Chapter 2
Understanding Distributed Systems and Their Management

Abstract An analysis of current implementation of distributed computing and its management using the von-Neumann stored program control computing model is presented to identify ways to improve the resiliency, efficiency and scaling of distributed transactions supporting the demands of communication, collaboration and commerce at the speed of light. A comparison with other distributed computing models such as the cellular organisms, human organizational networks and telecommunication networks points to a new computing model that leverages parallelism, signaling and end-to-end transaction management improving the resilience, efficiency and scaling of distributed transactions.

Stored Program Control Computing Model, Distributed Computing, and Management

> "Despite more than 30 years of progress towards ubiquitous computer connectivity, distributed computing systems have only recently emerged to play a serious role in industry and society. Perhaps this explains why so few distributed systems are reliable in the sense of tolerating failures automatically, or guaranteeing properties such as high availability, or having good performance even under stress, or bounded response time, or offer security against intentional threats. In many ways the engineering discipline of reliable distributed computing is still in its infancy.
>
> Reliability engineering is a bit like alchemy. The field swirls with competing schools of thought. Profound arguments erupt over obscure issues, and there is little consensus on how to proceed even to the extent that we know how to solve many of the hard problems."—Kenneth Paul Birman [1].

> "The sharing of resources is a main motivation for constructing distributed systems"—George Coulouris, Jean Dollimore, Tim Kindberghy [2].

R. Mikkilineni, *Designing a New Class of Distributed Systems*, SpringerBriefs in Electrical and Computer Engineering, DOI: 10.1007/978-1-4614-1924-2_2,

> "A distributed system is a collection of independent computers that appears to the users as a single coherent system."—Andrew S. Tanenbaum and Martin van Steen [3]

Since von Neumann [4] discussed, at the Hixon symposium in 1948, his views on the resilience of the cellular organisms and the shortcomings of the stored program control machines he designed, the quest for building reliable systems on an infrastructure that is not, often, reliable has been the holy grail of information technologies. As Birman points out, distributed systems pose more of a challenge because of the need for the management to coordinate the collaborating resources that span across different hardware and geographies to accomplish the goals of the overall system. A transaction to accomplish a system goal in a distributed system by definition spans across distributed shared resources transcending geographical boundaries. Fault, configuration, accounting (of who uses what resources), performance and security management of distributed resources that collaborate using different communication mechanisms and network connections determine the overall response time and success or failure of the end-to-end transaction. The shortcomings or, the aspects of alchemy as Birman [1] puts it, arise from a lack of consistent treatment of distributed computing and its management. In this chapter we revisit the stored program control computing model, traditional treatment of distributed systems, and their management strategies to show that many of the issues are a result of the coupling of computing tasks and their management tasks implemented using the stored program control computing model. We also study other distributed systems such as cellular organisms, human network organization and telecommunication networks that are proven to be very resilient, to define a new class of distributed computing systems.

Current distributed computing practices have their origin from the server-centric von Neumann Stored Program Control (SPC) architecture that has evolved over the last five decades. In its simplest form, the computation and storage are separated using CPU and memory devices. A single storage structure holds both the set of instructions on how to perform the computation and the data required or generated by the computation. Such machines are known as stored-program computers. The separation of storage from the processing unit is implicit in this model. The ability to treat instructions as data is what makes compilers possible. It is also a feature that can be exploited by computer viruses when they add copies of themselves to existing program code. The problem of unauthorized code replication can be addressed by the use of memory protection support. Virtual memory architectures have incorporated management of computing and storage resources in the operating systems. During last five decades, many layers of computing abstractions have been introduced to map the execution of complex computational workflows to a sequence of 1 s and 0 s that eventually get stored in the memory and operated upon by the CPU to achieve the desired result. These include process definition languages, programming languages, file systems, databases, operating systems etc.

While this has helped in automating many business processes, computing remained mainly centralized in islands of main frames and mini computers with

occasional time-sharing thrown in until Ethernet was introduced to connect multiple computers. Distributed computing utilizing networked computing resources came of age in the 1970s, starting from a client–server computing model, and has fully developed to current grid-computing and cloud computing implementations where hundreds of physical and virtual servers are used in distributed grids and clouds to provide orchestrated computational workflow execution. With the steady increase of computing power in each node and 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, failure of participating nodes, issues of trust, and management of latency and performance. These problems are well articulated in literature and the discipline of distributed computing is devoted to address them. There are three major attributes that must be considered in designing distributed systems:

1. 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 management allows proper allocation of resources to appropriate consumers 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:

    a. Measure the FCAPS parameters both at the individual resource level and at the system level and
    b. Control the resources system-wide based on the measurements, business priorities, varying workloads, and latency constraints of the distributed transactions.

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.

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

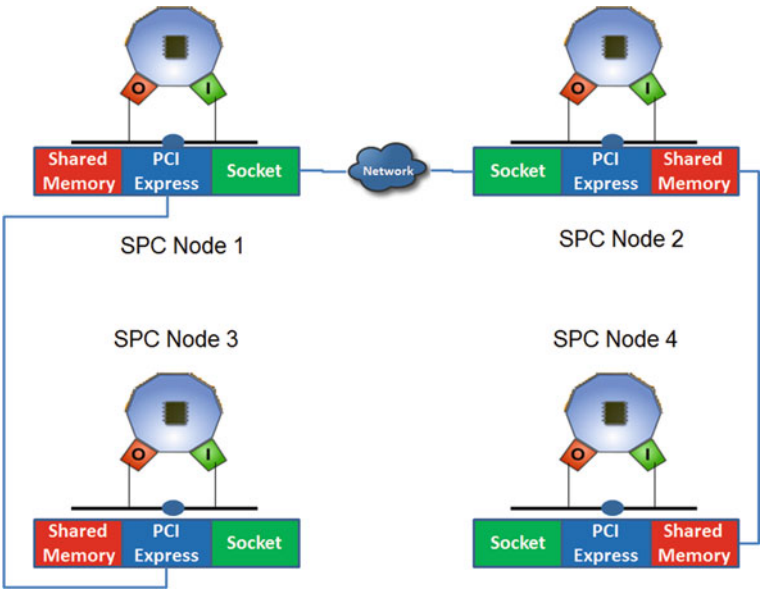

Fig. 2.1 A network of stored program control nodes implementing computational workflow and the resource management workflow

a. Individual resource utilization efficiency which measures the cost of executing a task by the component with a specified service level and
b. 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.

