Cloud Computing and Cloud Automata as A New Paradigm for Computation

Mark Burgin [1], Eugene Eberbach [2,*] and Rao Mikkilineni [3]

[1]UCLA; [2]Scopium AI; [3]Golden Gate University

markburg@cs.ucla.edu. eeberbach@gmail.com, rmikkilineni@ggu.edu,

Abstract:

Cloud computing addresses how to make right resources available to right computation to improve scaling, resiliency and efficiency of the computation. We argue that cloud computing indeed, is a new paradigm for computation with a higher order of artificial intelligence (AI), and put forward cloud automata as a new model for computation. A high-level AI requires infusing features that mimic human functioning into AI systems. One of the central features is that humans learn all the time and the learning is incremental. Consequently, for AI, we need to use computational models, which reflect incremental learning without stopping (sentience). These features are inherent in reflexive, inductive and limit Turing machines. To construct cloud automata, we use the mathematical theory of Oracles, which include Oracles of Turing machines as its special case. We develop a hierarchical approach based on Oracles with different ranks that includes Oracle AI as a special case. Discussing a named-set approach, we describe an implementation of a high-performance edge cloud using hierarchical name-oriented networking and Oracle AI-based orchestration. We demonstrate how cloud automata with a control overlay allows microservice network provisioning, monitoring, and reconfiguration to address non-deterministic fluctuations affecting their behavior without interrupting the overall evolution of computation.

Keywords: Turing Machine, Cloud Computing, Edge Cloud, Hierarchical Named Networks, Turing Oracles, Application QoS, AI, Computing Models.

1. Introduction

Cloud computing has been widely used in practice and has been studied extensively. In this paper, we will provide the justification to claim cloud computing as a new paradigm for computation.

We stress that we believe that using cloud computing can lead to the solution of problems unsolvable otherwise (see our next sections, comments and description on hypercomputation, super-recursive algorithms, and Oracles). We think that this is the biggest advantage of cloud computing (and its formal model - cloud automata) who has the roots in 1960s/1970s virtual machines from IBM360/370 series (see e.g., (Tanenbaum and Bos, 2015 [33], Tanenbaum and Wetherall, 2011 [34]). We stress the enormous potential of cloud computing, and this was possible due to introduction of cloud automata model. Note that cloud computing does not have de facto its formal model(s) (despite of its popularity and rich applications), and for these reasons some properties of cloud computing cannot be properly studied. Exactly by introducing cloud automata, we were able to harness the power of distributed computing in clouds. Our results are complete new and original, and by stressing the emergence of new computational paradigm we advocate to use cloud computing for problems that either cannot be solved at all or not to be solved effectively by traditional computers.

This is exactly the novelty of our approach - at least we were not able to find similar work by other authors on that subject. However, our references are extensive, including the references to our earlier publications and many other publications by other authors (because nothing is born completely in the vacuum). In this shape and formulation, this is the first attempt to provide a uniform formal model for cloud computing. In fact, we started already to work on extension of our journal paper to a more complete monograph/book.

It is possible to demarcate three stages of the Internet:

1. The address-oriented networking, when connection to systems and resources, is performed by their addresses in the network.

2. The name-oriented networking, when connection to systems and resources, is performed by their names.

3. The hierarchical name-oriented networking when connection to systems and resources is performed by connection to the relevant host, which provides systems and resources by their names.

Examples of the hierarchical name-oriented networking are cloud computing (networking), fog computing (networking), or mist computing (networking).

Now all three types of computing and networking coexist using the Internet as their communication space. However, current Internet architecture supports (is oriented at) only address-oriented networking. To achieve transition from address-oriented networking to name-oriented networking, it is possible to use two approaches. One of them demands transformation of the ground Internet architecture. The other one does not demand such immense transformations because it uses network agents, such as DIME from (Burgin and Mikkilineni, 2018 [14]) or middleware CORBA or Java rmi [34] to convert names to addresses.

Functioning of the Internet and the basic tendency of its development explicitly show that naming is a fundamental issue of growing importance in distributed systems. From the very beginning, names in the form of addresses in the network were crucial communication tools in any network. As the number of directly accessible systems and resources grows, it becomes increasingly difficult to discover the addresses of objects of interest, and it becomes vital to use content names instead of addresses.

Named Data Networking (NDN) (related to Content-Centric Networking (CCN), content-based networking, data-oriented networking or information-centric networking) is a Future Internet Architecture inspired by years of empirical research into network usage and a growing awareness of unsolved problems in contemporary internet architectures.

In an intentional naming and resolution architecture, applications describe their intent and specify what they are looking for but not where it is situated. This shifts the burden of resolving 'what is desired' to 'where it is' from the user to the network infrastructure. It also allows applications to communicate seamlessly with end-nodes, despite changes in the mapping from name to end-node addresses during the session.

Although Internet developers and users come to better understanding of the vital importance of names and naming, they see only the top part of the "naming iceberg" situated above the surface of common-sense knowledge. The major part is under the surface where the underlying structures, procedures, and their theories dwell. The key structure in this domain is a named set, the imperative procedures are naming renaming and other operations with named sets, and the fundamental theory is named set theory (Burgin, 2011 [5]). When people are discussing names, they often forget that names are names because they name something, that is, they are related to some things by naming relations. As a result, we come to the structure of a named, which consists of three components: a system of names, a collection of named objects and a naming correspondence connecting names with objects.

Thus, names are implicitly used in information processing and networking technology. At the same time, understanding possibilities opened by named sets and their explicit utilization can rapidly accelerate development and efficiency of all kinds of networks including the Internet.

2. Cloud computing as a new paradigm for computation

Historically, we can distinguish many paradigms of computation, including Mainframe, PC, Network, Internet, Distributed Computing, Grid Computing, and Cloud Computing. From those the youngest is Cloud Computing. New computing paradigms may involve various technologies besides VLSI, i.e., quantum computing, biologically

inspired computing, nanocomputing, optical computing, neurocomputing. Special role is played by superTuring computing, known also as hypercomputation, which is directed and controlled by superrecursive algorithms and can be implemented using described technologies as well as new emerging tools. We will concentrate in this section on cloud computing justifying that it is in essence a new computational paradigm. This is important because many scientists consider clouds simply as an extension of either client/server architectures, or distributed systems, or an extension of IBM virtual machines e.g., (Tanenbaum and Bos, 2015 [33], Tanenbaum and Wetherall, 2011 [34]).

Hypercomputation plays a special role in cloud computing. We will see in the next sections that cloud computing, because of its distributed nature combined with the presence of oracles can solve problems unsolvable by Turing machines. This may justify partially the popularity and attractiveness of cloud computing, although current clouds are designed mostly for higher efficiency and user convenience. Hypercomputational features have to be explored yet in a new environment.

Many scientists are convinced that computations going beyond Turing machines are possible, while contemporary and future computers can compute and will compute beyond the Turing limit using, for example, reactive programs such as operating systems or client/server Internet computing (Burgin, 2005 [2]; Eberbach, 2017 [23]). Cloud computing also belongs to this category leading researchers to more powerful models of computation and computing devices.

Computations that cannot be realized by Turing machines are called hypercomputation or superTuring computations. Algorithms for realization of hypercomputation are called superrecursive algorithms. Models of hypercomputation and computing devices (machines) that can perform hypercomputation are called superrecursive automata.

