ISBN 978-1-4503-3393-1/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2797433.2797446

II. Towards El Dorado for Cloud Computing: Virtualization, Lightweight VMs, Containers and Meta-Containers

In searching for El Dorado in cloud computing, there are two tendencies: to minimize overhead of cloud implementation by using containers and meta-containers, and secondly by exploring the expressiveness of distributed cloud itself, e.g., by adding the power of Oracles [19] or other hyper-computational means. The first approach, although not standardized yet, is quite well developed, and an ecosystem of supporting technologies has spawned to try and help manage the additional complexity of applications built on the cloud and with containers. The second approach is more controversial and subject to discussion. In this section we deal with VMs, containers and meta-containers, and "Oracle"-like cloud are discussed in section V and VII.

The concept of the cloud is really itself an extension of virtualization. As is stated in [17]: "Although the idea of using virtual machines was popular in 1960s and 1970s in both the computing industry and academia research, interest in virtualization was totally lost after 1980s and the rise of the personal computer industry. Only IBM's mainframe division still cared about virtualization. Indeed, the computer architectures designed at the time, and in particular Intel's x86 architecture, did not provide architectural support for virtualization (i.e., they failed the Popek/Goldberg criteria [14]). This is extremely unfortunate, since the 386 CPU, a complete redesign of the 286, was done a decade after the Popek-Goldberg paper [14], and the designers should have known better."

While VMware was the first to bring a compelling virtualization solution to the market in 1999 in their VMware Workstation where they managed to virtualize unvirtualizable x86 architecture with type 2 hypervisor, in 2003 the introduction of Xen with type 1 hypervisor, an open source micro-kernel based VM implementation greatly increased the speed at which the cloud began to spread. Since it was free and open source, companies were able to spread it across multiple commodity x86 machines and offer the underlying computing services to the market at a greatly reduced price. Prior to being able to rent a virtual server you either had to accept noisy neighbors in a truly shared host, or pay a large amount of money to either rent a full machine or pay someone else to watch yours for you.

Instead of buying or renting physical servers, customers can buy or rent virtual ones. In addition to general computing resources, resources for specific use like databases and webservers are also available. This space is dominated by Amazon's AWS, though Google's Cloud and Microsoft's Azure are also well known providers.

Running virtual machines has many benefits [15]. They utilize hardware much better, are easy to backup and exchange, and isolate services from each other. But running virtual machines also has downsides. Virtual machines require a fair amount of resources as they emulate hardware and run a full stack operating system. With Linux Containers there exists a lightweight alternative to full blown virtual machines while retaining their benefits.

Containers are *lightweight VMs*, and are very similar to them regarding functionality, but without using the full hardware abstraction of a virtual machine. Rather they are implemented as a set of process controls which themselves run on a host operating system without the need for an official hypervisor. On the other hand, virtual machines provide full abstraction of the hardware and as such are strictly more powerful than containers. The efficiency is the main reason that containers have received so much attention from industry lately. Containers are lighter-weight than VMs and this property has led them to be included in a number of successful projects which have led to the global spread in popularity.

Meta-Containers compared to containers are embedded with extra components (an extra meta-layer) that allows to reason and control containers. This concept is based on meta-programming, and the need for something to manage containers is real.

III. Docker Container Technology

Docker is the most popular instance of the open source container technology. Docker builds upon Linux LXC [15] and consists of three parts: *Docker Daemon*, *Docker Images*, and the *Docker Repositories* which together make Linux Container easy and fun to use. Linux Container (LXC) has been part of Linux since version 2.6.24 and provides system-level virtualization. It uses Linux *cgroups* and *name spaces* to isolate processes from each other so they appear to run on their own system.

Docker Deamon runs as root and orchestrates all running containers. Just as virtual machines are based on images, Docker Containers are based on *Docker Images*. These images are tiny compared to VM images. *Docker Images* can be exchanged with others and versioned like source code in public or private *Docker Repositories*.

Docker is an open source system that allows developers and system administrators to package applications into "containers" that can be moved between different systems. For example, an application can be moved between a developer's laptop and a testing server, and back again, and then finally to QA or production servers easily. Not just applications, but dependencies can be packaged into containers. Docker Hub already contains lots of common dependencies, such as versions of MySQL, Python and the Ubuntu (News - Alert) Linux distribution.

