Applying Amdahl's Other Law to the data center

D. Cohen
F. Petrini
M. D. Day
M. Ben-Yehuda
S. W. Hunter
U. Cummings

As computing system workloads become more distributed in nature, there is an increasing dependence on the networking interconnects between such systems. As stated by Amdahl's Other Law, this dependence not only exists on the I/O (input/output) subsystem, but also on the memory subsystems. In particular, as processor utilization increases, there is a direct, corresponding increase in memory and I/O utilization. At a broader level, the distribution of workloads is driving the need for computing based on locality (or pods) to achieve the appropriate balance of compute, network, and storage resources. This paper studies the applicability of Amdahl's Other Law to the data center to better understand the relationship between processor systems and the networks interconnecting them. This study is also relevant as multicore systems will become more prevalent to sustain growth of processing performance.

Introduction

Computer scientist Gene Amdahl is associated with many rules in the field of computing. His most famous rule involves his argument that the speedup realized from system parallelism is limited by the amount of sequential work done by the application [1]. Amdahl also argued for system balance. Referred to as Amdahl's Other Law, he specified this design principal in the late 1960s, stating that for an efficient computing system there must be a balance between the platform clock speed, capacity of main memory (i.e., random access memory, or RAM), and the bit rate (in seconds) of the I/O (input/output) bandwidth. If any one of these three resources becomes constrained, a computation will be forced to wait.

It is generally accepted that computing systems now and in the future will be built using multicore processors. What is less clear is the impact this shift has on the host memory hierarchy and the constraints it imposes on system designers as they strive for system balance with respect to processor speed, memory, and I/O bandwidth. These multicore processing systems will have independent, parallel threads of execution, forcing designers to address various issues involving concurrency and scalability within the system platform, the system rack, and the data center as a whole. This further complicates the platform designer's effort to achieve system balance. They must now account for independent threads contending for system resources (e.g., memory and I/O resources).

The scalability of computer systems is affected by multiple factors including the central processing unit (CPU), memory, I/O, and communication [2, 3]. Methods for achieving scalability with concurrency include symmetric multiprocessing (SMPs), loosely coupled computing clusters, and massively parallel processing (MPP) systems. An attribute that distinguishes between these three system architectures is the level of data coherency maintained. For example, an SMP machine executes a single operating system (OS) while maintaining coherency across the local caches of each processor. Loosely coupled computing clusters have little, if any, data coherency maintained across compute systems interconnected across a networking fabric. MPP systems may be viewed as residing between these two approaches, such that coherency may be realized across large numbers of interconnected compute systems often with a high-performance networking fabric and appropriate low-latency communication protocols. For example, workloads running on high-performance, cluster-based platforms frequently employ the message passing

0018-8646/09/$5.00 © 2009 IBM

interface (MPI), combined with remote direct memory access (RDMA) as the transport method. The concept of computing based on locality (or pods) is introduced as a model for making use of these architectures and technologies to achieve the appropriate balance of system resources in the data center.

This paper focuses on locality-based computing, multicore processors, the need for system balance, techniques for scaling compute systems, and the impact these have on enterprise data centers. In fact, the data center as a whole could be viewed as the compute system. Organizations have substantial investments in computer-related hardware and the software that runs on it. The composition of these investments will evolve with incremental changes to exploit new designs and maintain the appropriate system balance. Pragmatically, the vast majority of existing software is sequential and cannot take full advantage of multiprocessor platforms. Thus, although industry support of parallel application development will eventually be required, for the foreseeable future companies will take advantage of multicore platforms using a variety of virtualization techniques [4].

Motivation

Advancements in processor, networking, and storage technologies are leading system designers to re-think how compute resources should be deployed. That is, as networking bandwidth improves, system workloads and services are becoming more distributed across multiple computing systems, increasing the dependence on the network infrastructure. Similarly, enhancements in storage technology are encouraging a multitiered hierarchy across the I/O network. Also, multicore processing will continue to increase processing capability at the processor socket level, which will drive increases in both memory and I/O bandwidth. Finally, while virtualization will assist in transitioning through some of these technological shifts, hypervisors are also known to further drive memory bandwidth requirements. If not addressed, the combination of these advancements will lead to an imbalance in compute resources within the data center.

Thus, our task is to answer two questions: 1) How can the effective memory and I/O bandwidth of a single thread be increased, and 2) what can be done to minimize contention while maximizing concurrency? Of course, answers to these questions must reflect the broader, distributed nature of computation, both for communication between threads on a single host and for off-host communication.

The memory hierarchy

Densification is the process of increasing the number of software-based computing systems running in a fixed amount of space and time. Recently, this trend is reflected in the move to system rack mounts and blade servers. These are now being fitted with multiple processor sockets and multi-processor cores. Not surprisingly, this process is forcing system designers to focus on concurrency and access to shared resources.

The implication of a conceptual model of a computer involving processor, RAM, and I/O is a memory hierarchy that consists of three tiers: register, memory, and disk. This hierarchy involves the path used to shuttle data and instructions in and out of the processor. The tiers represent a tradeoff between access time and the cost of memory space; the closer data resides to the processor, the faster the access time and the more expensive the space. On a multicore platform, with several processes operating in parallel, this shuttling of data depends strongly on available bandwidth, both memory and I/O. The emphasis must be on minimizing contention for these resources.

From this perspective, the performance of a workload running on a balanced system will be limited by the most "expensive" component of the platform. This has led to a general focus on memory and the introduction of multiple levels of caches between the processor and RAM. The first levels, level 1 (L1) and level 2 (L2) caches, reside on the processor, while the level 3 (L3) cache typically resides on the motherboard. The L1 and L2 caches are dedicated to the processor and provide slightly more space, but they are slightly slower than processor register accesses. At the next level of the hierarchy is the L3 cache. It increases the amount of space at the cost of slower access times. Since it resides on the motherboard and is accessible by multiple processors, it also introduces the need to consider the effects of concurrency [5, 6].

