Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2012, Article ID 604018, 10 pages
doi:10.1155/2012/604018

Research Article

Service Virtualization Using a Non-von Neumann Parallel, Distributed, and Scalable Computing Model

Rao Mikkilineni,[1] Giovanni Morana,[2] Daniele Zito,[2] and Marco Di Sano[2]

[1] *Computer Science Department, Kawa Objects Inc., Los Altos, CA 95014, USA*
[2] *DIEEI, University of Catania, Catania, Italy*

Correspondence should be addressed to Rao Mikkilineni, rao@kawaobjects.com

Received 8 October 2011; Accepted 20 January 2012

Academic Editor: Yueh M. Huang

This paper describes a prototype implementing a high degree of transaction resilience in distributed software systems using a non-von Neumann computing model exploiting parallelism in computing nodes. The prototype incorporates fault, configuration, accounting, performance, and security (FCAPS) management using a signaling network overlay and allows the dynamic control of a set of distributed computing elements in a network. Each node is a computing entity endowed with self-management and signaling capabilities to collaborate with similar nodes in a network. The separation of parallel computing and management channels allows the end-to-end transaction management of computing tasks (provided by the autonomous distributed computing elements) to be implemented as network-level FCAPS management. While the new computing model is operating system agnostic, a Linux, Apache, MySQL, PHP/Perl/Python (LAMP) based services architecture is implemented in a prototype to demonstrate end-to-end transaction management with auto-scaling, self-repair, dynamic performance management and distributed transaction security assurance. The implementation is made possible by a non-von Neumann middleware library providing Linux process management through multi-threaded parallel execution of self-management and signaling abstractions. We did not use Hypervisors, Virtual machines, or layers of complex virtualization management systems in implementing this prototype.

1. Introduction

The advent of many-core severs with tens and even hundreds of computing cores with high bandwidth communication among them makes the current generation server, networking, and storage equipment and their management systems which have evolved from server-centric and bandwidth limited architectures completely unsuited to use in the next generation computing infrastructure efficiently. It is hard to imagine replicating current TCP/IP-based socket communication, "isolate and fix" diagnostic procedures, and the multiple operating systems (which do not have end-to-end visibility or control of business transactions that span across multiple cores, multiple chips, multiple servers, and multiple geographies) inside the next generation many-core servers without addressing their shortcomings. The many-core servers and processors constitute a network where each node itself is a subnetwork with different bandwidths and protocols (socket-based low bandwidth communication between servers, InfiniBand, or PCI Express bus-based communication across processors in the same server and shared memory-based low latency communication across the cores inside the processor). Figure 1 shows the many-core server network supporting multiple bandwidths.

In order to cope with the scaling issues and utilize the hierarchical many-core network of networks effectively, next generation service architecture has to emulate the architectural resiliency of cellular organisms that tolerate faults and implement command and control structures which enable execution of self-configuring, self-monitoring, self-protecting, self-healing, and self-optimizing (in short self-$*$) business processes.

Figure 2 shows the evolution of current computing infrastructure with respect to three parameters—system resiliency, efficiency, and scaling.

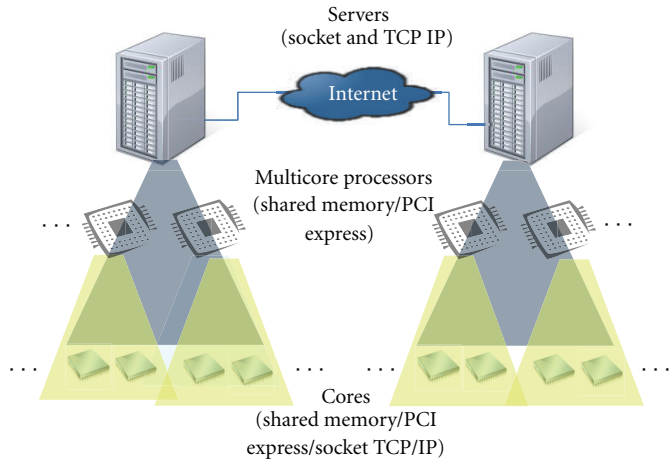

Figure 1: Many-core servers containing multiple processors connected via the Internet. Communication between cores must be context-sensitive to exploit the performance offered by the many-core servers with each core supporting multiple parallel threads of computation.

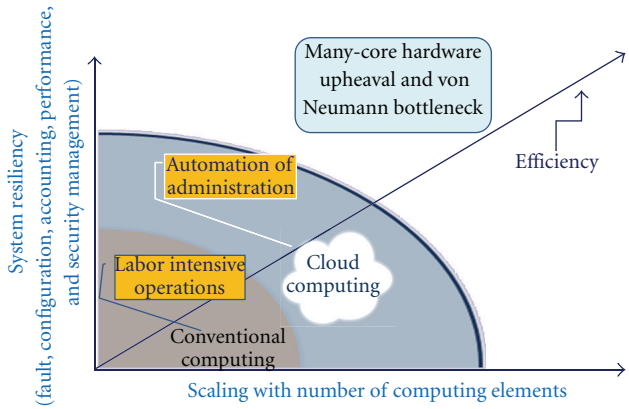

Figure 2: The resiliency, efficiency, and scaling of information technology infrastructure. Grid and cloud computing management brings automation of physical and virtual resources management.

The resiliency is measured with respect to a service's tolerance to faults, fluctuations in contention for resources, performance fluctuations, security threats, and changing business priorities. Efficiency is measured in terms of total cost of ownership and return on investment. Scaling addresses end-to-end resource provisioning and management with respect to increasing number of computing elements required to meet service needs.