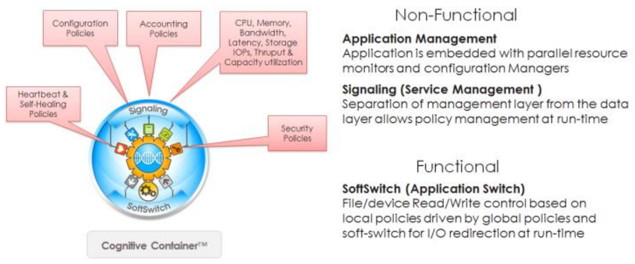
According to the developers of Docker, each container should run as few processes as possible with the ideal container running a single process. The concept of a meta-container certainly appears to violate this rule, though the rule is itself subject to debate. There are certain programs which rely on the presence of background services such as *syslog* and running these programs in isolation can lead to unexpected results. Accordingly there is an alternate view of container-scope which tries to provide a bare minimum configuration of an operating system to ensure that all “containerized” programs can run as their authors intended.

IV. DIME NETWORK ARCHITECTURE WITH META-CONTAINERS AND ORACLES

DIME Network Architecture (DNA) introduces the concept of a meta-container that abstracts applications from underlying infrastructure to remove dependencies on proprietary solutions, architectures and APIs. This coupled with DNA’s ‘Application Soft Switch’ technology “enables instant mobility of applications running across Clouds in Virtual Machines, Containers (LXC, Docker) or Physical Servers” [3].

A *DIME Network Architecture* (*DNA*), pioneered by Rao Mikkilineni and his group from C3DNA, is a cloud architecture with meta-containers providing services for autonomic clouds and grids and consisting of Distributed Intelligent Managed Element (DIME) nodes connected through signaling/controlling and input/output communication channels [11-13]:

- *DIME node,* called also *Cognitive Meta-Container,* consists of the *Managed Intelligent Computing Element* (*MICE*) performing computations controlled by meta-level controlling self-management of *Faults, reConfiguration, Accounting, Performance* and *Security* (*FCAPS*). The *MICE* can be in the form of simple (atomic) worker or hierarchically defined *DNA* subnetwork.
- *Communication channels* interconnecting the *DIME* nodes are of two types: the *signaling channels* for meta-level FCAPS management and the *input-output channels* for input/output of managed MICE computing elements/workers. Signaling channels connect meta-layers for reconfiguration, performance monitoring, fault-tolerance, security and accounting, whereas i/o channels connect MICE workers to perform their message-passing for the regular work.



Application Management
Application is embedded with parallel resource monitors and configuration Managers
Signaling (Service Management)
Separation of management layer from the data layer allows policy management at run-time
SoftSwitch (Application Switch)
File/device Read/Write control based on local policies driven by global policies and soft-switch for I/O redirection at run-time

Figure 1: A DIME Cognitive Meta-Container node with Non-Functional FCAPS meta-layer and Functional managed MICE simple element

In other words, the DIME Network Architecture (DNA) can be defined recursively: it consists of the meta-layer FCAPS and underlying either a simple atomic MICE element or another DNA.

Each DIME Cognitive Meta-Container with FCAPS meta-layer is programmable to provide continuous supervision of the execution of the programs executed by the MICE worker (can be implemented by a single processor or multicore processors, or form recursively the whole subnetwork of DIME cognitive containers). In a general case, the DIME nodes form a DAG (Directed Acyclic Graph). The DIME FCAPS meta-level management allows modeling and representing dynamic behavior of each DIME, the state of the MICE and its evolution as a function of time based on both internal and external stimuli. The blue print of what the intent, context, communications, constraints, control behaviors are specified for each application and are used to define the behavior of the container that prepares the environment using the local operating system and loads and executes the application.

The *Fault manager* (*F* from *FCAPS*) checks the health of the process and sends a heartbeat. The application as it is forked by the *Configuration manager* (*C* from *FCAPS*) in the container, it has the application's run-time control: to pause, restart, start the application at run time by controlling commands from the configuration manager. The *Accounting manager* (*A* from *FCAPS*) is responsible for all accounting on the cloud. The *Performance manager* (*P* from *FCAPS*) checks the vital signs (cpu, memory, bandwidth, storage capacity, throughput and IOPs usage periodically and institutes specified behavior based on policies. The *Security manager* (*S* from *FCAPS*) implements security policies including key management for read/write at file/device.

Some preliminary thoughts about formal modeling of DIME Network Architecture have been presented in [2,5,6].

