

Parallax – A New Operating System Prototype Demonstrating Service Scaling and Service Self-Repair in Multi-core Servers

Dr. Rao Mikkilineni
IEEE Member
Kawa Objects Inc., USA

and

Ian Seyler
Return Infinity, Canada

rao@kawaobjects.com, ian.seyler@returninfinity.com,

Abstract— This paper describes a prototype demonstrating a new operating system (called Parallax) for the new generation many-core servers. The operating system is designed to address scaling, resource monitoring, dynamic configuration, and self-repair of many-core chip based servers to support distributed computational tasks. The operating system is implemented in assembler language for efficiency and supports C/C++ programming interfaces for high-level programming.

The Parallax operating system is based on the DIME network computing model supporting distributed network-centric computing architecture. A parallel signaling network overlay over a network of von Neumann stored program control (SPC) computing nodes is utilized to implement dynamic fault, configuration, accounting, performance, and security management of both the nodes and the network based on business priorities, workload variations and latency constraints.

Keywords - Cloud Computing; Distributed Computing; Parallel Computing; Distributed Services Management; Operating System; Parallax; FCAPS (Fault, Configuration, Accounting, Performance, and Security) Management.

I. INTRODUCTION

Hardware advances such as Intel Xeon processors, low-power x86 and Xscale ARM RISC chips are making possible highly scalable many-core servers with large addressable memory¹. The upheaval in hardware is not matched by equal innovation in operating systems to take advantage of the abundance of computing, memory, network, and storage resources. While the new class of processors offers parallel processing and multi-thread architecture, the operating systems that have evolved over the past four decades are optimized to work with serial von Neumann stored program computers. While the need for a new OS to leverage many-core

architectures is acknowledged [1-5], the question remains: which one of the approaches being suggested will survive? The history of the evolution of current OSs is filled with lessons on wasted billions (does anyone remember Multics or OS2?), unmet expectations (who would have thought UNIX, the original System V, would vanish), surprise winners (Windows and Linux), and stealthy survivors (Mach in a Mac).

It is also true that the hardware upheaval is not matched by software advances [6]. For example, although hardware assisted virtualization technology, chip level management and high bandwidth wire-line and wireless networks have allowed applications or groups of applications to dynamically migrate from one location to another and become independent of underlying physical infrastructure, their management is still a slave to the underlying OS configuration, hardware and its management systems. This makes the service migration involving groups of applications completely dependent on the underlying OS images and associated configuration paraphernalia. The end result is that current IT infrastructure is overburdened with operation & management complexity that has to integrate a multitude of software systems from various vendors that manage computing, network, and storage resources independently.

There are three reasons why there is a new search for a different operating system strategy for many-core servers:

1. Application response time, unlike in a single-core server, is no longer a function of application software and the operating system that provides the computing, network, and storage resources. It depends on run-time workload fluctuations and latency constraints in a *shared* infrastructure. The wild fluctuations caused by massive consumer demand on web-based services place a heavy burden on the management of *shared* infrastructure. Conventional server-centric operating systems that have evolved from their single-core origins are no longer the

¹ Silicon Graphics Inc., uses Intel Xeon chips in its Altix systems to provide >256000 cores and 8 Petabytes of globally addressable memory in one huge container.

primary drivers of resource administration to influence the response time. The myriad intervening layered management systems have usurped their role.

2. The marriage, of server-centric OS security which, is mainly focused on isolating the shared resources allocated to different applications, with network-centric security which is, focused on isolating access to multiple servers, network devices and storage devices providing the required application resources, has resulted in a web of security management systems that require shared run-time resources along with applications. This has resulted in ad-hoc implementation of parallel application execution and application management workflows.
3. The evolution of single-thread operating system data structures to support multi-threaded operating system data structures has resulted, over time, in a large and complex code-base dealing with choosing correct lock granularity for performance, reasoning about correctness, and deadlock prevention. Resulting difficulties in extending the large-scale lock-based OS code² for multi-core systems is well documented [3].