As information technologies evolved from server-centric computing to Internet/Intranet-based managed grid and cloud computing technologies, the resiliency, efficiency, and scaling are improved by automating many of the labor-intensive and knowledge-sensitive resource management tasks to meet the changing application/service needs.

Unfortunately, extending current state of the art to develop applications that harness the full power of many-core systems is difficult and requires software developers to transition from writing serial programs to writing parallel programs [1]. Parallel applications share data, and current thread technologies used to modifying them behave in unpredictable ways resulting in a complex web of debugging and optimizing strategies. To obtain the required behavior, the access to shared data should be coordinated between all cores using proper synchronization techniques and applying suitable policies and patterns based on overall system goals, relative priorities of various tasks, and latency constraints. Current computing techniques, operating systems that have to effectively supply multicore resource management, and high-level application programming that supports distributed transaction management have to be reexamined to leverage parallelism of processing cores.

In addition, current approaches to resource management, albeit with automation, are not sensitive to the distributed nature of transactions, and contention resolution of shared distributed resources, at best, is complex involving many layers of management systems. As von Neumann [2] pointed out, current design philosophy that "errors will become as conspicuous as possible, and intervention and correction follow immediately" does not allow scaling of services management with increasing number of computing elements involved in the transaction. Comparing the computing machines and living organisms, he points out that the computing machines are not as fault tolerant as the living organisms. He goes on to say "it's very likely that on the basis of philosophy that every error has to be caught, explained, and corrected, a system of the complexity of the living organism would not run for a millisecond." More recent efforts, in a similar vein, are looking at resiliency borrowing from biological principles [3] to design future Internet architecture.

In this paper, we will revisit the design of distributed systems with a new non-von Neumann computing model (called Distributed Intelligent Managed Element (DIME) (DIME, Cloud-DNA, and Dime Network Architecture are trade marks of Kawa Objects Inc.) network computing model) [4–8] that integrates computational workflows with a parallel implementation of management workflows to provide dynamic real-time FCAPS management of distributed services and end-to-end service transaction management.

The DIME network architecture provides a new direction to harness the power of many core servers with the architectural resiliency of cellular organisms and a high degree of scaling, and efficiency. It also eliminates many of the shortcomings of current solutions being proposed for solving the scalability issue in these systems, that is, the use of SSI [9] or the introduction of multiple instances of the OS in a single enclosure with socket communication among them instead of high-speed shared memory or PCI Express (e.g., [10]), which are inefficient because they increase the management complexity. A review of other operating system approaches has been presented elsewhere by one of the authors [7].

The focus of this paper is the demonstration of resiliency, scaling and efficiency of the new computing model in a conventional Linux operating system environment by injecting a non-von Neumann middleware to introduce self-management (FCAPS) and network-aware signaling abstractions (alerting, addressing, supervision, and mediation). We have chosen a popular LAMP-based web-services infrastructure to demonstrate end-to-end transaction management based on business priorities, workload fluctuations, component failures, and latency constraints. We demonstrate autoscaling, self-repair, performance management, end-to-end transaction security assurance, and live-migration of services without the need for a hypervisor or other server virtualization technologies or any new standards. This implies that DIME networks offer computing, networking and storage on demand, without the need for an additional "Hypervisor Layer," simplifying the parallelization schema and, at the same time, speeding up the communication amongst various components.

In addition, using the features provided by FCAPS management, we have found that the DIME network infrastructure simplifies the managed services development. The developer, in fact, has to focus only on the algorithmic part (i.e., the computing workflow) of the service, leaving the management issues (such as fault, configuration, accounting, performance, and security management) to the DIME infrastructure. Above all, this approach offers to leverage current OSs by converting each process into a DIME while also allowing the development of a native distributed, parallel, and scalable operating system called parallax as discussed elsewhere [6, 7]. The parallax OS is implemented in assembler language (with C and C++ language support) and converts each core into a DIME (inside the many-core server), whereas the approach described in this paper encapsulates a Linux process into a DIME.

The paper is organized as follows. In Section 2, we briefly review the new computing model and the DIME network architecture (DNA) that allows programming self-$*$ services creation, delivery, and assurance frameworks. In Section 3, we use DNA to implement a Linux, Apache, MySQL, and PHP/PERL/PYTHON (LAMP) based services architecture to demonstrate end-to-end transaction management with autoscaling, self-repair, dynamic performance management, and distributed transaction security assurance. The implementation demonstrates a true decoupling of services and their management from the hardware infrastructure and its management and shows the resiliency, scaling, and efficiency that go beyond current state of the art. In Section 4, we present a discussion with some thoughts on future directions for continued research. In Section 5, we conclude the paper with some thoughts on DNA in biology and DNA in information technologies.

2. The DIME Network Architecture

The DIME computing model exploits the parallelism to implement a signaling network overlay over a network of von Neumann SPC computing nodes (cores in a multicore server using a new operating system [6, 7] or Linux Processes in conventional computing [5]). Multiple threads available in each core or a Linux process implementation are exploited to implement a self-managed computing element called the DIME. Each DIME presents a computing element that can execute a managed computing process with fault, configuration, accounting, performance, and security management. Figure 3 shows a comparison between the von Neumann SPC computing model and the DIME computing model.

The parallelism of service execution and service control allows real-time monitoring of service behavior and management based on policies and constraints specified by the regulators both at the node level and at the network level. The DIME network architecture thus allows the description and management of the service to be separated from the execution of the service (using a computing thread called Managed Intelligent Computing Element, MICE). The signaling control network allows parallel management of the service workflow. In step 1, the service regulator instantiates the DIME and provisions the MICE based on service specification. In step 2, the MICE is loaded, executed, and managed by the service regulation policies. At any time, the MICE can be controlled through its FCAPS management mechanism by the service regulator.