The DIME Network Architecture (DNA) node is assumed to be related to Turing O-machine:

- FCAPS meta-layer is assumed to be an extension of Turing’s Oracle
- MICE managed layer is either a regular atomic TM or the whole DNA subnetwork

In other words, DNA is a network of interconnected O-machines [19] at least at the level of modeling. However, current implementation of DNA did not demonstrate so far the need for Oracles solving TM undecidable problems, concentrating on self-repair, reconfiguration, auto-scaling and live-migration, i.e., on complexity issues. Perhaps, in the future, it might attempt to solve, or at least to approximate solutions of one of the TM undecidable problems to explore tremendous expressiveness of distributed systems. Note that DNA extended the original definition of Oracles from classical Turing’s Oracles [19] – calling them *Extension Oracles* (able to solve undecidable problems) to include also *Simplification Oracles* (decreasing time or space complexity of managed layer) and *Augmentation Oracles* (converting intractable managed layer to tractable/polynomial one). Additionally, DNA assumes that Oracles can control other Oracles, and use humans as the last instance if Oracle fails.

Probably, it would be a good idea, to avoid confusion with Turing's Oracles, to change the name of Oracle to DNA Oracle, or DNA meta-layer, to avoid expectations that DNA was primarily designed to solve TM undecidable problems, like original O-machine was [19].

V. Alternative Implementations of Meta-Containers

Two alternative implementations of meta-containers to implement an approximation to the FCAPS QoS design of DNA have been attempted in [16] known as *Straight Road*:

1. Docker containers with Nagios meta-layer,
2. Docker containers within Mesos meta-layer as PaaS.

The *Straight Road* is a construct from literature that is the path which will bring mortals to heaven by curving out through the atmosphere.

Straight Road - Nagios: Meta-containers were implemented using a Docker container that has *Nagios Remote Plugin Executor (NRPE). Nagios* is the best known open source server monitoring in the world. *Nagios* can be used to monitor network services (HTTP, SMTP, SSH, etc.), host resources (CPU utilization, disk load, etc.), probe data (temperature, alarms), or really anything that a remote agent can be configured to send data back to the host monitoring system for (somehow similar to DNA FCAPS or Zookeeper from Apache project that was initially part of Hadoop that provides services for distributed computing such as configuration, naming, leader election, synchronization and group services). The concept is to ensure that each service deployed into the system would use the meta-container as a base and thereby ensure that it could be monitored by *Nagios*.

Straight Road – Mesos framework is a sample implementation that uses *Docker* and Apache *Mesos* to build a private PaaS (*Heroku*) that could leverage the concept of a meta-container. Apache *Mesos* is a project inspired by Google's Project Omega which was created at UC Berkeley and is intended to provide a framework for clustering computing resources that can power your data center. *Heroku* is a cloud platform as a service (PaaS) supporting several programming languages, including Ruby, Java, Scala, Python, PHP, Perl and using Debian or Ubuntu Linux OS. In 2010 Yukihiro "Matz" Matsumoto, Ruby inventor, joined *Heroku* as a Chief Architect that stresses how important the area of cloud computing is. The abstraction that *Heroku* provides is quite popular among developers because it allows for applications to be installed into the cloud with relative ease. The *Straight Road* was a simple *Mesos framework* that uses meta-containers to stand in for the "build-packs" of *Heroku*.

Note that both *Straight Road – Nagios version* and *Straight Road – Mesos version* were two alternative to DNA implementations of meta-containers as the proof of concept only – an experiment that this is possible at all, without claiming any superiority over DNA meta-containers. We did not inject any hypercomputational means of the Oracle type to our project at this moment yet. We do not claim or compare DNA and Straight Road, because Straight Road represents a very preliminary experiment to implement meta-containers only - without live migration, load balancing, etc. that have been implemented in DNA already.

VI. Does El Dorado Necessitate Oracles?

The short answer is no – the work of distributed cloud can be approximated by Turing Machines or even finite automata if it is sufficient for our purpose, i.e., if we do not want to solve Turing Machine undecidable problems. The same is true to reply to the question whether we should use Turing Machines, pushdown automata or finite automata to model computation – everything depends what we want to achieve. For example, for design of sequential circuits or lexical analyzers – finite automata are sufficient. Nobody is using clouds to implement full-blown cognitive computing yet (it does not mean that hopefully we will not do that in the future). This means that we can use and implement clouds without Oracles. However, interactive distributed computing is capable to solve undecidable problems (see, e.g., [20]), thus if we want that - the Oracles could be quite handy.