It is becoming clear that from a datacenter point of view, the execution of applications and their management are two parallel processes that span across multiple geographical, physical infrastructure, and heterogeneous resource boundaries. The response time then becomes a function of application workloads, business priorities of various applications sharing the resources, and latency constraints at any given time, and depends on applications to spindle resource fault, configuration, accounting, performance and security management at run time. According to David Probert [7], perhaps a better way to deal with multiple cores is to rethink the way operating systems handle these processors. “The programs, or runtimes as Probert called them, themselves would take on many of the duties of resource management. The OS could assign an application a CPU and some memory, and the program itself, using metadata generated by the compiler, would best know how to use these resources.”

There are few parallel efforts that are underway to architect a new OS for many-core servers:

1. Tessellation [8]: It is predicated on two central ideas: Space-Time Partitioning (STP) and Two-Level Scheduling. STP provides performance isolation and strong partitioning of resources among interacting software components, called Cells. Two-Level Scheduling separates global decisions about the allocation of resources to Cells from application-specific scheduling of resources within Cells.

² It is estimated that the Redhat Linux 7.1 (2001) has about 30 million lines of code and Microsoft Vista (2008) has about 50 million lines of code.

2. Barrellfish [2]: It uses a multi-kernel model which calls for multiple independent OS instances communicating via explicit messages. Barrellfish factors the OS instance on each core into a privileged-mode CPU driver and a distinguished user-mode monitor process. CPU drivers are purely local to a core, and all inter-core coordination is performed by monitors. The distributed system of monitors and their associated CPU drivers encapsulate the functionality found in a typical monolithic microkernel such as scheduling, communication, and low-level resource allocation. The rest of Barrellfish consists of device drivers and system services (such as network stacks, memory allocators, etc.) which, run in user-level processes as in a microkernel. Device interrupts are routed in hardware to the appropriate core, demultiplexed by that core’s CPU driver, and delivered to the driver process as a message.
3. Factored Operating System (FOS) [3]: According to the authors, “FOS is a new operating system targeting many-core systems with scalability as the primary design constraint, where space sharing replaces time sharing to increase scalability. We describe fos which is built in a message passing manner, out of a collection of Internet inspired services. Each operating system service is factored into a set of communicating servers which in aggregate implement a system service. These servers are designed much in the way that distributed Internet services are designed, but instead of providing high level Internet services, these servers provide traditional kernel services and replace traditional kernel data structures in a factored, spatially distributed manner. fos replaces time sharing with space sharing. In other words, fos's servers are bound to distinct processing cores and by doing so do not fight with end user applications for implicit resources such as TLBs and caches.” They suggest redesigning traditional OSs using their approach for scalability.
4. Helios [5]: Helios is an operating system designed to simplify the task of writing, deploying, and tuning applications for heterogeneous platforms. Helios introduces satellite kernels, which export a single, uniform set of OS abstractions across CPUs of disparate architectures and performance characteristics. Access to I/O services such as file systems are made transparent via remote message passing, which extends a standard microkernel message-passing abstraction to a satellite kernel infrastructure. Helios retargets applications to available ISAs by compiling from an intermediate language. The authors compare their approach to Barrellfish as follows: Barrellfish focuses on gaining a fine-grained understanding of application requirements when

running applications, while the focus of Helios is to export a single-kernel image across heterogeneous coprocessors to make it easy for applications to take advantage of new hardware platforms”

5. Corey [9]: The authors argue that applications should control sharing: “the kernel should arrange each data structure so that only a single processor need update it, unless directed otherwise by the application. Guided by this design principle, this paper proposes three operating system abstractions (address ranges, kernel cores, and shares) that allow applications to control inter-core sharing and to take advantage of the likely abundance of cores by dedicating cores to specific operating system functions. Measurements of micro-benchmarks on the Corey prototype operating system, which embodies the new abstractions, show how control over sharing can improve performance. Application benchmarks, using MapReduce and a Web server, show that the improvements can be significant for overall performance: MapReduce on Corey performs 25% faster than on Linux when using 16 cores. Hardware event counters confirm that these improvements are due to avoiding operations that are expensive on multicore machines.”

All these efforts recognize the importance of application’s requirements in controlling the resources and provide a way to mediate between the many-core resources and fluctuating application needs. All these approaches implement application services and the resource mediation services using the same serial von Neumann SPC model.

A novel approach proposed in this paper takes a different route to leverage the parallelism offered by multi-core and many-core architecture to implement the service *management* workflow as an overlay over the service workflow implemented over a network of SPC nodes.