A large change is anticipated with respect to industry-standard servers in the area of memory and I/O bandwidths. These servers will support non-uniform memory access (NUMA). In this architecture, the adjacency of a processor socket to memory slots on the motherboard has an impact on its access to data. This, in turn, will bias processor cores toward accessing data in directly attached memory slots. Managing coherency of the L1 and L2 caches will be a key requirement. The shift to NUMA has also forced a redesign of the I/O subsystem so that it interfaces directly with the memory subsystem. This reduces the access time and increases bandwidth by removing an intermediate switch hop in the memory controller hub.

Increasing the bandwidth of the interface between memory and I/O means that the I/O subsystem bandwidth to external components must also increase. This is represented by a shift to the Peripheral Component Interconnect (PCI) next-generation design, referred to as "PCI Express** gen2." This will provide

substantially more I/O bandwidth. In isolation, however, there is still a need to consider the contention inherent in a multicore system.

The I/O bus must somehow work in conjunction with system virtualization technologies to allow multiple OSs, or containers, to run simultaneously within a single computer to natively share PCI Express devices. This will be accomplished using Single Root I/O Virtualization (SR-IOV), which focuses on single root topologies (e.g., a single computer that supports virtualization technology).

In the SR-IOV model, each virtualized system has one or more virtual PCI Express devices, referred to as virtual (PCI) functions (VFs). For example, a VF can be a virtualized Ethernet network adapter. A guest VM (virtual machine) using a VF is given a direct path (e.g., DMA) to the physical device and is assigned some priority in accessing that device. The implication is that the I/O adapter is providing some switch-like capability. The impact and opportunity of this is addressed in more detail later in this paper (see section "Hypervisors and I/O Scheduling across Domains").

The storage hierarchy

The data being shuttled into and out of these computers and the virtualized systems they host will ultimately reside on some type of persistent storage. This storage device consists of a head unit containing CPUs and RAM attached to disk drives, in the form of a storage array or a collection that is referred to as "just a bunch of disks" (JBOD). At one end of the spectrum, this is a single host, in which case the storage is referred to as "direct-attached storage" (DAS). At the other end, the storage head is connected to a network. This may be a block-based, storage area network (SAN) or file-based, network-attached storage (NAS).

Regardless of how data is accessed, a storage array has traditionally focused on providing what is referred to as "aggregate disk throughput." Data is striped across multiple disks to increase the overall bandwidth to the system. A storage engineer develops a solution based on particular workload capacity requirements (considering such questions as "how much data needs to be on disk?") and performance requirements ("how much data needs to be delivered and how fast?"). Understanding a workload capacity requirement involves a simple calculation in which the required capacity is divided by the single disk capacity. Answering the second question is more complicated.

A workload performance is limited by the number of disk drives available to it, not the amount of capacity. Performance is a function of the number of disks, a single disk access time, and the number of I/O operations that disk can deliver per second (IOPS). Access time represents the length of time it takes to move the disk head to where the data resides on the platter. This is also a function of the spin rate of the drive. A mechanical disk drive, spinning at 15,000 revolutions per minute (rpm) delivers 250 IOPS, representing the amount of data the drive can deliver at a given moment.

The introduction of solid-state drives (SSDs) as a viable alternative to mechanical drives represents a pivotal point in technology. Rather than an enhancement to storage, SSDs represent a new tier in the memory hierarchy. This is a level 4 (L4) cache that resides between host memory and I/O. An SSD is capable of delivering 50 times the performance of the fastest mechanical drive. This affects the storage engineering considerations in that a relatively small number of SSD devices will be combined with large capacity, Serial ATA (SATA) drives. The mix will be dependent on the tradeoffs between price, performance, and capacity [7–9].

In parallel, new storage products are becoming available on the market. These deliver exceptional performance and high capacity at a very low price. It is now possible to purchase a fully resilient storage array with more than 40 TB of usable capacity, which delivers more than 100,000 IOPS and occupies less than 10 RU (rack units) of data center rack space. For many enterprise workloads, it is very appealing to directly connect this storage to servers or share it among hosts on a local area network (LAN). Of concern, however, is how to ensure data on these devices can be backed up and, more importantly, restored.

Convergence of messaging, storage, and backup networking

The composition of the data center has evolved, progressing from a container of a small number of centralized, monolithic mainframe systems to one dominated by thousands of small form-factor rack-mounted blade servers. The implication of this evolution is that computers have become increasingly interconnected, resulting in a significant network inside the machine room, requiring costly network services in order to scale such resources. As the prominence of these smaller, modular computers has risen, their capabilities have also increased. Today, an enterprise-class x86 server shares increasing characteristics with a mainframe (e.g., increased memory and I/O capacity and parallelism through SMP).

However, so far, enterprise storage infrastructures have not entirely followed the trend toward decentralization. The case for centralized storage is reinforced by requirements such as backup and recovery, resiliency, and regulatory and compliance needs. This data protection infrastructure benefits from natural economies of scale associated with being centralized. Recently, the economic burden of attaching an increasing number of centralized

modular servers to the network is shifting the balance in favor of decentralization.

The cost of attachment to these networks has led to the need for convergence because the current two-tier, fan-in model of host access to data is increasingly expensive; the more systems accessing the centralized storage, the more bandwidth that is needed. Here, we use the phrase *fan-in model* to denote a many-to-one model of sharing such that some number of server systems are sharing a single centralized storage. Network cost is increased by connections (both number of instances and fan-out of the clients), protocol extent (i.e., the physical distance associated with a networking protocol), and bandwidth. Notably, except for bandwidth, these considerations also increase access latency. Obviously, there is a limit to scaling a monolithic resource for host access to data.

While it is exceedingly difficult to make the storage resource completely distributed, it is natural to cache or tier central resources to reduce the bandwidth and fan-in of accessing central resources. The platform shift to multicore processors, the use of NUMA, and increased host bandwidth will further motivate the need to increase the access to this data. This distribution means that application software and data must be deployed to these networked systems while the organization or company retains control over the integrity of this data.