It is commonly claimed that O-machine is not implementable by mechanical means (e.g., in the form of the digital computer (A-machine – automatic machine, known commonly as Turing Machine) as Turing said [19]), thus DNA would not be implementable either (i.e., nobody knows so far, how to implement efficiently the Oracle's black box). However, DNA has been implemented, but, so far, implementation is very far from expressiveness of TM with Oracle. Yes, DNA performs the function of meta-layer FCAPS, but it is very far from Oracle that can solve TM undecidable problems. Note, the O-machine without self-reflection is able to solve the halting problem of UTM, i.e., REnonREC (recursively enumerable but not recursive) problems, and O-machine with self-reflection even nonRE (non-recursively enumerable) problems. Thus even a single node – DIME (at the modeling level) becomes more powerful than TM. What about a network of O-machines – the whole DNA? Can they do more, or one DIME node with power of O-machine is sufficient and truly we do not need that whole distributed multi-node cloud?

Do we need such tremendous expressiveness at all? For future applications, e.g., cognitive computing, the answer would be yes (pending that cognitive science would be more mature and defined more precisely than it is now). For most current typical cloud applications and users, we do not need that yet. Can we achieve that at all? As was justified in [11-13], the designers wanted from DNA model to be complete and Kurt Gödel [7] was the first to prove that mathematics is undecidable and incomplete, followed by Church and Turing analogous results. The problem starts when the Universal Turing Machines (UTM) tries to decide about itself (i.e., self-reflection, self-reference). In other words, Gödel's, Church's and Turing's results point out to the limits of controlling underlying systems. UTM was able to decide halting problem until it was given as an input itself. UTM can simulate any other TM, but stops to be able to decide when it gets as its simulated input itself. Similarly von Neumann universal constructor can construct arbitrary new cellular automaton that will be able to construct another cellular automaton (including self-reproducing itself by creating an identical but separate copy of itself), but it hits its own "halting problem" when it will be given a universal constructor itself as an input. Note that self-managing distributed systems (the main topic of the workshop) are self-reflecting systems and that naturally leads to TM undecidable problems independently whether we wish or not to be so.

This would be equivalent to the requirement that DNA FCAPS meta-layer can manage itself, i.e., can recover from errors in fault-manager (in the style of self-correcting codes able to correct all bits in words), or configuration manager that configures itself, or security manager able to detect malignant software in itself, i.e., it is perfect. Adding imperfect error-prone human in the loop is not an elegant solution at all, but a pragmatic necessity. If we assume that humans are hypercomputers, i.e., more powerful than TM, we are referring to the help of Turing C-machines (choice machines [18]) and not O-machines [19].

Consider a related approach using off the shelf software with Docker containers with Nagios monitoring [16]. The project involved building a Nagios container as well as a base NRPE enabled container and then building a sample Nginx webserver container on top of that. The Nagios container would register the Nginx service as something to monitor by interacting with the NRPE plugin of the Nginx container. Consider the Nginx process that is running. We would like to know whether the Nginx process will run forever or if it will crash at some point. By monitoring the Nginx process with Nagios we can ensure, that should it deviate in any way from our expectations that Nagios will restart it. In the case of Nginx we can decide that it will run forever, because if it crashes, Nagios will restart it. But can we answer the question in the general case? Can Nagios ensure that all processes under its watch will run forever? What about itself?

If Nagios is asked to monitor itself it will not be able to determine when it has crashed. It cannot ask itself whether it is still running or not.

The trick is to use other Nagios nodes to control and inject failed Nagios nodes (something in the style of Floating Master from Master-Slave Parallel OS). Then with the big likelihood we can approximate "immortal", i.e., running forever Nagios, or OS or server, because distribution of control minimizes the chances of the catastrophic/fatal failure of the whole distributed system (however, it does not eliminate it completely).

Will FCAPS/Oracle from DNA be able to reply such question, i.e., to keep DNA "immortal" – running forever? Note that Oracle will also hit its own halting problem if it asked about Oracle itself. Note that DNA tries to solve this problem by Oracles controlling other Oracles, and the root Oracle getting the help from human operator. This means that at the level of formal modeling DNA is a hybrid model combining both Turing C-machines [18] and O-machines [19].

Did we want too much? Perhaps, the DIME node should be in the form of UTM with FCAPS meta-layer being UTM and MICE managed layer being TM input to UTM? The power of distributed systems is not in the complexity of nodes, but rather in the interaction of multiple cloud nodes. Note that typically meta-layers are simpler than managed structures. This is for example true for Genetic Programming Problem Solver (GPPS) [10] and $-Calculus [4]. However, meta-layer from DNA supposed to have power of Oracles, thus it is more complicated than controlled containers.