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

3. Scaling: 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 implemented by the SPC computing nodes and (b) the management workflow which addresses the management of the network of computing nodes to assure connectivity, availability, reliability, utilization, performance, and security (FCAPS) of the distributed resources. Figure 2.1, shows a group of SPC nodes connected by different mechanisms (shared memory, PCI Express and Socket communications) with different bandwidths.

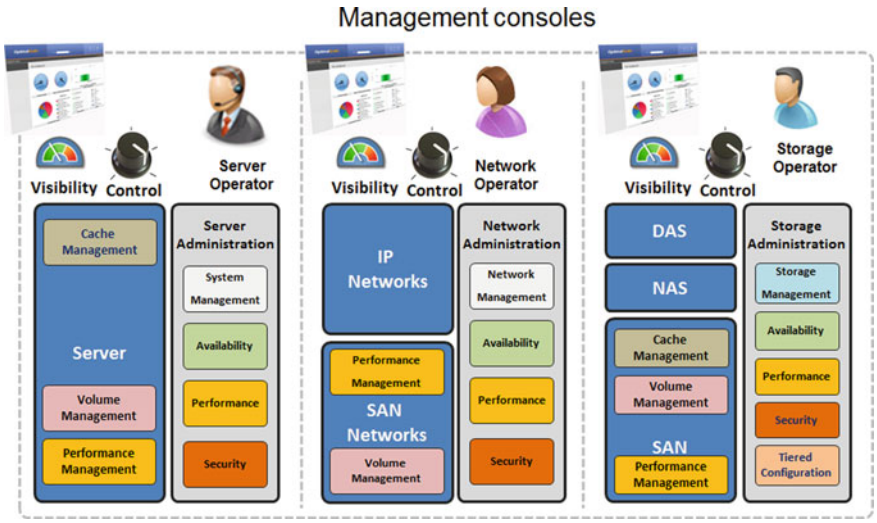

Fig. 2.2 Layers of management governing the behavior of computational workflows with system, server, network and storage management systems

Traditionally, the operating system provided the node resource management (memory, CPU cycles, bandwidth, storage capacity, throughput and its rate of Input/Output). Various network, storage and server management systems provided the FCAPS management of resources at the network or system level. As the distributed transactions start spanning across many nodes and networks across various geographies, the end to end response becomes a function of workloads on individual nodes, sub-networks, business priorities that control access to various applications sharing the resources and the overall latency constraints. In order to control the end-to-end transaction FCAPS, both management workflows and computational workflows are implemented using the same SPC computing network. Many of the functions controlling the provisioning, fault management, utilization control, performance and security monitoring and control are distributed in various nodes as workflows using the same SPC nodes.

The server, network and storage resources are monitored and controlled by different management workflows often duplicating many functions. The open systems approach, multiple vendor products evolving simultaneously to specialize in server, network and storage functions to improve resiliency, efficiency and scaling, and a lack of end-to-end systems optimization strategies in point products, all have contributed to creating a complex web of hardware and software systems in the IT data center. Figure 2.2 shows the evolution of the management workflows.

With the advent of virtualization technologies, the resource management is accessible through automation systems reducing the labor intensive server, network and storage management functions thus reducing the human latency involved

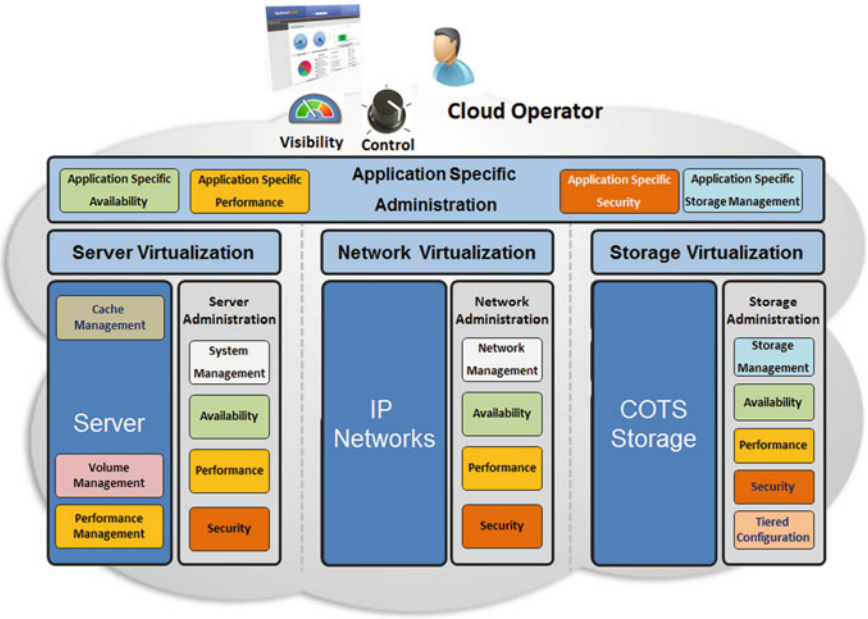

Fig. 2.3 Automation of application-centric management governing the behavior of computational workflows with virtual server, network and storage management systems

in responding to changing circumstances. Both grid and cloud computing technologies make use of system-wide resource management to control the resource allocation priorities based on specific application requirements such as business priorities, workload fluctuations and latency constraints. The management systems monitor the resource utilization characteristics across the system and implement appropriate control workflows to reconfigure the resources. Figure 2.3 shows the evolution of resiliency with the advent of grid and cloud computing.