There are three key features in this model that differentiate it from all other models.

(1) The self-management features of each SPC node with FCAPS management using parallel threads allow autonomy in controlling local resources and provide services based on local policies. Each node keeps its state information and history of its transactions. (The concept of state awareness and history of computational transactions provided at the node level and at the network level introduces a non-Markovian element into the DIME computing model which allows for diagnosis-after-the-fact and facilitates system level predictive corrections based on history.) The DIME node provides managed computing services, using the MICE to other DIMEs based on local and global policies.

(2) The network aware signaling abstractions allow a group of DIMEs to be programmed to manage themselves with subnetwork/network level FCAPS management based on group policies and execute a service workflow as a managed directed acyclic graph (DAG).

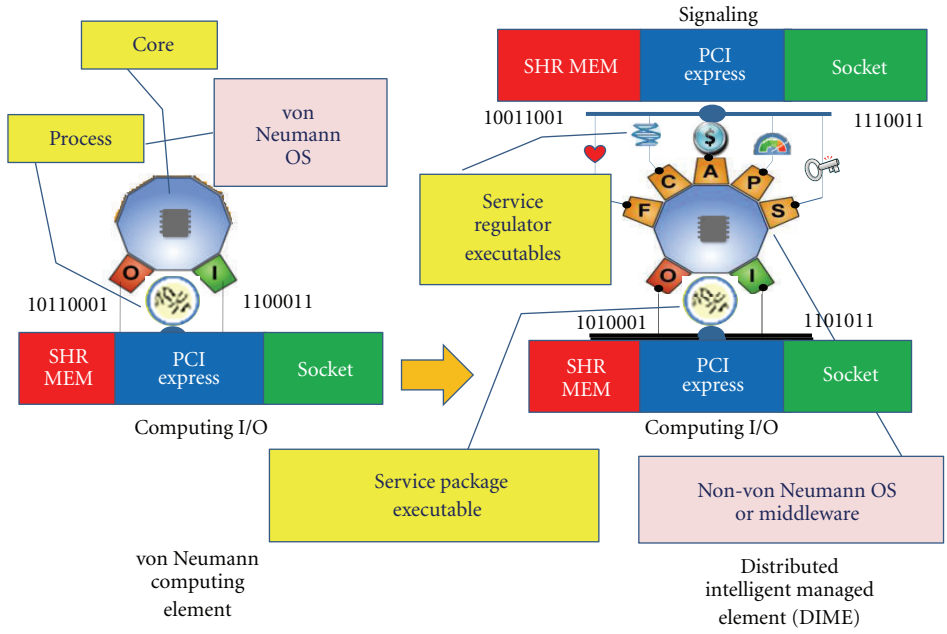

Figure 3: von Neumann computing model and DIME computing model. For a description of the DIME network architecture and the genetic transactions, please see the video http://youtu.be/Ft_W4yBvrVg.

(3) Run-time profile-based FCAPS management (at the group level and at the node level) allows a composition scheme by redirecting the MICE I/O to provide recombination and reconfiguration of service workflows dynamically at run time.

The MICE provides the logical type that performs everything that is feasible within that logical type (a Turing machine), and the DIME FCAPS management provides a higher logical type (management of the Turing machine) which describes and controls what is feasible in the MICE [11]. These features provide the powerful genetic transactions, namely, replication, repair, recombination, and reconfiguration that have proven to be essential for the resiliency of cellular organisms [11].

We have applied the DIME computing model to convert a Linux process into a DIME with self-management (FCAPS management of the Linux process) and signaling awareness to create a managed DIME network implementing a managed workflow. The details of injecting the DNA in Linux OS using a non-von Neumann middleware are described elsewhere [5], and it requires no special accommodations from the operating system. In fact, the non-von Neumann middleware uses standard OS services so that it can be easily ported to other operating systems offering multithreading capabilities.

The separation of the service design by splitting the service regulator component and the service execution package as shown in Figure 4 allows the description and control of the service to be separated and made available at run time providing the resiliency in services management. This separation also makes possible the genetic transactions of replication, repair, recombination, and reconfiguration that are the distinguishing characteristics of cellular organisms. Each DIME executes a set of tasks arranged in a DAG. Each node of this DAG contains both the task executables (which itself could be another DAG) and the profile DAG as a tuple (task (SP), profile (SR)): in this way, it is possible not only to specify what a DIME has to do or execute but also its management (how this has to be done and under what constraints). These constraints allow the control of FCAPS management both at the node level and the subnetwork level. In essence, at each level in the DAG, the tuple gives the blueprint for both management and execution of the downstream graph. Under these considerations, it is easy to understand the power of the proposed solution in designing self-configuring, self-monitoring, self-protecting, self-healing, and self-optimizing distributed service networks.

The DIME network architecture takes its cues from parallels in cellular biology where regulatory genes control the actions of other genes which allow them the ability to turn on or turn off specific behaviors. As affirmed by Stanier and Moore [12], "in essence, genes work in hierarchies, with regulatory genes controlling the expression of "downstream genes" and with the elements of "cross-talk" between the regulatory genes themselves." The same parallel, furthermore, exists between the task profile and the concept of gene expression. Gene expression is the process by which information from a gene is used in the synthesis of a functional gene product.

In the next section, we describe the use of a DIME network (each DIME encapsulating a Linux process with FCAPS management) to implement LAMP-based web services architecture to demonstrate end-to-end transaction management with autoscaling, self-repair, dynamic performance management, and distributed transaction security assurance.