However, cloud computing with or without Oracles already has enormous potential – both cloud computing and distributed systems are tremendously expressive. We proved recently both for Interaction Machines with simple nodes of the class of finite automata [20], and for Evolutionary Finite Automata [1] that they can decide both REnonREC as well as nonRE languages. Both models can be used as special instances of cloud computing. In fact, any computational systems using OS (and practically all computer systems use OS), virtual machines, hypervisors, servers are already more expressive than Turing Machines.

This includes both DIME Network Architecture and Nagios and Mesos Straight Road. They both demonstrate already the departure from the TM model, because in implementation they use client-server model and re-use OS or hypervisors. Both servers, operating systems or hypervisor VM provide non-terminating services at the level of processes - kind of algorithms that run forever [9], and thus violate classical definitions of terminating recursive algorithms (this leads to Infinite Time Turing Machines – another hypercomputational model more expressive than TM). Thus DNA, Mesos/Nagios Straight Road and cloud computing goes beyond conventional TM already (even without using the help of Oracles).

VII. Conclusion

Note that the main goal of this paper is to identify the current trends in cloud computing research and implementations. We do not compare here Docker, DNA, Mesos, Nagios, or Straight Road, because all of them have different goals and we do not have enough data to do fair comparison. In order to have better services offered by cloud computing, we have distinguished two approaches:

- To minimize overhead of virtual machines by using containers and meta-containers.
- To explore potential higher expressiveness of distributed cloud computing, perhaps using various hypercomputational means, including Oracles.

Whereas containers and meta-containers are quite well researched and mature, it is not clear to which degree distributed cloud and grid computing will allow to solve problems either unsolvable or intractable using Turing Machine models and computing.

Whether this will lead to El Dorado in cloud computing, the jury is out yet.

What the DNA architecture approach points to is that there are new types of applications which are missing a true operating system. As [8] points out - compare the process of installing a new application on your personal computer's operating system with the process of installing a new web application into the cloud. Applications which are installed on a consumer grade operating system take for granted that virtual memory exists for them, that context switching will provide dynamic multi-threading, and that physical storage can be relied on. Compare this with a web application that is installed on the cloud and must scale up to serve millions of users. This results in resource demands which cannot be met by a single computer, no matter how large. There are clever ways to dynamically provision resources, but eventually the single "monolithic" application must be split into component pieces which can run on separate computers with separate and distinct operating systems. These separately running components communicate with each other via some form of message passing, and start to resemble a form of supercomputer.

Take Facebook or Twitter or Google as examples of applications which require more resources than the architecture of a single computer could provide. They seem like good candidates for a supercomputer, and in fact it can be argued that in fact that is what ad-hoc has been developed .

Multicomputers as described by Tanenbaum [17] abstract a set of computing resources into a resource collection upon which programs can be written that will treat the collection as a consistent logical resource, a Universal Machine so to speak. While this description may sound a lot like cloud computing, in practice access to this universal machine does not exist, at least not at the consumer and small business level which we commonly know of as the domain of cloud computing. Instead what developers are forced to interact with are virtual machines, or containers, that themselves have strictly prescribed resource limits. A virtual machine with 4 or 6 or 12 virtual cores can be obtained, as can a virtual machine with 4 or 6 or 12 gigabytes of working memory, but a virtual machine with unlimited amounts of both cannot be obtained in cloud computing's current configuration.

At some level this process of scaling can be compared to threading, moving data access to a database, and offloading computing that can be parallelized into completely distinct systems all together. Keeping these distinct systems running in concert has proven to be quite difficult in practice, with developers and system administrators playing the role of operating system. In fact this is what is missing, a new operating system for multi-computers that can provide the kind of abstraction this relatively new class of applications has been missing. The DNA's approach and others are acknowledgements by the market of the need for an abstraction layer that will automatically respond to the kinds of failures that an application which has scaled becomes prone to. Conceptually DNA's cognitive computing model can be compared to another effort in this space, the Mesos Data Center Operating System (DCOS) project. Both aim to address the burgeoning complexity that running cloud-centric applications are known for, but they take different approaches. While both can certainly be considered as advances in the state of the art of cloud computing implementation, it is not clear to which degree they represent an advancement in computability theory as such.

Acknowledgements- The authors would like to thank the reviewers for their comments and criticism, and the workshop organizers for the time flexibility and understanding in paper preparation.