As the number of CPUs within an enclosure increase, the distributed resource control within the enclosure (processors and cores in each processor) is relegated to the local operating system in that enclosure and it implements appropriate management workflows to match the demands of applications that request the resources.

The system level management is implemented using a plethora of resource management systems. The resulting layers of computational and management workflows improve resiliency, efficiency and scaling of distributed systems. Figure 2.4 shows the evolution of resiliency, efficiency and scaling with the introduction of grid and cloud computing technologies.

The picture shows the FCAPS management ability and the resulting resiliency of a distributed system, its scaling ability to be able to add the number of computing elements and the efficiency (shown by the size of the sphere). Conventional computing where server, network and storage management are resource centric

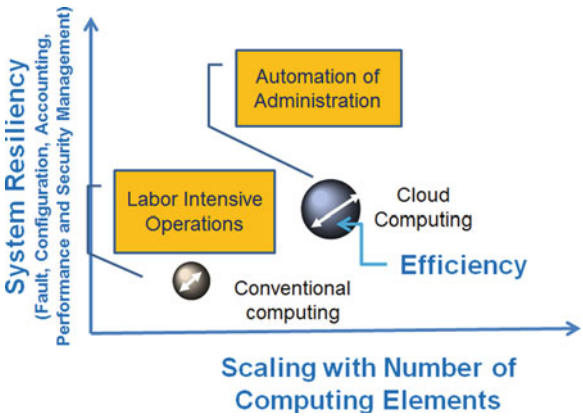

Fig. 2.4 Resiliency, Efficiency and Scaling of Computing systems

and are static (i.e., applications do not have the ability to request more or less resources based on their workloads and latency constraints), the resiliency and scaling are still limited by the human latency involved in resource management using various management systems. The physical server is the computing unit in conventional computing which provides FCAPS management. With the advent of virtualization technologies, another layer of application-centric dynamic resource management albeit using multiple management systems, improves the resilience, efficiency and scaling. In this case, a virtual server constitutes a computing unit with FCAPS management.

In both cases, the computational workflow and management workflow are implemented using the SPC computing node (either a physical server based application or virtual server based application) and are executed in a serial fashion as shown in Fig. 2.5.

In this approach, the end-to-end distributed transaction resilience is dependent on detecting changes required and correcting them and the serial nature of this process introduces an inherent latency in the process. If the timescale, in which the external requirements change because of the changes in business priorities, workloads fluctuations or latency constraints, is much larger than the time it takes to respond, the serial process does not pose a serious problem. On the other hand, if the scale of changes varies by orders of magnitude in a short period of time as experienced by current Internet based service delivery on a global scale, the von-Neumann bottleneck becomes pronounced and will adversely impact the resiliency, efficiency and scaling of distributed transactions.

The limitations of the SPC computing architecture were clearly on his mind when von Neumann gave his lecture at the Hixon symposium in 1948 in Pasadena, California [4]. He pointed out that "Turing's procedure is too narrow in one respect only. His automata are purely computing machines. Their output is a piece of tape with zeros and ones on it." However, he saw no difficulty in principle in dealing with the broader concept of an automaton whose output is other automata

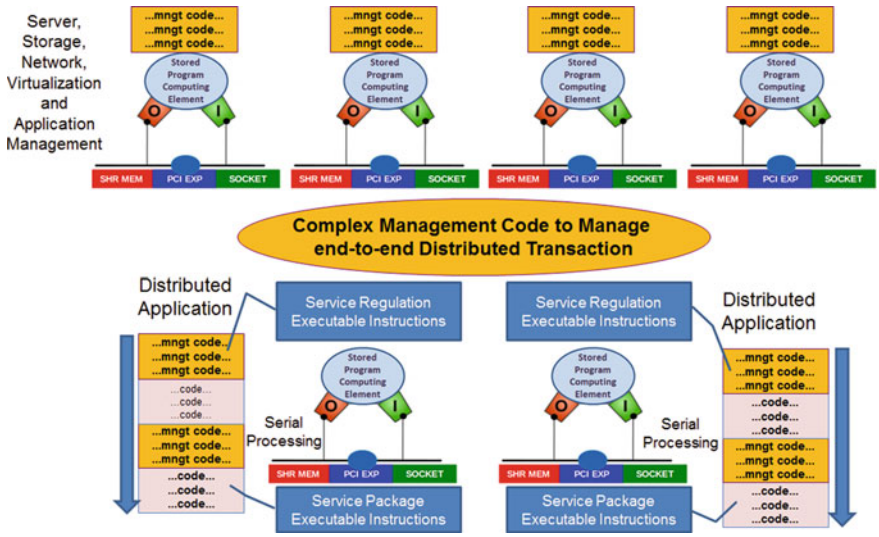

Fig. 2.5 Distributed computational and management workflow implementation using the stored program control computing model where the node is either a physical server or a virtual server

and in deriving from it the equivalent of Turing result. He went on to make this remark. "Normally, a literary description of what an automaton is supposed to do is simpler than the complete diagram of the automaton. It is not true a priori that this always will be so. There is a good deal in formal logic which indicates that when an automaton is not very complicated the description of the function of the automaton is simpler than the description of the automaton itself, as long as the automaton is not very complicated, but when you get to high complications, the actual object is much simpler than the literary description." He went on to say, "It is a theorem of Gödel that the description of an object is one class type higher than the object and is therefore asymptotically infinitely longer to describe." The conjecture of von Neumann leads to the fact that "one cannot construct an automaton which will predict the behavior of any arbitrary automaton [5]." This is the case with the Turing machine implemented by the SPC model.