Many such models (some of them, surprisingly, are as old as Turing machine), have been proposed including Alan Turing's o-machines (Turing machines with an Oracle), c-machines and u-machines, limit recursive functions, analog neural networks, interaction machines, π-calculus, $-calculus, inductive Turing machines, limit Turing machines, infinite time Turing machines, grammars with prohibition, correction grammars, periodic Turing machines, general Turing machines, accelerating Turing machines, inductive cellular automata, evolutionary Turing machines and evolutionary finite automata (Siegelman, 1999 [32]; Hamkins and Lewis, 2000 [27]; Case and Jain, 2011[16]; Case and Royer, 2016 [17]; Burgin, 2005; 2005a; 2014; 2015 [2,3,6,7]; Burgin and Eberbach, 2009; 2012 [11,12]; Eberbach, et al, 2004 [24]; Eberbach, 2005; 2007; 2015 [20-22]).

Superrecursive models derive their higher than the Turing machines computing power using three principles: interaction, evolution, or infinity (Eberbach, et al, 2004 [24]). In the interaction principle the model becomes open and the automaton (agent) interacts with either a more powerful component or with infinitely many components. In the evolution principle, the model can evolve to a more expressive one using non-recursive variation operators or by pure chance. In the infinity principle, models can use unbounded resources: time, memory, the number of computational elements, an unbounded initial configuration, an infinite alphabet, etc. These principles bring forth superrecursive models that derive their higher computational power either from infinite resources or from already superrecursive sources.

There are also two more principles, which allow perform hypercomputation utilizing conventional sources and resources. Using the principle of implicit result, the machine can reach higher computational power when obtaining the result, it does not inform about this. In the principle of randomness, higher computational power is achieved due to randomness in functioning.

All these innovations are coming to cloud computing, which is classified in the following way.

The functional typology of clouds:

1. A univalent cloud provides only one service or virtual system.

2. A specialized cloud provides only one type of services or virtual systems.

3. A multipurpose cloud provides several types of services or virtual systems.

Let us consider some examples.

Microsoft's Office Online, which provides Internet-only versions of Word, Excel, PowerPoint, and OneNote, is a specialized cloud accessed via browser without installing anything.

Google drive, which provides Google docs, Google sheets, etc., is a specialized cloud.

Apple iCloud, which provides online storage, backup and data synchronization service, is a univalent cloud

Amazon Cloud Drive, which provides the storage for anything digital bought on Amazon such as kindle books, music, as well as unlimited image storage, is a univalent cloud

Dropbox, which provides the synced version of user's files online, is a univalent cloud

Microsoft Azure, IBM Smart Cloud, Oracle Cloud, Google Compute Engine, Amazon Web Services (AWS) are examples of multipurpose clouds.

In addition, three kinds of univalent and specialized clouds are utilized:

- SaaS (Software as a service) clouds deliver an application over the Internet by a subscription. It is not installed on a company's servers or on a person's PC. Salesforce.com is the granddaddy of this category, but other examples include things like Google Apps, Microsoft's Office 365, or the human resources suite Workday. When the term "cloud" is used for consumers, it typically means SaaS such as Dropbox, iCloud, Evernote, and so on.

- PaaS (Platform as a Service) clouds form the next layer up, where it is possible to build user's cloud application by rent everything that is used including the runtime platform like Java, Ruby or .Net. Examples of PaaS are Google App Engine, Microsoft Azure, and Salesforce.com's Heroku.

- IaaS (Infrastructure as a Service) clouds are the most basic, where hardware (server with operating system, storage, and networking) is rented and user's applications are uploaded. The difference between IaaS and old-school hosting is that users are sharing the hardware with other renters. Amazon AWS is the biggest IaaS cloud, Rackspace is another while Linode is popular for Linux users.

One more cloud classification is based on their communication features.

The communication typology of clouds:

1. A public cloud offers communication with it over a public network such as the Internet.

2. A community cloud shares computing resources across several organizations, and can be managed by either organizational IT resources or third-party providers.

3. A private cloud restricts communication with it to an infrastructure for an organization maintained on a private network hosted internally or externally.

There are also four types of hybrid clouds, which combine the elements of two or all three other types of clouds. Usually they utilize public cloud for non-sensitive communication and private setup for sensitive communication.

Let us consider some examples.

Microsoft Azure, IBM Smart Cloud, Oracle Cloud, Google Compute Engine, Amazon Web Services (AWS) are examples of public clouds. On the other hand, private clouds are maintained by single company, e.g., military, government and other organizations for their internal needs with strictly controlled and very restricted access from outside.

3. Elements of the theory of oracles

This section is dedicated to hierarchical approach in artificial intelligence based on the mathematical theory of oracles (Burgin, 2016; 2017 [8,9]). This approach includes Oracle AI (Armstrong, et al, 2012 [1]) as a special case providing new tools for exploration of artificial intelligence in general and Oracle artificial intelligence in particular. This section is also a prerequisite for the cloud automata model from section 4.

Because there are so many meanings of the term Oracle coming from different areas, e.g., from an ancient Greece, from Turing o-machine (Turing, 1939 [35]), or an Oracle software company, we should justify its use, and why it is not called simply a management automaton, cloud IMP, Gateway, Router performing cloud management functions.

In fact, we can speak about many types of Oracles, and we can distinguish the whole hierarchy of Oracles, utilized by different researchers while being formalized and organized in the mathematical theory of Oracles (Burgin, 2017 [9]).

Very often people assume that oracles only give information. This opinion is true in general (Curnow, 2004 [19]; Leeson, 2014 [28]) and is also related to the role of oracles in artificial intelligence where an Oracle AI is a confined AI with no physical manipulators, with which we can only interact through text messages (Armstrong, et al, 2012 [1]). In the mathematical theory of oracles, it is postulated that oracles can have much wider functions (Burgin, 2017 [9]).

When M is an oracle for T, we call T a client of M.

It is necessary to distinguish various types of oracles as they can behave differently and have different properties. Knowledge of these properties is important for the development of high level artificial intelligence.

According to the relational classification, we have the following types of oracles.

- Independent oracles function independently of other systems.
- Functioning of partially dependent oracles partially relies on other systems.
- Functioning of dependent oracles completely relies on other systems.

Partially dependent oracles and dependent oracles can pursue their own goals or can do some work for other systems. In the latter case, we have agent oracles.

There are three types of agent oracles:

- Controlled agent oracles
- Regulated agent oracles
- Autonomous agent oracles

Control is mostly performed by prescribing actions, i.e., by instructing (telling) what to do, while regulation is typically organized by restricting actions, i.e., by telling (instructing) what not to do. Restrictions play an important role in the brain functioning, and this has to be reflected in artificial intelligence based on mathematical models, which incorporate not only generating rules but also excluding imperatives. An example of such models is given by grammars with prohibition (Burgin, 2005a; 2015 [3,7]).

A useful structure for AI is a hierarchy of oracles when machines (systems) are organized in layers so that a machine (system) in the layer n can be an oracle only for machines (systems) in the layer n – 1 and lower. Naturally, when such a hierarchy exists, machines (systems) in the lowest layer are not oracles for machines in higher levels in this hierarchy. This structure allows ascribing ranks to oracles in the following way. An oracle of the rank 1 is an oracle for one or several basic systems (Burgin and Mikkilineni, 2018 [14] [; Burginl, 2015; 2016 [7,8]). Assuming that functioning of machines is organized according to the hierarchical structure, an oracle of the rank n is an oracle for one or several oracles of the rank n – 1.