Figure 1 shows the Distributed Intelligent Managed Element Network (DIME) computing model. The DIME network computing model and resulting services architecture are derived from what we learn about the genes in living organisms and how they replicate, repair, recombine and reconfigure to create self-configuring, self-monitoring, self-healing, self-protecting and self-optimizing (self-*) distributed systems [10, 11]. The figure shows the DIME components which define self-management (FCAPS) features and a computing element MICE (Managed Intelligent Computing Element).

Each service contains two components:

1. A service workflow that is encapsulated as executable modules (which is described as a directed acyclic graph (DAG) and implemented using a single MICE or a network of MICEs belonging to a DIME network) and

The service regulator (which is similar to the meta-model proposed by Dave Probert mentioned earlier) that specifies a control workflow that is defined by the service component developer on run time behavior.

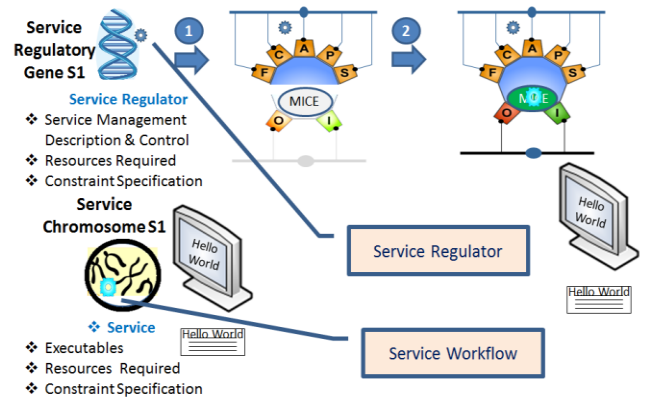


Figure 1: The DIME components and Service Specification

See <http://www.youtube.com/watch?v=KYIbH1P247w>

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 regulator. The DIME network architecture thus allows the description of the service to be separated from the execution of the service and 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 allows autonomy in controlling local resources and provide services based on local policies. Each node has to keep its state information and history of its transactions. 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 sub-network/network level FCAPS management based on group policies and execute a service workflow as a managed DAG.
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.

These features provide the powerful genetic transactions namely, replication, repair, recombination and reconfiguration that have proven to be very useful in cellular organisms [12].

As George Dyson, in his book ‘Darwin among the Machines,’ observes [13] “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 themselves. 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 well-managed one at that.

This paper describes the dynamic configuration, scaling, and fault management features implemented using the DIME network architecture (DNA) to implement a proof of concept on distributed multi-core servers and to evaluate its validity and shortcomings. In section 2, we describe the Parallax operating system implementation demonstrating its scaling, fault management and dynamic service regulation. In Section 3, we describe our conclusions and future research plans.

II. PARALLAX OPERATING SYSTEM AND THE SERVICE ORCHESTRATOR

Parallax is implemented using the assembler language at the lowest level for efficiency and provides a C/C++ programming API for higher level programming. Figure 2 shows the Parallax implementation schematic.

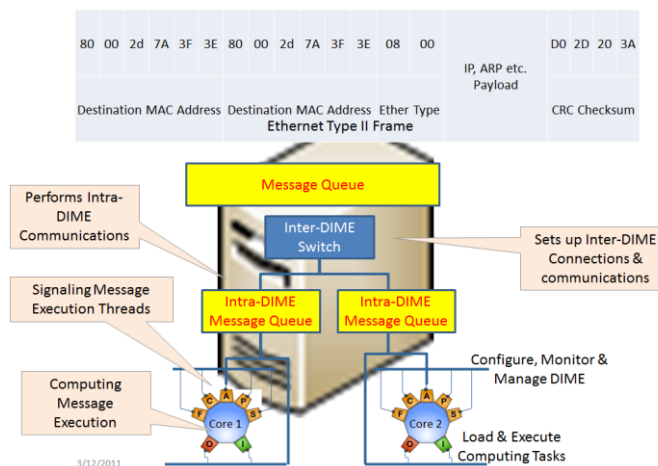


Figure 2: The Parallax OS Components in a Two-core Server

It is implemented to execute on 64-bit multi-core Intel processors. Each core is encapsulated as a DIME addressable as a network element with its own FCAPS management module. Each DIME has two communications channels supported by Ethernet, one for signaling and another for MICE I/O communications. The signaling Channel is used to execute

FCAPS commands and change FCAPS parameters at run time. The data channel is dynamically reconfigurable to set up inter-MICE communications, I/O paths and network and storage paths. This allows a composition scheme for creating a network of MICEs to execute a DAG very similar to using PIPE in UNIX but while the applications are running.