The shift to decentralization emphasizes data diffusion and locality-based computing. This paradigm is predicated upon changes in the enterprise data center network and the introduction of an intermediary caching tier. Together, these work to reduce the latency between communicating nodes by optimizing the proximity of participants in the communication. By allocating applications in a rational manner, the bandwidth needed in the network core is reduced [10, 11].

Data center networks

Data center networks employ a hierarchy of switches in order to scale capacity (i.e., bandwidth). The original assumption, from which evolved this hierarchical network model for providing sufficient bandwidth for centralized data, is that storage and data protection infrastructure are centralized, while hosts are increasingly decentralized. When viewed in the aggregate, communication between this centralized storage infrastructure and hosts accessing data that resides on that infrastructure has a fan-in pattern.

To accommodate this pattern, network designers scale the capacity of the communication channel at the network core. This approach to scalability employs a layered architecture that calls for hosts to attach to a switch at a bottom layer, or *access layer*. These are then aggregated via an intermediate, or aggregation, layer. Switches in this middle layer then attach to the "spine" of a network, or core layer switches. This top layer is used to scale the bandwidth needed to support access to the storage and data protection infrastructure of a data center.

This approach is becoming untenable for several reasons. First, the bandwidth of a host I/O adapter, which connects to the access layer, has risen from 10 Mb/s to 100 Mb/s to 1 Gb/s in the span of the past ten years, and it is expected that new servers will come equipped with a dual-port 10-Gb/s I/O adapter on the motherboard. For a host to take advantage of this expanded I/O capacity, the capacity of the network must also be expanded.

Second, this economic burden is increased as the rate of host attachment increases. As more hosts come online, there is an increase in demand for network capacity, increasing the network utilization. This increases the need to expand the network. Third, the core-level switches are predominantly proprietary, with ASIC (application-specific integrated circuit)-based processors and custom OSs. With limited availability, suppliers have been able to demand a premium for this class of switch.

Finally, the current model emphasizes bandwidth over latency (i.e., the time it takes for a processor to access data needed to complete a computation). Latency is a function of distance and the number of switches and other network devices that must be traversed as a result of the access. At any of these network devices, there may also be congestion that results in some form of buffering that further increases the latency.

Architecture Based on Clos-based fabrics

In response, the Clos architecture is emerging as a viable alternative for enterprise data center networks. This is a scalable switch stage that is implemented via a cascade of increasingly smaller switches. These switches are arrayed in a butterfly network. While the original Clos concept is non-blocking with packet reordering, modern Clos fabrics are statistically fully provisioned and maintain packet ordering within flows. Similar to the trend of arrays of commodity processors, a Clos fabric reduces cost by making use of commodity volume switches [12, 13].

A Clos fabric can have local switching at each tier. This allows a balance of oversubscription from the access layer to the spine (or core). Furthermore, a Clos can be implemented with different switches at each tier, allowing performance-optimized switches in the access layer and service-optimized switches in the core.

This is leading to a design in which a relatively small number of hosts are connected directly to a set of aggregation dedicated switches. In this design, the access layer resides on each computer. The aggregation switches of the pod serve as the second stage of the Clos. The core

network of the data center is at the top or spine of the Clos. Note that the access and top-of-rack layers are either fully subscribed or undersubscribed, while the core is oversubscribed.

Over time, access layer switching grows at an independent and faster rate than the second layer in the top-of-rack switch, which is also growing at a faster rate than the core. The growth of the access layer switching function is evidenced in the switching logic of the hypervisor, the queuing (and scheduling) features in SR-IOV network interface cards (NICs), and the introduction of small Ethernet or multi-root I/O virtualization (MR-IOV) PCI Express switches on individual compute elements [14–16].

File system caching tier

In the context of a single computer, a cache is generally defined as a storage medium located in between the main memory and the host computer processors [16]. When scaling beyond a single host, however, a different type of cache is employed. This cache, or rather series of caches, resides in between a host main memory and external storage. If that storage is locally attached, then this cache is simply the page cache. On the other hand, if the storage is remote, then this is a file system cache (or series of caches). One tier is local to the host and one tier is local to the network distribution layer [17, 18].

The caching tier that resides on the host is a client-side cache that caches data locally. The mechanism trades disk space to gain performance by avoiding access to data over a slow network. The second tier of caching resides in the network, between the host and a distributed file system. This is an intermediate proxy cache that serves to reduce utilization at the core level of the network.

The proxy cache employs a technique referred to as data shipping. File access is partitioned across nodes in a cluster or parallel file system. File blocks are assigned to nodes in a round-robin fashion, so that each data block will be read or written only by one particular node. Read and write operations originating from other nodes are forwarded to the node responsible for a particular data block. This communication is performed over a high-speed channel that is private to the cluster [19–21].

Data always originates from a centralized, authoritative source. The implication is that the intermediary and host-side caches hold replicas of the data. If the original data is changed, then there must be some model for keeping the replicas up to date. Changes to the original data are eventually reflected in the caches. In the case of read (and metadata) operations, this consistency model applies one or more policies to the file system metadata. If a replica in a cache is inconsistent with the origin, the replica is ejected and the newer data retrieved and sent to the cache [22–25].

The cache is primarily concerned with the consistency of the replicated data it contains and the authoritative source of that data. The implication is that instances of the cache communicate with the centralized source. This consistency model is tunable between the characteristics of synchronous (i.e., always consistent) and asynchronous (i.e., consistent over some specified time interval).

For read operations, the cache simply contains replicas. For write operations, the cache supports write-through and asynchronous semantics. The former simply writes data back to the centralized, authoritative source, retaining a local replica as part of the operation. When configured for asynchronous write operations, data is written to the persistent store of the cache and subsequently migrated back to the centralized, authoritative source. Again, a local replica is retained in the cache as part of the operation.

Locality-based computing

Locality-based computing is an enterprise IT model that attempts to harmonize two primary and conflicting goals. The first goal involves moving data as close as possible to the compute resources manipulating that data. The second goal involves the minimization of the movement of data by preferring local compute resources over remote ones.