Note that in general, it is possible that a machine A plays the role of an oracle for a machine B in some situations while in other situations, the machine B plays the role of an oracle for a machine A.

Hierarchic intelligence forms a hierarchy of oracles over the base information processing system. There are several architectural solutions for implementation of this structure. A linear hierarchy of oracles is presented in Figure 1.

Figure 1: A linear hierarchy of oracles

In this figure:

– B is the base information processing system

– O_1 is an oracle for B

– O_2 is an oracle for O_1

– O_3 is an oracle for O_2

……

– O_n is an oracle for O_{n-1}

The organization of oracle interactions when an oracle interacts with, e.g., supervises, several systems of lower ranks is more efficient than linear ranking of oracles presented in Figure 1. An expanding 3-tier hierarchy of oracles is presented in Figure 2.

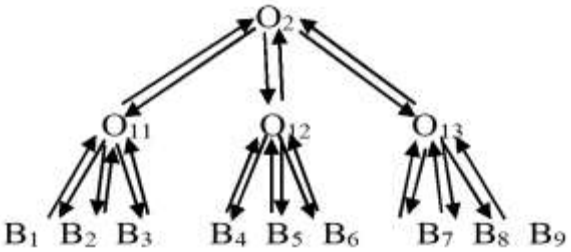


Figure 2: An expanding 3-tier hierarchy of oracles

In this figure:

– B_1 , B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9 are the base information processing systems

– O_{11} is an oracle for B_1, B_2, B_3

– O_{12} is an oracle for B_4, B_5, B_6

– O_{13} is an oracle for B_7, B_8, B_9

– O_2 is an oracle for O_{11}, O_{12},O_{13}

An expanding hierarchy of oracles allows better organization of information processing and artificial intelligence than a linear hierarchy of oracles.

Different types of oracles have different properties. Let us consider transitivity of the relation of being an oracle.

We remind that according to the operational classification of oracles, there are the following types of oracles (Burgin, 2017 [9]):

1. A *simplification oracle M* for *T* in the domain *D* of problems is a system that solves all problems from *D* more efficiently (with less complexity) than *T*.

2. An *augmentation oracle M* for *T* in the domain *D* of problems is a system such that all problems from *D* are intractable for *T* but tractable for *M*.

3. An *extension oracle M* for *T* in the domain *D* of problems is a system such that all problems from *D* are unsolvable for *T* but solvable for *M*.

Theorems 4.1, 4.2, and 4.6 from (Burgin, 2017 [9]) imply the following result.

Proposition 3.1. Relation "to be a simplification oracle in a fixed domain" is transitive.

This property of oracles allows proving the following result.

Theorem 3.1. Simplification oracles in a fixed domain form a hierarchy.

We remind that:

– A hierarchy is a strict partial order

– A descending hierarchy is a DAG (directed acyclic graph) with arrows going from the top to the bottom

– An ascending hierarchy is a DAG (directed acyclic graph) with arrows going from the bottom to the top

Theorems 4.10 and 4.12 from (Burgin, 2017 [9]) imply the following result.

Proposition 3.2. Relation "to be an augmentation oracle" is transitive if all problems intractable for an oracle are also intractable for its client.

This property of oracles allows proving the following result.

Theorem 3.2. Augmentation oracles form a hierarchy if all problems intractable for an oracle are also intractable for its client.

Theorem 4.21 from (Burgin, 2017 [9]) implies the following result.

Proposition 3.3. Relation "to be an extension oracle in a fixed domain" is transitive if all problems unsolvable for an oracle are also unsolvable for its client.

This property of oracles allows proving the following result.

Theorem 3.3. Extension oracles in a fixed domain form a hierarchy if all problems unsolvable for an oracle are also unsolvable for its client.

However, similar results are not true for some types of oracles.

We remind (Burgin, 2017 [9]) that according to the functional classification of oracles, there are the following types of oracles.

1. An *informative oracle M* for *T* is a system that provides some necessary information for *T*, which *T* does not have.

2. A *performing oracle M* for *T* is a system operation of which improves the functioning of *T*.

3. A *service oracle M* for *T* is a system that provides services for *T* that cannot be done by *T*.

The functional classification of oracles describes the functions or roles of oracles.

Proposition 3.4. Relation "to be an informative oracle" is not transitive.

Indeed, let us assume that a computing device *M* computes a function *f* such that a computing device *T* cannot compute and a computing device *B* computes a function *g* such that a computing device *M* cannot compute, but the computing device *T* can compute all function computable by *B*. Then *M* is an informative oracle for *T* and *B* is an informative oracle for *T*, but *B* is not an informative oracle for *T*.

Definition 3.1. A *forecasting oracle M* for *T* is a system that forecasts the future of *T*.

Proposition 3.5. Relation "to be a forecasting oracle" is not transitive.

Indeed, let us assume that a system *M* forecasts future of *T* and a system *B* forecasts future of *M*. However, it is possible that *B* cannot forecast future of *T* and thus, *B* is not a forecasting oracle for *T* although *B* is a forecasting oracle for *M* and *M* is a forecasting oracle for *T*.

Definitions of simplification oracles employ an arbitrary complexity function *c* (Burgin, 2017). This allows considering specific simplification Oracles for various complexity functions, such as time complexity, space complexity, and operational complexity.

Definition 3.2. A simplification Oracle *M* is called:

a) a *temporal simplification Oracle* if *c* is time complexity;

b) a *space simplification Oracle* if *c* is space complexity;

c) an *operational simplification Oracle* if *c* is operational complexity.

Oracles are used in a variety of areas (Burgin, 2017). Here we use the concept of an oracle to construct models of cloud computation.

4. **Cloud automata as a new model for computation**

To implement and investigate cloud computing, we introduce the cloud automata model. To better understand the modeling approach in science and engineering, we describe a classification of models

There are three basic types of system models (Burgin, 2005 [2]):

1. A *static structural model* represents the structure of the system.
2. A *functional model* describes functions of the system.
3. A *process model* portrays processes in the system.

Now let us apply this classification to cloud computing.

A *functional model* of cloud computing describes functions of the elements and components involved in this process.

Actor model and system actor model are appropriate functional models of cloud computing. In this aspect, system actor model provides more tools in comparison with actor model for modeling cloud computing.

A *process model* of cloud computing describes dynamics of cloud computing in the form of a system of interacting processes.

The π-calculus and CCS (Milner, 1980 [30]), the $-calculus (Eberbach, 2007 [21]), and EAP (event-action-process) model of concurrent processes (Burgin and Smith, 2010 [15]) are proper process models of cloud computing.

There are three basic types of control/ management organization in a cloud. A cloud is:

- *user-managed* if the user manages functioning computers (devices), which provide services to this user through the cloud
- *provider-managed* if the provider manages its functioning
- *autonomic* if it itself manages its functioning

In our exploration of cloud computing, we start with elaboration of a static structural model of cloud computing in the form of a specific automaton called a cloud automaton.