Cellular Organisms, Genes, Chromosomes and Distributed Computing

It turns out that the description and the execution of the described function play a crucial role in cellular organisms giving them the capability to replicate, repair, recombine and reconfigure themselves. These genetic transactions are supported by DNA (Deoxyribonucleic acid), genes and chromosomes. As Mitchell Waldrop

explains in his book on Complexity [6], "the DNA residing in a cell's nucleus was not just a blue-print 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, 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." The conjecture of von Neumann leads to the fact that the SPC computing model alone is not adequate for self-replication and self-repair. Organisms somehow have managed to precisely encapsulate the descriptions of building and running a complex system such as a human being in a simpler vehicle such as a set of genes and chromosomes. They have also managed to invent mechanisms for replication, repair, recombination and rearrangement to execute the descriptions precisely. According to Jacob and Monod [7], "The gene circuitry of an organism connects its gene set (genome) to its patterns of phenotypic expression. The genotype is determined by the information encoded in the DNA sequence, the phenotype is determined by the context dependent expression of the genome, and the circuitry interprets the context and orchestrates the patterns of expression. Gene circuits sense their environmental context and orchestrate the expression of a set of genes to produce appropriate patterns of cellular response."

The relationship between von Neumann's self-replication and genetic behavior was recognized by Chris Langton [8] who created a new field called artificial life, which evolved to throw light on self-organization and the emergence of order from chaos. The field of artificial life and genetic programming focus on self-organization under probabilistic evolution rules that reduce overall entropy of the system. However, equally fascinating feature of the genome is its ability to reproduce itself with a precision that is unparalleled.

As George Dyson, in his book 'Darwin among the Machines,' observes [9] "The analog of software in the living world is not a self-reproducing organism, but a self-replicating molecule of DNA. Self-replication and self-reproduction have often been confused. Biological organisms, even single-celled organisms, do not replicate themselves; they host the replication of genetic sequences that assist in reproducing an approximate likeness of them. For all but the lowest organisms, there is a lengthy, recursive sequence of nested programs to unfold. An elaborate self-extracting process restores entire directories of compressed genetic programs and reconstructs increasingly complicated levels of hardware on which the operating system runs." Life, it seems, is an executable directed acyclic graph (DAG) and a managed one at that.

A cell is formed by a stable pattern of chemical molecules that establish equilibrium with its environment and optimize resource utilization to maintain its equilibrium. According to Richard Dawkins [10], "DNA molecules do two important things. Firstly, they replicate, that is to say they make copies of themselves. This has gone on non-stop ever since the beginning of life and the DNA molecules are now very good at it indeed."

It is one thing to speak of the duplication of DNA. But if the DNA is really a set of plans for building a body, how are the plans put into practice? How are they translated into the fabric of the body? Dawkins poses these questions and answers

them. "This brings me to the second important thing DNA does. It indirectly supervises the manufacture of a different kind of molecule—protein. The coded message of the DNA, written in the four-letter nucleotide alphabet, is translated in a simple mechanical way into another alphabet. This is the alphabet of amino acids which spells out protein molecules."

The indirect supervision task mentioned by Dawkins is related to the function of the Gene, which is a piece of DNA material that contains all the information needed to "build" specific biological structure. Genes thus contain the information to make proteins, the body's building blocks. Proteins make up the structure of the organs and tissues; they are also needed for the body's chemical functions and pathways. Each protein performs a specific job in the body's different types of cells, and the information for making at least one protein is contained in a single gene. The pattern or sequence of the genes is like a blueprint that tells the body how to build its different parts. The Genes constitute the workflow components for building the biological system consisting of a group of cells that act with a single purpose which is to propagate the equilibrium patterns they have found to survive.

Cells may have a variety of fates: they may divide and increase in number, differentiate into different kinds of cells, or die (apoptosis). "Determination of the fate of a cell starts when a protein called a signaling molecule binds to a receptor embedded in the cell membrane," says Yasushi Sako, [11] Chief Scientist, Cellular Informatics Laboratory, RIKEN Advanced Science Institute in Japan. When bound by a signaling molecule, the receptor is activated and information is transmitted into the cell. The information is then conveyed from one protein to another within the cell through repeated binding, dissociation and migration until it eventually reaches the cell nucleus, where it induces the expression of a specific gene. This gene triggers various cellular responses, including proliferation, growth inhibition, differentiation, apoptosis, and oncogenic transformation.

Even the simplest unicellular organism provides a good example of self-management and reproduction to sustain life. However, more recent studies in evolutionary developmental biology throw fundamental insights into the inner workings of how groups of cells are organized and orchestrated to create what the biologist Sean B. Carroll calls "endless forms more beautiful" [12]. He points out that "just about 1.5%, codes for the roughly 25,000 proteins in our bodies. So what else is there in the vast amount of our DNA? Around 3% of it, made up of about 100 million individual bits, is regulatory. This DNA determines when, where, and how much of a gene's product is made." With modularity, multi-functionality and redundancy built in its architecture, how does orchestration take place? The orchestration is accomplished by "parallel and sequential actions of tool kit genes—dozens of genes acting at the same time and same place, many more genes acting in different places at the same time, and hundreds of toolkit genes acting in sequence." Each gene may have multiple switches which can be switched on or off by the gene toolkit thus controlling the behavior of the gene. "The developmental steps executed by individual switches and proteins are connected to those of other genes and proteins. Larger sets of interconnected switches and proteins form local "circuits", which are part of still larger "networks" that govern the development

of complex structures. Animal architecture is a product of genetic regulatory network architecture."