Locality-based computing minimizes the movement of data by placing the different processes of a distributed application as close as possible to each other, within the same physical computer if possible. If this is not possible, processes are placed in the same rack. To formalize the goal of minimizing data movement, the pod concept is useful. A pod is a collection of all the resources needed by a distributed application within the same network segment and shared switch, augmented by storage caching appliances.

By introducing a pod, this architecture combines the Clos network design with the two-tier, distributed file system cache. This is a unit of deployment composed of distribution layer switches and a set of host computers. The majority of these hosts are dedicated to running a hypervisor. The remaining hosts serve as the proxy cache of the pod and the hypervisor implementation is extended to serve as a client-side cache.

This pod-based configuration is engineered to accommodate workload demand for data and reduce the amount of time these workloads spend waiting on I/O. The idea is to stage, or cache, more frequently accessed data (and application-related software elements) within the partition of the network distribution layer where the workload is running.

As depicted in **Figure 1**, locality-based computing considers the memory hierarchy at all points in the system: Data needed for computation should be fetched

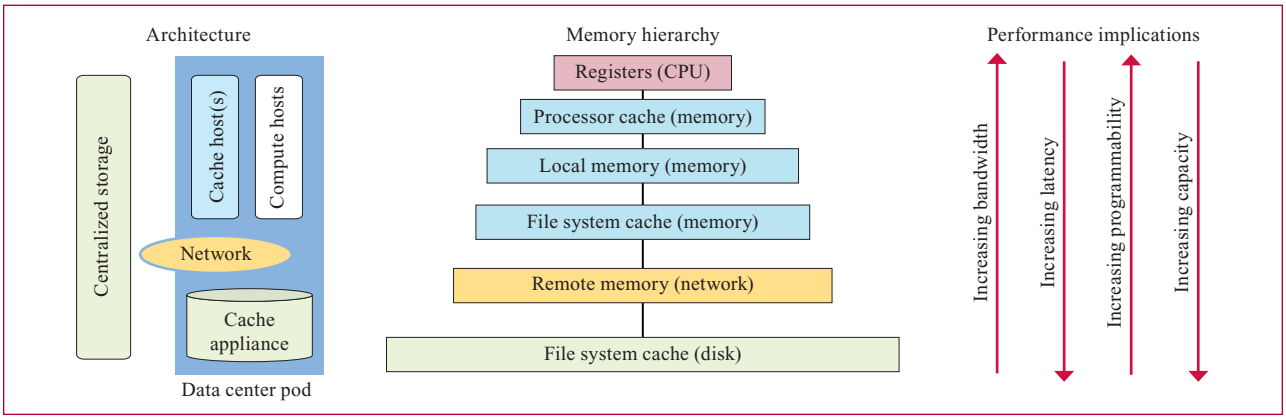

Figure 1

Positioning a pod within the memory hierarchy.

from storage over the network. Once fetched, that data should remain in the local secondary storage cache. The networking hardware should place the fetched data in the main memory of the computer close to the processor and core that will manipulate that data. The core performing the manipulation should cache the data within its local private cache and keep it there for as long as possible. The OS (and by extension the hypervisor) should keep a computational job running on the same core to maximize the benefits of local caching.

Data residing on this network is transient. Long-term persistence and data protection (i.e., archival and backup/recovery) are centralized and external to a pod. The implication is that some form of data movement, for example, policy-based data life cycle or hierarchical management technologies, is used to move data into and out of a pod.

This model is depicted in **Figure 2**. Here, the data protection infrastructure resides at the center of the data center and is accessible over the core layer of the network. One or more pods are deployed, each isolated by a set of distribution layer switches and containing a caching appliance. The data protection infrastructure and the caching appliances share a single, global file system namespace. This allows for a distributed policy engine that carries out replication and migration policies of an organization [26].

Next steps and challenges

The blueprint for locality-based computing calls for the distribution and access layers of a data center-wide Clos network to reside within a pod along with a tier of file system caches. Workloads are then dispatched on guest systems running within the pod. While technically feasible today, there are a few challenges that need to be overcome before this vision can be fully realized. This section highlights what exists, what needs to be developed, and which problems are open research problems and which problems are likely to be solved simply through implementations. In essence, we provide a model for how to build a pod.

Optimized communication protocols

To date, the performance of system virtualization has been limited by the I/O subsystem. This has largely been a function of the I/O subsystem and the duality of the I/O schedulers associated with the hypervisor and the guest OS. In this model, the hypervisor is interposed with the I/O adapter and the guest. Packets and related interrupts must be processed by the hypervisor I/O scheduler. The result has been that, even under ideal conditions, a virtualized system struggles to saturate a gigabit Ethernet link.

Two separate problems exist. First, if we have all of the I/O transactions pass through the hypervisor and the software IOMMU (I/O memory management unit) function, then this leads to a significant increase in I/O latency and reduction in bandwidth. Second, a lack of rational coordination between the guest I/O scheduler, the hypervisor I/O scheduler, and the aggregate effect of hypervisor I/O schedulers across machines leads to nondeterminism and poor performance of any particular guest.

The access layer switching function of the new platform operates in conjunction with the second layer switching function, forming the bottom two layers of the Clos topology. As mentioned, the key is that these layers are undersubscribed (or fully subscribed) while the top layer is oversubscribed. This places a significant load on the storage subsystem, both on the local host and on those