A *cloud automaton* consists of three components (M, C, O):

- The user (basic) automaton M
- The communication space C
- The Oracle O (in a Cloud)

In a general case, the *user automaton* is a distributed system of *unit user automata*, each of which the cloud user utilizes for computation or more general, for solving problems. The most convenient and flexible mathematical model of a user automaton is a grid automaton.

In a cloud automaton, the *Oracle* can serve for providing information, services and/or virtual systems, e.g., computing and storage facilities, virtual software and hardware, data, and infrastructures, to the user automaton via the *communication space* (Burgin, et al, 2001 [13]).

The most popular communication space for clouds is the Internet. However, local, community or private networks are also used for this purpose.

The user (basic) automaton can be a distributed system such as a grid automaton or a structural machine in the form of multi-core or multicomputer.

In Turing machines with an oracle, the oracle and communication space are not specified (cf., for example, (Rogers, 1987 [31]). In advice Turing machines, the oracle is the tape with advice and communication space is the same tape.

There are various kinds of oracle substance giving us distinct classes of oracles – human beings, physical machines, abstract automata, functions, etc. (cf., for example, (Burgin, 2017 [9]). Here we are interested mostly in oracles in the form of an automaton as we build a flexible model of computing.

Definition 4.1. A cloud automaton is called *confined* if its Oracle is also an automaton.

In what follows, we study only confined cloud automata.

Let us consider two classes of automata ***A*** and ***B***.

Definition 4.2. A confined cloud automaton is called an ***AB***-cloud automaton if its Oracle belongs to ***B*** and its basic automaton belongs to ***A***.

Example 4.1. If **IT** is the class of all inductive Turing machines (Burgin, 2005) and **T** is the class of all Turing machines, then **T,IT**-cloud automaton has a Turing machine as its user machine and an inductive Turing machine as its Oracle.

Note that that both the user machine and its oracle in a cloud automaton can belong to the same class of automata. For instance, a universal inductive Turing machine can be an oracle for an inductive Turing machine, which is equivalent to a Turing machine.

The Oracle can be informative, performing, and/or service Oracle although usually it is an extension present-time service Oracle in the sense of (Burgin, 2017). This shows that Turing Oracle (Turing, 1939) is too weak and inadequate for modeling cloud computing. The most adequate model uses Oracle in the sense of (Burgin, 2017 [9]).

The Oracle in physical clouds is, as a rule, a distributed system such as a grid automaton or a structural machine.

The Oracle resides in a cloud outside the prime automaton.

A *cloud* is not exactly defined being a dynamic part of (a place in) the information space where the Oracle or several Oracles of a cloud automaton function.

To have an efficient model of cloud computing, we generalize the concept of a cloud automaton.

A *multicloud automaton* consists of three components (***M***, *C*, ***O***):

- A system ***M*** of user automata

- The communication space *C*
- A system ***O*** of Oracles

If a cloud is a union of several collaborating clouds, it is called a *cloud federation*. A multicloud automaton is a model of a cloud federation.

Let us consider relations between cloud automata and other types of abstract automata from the oracle perspective in the context of artificial intelligence.

To build a high-level artificial intelligence, it would be useful to incorporate certain features of the human brain. One of the central features is that the brain learns all the time and learning is incremental (Hawkins, 2017). Consequently, for artificial intelligence, we need to use computational models, which reflect incremental learning without stopping. These features are inherent in reflexive Turing machines, inductive Turing machines and limit Turing machines (Burgin, 2005 [2]).

In his talk at the International Congress of mathematicians in 1958, Kleene formulated a conjecture that it might be possible that algorithms that change their programs while computing would be more powerful than Turing machines. Such algorithms were formalized by building the mathematical model called *reflexive Turing machine*. However, it was proved in by Burgin that the class of reflexive Turing machines is equivalent to the class of Turing machines, i.e., these machines have the same computing power. This result disproved Kleene's conjecture. At the same time, it was also proved that reflexive Turing machines are more efficient than Turing machines. Namely, a reflexive Turing machine can outperform any Turing machine that computes the same function. The latter assertion gives us the following result.

Proposition 4.1. For any Turing machine *T*, there is a reflexive Turing machine *R*, which is a temporal simplification oracle for *T*.

In essence, a reflexive Turing machine learns continuously and its learning is incremental similar to the processes in the brain. Moreover, a reflexive Turing machine acquires operational knowledge, which allows improving its functioning without stopping the computational process.

Another important mathematical model of computation connected to cloud automata is called an agent machine.

Definition 4.3. An *agent machine* is an automaton with two components – the basic machine and the agent.

There are different types of agent machines and their agents:

A *memory agent* prepares memory, e.g., establishes links between memory cells or adds new cells to the memory, for the basic machine.

A *data agent* prepares data for the basic machine performing preprocessing of data.

A *software agent* prepares the program, e.g., changes instructions or adds new instructions, for the basic machine.

Example 4.2. Inductive Turing machines of the second and higher orders have memory agents (Burgin, 2005).

Example 4.3. Reflexive Turing machines can organize their functioning with software agents (Burgin, 1993a).

Example 4.4. The nervous system in the human eye is a data agent.

Indeed, information from visual receptors (rods and cones) is not directly transmitted to the brain, but at first, it is preprocessed by three kinds of neurons (bipolar cells, horizontal cells, and amacrine cells) in the retina, which play the role of data agents in the first stratum. Then it is preprocessed by ganglion cells, which plays the role of data agents in the second stratum and transmit the resulting information to the brain (Gray, 1994).

Definition 4.4. An *agent Turing machine* is an agent machine, in which both components – the basic machine and the agent machine – are Turing machines.

For instance, inductive Turing machines of the first order are agent Turing machines (Burgin, 2005). It is proved that agent Turing machines can have higher computing power than conventional Turing machines.

Definition 4.5. An *agent inductive Turing machine* is an agent machine, in which both components – the basic machine and the agent machine – are inductive Turing machines.

For instance, inductive Turing machines of the second and higher orders are agent inductive Turing machines (Burgin, 2005).

Proposition 4.2. For any Turing machine T with time complexity higher than $O(n)$, there is an agent machine A, which is a temporal oracle for T.

Proof is based on the results from (Burgin, 2005 [2]).

5. Properties of cloud automata

In this section, we explore computational power of cloud automata, their user automata and Oracle taking into account that they can be finite automata, pushdown automata, Turing machines, o-machines, inductive Turing machines, π-calculus, etc.

We begin with the role and properties of the communication space of cloud automata detaching several classes of communication spaces.

Definition 5.1. A communication space is called a *communication channel* if it is a transmitter, i.e., all its inputs are outputs of the communicating systems (user automata and the oracle of a cloud automaton in our case), and all its outputs are inputs of the communicating systems.

Informally, it means that a communication channel as a communication space does not get inputs from and does not produce outputs going to its environment.

At the same time, a communication channel as a communication space can transform and even additionally process data that go through it. For instance, e-mail systems are used as communication channels, and many of them perform correction of misprints in e-mails. The screen of a contemporary computer as a communication channel executes many other operations with data.

This feature of communication spaces allows differentiation of special types of channels.

Definition 5.2. A communication channel is called *plain* if it only transmits data.

However, in practice, there exists distortion of transferred data. For instance, data can be corrupted by noise in channels with noise (Shannon,).

Definition 5.3. A communication channel is called *exact* if there is no distortion of transferred data.