He goes on to say "another class of toolkit members belong to so-called signaling pathways. 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."

To summarize, the key abstractions the cell architecture supports are:

4. The spelling out of computational workflow components as a stable sequence of patterns that accomplishes a specific purpose,
5. A parallel management workflow specification with another sequence of patterns that assures the successful execution of the system's purpose (the computing network) and
6. A signaling mechanism that controls the execution of the workflow for gene expression (the regulatory network)
7. Real-time monitoring and control to execute genetic transactions which provide the self-* properties

Human Networks and Distributed Computing

Another example of a distributed system is the human network which has developed sophisticated abstractions and complex patterns for individuals to collaborate together and accomplish a common objective. The human networks are considered intelligent because they accomplish their goals in multiple ways using information collected from the external world and using it to control it. Just as cellular organisms have developed evolutionary best practices and pass them on from survivors to their successors, humans have evolved organizational best practices to leverage their individual capabilities as part of a group by defining and accomplishing common goals with high resiliency, effectiveness and scaling. A human network consists of a group of individuals organized to communicate and collaborate globally to carry out individual tasks executing a part of a distributed transaction using local intelligence and resources. Each group has a purpose and can be part of a larger group with its purpose consistent with that of the larger group. Each individual contributes (taking advantage of specialization) to the overall execution of the distributed transaction (implementing separation of concerns) as a part of the workflow as a managed directed acyclic graph. The group implements both the node-level and the network-level FCAPS management giving it the self-* management capabilities.

The effectiveness of the human network depends on the connections, communication and mastery (or specialization) of the individual human object. Better the quality of mastery of the individual node, the quality of connection and

communication, higher the effectiveness. Humans have created organizational frameworks through evolution. According to Malone [13], organization consists of connected "agents" accomplishing results that are better than if they were not connected. An organization establishes goals, segments the goals into separate activities to be performed by different agents, and connect different agents and activities to accomplish the overall goals. Scalability is accomplished through hierarchical segmentation of activities and specialization.

There is always a balance between the cost of coordination of the agents and economies of scale obtained from increasing the network size which defines the nature of the connected network. Efficiency of the organization is achieved through specialization and segmentation. On the other hand agility of an organization depends on how fast the organization can respond to changes required to accomplish the goals by reconfiguring the network. Dynamic reconfiguration is accomplished using signaling abstractions such as addressing, alerting, supervision and mediation.

Both efficiency and agility are achieved through a management framework that addresses FCAPS of all network elements (in this case the agents). Project management is a specific example where Fault, configuration, accounting, performance and security are individually managed to provide an optimal network configuration with a coordinated work-flow. Functional organizations, and hierarchical and matrix organizational structures are all designed to improve the resiliency, efficiency, scaling and agility of an organization to accomplish the goals using both FCAPS management and signaling.

Connection management is achieved through effective communications framework. Over time, human networks have evolved various communications schemes and signaling forms the fundamental framework to configure and reconfigure networks to provide the agility. There are four basic abstractions that comprise signaling:

1 Alerting,
2 Addressing,
3 Supervision and
4 Mediation

Organizational frameworks are designed to implement these abstractions using distributed object management in human networks. Signaling allows prioritization of the network objectives and allocates resources in the form of distributed agents to accomplish the objectives and provides management control to resolve contention and mitigate risk. Elaborate workflows are implemented using the signaling mechanism to specialize and distribute tasks to various agents. The agents are used to collect information, analyze it and execute controls while organizing themselves as a group to accomplish the required goals.

Thus organizational hierarchies, project management, process implementation through workflows are all accomplished through the network object model with FCAPS abstractions and signaling. It is important to note that the signaling abstractions, while commonly used, have not been discussed widely in the

distributed computing domain. First clear articulation is found in the description of SS7 signaling in telecommunications domain and a reference to it by Gartner group [14].

Telecommunications and Distributed Computing

For much of its history, AT&T and its Bell System functioned as a legally sanctioned, regulated monopoly. The fundamental principle, formulated by AT&T president Theodore Vail in 1907, was that the telephone by the nature of its technology would operate most efficiently as a monopoly providing universal service. Vail wrote in that year's AT&T Annual Report [15] that government regulation, "provided it is independent, intelligent, considerate, thorough and just" was an appropriate and acceptable substitute for the competitive marketplace." From the beginning of **AT&T** to today's remaking of **at&t**, much has changed but two things that remain constant are the universal service (on a global scale) and the telecom grade "trust" (providing reliable, secure and high performance connection at a reasonable cost) that are taken for granted. The Plain Old Telephone System (POTS) altered the communication landscape by connecting billions of humans anywhere any time at a reasonable cost. It provided the necessary managed infrastructure to create the voice dial tone, deliver it on demand and assure the connection to meet varying workloads and individual preferences with high availability, optimal performance and end-to-end connection security. The service assurance set a standard known as "telecom grade trust". Two major factors that contributed to the telecom grade trust are the end-to-end network management of various elements and the signaling network [14] that is used to dynamically manage the resources for every connection based on profiles.

Entry for FCAPS in Wikipedia, [16] states that it "is the ISO Telecommunications Management Network model and framework for network management. FCAPS is an acronym for fault, configuration, accounting, performance, and security which are the management categories into which the ISO model defines network management tasks. In non-billing organizations, accounting is sometimes replaced with administration. The comprehensive management of an organization's information technology (IT) infrastructure is a fundamental requirement. Employees and customers rely on IT services where availability and performance are mandated, and problems can be quickly identified and resolved. Mean time to repair (MTTR) must be as short as possible to avoid system downtimes where a loss of revenue or lives is possible".

All intelligent telecommunication network elements today are FCAPS enabled and Operation Support Systems are designed to provide day to day operations and management. The operation support systems and the network elements utilize the signaling abstractions to provide an elaborate communication infrastructure that enables collection, analysis and control of various elements to accomplish the business goals.