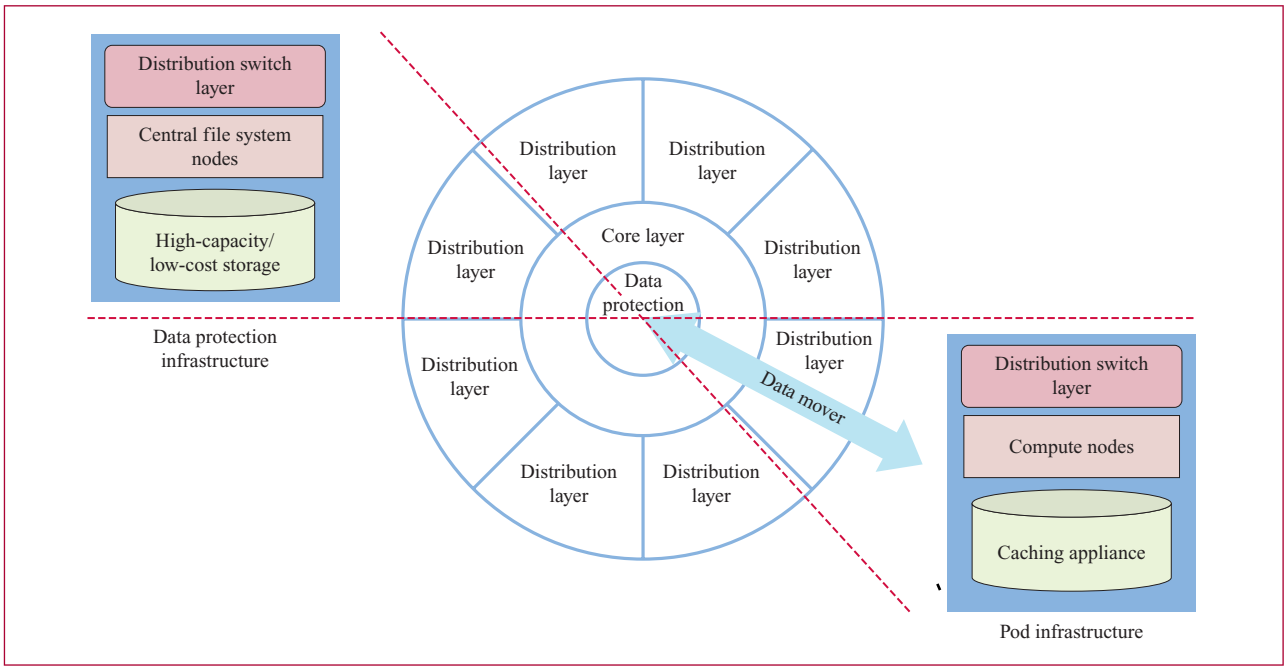

Figure 2

Overview of a pod within the data center.

that are accessed via the network. If this load is not addressed, the performance gains of the new platform will be squandered as guests spend an increasing amount of time waiting on I/O.

Pod-based communications

There are three patterns of communication guests-based application software running within a pod. In this context, the virtual NIC (vNIC) of a guest can be configured to use virtual LAN (VLAN) tagging as an alternative to MAC (Media Access Control) addresses. In this case, hardware support in the adapter binds an incoming packet into a receive (RX) queue based on a VLAN tag instead of a MAC address. On the transmit (TX) side, the adapter can tag the packet with a VLAN prior to forwarding. Source address anti-spoofing, both networking layers 2 and 3, is also supported.

Communication with a domain external to the pod

The first of the three communication patterns is associated with how an application communicates with another application that is external to the pod. In this case, messages exchanged between the two end-points traverse the pod distribution switching layer and the core network of the data center. **Figure 3** depicts the remaining two patterns of communication in which message traffic remains within the boundaries of the pod.

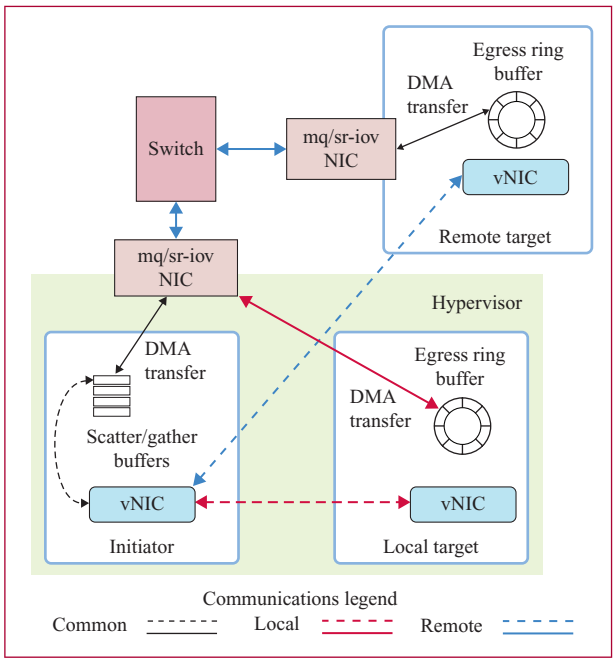

Figure 3

Overview of a virtual network interface card (vNIC). The dashed arrow lines indicate peer-to-peer communication. (DMA: direct memory access; mq: message queue; SR-IOV: Single Root I/O Virtualization.)

Communication between domains on the same compute element

The second of these communication patterns is between guests on the same compute element. As depicted in Figure 3, messages flow between the initiator and the local target. The NIC of the compute element serves as the access layer switch. Its switching capabilities are employed to identify that a packet transmitted from one particular guest is destined for a guest on the same hypervisor.

The packet is looped back internally without having to go to the next level of switch. This operation is based on the destination layer 2 address of the packet, expressed as {MAC, VLAN}. Multicast packets are looped back and potentially are delivered to all local vNICs on same VLAN. Optionally, hardware filtering can be applied via the destination MAC. This feature can be exploited to enable an efficient intra-node MPI. This concept can be generalized to the pod or access layer, provided there is a binding to an inter-node MPI stack for communication that needs to leave the pod.

Note that using Ethernet hardware to transfer data among domains residing on the same host is counter to the prominent virtual-DMA method used by hypervisors today. Most hypervisors implement an internal virtual network switch over a shared memory medium. All guest-to-guest communication goes over the virtual switch. Messages are copied from one physical memory address to another by a host CPU (see the section on memory and the access layer switch).

Communication between domains on different compute elements

The third communication pattern is between two guests hosted by independent compute elements within the pod, for example between a guest and the caching appliance of the pod. In this pattern, the distribution and access layer switching tiers of the pod collaborate in order to ensure quality of service. This collaboration motivates changes in the I/O scheduling mechanism of the compute element as well as enhancements to the Ethernet protocol.