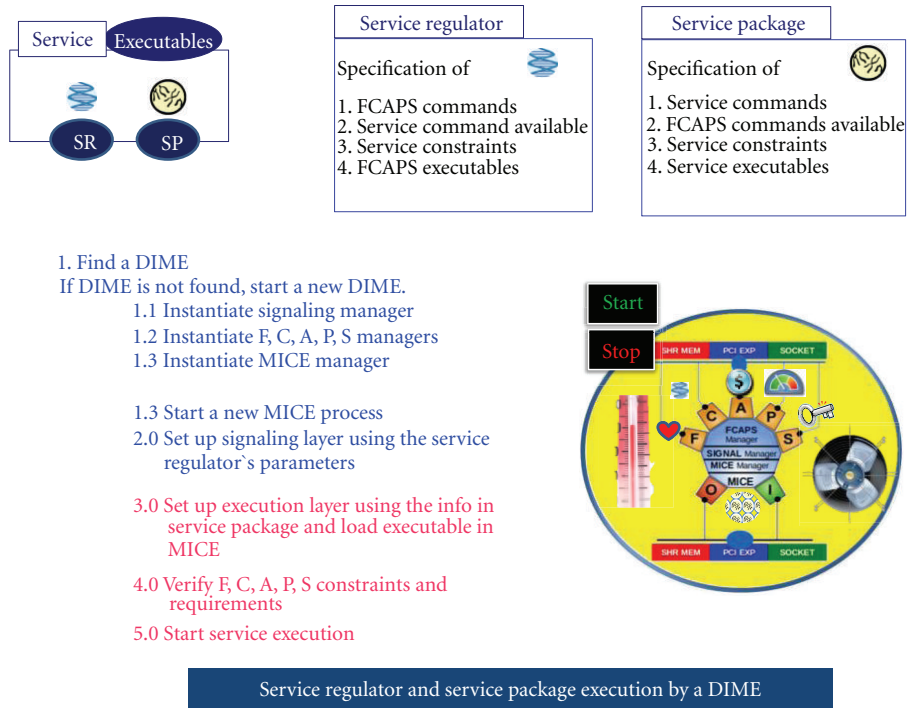

FIGURE 4: The anatomy of a DIME and the separation of service regulation and service execution workflows.

3. LAMP Web Services Using DNA

Figure 5 shows a DIME network prototype of Linux processes implementing a web services workflow using an Apache web server, MySQL database, and Python/PHP/perl services.

The Apache web server and the DNS server are executed by encapsulating each of them in a single DIME. The DIME network architecture enables the application or service running in the MICE under the control of the FCAPS manager. The fault and performance information from each DIME is collected through the signaling channel and is utilized by the end-to-end DIME network management infrastructure. The policies are implemented by the DIME network configuration manager (including the Supervisor, Fault Manager, Performance Manager, Accounting Manager, and the Security Manager functions designed to execute policy implementation workflows.)

The workflow is as follows.

(1) A graphical user interface allows configuring the DIME network by assigning

  (a) the primary role to an instance of DIME implementing the Apache webserver,
  (b) the secondary instances of DIMEs with Apache for backup in executing the self-repair policy,
  (c) the DNS server with primary and secondary DIME addresses that are configured automatically.

(2) When the primary is killed, the fault manager in the DIME configuration detects the lack of heartbeat from the DIME containing the primary and immediately sends a signal to the DNS to replace the primary Apache address with an available secondary. In Figure 5, the self-repair scenario replaces the primary which is killed with the secondary from the DNS server list.

(3) The local performance manager in the DIME implementing the Primary Apache receives notifications about the response time of the transactions going through the Apache web server.

(4) When the performance exceeds a threshold, the signaling channel is used to notify, via network Performance Manager, the Supervisor that instantiates an additional Apache (a consistent copy of the first Apache.) in a new DIME. Figures 6(a), 6(b), and 6(c) show the trend of response time using more than one apache server. You can see as response time decreases when a new DIME is added.

(5) The Supervisor, based on business priorities, workload management policies, and latency constraints, coordinates with Configuration and Security Managers of the network to instantiate the new instance of Apache with appropriate configuration.

(6) The network Configuration Manager instantiates the new service and adjusts the work-loads modifying the DNS rules as required.

Let us analyze the results obtained from the graphs in Figures 6(a), 6(b), and 6(c). They were generated by using Apache JMeter, a graphical server performance testing tool.

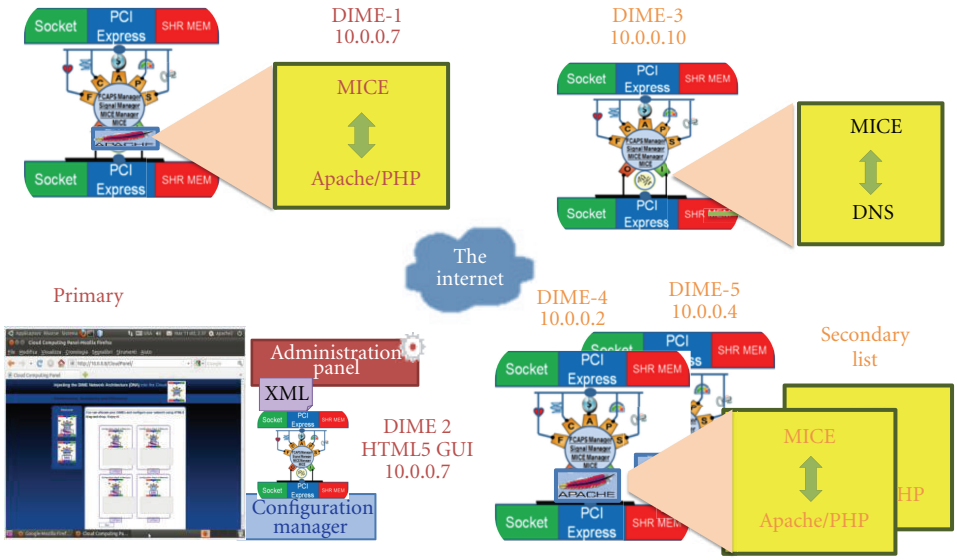

Figure 5: DIME network implementing web services using LAMP services with FCAPS management at both the node level and at the network level. When the primary Apache fails, it is replaced by the secondary by the DIME network.