The kernel provides memory management, CPU resource management, storage/file management, network routing and switching functions, and signaling management.

Each DIME maps to different pages in the linear memory system and cannot access pages to which it is not assigned. Security is provided at the hardware level for this memory protection. Once a program has completed its execution, all memory that it had in use is returned to the system. Limits can be set on how much memory each DIME is able to allocate for itself. Memory is divided into a shared memory partition where the Parallax Kernel resides and partitions that are devoted to each core. Memory can be dynamically adjusted on each core on an as needed basis in 2 MiB chunks. Memory allocated to a DIME (core) can only be accessed by that DIME based on its security configuration. With dedicated resources, each DIME can be viewed as its own separate computing entity. If a DIME completes its task and is free, it is given back to the pool of available resources. The network management assures discovery and allocation of available DIMEs in the pool for new tasks. The signaling allows addressability at the thread level.

Parallax offers local storage per server (Shared with each DIME within the system) as well as centralized file storage shared via the Orchestrator between all servers. Booting the OS via the network is also a possibility for systems that do not need permanent storage or for cost saving measures.

Under Parallax, all network communication is done over raw Ethernet frames. Conventional operating systems use TCP/IP as the main communication protocol. By using raw packets we have created a much simpler communication framework as well as removed the overhead of higher-level protocols, thereby increasing the maximum throughput. The use of raw Ethernet packets has already seen great success with the ATAoE protocol invented by CoRaid for use in their network storage devices. Eventually, PCIExpress, and TCP/IP will be added along with Shared Memory.

Under Parallax, each DIME is addressable as a separate entity via the signaling and data channels. With the signaling layer, program parameters can be adjusted during run-time. DIMEs have the ability to communicate with other DIMEs for co-operation. The Orchestrator, from which the policies are implemented, communicates with the DIMEs for the purpose of coordination and control. Instruction types can be directly encoded into the 16-bit Ether-Type field of each Ethernet frame shown in figure 2. By making use of the EtherType field for specific purposes we can streamline the way in which packets are routed within a system. Packets can be tagged as Signaling/Data packets as well as whether or not they are destined for the overall system or rather just a specific DIME.

The proof-of-concept prototype system consists of three components:

1. A service component development program that takes assembler or C/C++ programs and compiles them to be executed on an Intel Xeon multi-core Servers
2. Parallax Operating System that is used to boot the servers with Intel Xeon cores and create the DIME Network with each core acting as a DIME. The DIME allows dynamic provisioning of memory for each DIME. It supports executing multiple threads concurrently to provide DIME FCAPS management over a signaling channel. It enables fault management by broadcasting a heartbeat over the signaling network. It allows loading, executing, and stopping an executable on demand. It supports DIME discovery through signaling channel.
3. A run-time service orchestrator that allows DIME network management.

Figure 3 shows the Proof-of-concept set up using three servers with Intel-Xeon processors where a DIME networks deployed and various features such as discovery, service scaling, fault management and dynamic reconfiguration are demonstrated.

The servers are booted with Parallax OS which sets up the hardware abstraction layer to allocate, CPU, memory, network and storage resources for each DIME and initialize the DIME FCAPS and MICE modules. In addition, the operating system provides the signaling and MICE I/O drivers including the Ethernet interface for communication over the network. The signaling channel and the Orchestrator FCAPS management commands are initiated. The operating system allows for each DIME Kernel level FCAPS management that is part of the bootable operating system. In addition, it also allows user level DIME FCAPS executables can be loaded and executed with appropriate security protocol to deploy user level FCAPS management.