Hypervisors and I/O scheduling across domains

Figure 4 depicts the model of the I/O scheduling in the hypervisor. Note, however, that when this paper was written, the hypervisor was situated between guests it hosted and the I/O subsystem of the computer. All I/O passed through the hypervisor. When the hardware adapter is functioning as a switch, it does all I/O scheduling within its own networking domain. That is, the hardware I/O adapter does all the scheduling previously done in the hypervisor, leaving the hypervisor to handle intra-domain scheduling.

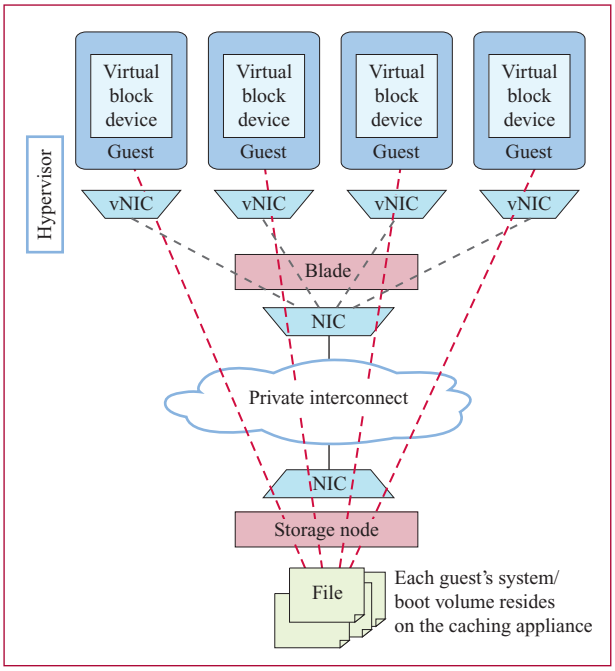

Figure 4

Distribution and access layer switching in the pod. The red dashed lines indicate the logical paths between a guest and the storage.

The hardware adapter is responsible for scheduling packets associated with different domains. All ingress traffic is processed at wire speed, meaning packets are forwarded on arrival. During transmission, the hardware scheduler handles scheduling across the TX queues, forwarding packets to their destination. The scheduler of the adapter has configurable features such as QoS (quality of service) with respect to priority and bandwidth (e.g., maximum bandwidth and best effort). Both relative and absolute allocations on bandwidth are supported. The role of the hypervisor/OS in this model is to configure the adapter's scheduler and monitor its hardware counters.

The I/O resources (in terms of priority and bandwidth) are allocated to a vNIC when it is created, and from that point on, the domain can access hardware directly without the hypervisor being in its way. Once a domain has registered its vNIC with the adapter, the adapter forwards interrupts directly to that domain via MSI-X (Message Signal Interrupt Extension). Interrupt and packet processing are handled directly by the domain with no interference from the hypervisor or OS.

Enhancements to the Ethernet protocol

In terms of Converged Enhanced Ethernet (CEE) features, the feature most commonly supported by the

new adapters is *per-priority-pause* (PPP). In addition, many of the adapters provide early (pre-standard ratification) support for congestion notification including some form of rate limiting and filters to trap congestion notifications. This latter feature is dependent on packet format being standardized. Finally, the adapters may support enhanced transmission selection that schedules priorities and converged types of service groups across each port.

As far as support for PPP, there are three bits in an Ethernet frame for pause support. This allows up to eight queues to be addressed directly; however, most of the adapters have more than eight queues. This leaves the scheduler of the adapter responsible for mapping the packet to the appropriate queue and applying pause as required. Each queue, RX and TX, is associated with a priority, with *best effort* being the default. A particular queue may have "pause" enabled.

Care needs to be taken to separate or segment a queue with pause enabled from other classes of traffic. For example, if Fibre Channel over Ethernet (FCoE) and remote DMA (RDMA) transport protocols are to be supported, these should be assigned a dedicated vNIC. Using this approach, pause can be enabled without affecting or having an impact on other traffic flowing in and out of that domain.

For this type of specialized traffic (e.g., FCoE and RDMA), there is a perceived need for encoding information in the frame beyond what is available via VLAN tagging or MAC address. The desire here is to be able communicate the identity of a domain between the NIC and the first-level switch without using MAC addresses and without overwriting any important information in the packet.

As mentioned, a new alternative to using MAC addresses or VLAN tagging has been proposed. Called VN-Tagging, this proprietary approach has elicited a great deal of controversy. The debate centers on the deficiencies of the standard VLAN/MAC approaches and whether these are sufficient. Widespread consensus suggests that there is no need for a new packet format or tagging method.

Memory and the access layer switch

Using physical DMA instead of a virtual shared memory switch has several key advantages when used with SR-IOV-capable adapters. Typical advantages of using physical DMA compared to virtual DMA to move data among domains on the same physical host include the following:

- The host CPUs can perform other work while the data is being moved by the I/O adapter, mitigating the effects of Amdahl's Other Law.
- The hypervisor interferes less with the guest I/O scheduling strategy than it does when using virtual DMA.
- Multi-queue and self-virtualizing I/O hardware allows linear scaling, whereby physical queue-pairs can be allocated per vNIC. With appropriate under-provisioning of switch capacity, DMA transfers can occur in parallel. This enables the multichannel QoS strategies described earlier.
- There is symmetry in the latency, throughput, and CPU utilization for transfers that remain within the pod regardless of whether they stay on the same physical host.

Host CPU may continue with non-I/O related work

When transferring data between two domains on the same physical host, physical DMA moves the data using I/O processors on the NIC. This mostly frees the CPU to continue performing work that is not related to I/O transfers. The CPU does need to do some work, however, as listed in the following steps.

1. A CPU for the target domain needs to work with the hypervisor to program the IOMMU of the server with a translation table entry that enables DMA into the memory of the target domain.
2. A CPU for the initiating domain needs to program the NIC to perform the DMA transfer. This includes marshalling the data into a scatter-gather buffer set and using the device driver of the NIC to set up the DMA.
3. A CPU for the target domain needs to process the incoming interrupt event that signals a DMA. This probably involves moving the received data out of the egress ring buffer and maintaining ring buffer pointers.
4. A CPU for the target domain may reprogram the IOMMU to remove the translation table entry that allowed the DMA to occur. This is an optional step that depends on the IOMMU strategy of the hypervisor and domain.
5. A CPU for the initiating domain needs to clean up after the transfer. This involves asynchronously destroying or reinitializing the scatter-gather buffer and cleaning up any other resources associated with the transfer.