In abstract computing devices, such as abstract automata, communication channels are, as a rule, exact. In the case of physical computing and especially communicating devices, engineers elaborate various tools to make communication channels exact.

An interesting question is whether a cloud automaton can have higher computing power than its oracle and user automaton. The answer is given by the following result.

Proposition 5.1. There is a cloud automaton with higher computing power than its oracle and user automaton.

Let us consider some examples of cloud automata with that property.

Example 5.1. A Turing machine with advice *M* () is a cloud automaton in which a conventional Turing machine *T* is the user automaton and a tape *L* with information is the oracle. It is known that a Turing machine with advice has higher computing power than a conventional Turing machine. Besides, *M* can compute more than the tape *L* because a tape cannot compute at all.

Example 5.2. Let us consider a cloud automaton *C* in which a finite automaton *A* is the user automaton, and a conventional Turing machine *T* is the oracle. The communication space of *C* is a moving tape. It is demonstrated in (Burgin, 2007), *C* can compute what is incomputable by any Turing machine.

One more example describes how the communication channel can increase the computing power of a cloud automaton. In this case, the cloud automaton *C* has a finite automaton *A* as the user automaton and a conventional Turing machine *T* as the oracle. At the same time, the communication space of *C* is a channel *ch* that includes another conventional Turing machine *M*. It is demonstrated in (Burgin, 2007 [4]), *C* can compute what is incomputable by any Turing machine. The cloud automaton *C* works in the following way. The automaton *A* sends its input directly to the machine *T* and when *T* obtains the result, it sends it back to *A* through the communication channel *ch*. Then the machine *M* works with this result *r* excluding it when *M* accepts *r* and sending *r* to when *M* does not accept *r*. When *A* receives *r*, it gives *r* as the output of the cloud automaton *C*.

Proposition 5.2.

a) The constructed cloud automaton *C* is linguistically equivalent to a grammar with prohibition.

b)) Any grammar with prohibition is linguistically equivalent to a cloud automaton with the same structure as the constructed cloud automaton *C*.

It is proved in (Burgin,) that grammars with prohibition computationally are more powerful than Turing machines. Consequently, the cloud automaton *C* can have more computational power than its oracle in the form of a Turing machine *T* and its basic automaton *A*.

In Section 3, we considered hierarchies of oracles. These hierarchies induce hierarchies of cloud automaton.

Let us consider two classes of automata ***A*** and ***B***.

Definition 5.42. A confined cloud automaton is called an ***AB***-cloud automaton if its Oracle belongs to ***B*** and its basic automaton belongs to ***A***.

Taking two classes of automata ***A*** and ***B***, we prove the following result.

Theorem 5.1. If ***B*** is a hierarchy of oracles, then ***AB***-cloud automata with exact plain communication channels also form a hierarchy.

In particular, if we have ranks of oracles, e.g., those that are defined in Section 3, we can define corresponding rank of cloud automata.

Definition 5.5. The *oracle induced rank* of a cloud automaton *C* is equal to the rank of its oracle.

However, it is necessary to understand that cloud automata with higher oracle induced rank have higher computing power or lower complexity of computing in comparison with cloud automata with lower oracle induced rank. This is caused by utilization of sophisticated communication spaces, which can inverse oracle induced ranking.

6. A New High-Performance Edge Cloud Using Named Information Processing Networks

A distributed information processing system (known as an *application* in the current IT parlance) is a collection of independent computation structures (micro- or mini- or mega- services, immutable or mutable, stateless or stateful) that appear to its users as a single coherent system executing functional requirements defined by business requirements. An example is a three-tier application consisting of a web server, business logic executor, database, etc., or a machine learning stack. Distributed systems often have such characteristics as:

1. Concurrency of components, lack of a global time scale, scalability and independent failures of components,

2. Underlying transparency (inherited by cloud computing), hiding unnecessary details of management, networking, multi-computer implementation and a network of services, and

3. A uniform representation of the whole computing system.

Distributed computing utilizing networked computing resources, starting from a client-server computing model, has fully developed to current cloud computing implementations where hundreds of physical and virtual servers are used in clouds to provide orchestrated computational workflow execution. With the steady increase of computing power in each node and increasing connectivity bandwidth among the nodes, during the last three decades, sharing of distributed resources by multiple applications to increase utilization has improved the overall efficiency in implementing business processes and workflow automation.

Sharing of resources and collaboration provide leverage and synergy, but also pose problems such as contention for same resources from different applications, failure of participating nodes, issues of trust, and management of latency and performance. There are three major attributes that must be considered in designing distributed systems:

Resiliency: Collaboration of distributed shared resources can only be possible with a controlled way to assure both connection and communication during the period of collaboration. In addition, the reliability, availability, accounting, performance, and security of the resources have to be assured so that the users can depend on the service levels they have negotiated for. The FCAPS (fault, configuration, accounting, performance and security) management allows proper allocation of resources to appropriate computations consistent with business priorities, requirements and latency constraints. It also assures that the connection maintains the service levels that are negotiated between the consumers and the suppliers of the resources. Resiliency therefore consists of the ability to:

1. Measure the FCAPS parameters both at the individual resource level and at the system level and

2. Control the resources system-wide based on the measurements, business priorities, varying workloads, and latency constraints of the distributed application.

In an ideal environment, resources are offered as services and consumers who consume services will be able to choose the right services that meet their requirements or the consumers will specify their requirements and the service providers can tailor their services to meet consumer's requirements. The specification and execution of services must support an open process where services can be discovered and service levels are matched to consumer requirements without depending on the underlying mechanisms in which services are implemented. In addition, service composition mechanisms must be available to dynamically create new value-added services by the consumers.

Efficiency:

The effectiveness of the use of resources to accomplish the overall goal of the distributed system depends on two components:

1. Individual resource utilization efficiency which measures the cost of executing a task by the component with a specified service level associated with functional requirements, and

2. The coordination and management cost which assures that the distributed components are contributing to the overall goals of the system with specified end-to-end transaction service level defined by non-functional requirements.

The efficiency is measured in terms of return on investment (ROI) and total cost of ownership (TCO) of the distributed application.

Scalability:

As the requirements in the form of business priorities, workload variations or latency constraints change, the distributed system must be designed to scale accordingly. The scaling may involve dialing-up or dialing-down of resources, geographically migrating them and administratively extending the reach based on policies that support centralized, locally autonomous or a hybrid management with coordinated orchestration

Therefore, strictly speaking, distributed computing, should address, a) the computational workflow execution and b) the management workflow which addresses the management of the network of computing nodes to assure connectivity, availability, reliability, utilization, performance, and security (known as fault, configuration, accounting, performance and security, FCAPS management) of the distributed application.

Today, everyone unequivocally agrees that the current datacenter, Internet and Cloud based services are poor in providing data security. One only has to look at the news about how billion-dollar companies were the subjects of massive security breaches. In addition, assuring right resources to the application in terms of CPU and memory to execute the computation, bandwidth to assure network latency requirement and storage capacity, throughput and IOPs to manage access to data is costly, complex and involves myriad silo management systems. Assuring availability, performance efficiently utilizing multiple distributed resources offered by various service providers without vendor lock-in increases complexity involving myriad stacks of computing, networking and storage infrastructure along with a plethora operating systems, middleware and orchestrations systems evolved over decades.