You can see how response time decreases when a new DIME is added as Primary, to achieve load balancing. Figure 6(a) refers to a scenario with four Apache servers, each of which is encapsulated in a DIME.

The number of simulated users make requests at the same time is 500 and the ramp-up period is 600 seconds. This means that the simulator activates each thread (user) every 600/500 seconds.

Figure 6(b) shows the trend of response time using three DIMEs. In this scenario, we have configured the performance tool to generate 1000 requests with a ramp-up period of 1000 seconds; this means that each request will be performed every second. This is why the response time trend in Figure 6(b) grows more rapidly than the first one: in second scenario, there are many more request and the activation rate of each request is larger. However, the trend is the same in both figures which means that the load balancing is working properly.

Finally, Figure 6(c) shows how we can handle response time, adding two DIMEs instead of one DIME at a time. The graph in Figure 6(c) refers to the same scenario of Figure 6(a), when the number of users is 500 and the ramp-up period is 600 seconds. In this way, load balancer can add two DIMEs (or N DIMEs) according to the latency incurred on the first Apache server. So, as shown in Figure 6(c), we can greatly improve performance of the services infrastructure by monitoring and controlling the response time using the signaling-enabled DIME network architecture.

The policies are implemented at various levels; at the node level and at the subnetwork or network level. In addition to domain specific service management workflow, each DIME implements its own local FCAPS management independent of what MICE processes are doing. This allows programming DNA level DIME instantiation, and its life-cycle management to assure 100% infrastructure availability, performance, and security service levels. While a simple end-to-end transaction security check is performed with a login and password scheme that allows service network management, a more elaborate authentication, authorization, and accounting scheme is discussed in another paper by Tusa et al. [13].

We believe that the DIME network architecture represents a major departure from the current cloud approaches [14, 15]. In conventional computing, the physical server serves as a managed computing element with node level resource management performed by the operating system. In cloud computing, the virtual server serves as a managed computing element also with operating system managing the local resources. In the DIME computing model, a process in an operating system serves as managed computing element. The non-von Neumann approach radically improves the resource exploitation using parallelism and coordinated management both at the node and network level even within the cloud environments, hiding the complexity of the management of FCAPS issues both from the developers and users of cloud services. Table 1 summarizes some of the attributes that distinguish DIME network architecture from conventional managed server computing and managed cloud computing.

4. Discussion and Future Direction

The DIME computing model attempts to fill the need to break the von-Neumann bottleneck and leverage the hardware upheaval to improve the resiliency, efficiency, and scaling of future services infrastructure as shown in Figure 7.

While the paper discusses the use of DNA injection into Linux OS, we see no technical obstacle to do the same in other operating systems. The key requirement is