As the steps above show, the domain CPUs are not entirely relieved of work even when using physical DMA. In fact, it may be the case that for many very small (cache-line size) transfers, physical DMA is more CPU intensive than virtual DMA. For all other cases, however,

the "housekeeping" associated with physical DMA is easily amortized across all the bytes that are moved by the NIC hardware, and the result is that the CPUs of the domains can continue processing data while high-throughput transfers are proceeding. The payoff is that Amdahl's Other Law may be addressed even for high-throughput applications.

Hypervisor interferes less with I/O scheduling

Physical DMA among domains mostly removes the hypervisor from the I/O scheduling process. Using virtual DMA, the hypervisor is responsible for transferring the data from the memory of one domain to that of the other. Using physical DMA, the hypervisor does not perform this step. I/O scheduling is totally under the control of the initiating domain and also mostly under the control of the target domain.

Once the initiating domain sets up the DMA transfer (Step 2 above), the NIC takes control and moves the data independently of the hypervisor. The data will go to the target even if the hypervisor de-schedules the initiating domain. This is a key point because this feature preserves the I/O scheduling policies of the domain.

On the target domain, the hypervisor has a minor influence in the delivery of the received data to its ultimate destination. Before the target domain can service its egress ring buffer, the hypervisor must receive the device interrupt of the NIC and deliver an interrupt event to the target domain. Interrupt events are usually delivered whenever the hypervisor reschedules a domain CPU. At that point, the target domain performs the steps outlined in Step 3 above.

The latency at the target domain introduced by the hypervisor will be at least one scheduling interval (for a domain with the number of virtual CPUs equal to the number of physical CPUs) and potentially several scheduling intervals (for a domain with the number of virtual CPUs greater than the number of physical CPUs). Domains that are running applications sensitive to I/O latency should be provisioned with the number of virtual CPUs equal to the number of physical CPUs.

Linear I/O scaling per domain

With physical DMA and smart NICs, I/O transfer capacity can approach linear scalability as more physical NICs are added to the initiating and target domains. (Here, *smart NICs* refers to NICs that collect state information regarding the network connections in order to offload network functionality from the processor core.) It is also necessary to maintain an under-provisioning of switch capacity, or the NICs will overwhelm the capacity of the switch to transfer the bits across the fabric.

Using physical DMA, memory transfers will occur in parallel. This is true within a single domain when that domain has control over multiple physical NICs, and across all the domains of a hypervisor.

With appropriate switch capacity, I/O scalability will likely remain linear up to the physical provisioning capacity of industry-standard hardware for the coming several years. The requirements placed on the CPU by physical DMA are relatively small enough not to influence the scalability of I/O for conventional and larger I/O sizes.

Locality of memory reference within a physical host computer

As mentioned, locality-based computing aims to move data as close as possible to the compute resource. Once data is close to the compute resource, the aim is to keep the data and resource together for as long as possible. In other words, once data is being processed, further movement of that data should be restricted.

When data arrives within the main memory of the physical host computer, there are some important issues with hypervisors of today that interfere with the goals of locality-based computing. The hypervisor issues are each related to the overprovisioning of memory and processor resources. The remedies include making use of the physical memory topology of the host, minimizing the movement of host processes across cores, processors and sockets, and preventing the fragmentation of physical memory.

In the future, all commercial server processors will have NUMA characteristics. Server processors have multiple sockets and cores, and each socket and core may have an integrated memory controller with attached memory. Since the entire processor is a cache-coherent memory domain, memory access times depend on the locality of the memory controller in reference to the core fetching the memory. These access time differences must be taken into account by the hypervisor when allocating memory for domains and when scheduling domains to run on processor cores.

The hypervisor requires information on the relationship between physical memory addresses and processor cores, and it should always attempt to place data in memory adjacent to the core that will manipulate the data. Once data is being processed, the hypervisor should attempt to keep an executing domain running on the same processor core. Doing so increases the locality of memory reference by the domain, which corresponds to better cache usage and higher performance.

Hypervisors of today are tuned to minimize the consumption of physical memory using techniques such as guest swapping, page coalescing, and memory ballooning. Guest swapping evacuates pages within a domain from main memory to secondary storage. Page coalescing identifies pages among domains that hold the

same data, and points all such pages to the same physical memory page. Ballooning allows domains to temporarily increase their use of physical memory by borrowing physical pages from each other.

All of the memory techniques described above are counter to the goals of locality-based computing because they involve the movement of data away from the processor currently accessing that data. Guest swapping is particularly counter to the goals of locality-based computing because it moves data all the way to secondary storage. Page coalescing tends to increase memory access times, because the single shared page may reside on a distant memory controller. Ballooning also tends to distribute data on pages that are less desirable from a NUMA viewpoint.

For a balanced analysis, we should point out that memory and I/O overprovisioning using the techniques described above has been a critical factor in the economics of computing for general-purpose data center applications. Traditionally, hypervisors have helped to make very expensive computers economical by amortizing the greatest possible amount of work across a finite set of expensive physical resources.

For many applications, however, higher throughput and lower latency are more important than overprovisioning memory, I/O, and CPU resources. In this environment, hypervisors need to change their memory allocation and processor usage policies in the following ways:

1. Hypervisors must be easily configurable such that they do not overprovision memory. Overprovisioning of memory and processor resources by a hypervisor reduces the locality of memory reference and works against the goals of locality-based computing.
2. A hypervisor must allocate memory for a domain using physically contiguous memory. Most hypervisors today allocate memory for a domain using discontinuous physical pages, and with no regard for the proximity of a socket or core to a dual inline memory module (DIMM).
3. Each virtual CPU must be backed by at least one physical processor core. Hypervisors should schedule multicore domains with a consideration toward the physical topology of the server. In other words, the virtual CPUs of a domain should all be scheduled on the same socket whenever possible.
4. Once a domain is executing on a given core, the hypervisor needs to keep that domain executing on the same core for as long as possible. When scheduling the domain to run on a different core, the first choice should be another core within the same socket.