In spite of large investments, there is no single architecture or solution that confidently addresses the shortfall. As George Gilder says in his book Life after Google, "You start by defining not the goal but the ground state. Before you build the function or the structure, you build the foundation."

A new foundation has to encompass the dynamic management, with end to end system-wide visibility and control, of the specific application. Before we build function or structure, we must build the foundation that lays an architecture for realization of these functions. In this paper, we present a new architectural foundation

that addresses some of the weakness of current application provisioning, monitoring and QoS assurance infrastructure.

Cloud computing in a named network:

Current distributed application (consisting of service components) uses shared resources from different sources and they are communicating with each other using an address-oriented networking. Figure 3 shows the evolution of an address-oriented network of information processing and networking structure.

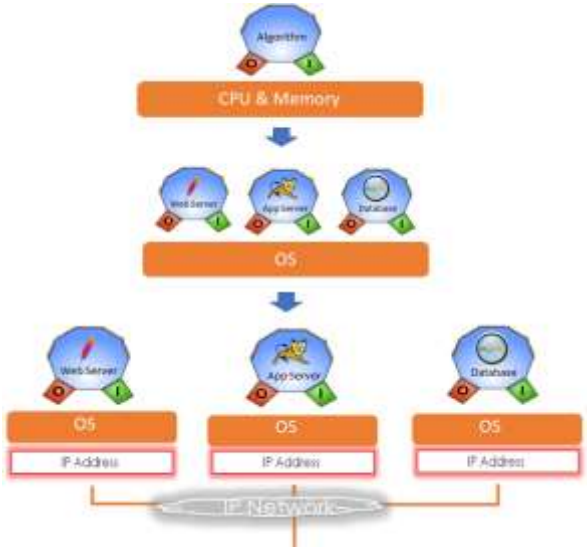


Figure 3: Evolution of computational structures from the von Neumann (universal Turing machine) architecture to Addressed information processing and Networking organization.

Information processing, over the last six decades, has evolved from the von Neumann (universal Turing machine) architecture based computations to information networking architecture with addressed network components in the form of IP addresses. An operating system facilitates shared resources (CPU and memory) among multiple information processing structures within a computing device such as a server. The information networking structure provides shared resources across a set of distributed information processing structures with their own addresses.

Cloud computing manages the resources across multiple distributed information networking structures with a hierarchical addressing to provision, monitor and adjust the resources as required when there are fluctuations in the demand or availability based on non-deterministic events. Figure 4 shows a hierarchical named network managed by a resource management Oracle discussed in this paper.

Figure 4: Multiple named networks are managed by Resource Oracle or a Resource Oracle network

Raison d'être for a new approach

There are four major business drivers influencing the current information technology architecture:

1. Velocity of information is constantly increasing with global communication, collaboration, and commerce demanding "always-on" applications and connectivity between them.

2. The explosive growth of the Internet of Things (IoT) and the associated data sets is creating a demand for pushing computing close to the sources of data. The trend towards more immersive and interactive user interfaces is flipping the center of gravity of data production and processing away from centralized data centers and public clouds and out to the edge.

3. New generation of artificial intelligence and machine learning with data analytics from multiple sources is creating a demand for high-performance and low latency information processing both at the edge and in the core multi-cloud back-end.

4. Non-deterministic fluctuations in the demand for or availability of the resources and the velocity of information are driving the complexity of current resource orchestration-based cloud computing architecture with overlays of legacy networking stacks, virtual machine networking stacks and myriad resource orchestration tools and point solutions.

In addition, new networking stacks supporting high bandwidth and low-latency memory with backend SSD/DRAM storage technologies are offering new information processing and networking architectures supported by micro-service orchestration tools to deploy applications, monitor them and rearrange them on any executable computing venue. In essence, while existing state of the art locks the application in the operating system and orchestrates the reconfiguration of infrastructure, the new architectures provide decoupling of the application from the operating system using named service microservice networks In the next section we describe an edge computing solution that uses named micro-service networks operating on static named resource networks that provide order of magnitude improvement in resiliency, efficiency and scaling.

Hierarchical Named Information Processing Networks

As we have discussed, the foundation for an information processing entity (IPE) comes from the stored program implementation of the Turing machine. However, IPE networking architecture augmented the information processing power by facilitating communication among IPEs and allowing shared resources for computation within a computing machine or a network of these machines. First phase of information networking used their host (or logical) address (usually an IP address in a private or public networks). A domain name server (DNS) provides the mapping between the location and the IPE. Inside each host, multiple IPEs are distinguished by various ports. During the past five+ decades, IP networking using IP addresses has been the dominant protocol providing the information networking services establishing links between the IPE nodes.

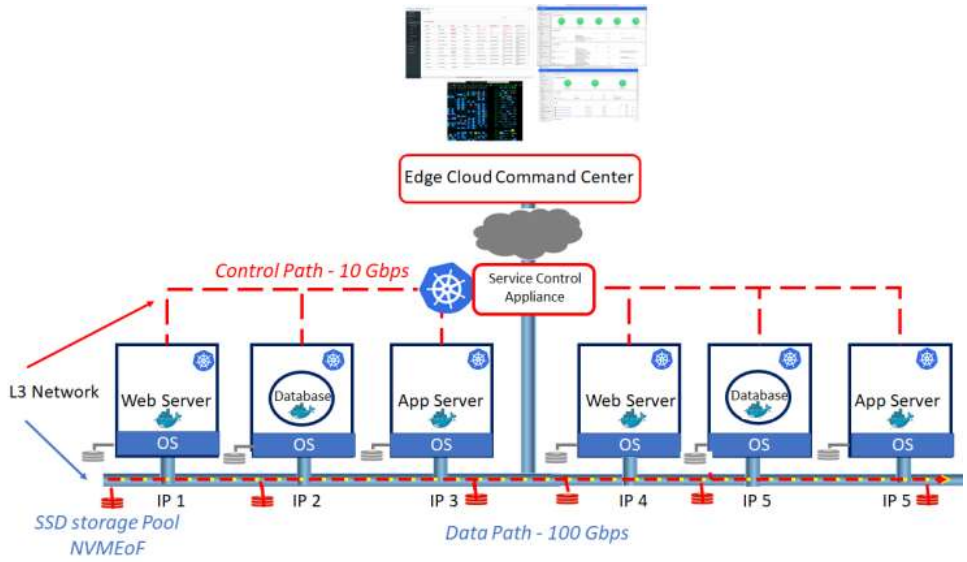
Global connectivity of interacting IPEs, the ephemeral nature of some of their communications impacted by non-deterministic fluctuations in their environment and the nature of data they exchange (text, voice, and video) are necessitating a new approach going beyond the physical address-based networks. The second phase provides location transparency by addressing data or content ("what" instead of "where" to access). Data-Oriented Network Architecture replaces DNS names with flat, self-certifying names and a name-based any-cast primitive above the IP layer. CCN uses an overlay network where servers and routers can find content packets located anywhere on a CCN network by consulting two specialized tables: the pending interest table and the forwarding information base. The FIB lists where content is currently stored, while the PIT traces how past requests were forwarded. Nodes can also pluck content packets they've cached in their own content store (CS) to satisfy requests.