Similarly, the network infrastructure that forms the backbone of the Internet (consisting of the servers, routers, switches and other network elements) has liberally borrowed the FCAPS and signaling abstractions to implement agility and resiliency required. In fact FCAPS awareness has become a mandatory requirement to be a network element in both telecommunications and Information Technology infrastructure. As an example, as storage also became networked, various storage network elements have started to become FCAPS aware and participate in a common management framework.

Client Server, Peer-to-Peer, Grid and Cloud Computing Management

Resiliency, efficiency and scaling of a distributed system are very much dependent on the division of responsibilities between individual computing nodes, the placement of them, and connectivity between the nodes. In the SPC architecture, as we discussed earlier, the computational workflow and the management workflow are distributed among the various nodes. An SPC node provides an atomic computing unit and is programmed to perform useful activity with well-defined responsibility and interact with each other using a communication channel. Different placements of the responsibilities and communication schemes define various derived computing models [3]. We will discuss some of these models with respect to their resiliency, efficiency and scaling characteristics.

Client–Server and Peer-to-Peer Models

Each node in this model is a physical container managed by a single operating system to allocate resources to various hosted processes executing different tasks contributing to either the computing workflow or management workflow constituting a distributed transaction. The container may be a server providing services or a client receiving the services or both. Over a period of time, TCP/IP has become the standard for communication between processes located on different containers. Within the container the processes communicate via high speed shared memory or the PCI bus depending on the physical architecture. In the client server model, a server can be a client or a client a server depending on the context.

Services are implemented using multiple processes supported by the local operating system which communicate and collaborate with each other to implement the distributed transactions. Various FCAPS management workflows are implemented both using code that is mixed with the service workflow and separate processes in various nodes. For example replication is used to improve fault tolerance, performance and availability.

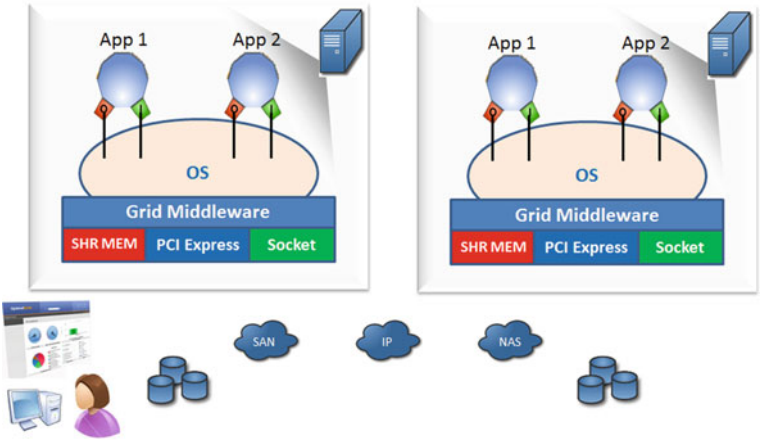

Fig. 2.6 Grid Computing Model

In peer-to-peer model, applications as processes in the operating system play a similar role, interacting cooperatively as peers to perform distributed transactions or computations without any distinction between clients and servers. The code in the peer processes also provides FCAPS management by maintaining consistency of application-level resources and synchronizes application-level actions when required.

Grid and Cloud Computing

Grid computing is designed to share disparate, loosely coupled IT resources across organizations and geographies. Using a grid middleware, the IT resources are offered as services. These resources include almost any IT component—computer cycles, storage spaces, databases, applications, files, sensors, or scientific instruments. Resources can be shared within a workgroup or department, across different organizations and geographies, or outside the enterprise.

Figure 2.6 depicts the role of Grid middleware and associated management software. The resources can be dynamically provisioned to users or applications that need them on demand.

With the advent of virtualization technologies, the physical enclosure that supported one operating system now allows multiple operating systems.

Figure 2.7 shows the new cloud computing model where multiple managed virtual servers contained in a physical enclosure can be dynamically provisioned, an operating system installed in every virtual server and applications can be run to share the CPU, network and storage resources.

Grid and cloud computing advances attempt to compensate for the deficiencies of the resource-centric management systems of conventional computing at the

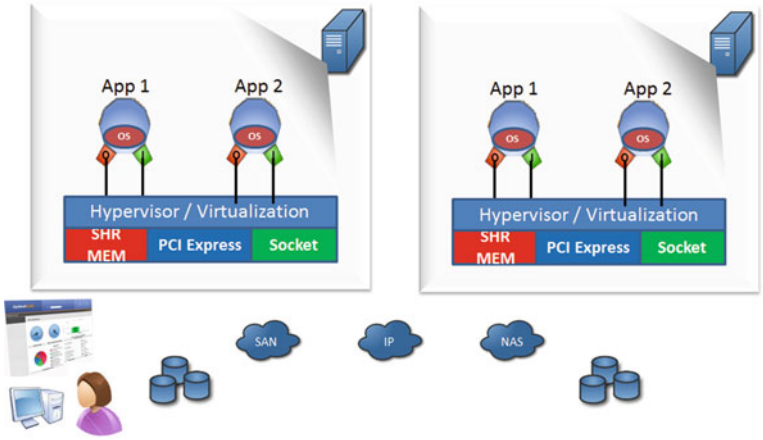

Fig. 2.7 Cloud computing model

application level by creating additional layers of resource utilization monitoring and management [17, 18]. Resulting automation and end-to-end visibility offered through collaborating management systems compensate for the inadequacy of the individual node operating system to see the global resource utilization involved in the distributed transaction. The distributed transaction resiliency, efficiency and scaling are improved through the control of end-to-end resources involved in the transaction.