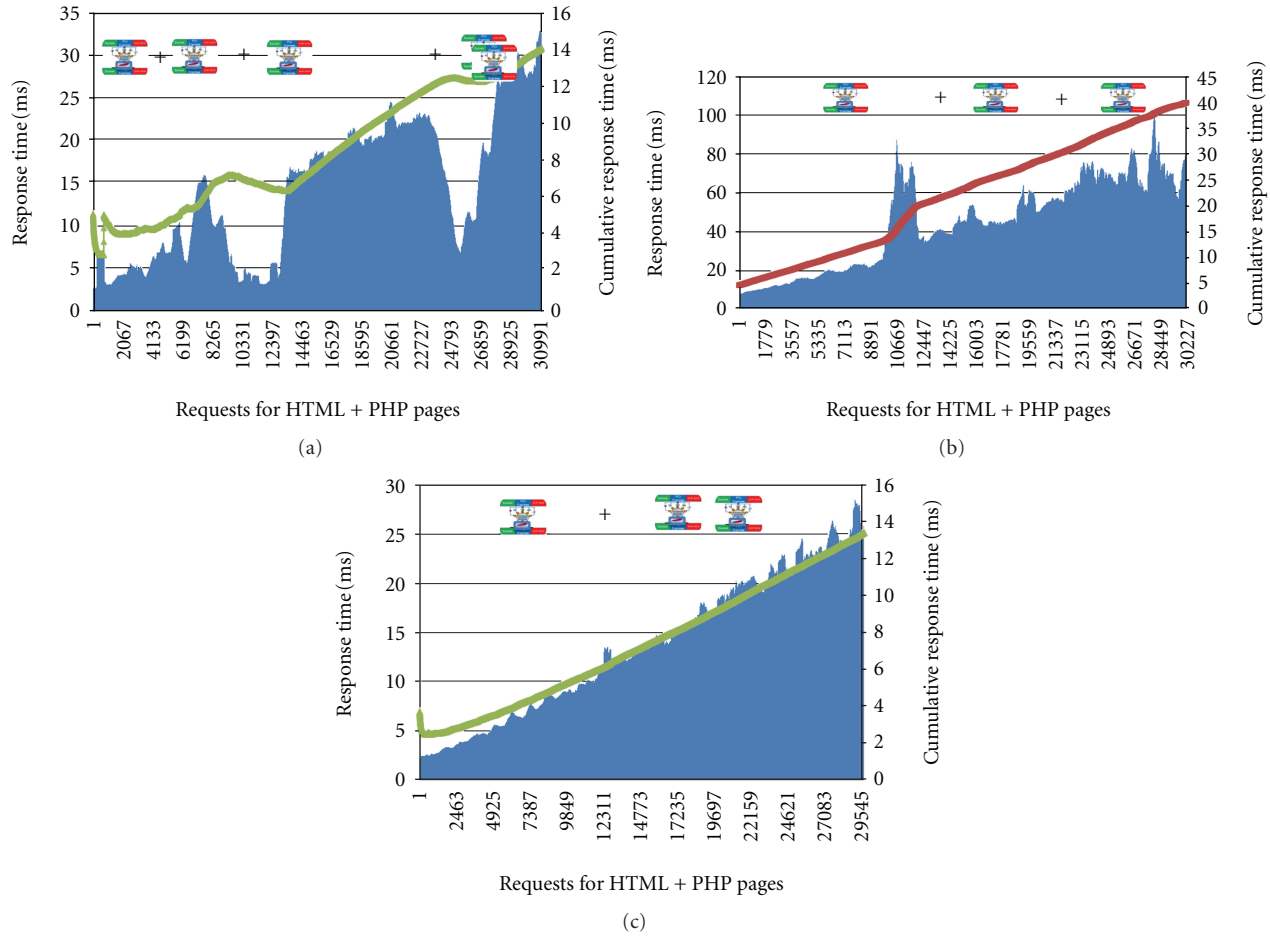

Figure 6: (a) DIME network implementing web services using LAMP services with performance management at both the node level and at the network level. When the primary Apache response time degrades, a new DIME is added using the signaling network that manages the DIME computing network. The role of increasing DIMEs controlling Apache is illustrated in controlling the response time as number of transactions increase. (b) Response time using three DIMEs, involved to manage a greater quantity of requests than the first scenario in Figure 6(a). (c) Response time trend adding two DIME using signaling to improve performance without Hypervisors or other Virtual Machine Management.

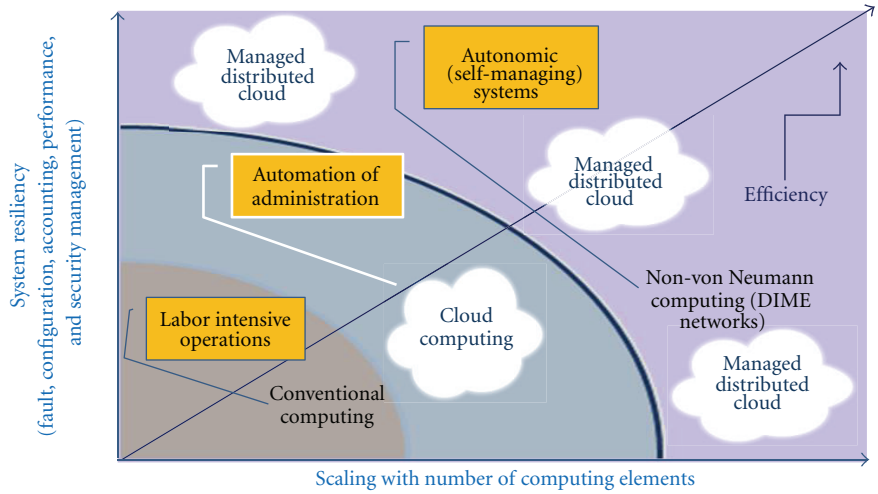

Figure 7: The resiliency, efficiency, and scaling with non-von Neumann Middleware with Linux operating system showing the transition from a physical server to virtual server to a virtual service container as an atomic managed computational unit.

Table 1: Comparison between conventional, cloud and DIME network computing models.

	Conventional computing	Cloud computing	DIME computing	
			Linux	Parallax
Managed computing entity	Physical server	Virtual server	A process	A Core
Computing network	IP/shared memory/PCI express	IP/shared memory/PCI express	IP/shared memory/PCI express	IP/shared memory/PCI express
Management network	Uses same computing network	Uses same computing network	Parallel signaling network overlay	Parallel signaling network overlay
Computing model	von Neumann Turing machine	von Neumann Turing machine	Managed Turing machine	Managed Turing machine
Server hardware management (FCAPS)	Labor intensive	Automated	Service management decoupled from server management	Service management decoupled from server management
Network hardware management (FCAPS)	labor intensive	Automated	Service management decoupled from network management	Service management decoupled from network management
Storage hardware management (FCAPS)	Labor intensive	Automated	Service management decoupled from storage management	Service management decoupled from storage management
Operating system	Server-centric	Virtual server-centric	Network-centric Process management in nude-centric Linux OS	Network-centric core management transcending physical server boundaries
Service management (FCAPS)	Labor intensive	Automated workflows	Autonomic policy based transaction FCAPS management	Autonomic policy based transaction FCAPS management
Management latency	T(management) >> T(service transaction)	T(management) > T(service transaction)	T(management) = T(service transaction)	T(management) = T(service Transaction)

the multithreading capability to implement parallel management workflow to control the Turing machine implemented as a process in the conventional OS. It is also proven [7] that the DNA can be injected at the core with a native OS written from scratch that scales and provides the resiliency. It is also proven that we can leverage current service-oriented architecture, development environments, and workflow implementations using a network of Turing machines by migrating them to a managed network of Turing machines using DNA. Using the DIME network, we have provided multitenancy with FCAPS management at a process (implementing a service component) without the need for current generation Hypervisors.