The proof of concept scenario demonstrates the following:

1. Discovery of DIME nodes across multiple servers
2. Service deployment and replication in a workflow and
3. Service fault management using dynamic reconfiguration of the DIME network

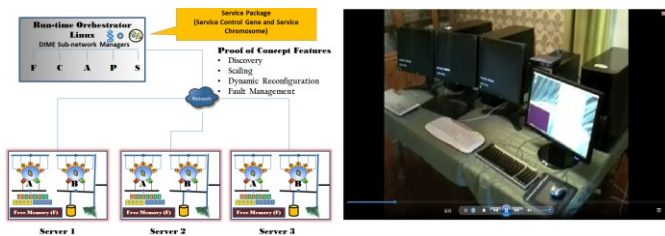


Figure 3: The Proof-of-Concept setup. A video of the demo is available at

<http://www.youtube.com/watch?v=y-0R-cRLfSk>

III. CONCLUSION

The impact of the *operating system gap* (the difference between the number of cores available in a server and the number of cores visible to a single instance of the operating system deployed in it) is dramatic when you consider current deployment scenarios. In one instance, a 500 core server is used as 250 dual core servers with 250 Linux images. In this case, in spite of proximity and high bandwidth, the TCP/IP based socket abstractions limit the performance by not utilizing the hardware resources and parallelism made available with new many-core architectures.

Many-core systems and cloud computing remind one of mainframes and time-sharing. What is different today is the abundance of computing power and bandwidth which change the way we use the resources in a profound fashion. The evolution of operating systems is reaching a new level where they are being challenged with the hardware upheaval and an opportunity presents itself where the von Neumann bottleneck potentially can be broken using parallelism supported in the hardware. In this paper, we have presented a proof of concept of a new generation operating system called Parallax that utilizes a novel extension to the SPC model by leveraging the parallelism and lessons learned from cellular organisms. The new computing model provided by the DIME network architecture (DNA) is implemented using a signaling control network overlay over a computing network consisting of computing nodes called MICEs which are programmed to provide self-management of fault, configuration, accounting, performance and security. In addition the signaling network overlay allows groups of DIMEs to be programmed to execute managed workflows as DAGs. The new approach provides four differentiators from other OS efforts:

1. The operating system encapsulates each core in a multi-core or many-core server with FCAPS management providing finer granularity for multi-tenancy down to a single core.
2. The operating system by providing a network management paradigm inside the server with many cores acting as nodes in a network enables visibility and control across all resources with uniformity. The same network management capability provides uniformity across servers and across geographical locations.
3. The signaling overlay and parallel implementation of service control based on service description at run-time provides the dynamism using the genetic transactions of replication, repair, recombination and reconfiguration.
4. The inclusion of service description, service regulation and service execution modules in the run-time service package provides the ability to implement global and local policies in the service workflow using network management.

By changing the computing paradigm, this approach complements many efforts to develop parallel language compilers and optimizers that utilize parallel processing at the core while addressing the von Neumann bottleneck [14]. The

signaling control network also addresses many of issues that concern distributed systems design [15, 16] such as racing conditions, starvation of resources etc. Traditionally, operating system takes a long time to get acceptance because of its central role in mission critical workflow execution. In spite of multi-million dollar investments from large corporations, Multics and OS2 did not survive. Even the unusual simplicity, power and elegance of UNIX operating system could not guarantee its commercial success. Linux is not derived from UNIX source code, but their interfaces are intentionally like UNIX and have enjoyed commercial adoption because of its free availability from Open Source. Windows operating system has borrowed freely many concepts from other operating systems before it, and has successfully penetrated both enterprise and consumer computing hardware. However, both Linux and Windows now suffer from their success and size and are not able to rapidly adapt to the many-core parallel computing [1, 3]. Trying to fit these operating systems in the new hardware is like fitting a square peg in a round hole.

New approaches are currently being addressed – some of them dividing the space and time taking the relativity theory route creating space-time continuum. Others take the route of multiple distributed kernels communicating with each other to provide resource management to match application needs in a distributed environment. This paper takes a different approach based on lessons learned from genes and chromosomes. As Mitchell Waldrop explains in his book on Complexity [17], “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 DIME computing model supports a service package where the service workflow is encapsulated as an executable DAG and is controlled by a service control executable managing the FCAPS of the computing network executing the service workflow. Parallax operating system implements the DIME computing model in a many-core system by creating a hardware abstraction layer for each core and converting it into a DIME which is self-manageable and network aware through signaling. The orchestrator and the associated service creation, packaging, delivery and assurance environment are designed to leverage current development environments using conventional operating systems to create the packages, deliver service workflows and assure service quality at run-time. Perhaps the role of current server-centric operating systems will be relegated in the future to service creation and workflow orchestration and the run-time operating system will be a network-centric OS such as Parallax implementing the DIME network architecture (DNA) providing service assurance.