Hardware Upheaval and The von Neumann Bottleneck

According to András Vajda [19], "The chip industry has recently coined the 'new' Moore's law, predicting that the number of cores per chip will double every two years over the next decade, leading us to the world of many-core processors-loosely defined as chips with several tens but more likely hundreds, or even thousands of processor cores." The reason for this shift from increasing clock speed to improve performance to increasing the number of cores in the same chip is two-fold:

1. Around the year 2004, the single processor clock-speed reached a maximum at about 4 GHz even though, the Moore's law allowed more transistors in smaller areas. The chip vendors decided to address this problem by increasing the number of cores in a chip thus improving the overall computing power.
2. This had the added advantage of saving power consumption and improving power dissipation, required to support higher speeds.

As the capabilities and speed of cores kept improving, the memory access speed lagged behind which led to the implementation of hardware multi-threading, a

mechanism through which a core could support multiple thread contexts in hardware (including program counter and register sets, but sharing for example the cache memory) and fast switching between hardware threads stalled due to high latency operations. In addition, the many-cores in a chip also provide high-bandwidth interconnect which links multiple cores together to provide a single logical processing unit together.

The hardware advances implementing a large network of cores in a chip and a large network of chips in a server (often connected by a high-speed PCIExpress bus) create a new challenge to the current operating systems, and software practices that have their origins from the von Neumann serial computing model. The many-core architecture pushes the network of computing nodes inside a single enclosure with far higher bandwidths and faster access to stored program instructions than is possible by connecting different devices as is the state-of-the-art practice today in grid and cloud computing. This adds a new dimension in distributed transaction processing where resources now collaborate and cooperate inside the enclosure at high speed and the hardware multi-threading compounds the software challenge to take advantage of the inherent parallelism offered. There are three major reasons why the hardware upheaval unleashed by the many-core processors is making the current distributed software architectures to be reexamined:

1. The role of an operating system is to provide an abstraction layer of underlying hardware, taking care of interrupts, processor management, low-level interaction with peripherals, and allocate resources to applications with a unified interface for their use and management. Current generation operating systems which are node centric are not scalable because they do not have visibility of resources across the entire network. Figure 2.8 shows the recursive nature of networked SPC nodes with different resource allocation and management requirements. It is easy to see how the current device-centric OSs with their evolution from single-thread operating system data structures to support multi-threaded operating system data structures over time, with a large and complex code-base dealing with choosing correct lock granularity for performance, reasoning about correctness, and deadlock prevention are inadequate for network-centric resource management [20].
2. Application response time, in a many-core system depends on run-time workload fluctuations and latency constraints in a ***shared processor and core network***-infrastructure as shown in Fig. 2.8. It therefore, becomes imperative to bring distributed transaction management infrastructure inside the many-core device allocating appropriate resources to various services that consume them based on business priorities, workload fluctuations and latency constraints. For example, if two computing nodes involved in collaboration with each other are in two different devices, the communication channel must be switched to socket communication, where as if they are communicating across two cores, shared memory would be an appropriate resource to be allocated. In a web based distributed transaction that spans across multiple geographies, the dynamic nature of the transaction demands dynamic resource allocation to optimally

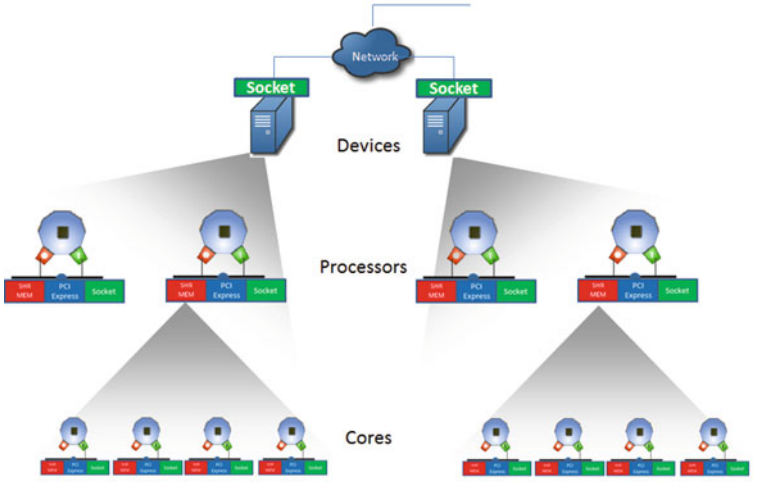

Fig. 2.8 The many-core servers and the recursive network-centric resource allocation and management requirements

execute the transaction. Current operating systems and management systems fall short in providing dynamic resiliency, efficiency and scaling for two reasons:

- The serial nature of von-Neumann computing node forces both service execution and service management to be intermixed as discussed above and introduces a level of complexity in dynamically managing resources globally based on changing business priorities, workload fluctuations and latency constraints.
- Current generation management systems have their origins in TCP/IP based narrow bandwidth environment and are not suited to leverage the high bandwidth and fast access to memory made available in many-core servers.

3. It is hard to imagine replicating current TCP/IP based socket communication, "isolate and fix" diagnostic procedures, and the multiple operating systems (that 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. In order to cope with the scaling issues and utilize many-core technologies 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 2.9 shows the need for improving the resiliency, efficiency and scaling of many-core software services and their management architecture.