It is interesting to note that the services management decoupling from the infrastructure hardware management using DNA does not require any new standards or approaches except exploiting parallelism to separate the management workflow and computing workflows at the core or at the process level using self-management, signaling, and network management abstractions.

In designing this new class of distributed systems, it behooves us to go back and seriously study von Neumann's views on the subject [2]. Talking of cellular organisms and how they operate across errors, he points out that "the system is sufficiently flexible and well organized that as soon as an error shows up in any part of it, the system automatically senses whether this error matters or not. If it does not matter, the system continues to operate without paying any attention to it. If the error seems to the system to be important, the system blocks that region out, by-passes it, and proceeds along other channels. The system then analyzes the region separately at leisure and corrects what goes on there, and if correction is impossible the system blocks the region off and by-passes it forever. The duration of operability of the automaton is determined by the time it takes until so many incurable errors have occurred, so many alterations and by-passes have been made, that finally the operability is really impaired. This is a completely different philosophy which proclaims that the end of the world is at hand as soon as the first error occurred."

In order to benefit from the approach adopted by the cellular organisms, current services management approaches must implement two features at the core computing element (a von Neumann computing node). First, they must implement self-management based on local history and local policy requirements. Second, it must provide a parallel signaling channel for a network of self-managed computing elements to communicate and collaborate to implement global policies. This defines a non-von Neumann computing model that implements a managed network of self-managed Turing computing nodes.

While current cloud and grid management systems implement services management by monitoring various application or service characteristics with the use of various resource management systems, the applications or services that use local operating systems in each node still have their

resource and service management serialized using the von-Neumann SPC computing model. The DNA addresses this by implementing the separation at the computing node by exploiting parallelism.

Discussing the work of Francois Jacob and Jacques Monod on genetic switches and gene signaling, Mitchell Waldrop [16] points out that "DNA residing in a cell's nucleus was not just a blueprint for the cell—a catalog of how to make this protein or that protein. DNA was actually the foreman in charge of construction. In effect, DNA was a kind of molecular-scale computer that directed how the cell was to build itself and repair itself and interact with the outside world."

We believe that the DIME network architecture enabling the execution of a workflow as a managed directed acyclic graph provides at least a mechanism for a blueprint for enterprise business process description, replication, execution, and control using a lengthy recursive sequence of nested programs which unfold in the von Neumann computing world using a non-von Neumann computing model.

Future directions of this research are self-evident. First, the non-von Neumann middleware can be exploited to improve resiliency, efficiency, and scaling of current grid and cloud services by decoupling services management from the infrastructure hardware management. This approach allows implementing reliable and resilient services using unreliable hardware just as the cellular organisms do.

Few immediate applications present themselves.

(1) Dynamic many-core cluster communication management across multiple Linux images to choose the type of communication based on available resources and service requirements.

(2) Implement WAMP (Windows, Apache, MySQL, and PHP/Perl/Python) services architecture using DIME network architecture.

(3) Application aware resource allocation (dial-up and dial-down) at run time.

(4) Resilient services oriented architecture (RSOA) implementation through the migration of the services microcontainer [17] into a DIME.

(5) High-performance computing (HPC) resource scheduling and management.

Secondly, the hardware infrastructure itself can be redesigned (exploiting the many-core architecture) to become signaling aware and respond to application requests at run time. Future storage and networking hardware thus can be simplified with hardware-assisted DIME architecture to eliminate current layers of management software and special purpose ASIC implementations. They can be designed to dial-up and dial-down raw resources (number of cores, memory, bandwidth, throughput, storage capacity, etc.) based on application requests at run time.

Finally, DNA can be implemented by chip vendors in hardware to provide self-management and signaling awareness exploiting parallelism at the core. This allows uniformity in hardware device drivers with self-management and signaling awareness.

On the theoretical side, it is worth examining the intriguing remarks of von Neumann about Gödel's theorem and its implications on the descriptions of complexity [18]. In his Hixon Symposium talk, von Neumann remarks "it is a theorem of Gödel that the next logical step, the description of an object, is one class type higher than the object and is therefore asymptotically infinitely longer to describe." He goes on to say "it is one order of magnitude harder to tell what an object can do than to produce the object." The DNA attempts to describe and assure what an object does; in this case, the object happens to be a von Neumann computing node. In the light of the new resiliency of DNA (e.g., the DIME can be instantiated and managed to provide 100% availability and recoverability), it is worthwhile to revisit the classic distributed computing issues such as the dining and drinking philosopher problems and the CAP theorem.

5. Conclusion

In conclusion, we observe that the evolution of living organisms has taught us that the difference between survival and extinction is the information processing ability of the organism to

(1) discover and encapsulate the sequences of stable patterns that have lower entropy, which allow harmony with the environment providing the necessary resources for its survival,

(2) replicate the sequences so that the information (in the form of best practices) can propagate from the survived to the successor,

(3) execute with precision the sequences to reproduce itself,

(4) monitor itself and its surroundings in real-time,

(5) utilize the genetic transactions of repair, recombination and rearrangement to sustain existing patterns that are useful.

That the computing models of living organisms utilize sophisticated methods of information processing was recognized by von Neumann who proposed both the SPC computing model and the self-replicating cellular automata. Later Chris Langton created computer programs that demonstrated self-organization and discovery of patterns using evolutionary rules which led to the field of artificial life and theories of complexity.