More recently, the focus has been shifting to IPEs as microservices which provide specific business (or other) functions each acting as an "active" node hiding the details of implementation (which itself could be a subnetwork of IPEs). This provides a new class of hierarchical name-oriented microservice networking where

each node executes a specific behavior based on the input. A control overlay allows microservice network provisioning, monitoring, and reconfiguration as required to address fluctuations in their behavior.

In essence, such a hierarchical named network of IPEs allows composition of micro-services to create a managed process workflow by enabling dynamic configuration and reconfiguration of the micro-service network. Examples are Kubernetes orchestrated microservice networks and DIME (distributed intelligent managed element) networks discussed in the literature. Both use hierarchical named micro-service networks (and subnetworks) to provision, monitor and control information processing structures and their resources while addressing non-deterministic fluctuations in the demand or availability of needed computing resources. In the next section, we describe an implementation of hierarchical named microservice network and its management which reduces current complexity in the datacenter.

New Edge Computing Cloud with Named IPE Network Provisioning, Monitoring, and Control



Fugure 5: A New Edge Cloud with Named Micro-Service Network

Figure 5 shows an implementation of a hierarchical named service network (consisting of various browsers accessing services delivered by an Apache web server, an application or business logic server and a database) provisioning, monitoring and connection management to address fluctuations in both functional and non-functional behavior at run-time without disrupting user experience.

The service control appliance is a platform from Platina Inc., which provides a microservices management bringing together multiple innovations: (i) Aggressive de-layering of networking software with a focus on containerized L3 protocol stacks and traffic slicing that can be deployed and managed without LAN L2 operational complexity; (ii) Elastic compute and flash storage providing data access at in-memory speed reducing the latency of computation by order of magnitude, and (iii) Zero-touch service orchestration using Kubernetes and a service layer control overlay architecture. The new architecture implemented using industry standard, off-the-shelf CPU and networking chips provides required application availability, performance, and security at the edge or the core with a highly scalable commercially off-the-shelf COTS) server footprints and offers compelling operational simplicity with a unified compute, network and storage stack management.

The hierarchical named micro-service network is defined consisting of a web server, application server and a database and a replica with various oracles defining their availability, performance, and security behaviors. Based on this "blueprint" a resource network blueprint is derived which defines a hierarchical named resource network configuration. The configuration oracles with the knowledge of available infrastructure, configures the

server, network, storage, and microservice management infrastructure and the microservice network. A set of monitoring oracles continuously watch the vital signs of both the micro-service network and the infrastructure that provides the resources and address any deviations from equilibrium caused by fluctuations either in the demand or availability of resources. The result is a highly available micro-service network that provides auto-failover, auto-scaling, and mobility to provide resilient information processing networks.

7. Comparison with the Current State of The Art

Current IT state of the art is becoming costly and complex with layers of evolutionary technologies accumulating without shedding the past. Figure 6 shows the multiple overlay technology stacks that have accumulated over time from their memory-starved, low-bandwidth computing origins, Compute, network and storage stacks have evolved over the years as silos with redundant and often competing functionality without an architecture that optimizes end-to-end service transaction availability, security, and efficiency. In a datacenter with high bandwidth fiber connectivity, the legacy OSI layers become inefficient at best. Addition of Virtual Machine networking complexity makes multi-cloud distributed resource utilization ineffective. The service orchestration and container technology overlay makes Virtual Machine networking redundant.

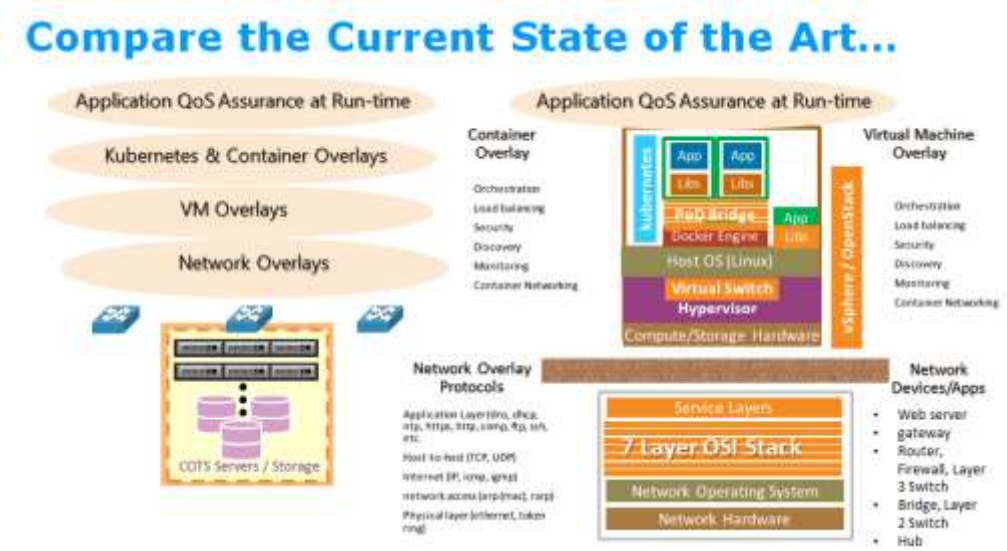


Figure 6: Current state of the art IT with layers of technology stacks (see the video https://www.youtube.com/watch?v=nmdVXVXQH8Y)

The new architecture discussed here learns from the past and provides a unified framework that simplifies computing, networking and storage layers to connect service components on any execution venue and provides visibility and control to reconfigure their connections based on business policies and priorities. It decouples application design and development process from application deployment and its QoS assurance processes. Figure 7 shows the new architecture that simplifies the technology stacks using a control appliance with hierarchical named micro-service connection management coupled with independent infrastructure connection management. The decoupling of infrastructure and service connection management with a control overlay provides the resiliency, efficiency, and scaling of distributed computing structures in the face of fluctuations both in the demand and the availability of the resources for assuring the end-to-end quality of service.

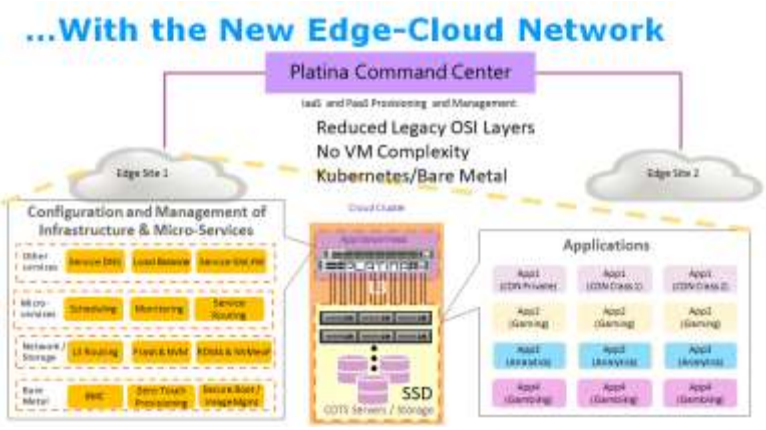


Figure **7**: Simplified and integrated compute, network and storage technology stack

8. Conclusion

Function, structure, and fluctuations play an important role in the evolution of information processing. Just as in physics, we have reversible and irreversible processes, we have deterministic and non-deterministic processes in information processing. Advanced information processing units and their network structure delimit computations while their evolution depends on physical computing devices, which provide the required resources. Fluctuations in the demand for and the availability of these resources cause deviations resulting in bringing non-determinism to their evolution. These deviations manifest themselves macroscopically as "speed of computation," or "response time" in their interaction between the components of the structure.