It is clear that current approaches to resource management, albeit with automation, are not sensitive to the distributed nature of transactions and contention

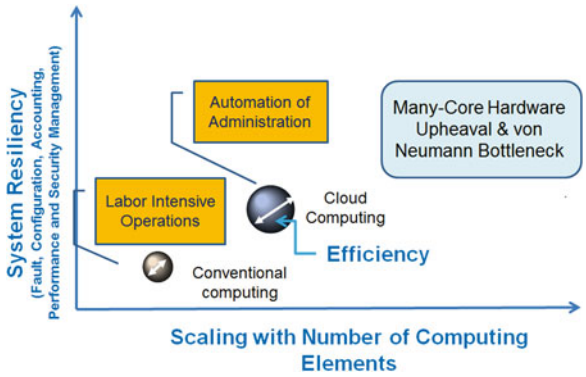

Fig. 2.9 The resiliency, efficiency and scaling diagram with many-core hardware upheaval and the von Neumann Bottleneck

resolution of shared distributed resources, at best, is complex involving many layers of management systems. As von Neumann [4] 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. More recent efforts, in a similar vein, are looking at resiliency borrowing from biological principles [21] to design future Internet architecture.

In the next chapter, we will revisit the design of distributed systems with a new non-von Neumann computing model (called Distributed Intelligent Managed Element (DIME[1]) Network computing model) 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. The DIME network architecture was first presented in WETICE 2010 in Larissa, Greece based on the workshop discussions started in WETICE 2009 in Groningen, Netherlands [22–24].

References

1. K.P. Birman, *Reliable Distributed Systems: Technologies, Web Service and Applications* (Springer, NY, 2005), p. 19
2. G. Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems, Concepts and Design*, 3rd edn. (Addison Wesley, New York, 2001), p. 1

[1] DIME™, Cloud-DNA and Dime Network Architecture are Trade Marks of Kawa Objects Inc.

3. A.S. Tanenbaum, M. van Sheen, *Distributed Systems, Principles and Paradigms* (Prentice Hall, New Jersey, 2002), p. 2
4. J.V. Neumann, *General and Logical Theory of Automata*, William Aspray and Arthur Burks (eds.), (MIT Press, Cambridge, 1987), p. 408
5. J.V. Neumann, Papers of John von Neumann on Computing and Computer Theory, in *Charles Babbage Institute Reprint Series for the History of Computing*, ed. by William Aspray, Arthur Burks (MIT Press, Cambridge, MA, 1987), pp. 409–474
6. M.M. Waldrop, *Complexity: The Emerging Science at the Edge of Order and Chaos* (Penguin Books, London, 1992), p. 218
7. F. Jacob, J. Monod, Genetic regulatory mechanisms in the synthesis of proteins. J. Mol. Biol. **3**, 318–356 (1961)
8. C.G. Langton, *Artificial Life*, Santa Fe Institute Studies in the Sciences of Complexity, in Proceedings **6**. (Addison Wesley, Redwood City, CA, 1989)
9. G.B. Dyson, *Darwin among the Machines, the evolution of global intelligence* (Addition Wesley, Reading, MA, 1997), p. 123. Helix Books
10. R. Dawkins, *The Selfish Gene* (Oxford University Press, New York, 1989), p. 23
11. Y. Sako, (2010), http://www.researchsea.com/html/article.php/aid/5501/cid/1/research/studying_cell_signaling_using_singlemolecule_imaging.html?PHPSESSID=rlufujxhvtg. (Accessed 27 November 2010) from Asia Research News: http://www.researchsea.com/html/article.php/aid/5501/cid/1/research/studying_cell_signaling_using_singlemolecule_imaging.html?PHPSESSID=rlufujxhvtg
12. S.B. Carroll, *The New Science of Evo Devo—Endless Forms Most Beautiful* (W. W. Norton & Co, New York, 2005), pp. 12 ,106 ,113, 129
13. T.W. Malone, Organizing Information Processing Systems: Parallels Between Human Organizations and Computer Systems, in *Cognition, computing, and cooperation*, ed. by S.P. Robertson, W. Zachary, J.B. Black (Ablex Publishing, New Jersey, 1990)
14. Gartner Dataquest Says Next Generation Signaling to Prosper with Emergence of Next Generation Networks. Business Wire. Accessed 12 February 2001. (http://www.allbusiness.com/technology/software-services-applications-information/6029582-1.html)
15. AT&T, Brief history (2010), http://www.corp.att.com/history/history3.html. Accessed 28 November 2010 from AT&T history
16. Wikipedia: http://en.wikipedia.org/wiki/FCAPS (Retrieved 11 29, 2010)
17. R. Buyya, R. Ranjan, Special section: Federated resource management in grid and cloud computing systems. Fut. Gen. Comput. Syst. **26**, 1189–1191 (2010)
18. R. Buyyaa, C.S. Yeoa, S. Venugopala, J. Broberga, I. Brandicc, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Fut. Gen. Comput. Syst. **25**(6), 599–616 (2009)
19. A. Vajda, *Programming Many-Core Chips* (Springer, New York, 2011), p. 3
20. D. Wentzlaff, A. Agarwal, Factored operating systems (fos): the case for a scalable operating system for multicores. SIGOPS Oper. Syst. Rev. **43**(2), 76–85 (2009)
21. S. Balasubramaniam, K. Leibnitz, P. Lio', D. Botvich, M. Murata, Biological Principles for Future Internet Architecture Design. IEEE Comm. Mag. **49**(7), 44 (2011)
22. P. Goyal, The Virtual Business Services Fabric: An Integrated Abstraction of Services and computing Infrastructure, in *Proceedings of WETICE 2009: 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2009, pp. 33–38
23. P. Goyal, R. Mikkilineni, M. Ganti, Manageability and Operability in the Business Services Fabric, in *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09*, 29 June–1 July 2009, pp. 39–44
24. P. Goyal, R. Mikkilineni, M. Ganti, FCAPS in the Business Services Fabric Model, in *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09*, 29 June–1 July 2009, pp. 45–51

http://www.springer.com/978-1-4614-1923-5

Designing a New Class of Distributed Systems

Mikkilineni, R.

2011, XII, 65 p. 30 illus., Softcover
ISBN: 978-1-4614-1923-5