References

1. Burgin, M., Eberbach, E., Evolutionary Automata: Expressiveness and Convergence of Evolutionary Computation, Computer Journal, vol. 55, no.9, 2012, 1023-1029 (doi: dx.doi.org/10.1093/comjnl/bxr099).
2. Burgin M., Mikkilineni R., Morana G., Intelligent Organization of Semantic Networks, DIME Network Architecture and Grir Automata, Int. J. of Embedded Systems, Vol.x, No.x, 20xx.
3. Delony D., C3DNA Nets $2M in Seed Funding to Build 'Meta-Containers' for Docker, TNSnet.com, Oct.8, 2014, http://technews.tmcnet.com/channels/softswitch/articles/390820-c3dna-nets-2m-seed-funding-build-meta-containers.htm
4. Eberbach E., The $-Calculus Process Algebra for Problem Solving: A Paradigmatic Shift in Handling Hard Computational Problems, Theoretical Computer Science, vol.383, no.2-3, 2007, 200-243 (doi: dx.doi.org/10.1016/j.tcs.2007.04.012).
5. Eberbach E., Mikkilineni R., Morana G., Computing Models for Distributed Autonomic Clouds and Grids in the Context of the DIME Network Architecture, Proc. of 21st IEEE Intern. Conf. on Collaboration Technologies and Infrastructures WETICE 2012, Track on Convergence of Distributed Clouds, Grids and Their Management, Toulouse, France, June 25-27, 2012, 125-130, doi: 10.1109/WETICE.2012.10.
6. Eberbach E., Mikkilineni R., Cloud Computing with DNA Cognitive Architecture in the Context of Turing's "Unsinkable" Titanic Machine, Proc. of 23rd IEEE Intern. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises WETICE 2014, Track on Convergence of Distributed Clouds, Grids and Their Management, Parma, Italy, June 23-25, 2014, 125-130, DOI: 10.1109/WETICE.2014.24 .
7. Gödel K., Über formal unentscheidbare Sätze der Principia Mathematica und verwander Systeme, Monatschefte für Mathematik und Physik, 38:173-198, 1931.
8. Hindman B., "Why the Data Center Needs an Operating System", http://radar.oreilly.com/2014/12/why-the-data-center-needs-an-operating_system.html
9. Kleinberg J., Tardos E., Algorithm Design, Pearson Addison-Wesley, 2006.
10. Koza J.R., Bennett III F.H., Andre D., Keane M.A., Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann, 1999.
11. Mikkilineni, R., Comparini A., Morana G., The Turing O-Machine and the DIME Network Architecture: Injecting the Architectural

Resiliency into Distributed Computing, Proc. The Turing Centenary Conference, Turing-100, Alan Turing Centenary, EasyChair Proc. in Computing, EPiC vol. 10 (ed. A. Voronkov), Manchester, UK, June 2012, 239-251.

12. Morana G., Millilineni R., Scaling and Self-repair of Linux Based Services Using a Novel Distributed Computing Model Exploiting Parallelism, 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp 98 – 103, 2011.

13. Mikkilineni R, Morana G., Cognitive Distributed Computing: A New Approach to Distributed Datacenters with Self-Managing Services on Commodity Hardware, Int. J. Grid and Utility Computing, Vol.x, No.x, 20xx.

14. Popek G.J., Goldberg R.P., Formal Requirements for Virtualizable Third Generation Architectures, CACM, vol.17, July 1974, 412-421.

15. Pustina L., Lightweight Virtual Machines Made Simple with Docker or How to Run 100 Virtual Machines, codecentric, Jan.6, 2014, https://blog.codecentric.de/en/2014/01/lightweight-virtual-machines-made-simple-docker-run-100-virtual-maschines/

16. Reuter A., "The Straight Road: Cloud Computing", Master Project, Rensselaer Polytechnic Institute at Hartford, 2015.

17. Tanenbaum A.S., Bos H., Modern Operating Systems, 4th ed., Pearson, 2015.

18. Turing A., On Computable Numbers, with an Application to the Entscheidungsproblem, Proc. London Math. Soc., 42-2, 1936, 230-265; A correction, ibid,43, 1937, 544-546.

19. Turing A., Systems of Logic based on Ordinals, Proc. London Math. Soc., 45-2, 1939, 161-228.

20. Wegner P., Eberbach E., Burgin M., Computational Completeness of Interaction Machines and Turing Machines, Proc. The Turing Centenary Conference, Turing-100, Alan Turing Centenary, EasyChair Proc. in Computing, EPiC vol. 10 (ed. A. Voronkov), Manchester, UK, June 2012, 405-414.