Many network management lessons learned from POTS (Plain Old Telephone Service), PANS (Pretty Amazing New Services based on Internet) and SANs (Storage Area Network) in the past have to be incorporated in its implementation. Parallax has to be implemented in a many-core server to validate, FCAPS management with scaling and tested across multiple distributed servers. We see this effort as a first step in evaluating a novel and interesting approach.

IV. ACKNOWLEDGMENT

This research effort is not funded by any government agencies, universities, corporations or venture capitalists. It is solely developed by the sweat capital of a few individuals, inspired by the great masters who pushed the boundaries without the concerns of immediate commercial benefits.

V. REFERENCES

- [1] A. Baumann, S. Peter, A. Schüpbach, A. Singhanian, T. Roscoe, P. Barham, and R. Isaacs. Your computer is already a distributed system. Why isn't your OS? In Proceedings of the 12th Workshop on Hot Topics in Operating Systems, May 2009.
- [2] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schupbach, and Akhilesh Singhanian, "The Multikernel: A new OS architecture for scalable multicore systems", In Proceedings of the 22nd ACM Symposium on OS Principles, Big Sky, MT, USA, October 2009
- [3] D. Wentzlaff and A. Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. SIGOPS Oper. Syst. Rev., 43(2):76–85, 2009.
- [4] K. Klues et al. Processes and Resource Management in a Scalable Many-core OS. In HotPar10, Berkeley, CA, June 2010.
- [5] Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt, "Helios: Heterogeneous Multiprocessing with Satellite Kernels", ACM, SOSP'09, October 11–14, 2009, Big Sky, Montana, USA
- [6] Vijay Sarathy, Purnendu Narayan and Rao Mikkilineni, "Next Generation Cloud Computing Architecture - Enabling Real-time Dynamism For Shared Distributed Physical Infrastructure", Proceedings of IEEE WETICE2010-Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on , p48-53.
- [7] Joab Jackson, "Multicore requires OS rework, Windows architect advises", IT World, IDG Network 2010 (<http://www.itworld.com/hardware/101580/multicore-requires-os-rework-windows-architect-advises?page=0%2C1&source=smlynch%22>)
- [8] Juan A. Colmenares, Sarah Bird, Henry Cook, Paul Pearce, David Zhu, John Shalf, Steven Hofmeyr, Krste Asanovic, and John Kubiatowicz. In Proc. 2nd USENIX Workshop on Hot Topics in Parallelism (HotPar'10). Berkeley, CA, USA. June 2010.
- [9] Ong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, Zheng Zhang, "Corey: an operating system for many cores", Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation OSDI '08, San Diego, California, December 2008.
- [10] Rao Mikkilineni and Ian Seyler, "Parallax – A New Operating System for Scalable, Distributed, and Parallel Computing", The 7th International Workshop on Systems Management Techniques, Processes, and Services, Anchorage, Alaska, May 2011 (Accepted)
- [11] Rao Mikkilineni, and Giovanni Morana "Is the Network-centric Computing Paradigm for Multicore, the Next Big Thing?" <http://computingclouds.wordpress.com>
- [12] Maxine Singer and Paul Berg, "Genes & genomes: a changing perspective", University Science Books, Mill Valley, CA, 1991, p 73
- [13] George B. Dyson, "Darwin among the Machines, the evolution of global intelligence", Helix Books, Addition Wesley Publishing
- [14] David Patterson, "The trouble with multi-core", IEEE Spectrum, July 2010, p28
- [15] G. Couloris, J. Dollimore, and T. Kinberg, Distributed Systems - Concepts and Design, 4th Edition, Addison-Wesley, Pearson Education, UK, 2001.
- [16] A. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, Prentice Hall, Pearson Education, USA, 2002.
- [17] Mitchell Waldrop, M., "Complexity: The Emerging Science at the Edge of Order and Chaos", Penguin Books, London, 1992, p218.