The three-tier Oracle structure overlay for provision, monitoring and control the evolution of the information processing structures provides a powerful way to manage the non-deterministic behaviors of the evolution of information processing structure. In this paper we have presented both the theory and implementation of such oracles aimed at improving the scaling, resiliency, and efficiency of these information processing structures.

There are also theoretical problems related to cloud automata. For instance, it is an open research problem how to classify languages defined by cloud automata.

References

1. Armstrong, S., Sandberg, A. and Bostrom, N. (2012) Thinking Inside the Box: Controlling and Using an Oracle AI, Minds and Machines, v. 22, No. 4, pp. 299-324

2. Burgin, M. (2005) Super-recursive Algorithms, New York: Springer.

3. Burgin, M. Grammars with Prohibition and Human-Computer Interaction, in "Proceedings of the Business and Industry Simulation Symposium," Society for Modeling and Simulation International, San Diego, California, 2005a, pp. 143-147

4. Burgin, M. Interactive Hypercomputation, in Proceedings of the 2007 International Conference on Foundations of Computer Science (FCS'07), CSREA Press, Las Vegas, Nevada, USA, 2007, pp. 328-333

5. Burgin, M. (2011) Theory of Named Sets, New York: Nova Science.

6. Burgin, M. (2014). Periodic Turing Machines, Journal of Computer Technology & Applications, v. 5, No 3, pp. 6 – 18

7. Burgin, M. (2015) Inductive Cellular Automata, International Journal of Data Structures and Algorithms, v. 1, No 1, pp. 1-9

8. Burgin, M. (2016) On the power of oracles in the context of hierarchical intelligence, Journal of Artificial Intelligence Research & Advances, v. 3, No. 2, pp. 6 - 17

9. Burgin, M. (2017) Inaccessible Information and the Mathematical Theory of Oracles, in Information Studies and the Quest for Transdisciplinarity: Unity through Diversity, World Scientific, New York/London/Singapore, pp. 59 - 114

10. Burgin, M. (2017). Swarm Superintelligence and Actor Systems, International Journal of Swarm Intelligence and Evolutionary Computation, v. 6, No. 3, open access.

(https://www.omicsonline.org/open-access/swarm-superintelligence-and-actor-systems-2090-4908-1000167-97430.html?view=mobile)

11. Burgin, M. and Eberbach, E. (2009) On foundations of evolutionary computation: an evolutionary automata approach, in Hongwei Mo (Ed.), Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies, IGI Global, Hershey, Pennsylvania, pp. 342-360

12. Burgin, M. and Eberbach, E. (2012) Evolutionary Automata: Expressiveness and Convergence of Evolutionary Computation, Computer Journal, vol. 55, no. 9, 1023-1029 (doi: dx.doi.org/10.1093/comjnl/bxr099).

13. Burgin, M., Liu, D., and Karplus, W. (2001) The Problem of Time Scales in Computer Visualization, in "Computational Science," Lecture Notes in Computer Science, v. 2074, part II, pp.728-737

14. Burgin, M. and Mikkilineni, R. (2018) Cloud computing based on agent technology, super-recursive algorithms, and DNA, Int. J. Grid and Utility Computing, v. 9, No. 2, pp.193–204

15. Burgin, M. and Smith, M.L. (2010) A Theoretical Model for Grid, Cluster and Internet Computing, in Selected Topics in Communication Networks and Distributed Systems, World Scientific, New York/London/Singapore, pp. 485-535

16. Case, J. and Jain, S. Rice and Rice-Shapiro (2011) Theorems for Transfinite Correction Grammars, Mathematical Logic Quarterly, 28 October 1-13

17. Case, J., and Royer, J. Program Size Complexity of Correction Grammars in the Ershov Hierarchy. In (A. Beckmann, L. Bienvenu, and N. Jonoska, Eds) (2016) Pursuit of the Universal - Twelfth Conference of Computability in Europe (CiE 2016), Proceedings}, Lecture Notes in Computer Science}, Volume~7921, pp.~240--250, Springer, Heidelberg.

18. Copeland, B.J. (2002) Accelerating Turing Machines. Minds and Machines 12, pp. 281–301

19. Curnow, T. (2004) The Oracles of the Ancient World: A Comprehensive Guide, Duckworth, London,

20. Eberbach, E., (2005) Toward a Theory of Evolutionary Computation, BioSystems, vol.82, no.1, 2005, 1-19.

21. Eberbach, E., (2007) The $-Calculus Process Algebra for Problem Solving: A Paradigmatic Shift in Handling Hard Computational Problems, Theoretical Computer Science, vol.383, no.2-3, 200-243

22. Eberbach E. (2015) On Hypercomputation, Universal and Diagonalization Complete Problems, Fundamenta Informaticae, IOS Press, 139 (4), 2015, 329-346, DOI 10.3233/FI-2015-1237.

23. Eberbach E. (2017) State of the Art of Information Technology Computing Models for Autonomic Cloud Computing, Proc. MDPI 2017, 1(3), 190; doi:10.3390/IS4SI-2017-04028; IS4SI 2017 Summit: Digitalisation for a Sustainable Society, Gothenburg, Sweden, June 12-16, 2017.

24. Eberbach E., Goldin D. and Wegner P. (2004), Turing's Ideas and Models of Computation, in: (ed. Ch. Teuscher) Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, 2004, 159-194.

25. Farley J. (1998), Java Distributed Computing, O'Reilly, 1998.

26. Gödel K. (1931) Über formal unentscheidbare Sätze der Principia Mathematica und verwander Systeme, Monatschefte für Mathematik und Physik, 38:173-198, 1931.

27. Hamkins, J.D., and Lewis, A. (2000) Infinite time Turing machines, Journal of Symbolic Logic, v. 65, No. 3, pp. 567-604

28. Leeson, P.T. (2014) Oracles, Rationality and Society, v. 26(2) 141–169

29. Mikkilineni, R., Morana, G. and Burgin, M. Oracles in Software Networks: A New Scientific and Technological Approach to Designing Self-Managing Distributed Computing Processes, Proceedings of the 2015 European Conference on Software Architecture Workshops, Dubrovnik/Cavtat, Croatia, September 7-11, 2015, ACM, 2015, pp. 11:1-11:8

30. Milner, R., A Calculus of Communicating Systems, LNCS 94, Springer-Verlag, 1980

31. Rogers, H. Theory of Recursive Functions and Effective Computability, MIT Press, Cambridge Massachusetts, 1987

32. Siegelmann, H.T. (1999) Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhauser, Berlin

33. Tanenbaum, A. and Bos, H. (2015) Modern Operating Systems, Fourth Ed., Pearson

34. Tanenbaum, A. and Wetherall, D. (2011) Computer Networks, Fifth Ed., Pearson

35. Turing, A. M. (1939) Systems of logic defined by ordinals, Proc. Lond. Math. Soc., Ser. 2, v. 45, pp. 161-228

Conflicts of Interest

All authors of the journal article, namely, Mark Burgin, Eugene Eberbach, and Rao Mikkilineni, declare no conflict of interest.

Funding Statement

Research described in our article has been self-funded.