In this paper, we focus on another aspect that we learn from the genes in living organisms that deal with precise replication and execution of encapsulated DNA sequences. We describe a computing model, recently proposed, extending the stored program control computing model to create self-configuring, self-monitoring, self-healing, self-protecting, and self-optimizing (self-managing or self-$*$) distributed software systems. As opposed to self-organizing systems that evolve based on probabilistic considerations, this approach focuses on the encapsulation, replication, and execution of distributed and managed tasks that are specified precisely.

According to biologist Carroll [19], "cells communicate with one another by sending signals in the form of proteins that are exported and travel away from their source. Those proteins then bind to receptors on other cells, where they trigger a cascade of events, including changes in cell shape, migration, the beginning or cessation of cell multiplication, and the activation or repression of genes." Signaling also has proven to be a critical element in telecommunications networks and human network communications.

Signaling, a new element introduced in the DIME network computing model, is as important as it is in cellular organisms to provide resilience. In this paper, we demonstrate its use in building resilient LAMP services using conventional computing infrastructure. We have demonstrated that the signaling enables dynamic service management while decoupling the service management from the underlying hardware infrastructure management. We have also demonstrated autoscaling, self-repair, and live migration of Linux processes without Hypervisors or the Virtual Machine Management required in current cloud implementations. This we hope will reduce the current management complexity in conventional and cloud computing realms that has evolved over last six decades from their server-centric and low-bandwidth origins. We also believe that the end-to-end transaction FCAPS management introduces a new level of telecom-grade trust [13] and brings the architectural resilience of cellular organisms to distributed computing exploiting the parallelism and performance offered by the new class of many-core servers.

Acknowledgments

The authors wish to acknowledge many valuable discussions with and great encouragement from Kumar Malavalli, and Albert Comparini from Kawa Objects Inc. Without their continued support, this work would not have been possible.

References

[1] D. Patterson, "The trouble with multi-core," *IEEE Spectrum*, vol. 47, no. 7, pp. 28–53, 2010.

[2] J. V. Neumann, "Theory of natural and artificial automata," in *Papers of John Von Neuman on Computers and Computer Theory*, W. Aspray and A. W. Burks, Eds., vol. 12 of *Charles Babbage Institute Reprint, Series for the History of Computing*, pp. 408–474, The MIT Press, Cambridge, Mass, USA, 1986.

[3] S. Balasubramaniam, K. Leibnitz, P. Lio, D. Botvich, and M. Murata, "Biological principles for future Internet architecture design," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 44–52, 2011.

[4] R. Mikkilineni, "Is the Network-centric Computing Paradigm for Multicore, the Next Big Thing?" Convergence of Distributed Clouds, Grids and Their Management, 2010, http://www.computingclouds.wordpress.com/.

[5] G. Morana and R. Mikkilineni, "Scaling and self-repair of Linux based services using a novel distributed computing model exploiting parallelism," in *Proceedings of the 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '11)*, pp. 98–103, June 2011.

[6] R. Mikkilineni and I. Seyler, "Parallax—a new operating system for scalable, distributed, and parallel computing," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW '11)*, pp. 976–983, 2011.

[7] R. Mikkilineni and I. Seyler, "Parallax—a new operating system prototype demonstrating service scaling and service self-repair in multi-core servers," in *Proceedings of the 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '11)*, pp. 104–109, 2011.

[8] R. Mikkilineni, *Designing a New Class of Distributed Systems*, Springer, New York, NY, USA, 2011.

[9] R. Buyya, T. Cortes, and H. Jin, "Single system image," *International Journal of High Performance Computing Applications*, vol. 15, no. 2, pp. 124–135, 2001.

[10] http://www.seamicro.com/.

[11] J. V. Neumann, "Theory of natural and artificial automata," in *Papers of John Von Neuman on Computers and Computer Theory*, W. Aspray and A. W. Burks, Eds., vol. 12 of *Charles Babbage Institute Reprint, Series for the History of Computing*, p. 454, The MIT Press, Cambridge, Mass, USA, 1986.

[12] P. Stanier and G. Moore, "Embryos, genes and birth defects," in *The Relationship Between Genotype and Phenotype: Some Basic Concepts*, P. Ferretti, A. Copp, C. Tickle, and G. Moore, Eds., p. 5, John Wiley & Sons, London, UK, 2nd edition, 2006.

[13] F. Tusa, A. Celesti, and R. Mikkilineni, "AAA in a cloud-based virtual DIME network architecture (DNA)," in *Proceedings of the 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '11)*, pp. 110–115, Paris, France, 2011.

[14] R. Buyya and R. Ranjan, "Special section: federated resource management in grid and cloud computing systems," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1189–1191, 2010.

[15] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[16] M. M. Waldrop, *Complexity: The Emerging Science at the Edge of Order and Chaos*, Simon and Schuster, New York, NY, USA, 1992.

[17] M. Mohamed, S. Yangui, S. Moalla, and S. Tata, "Web service micro-container for service-based applications in cloud environments," in *Proceedings of the 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '11)*, pp. 61–66, IEEE Computer Society, 2011.

[18] J. Von Neumann, "The General and Logical Theory of Automata," in *Cerebral Mechanisms in Behavior, The Hixon Symposium, Edited by L. A. Jeffress*, W. Asprey and A. Burks, Eds., Reprinted in Papers of John von Neumann on Computers and Computing Theory, pp. 456–457, The MIT Press, Cambridge, Mass, USA, 1987.

[19] S. B. Carroll, *The New Science of Evo Devo—Endless Forms Most Beautiful*, W. W. Norton & Company, New York, NY, USA, 2005.