Conclusion

With significant advancements in processing, networking, storage, and virtualization technologies, disruption is occurring in the data center on a large scale. As these advancements continue, the importance of a flexible and dynamic infrastructure is becoming apparent. When trying to understand and meet application requirements such as latency and bandwidth, Amdahl's Other Law can be applied at the various levels of the data center: the compute node, the pod, and even across the data center itself.

The pod level is the ideal level of optimization. By optimizing at the pod level, we can reduce the cost and increase performance, reduce the complexity of the technology outside of the pod, and establish a common external network interface for pod connectivity to the data center networking core, thereby protecting customer investment and maintaining a path for deploying future technology advancements. Optimizing at the pod level also enables us to exploit virtualization technology at multiple levels.

Naturally, many challenges still exist. We believe that these are manageable and should be approached in a manner that may provide efficient access across various different industries.

References

1. J. Gray and P. Shenoy, "Rules of Thumb in Data Engineering," *Proceedings of the 16th International Conference on Data Engineering* (ICDE), Washington, DC, February 28–March 03, 2000, p. 3.
2. K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill Book Co., Inc., New York, 1993.
3. K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill Book Co., Inc., San Francisco, CA, 1998.
4. K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, et al., "The Landscape of Parallel Computing Research: A View from Berkeley," University of California, Berkeley, Technical Report No. UCB/EECS-2006-183, December 2006.
5. U. Drepper, "What Every Programmer Should Know about Memory," Technical Report, Red Hat, Inc., November 2007; see *http://people.redhat.com/drepper/cpumemory.pdf*.
6. R. van der Paas, "Memory Hierarchy in Cache-Based Systems," Technical Report 817-0742-10, Sun Microsystems, November 2002.

7. K. S. Yim and J. C. Son, "SynergyFS: A Stackable File System Creating Synergies between Heterogeneous Storage Devices," *Proceedings of the Linux Symposium*, 2008, pp. 255–259.
8. B. Gregg, ZFS L2ARC; see *http://blogs.sun.com/brendan/entry/test*.
9. A. Leventhal, "Flash Storage Memory," *Communications ACM* **51**, No. 7, 47–51 (2008).
10. J. Szalay, J. Bunn, I. Gray, I. Foster, and I. Raicu, "The Importance of Data Locality in Distributed Computing Applications," NSF Workflow Workshop 2006.
11. I. Raicu, Y. Zhao, I. Foster, and A. Szalay, "Accelerating Large-Scale Data Exploration through Data Diffusion," *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing*, Boston, MA, June 24, 2008, pp. 9–18.
12. W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers, San Francisco, CA, 2003.
13. M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity, Data Center Network Architecture," *Proceedings of the ACM SIGCOMM Conference*, Seattle, WA, August 2008.
14. R. Russell, "Virtio: Towards a De-Facto Standard for Virtual I/O Devices," ACM *SIGOPS Operating Systems Review* **42**, No. 5, 95–103 (2008).
15. S. Tripathi, K. Belgaied, and N. Droux, OpenSolaris Project Crossbow: Network Virtualization & Resource Partitioning; see *http://opensolaris.org/os/project/crossbow/Docs/Crossbow_WP.pdf*.
16. Y. Weinsberg, D. Dolev, T. Anker, M. Ben-Yehuda, and P. Wyckoff, "Tapping into the Fountain of CPUs: On Operating System Support for Programmable Devices," *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), Seattle, WA, March 1–5, 2008, pp. 179–188.
17. S. Raina, "Virtual Shared Memory: A Survey of Techniques and Systems," University of Bristol Technical Report. UMI Order Number: CSTR-92-36, 2009.
18. D. Howells, "FS-Cache: A Network Filesystem Caching Facility," *Proceedings of the Linux Symposium*, Ottawa, Canada, July 19–22, 2006.
19. B. Callaghan, *NFS Illustrated*, Addison-Wesley Longman Ltd., 2000.
20. R. Noronha and D. K. Panda, "IMCa: A High Performance Caching Front-End for ClusterFS on InfiniBand," *Proceedings of the 2008 37th International Conference on Parallel Processing* (ICPP), Washington, DC, September 9–11, 2008, pp. 462–469.
21. F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA, January 28–30, 2002, USENIX Association, Berkeley, CA, p. 19.
22. P. Deniel, T. Leibovici, and J. C. Lafoucriere, "GANESHA, A Multi-usage with Large Cache NFSv4 Server," *Proceedings of the Linux Symposium*, Ottawa, Canada, July 27–30, 2007.
23. R. Ananthanarayanan, M. Eshel, R. Haskin, M. Naik, F. Schmuck, and R. Tewari, "Panache: a Parallel WAN Cache for Clustered Filesystems," *ACM SIGOPS Operating Systems Rev.* **42**, No. 1, 48–53 (2008).
24. A. Gulati, M. Naik, and R. Tewari, "Nache: Design and Implementation of a Caching Proxy for NFSv4," *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, February 13–16, 2007, USENIX Association, Berkeley, CA, p. 27.
25. M. Turner, "FlexCache Caching Architecture," Technical Report TR-3669, NetApp, Inc., June 2008, see *http://media.netapp.com/documents/tr-3669.pdf*.
26. S. Oehme, J. Deicke, J. Akelbein, R. Sahlberg, A. Tridgell, and R. L. Haskin, "IBM Scale Out File Services: Reinventing Network-Attached Storage," *IBM J. Res. & Dev.* **52**, No. 4, 319–328 (2008).

Received December 9, 2008; accepted for publication January 12, 2009

David Cohen *EMC, 11 Cambridge Center, Cambridge, Massachusetts (cohen_david2@emc.com)*. Mr. Cohen's career spans more than 25 years. He recently joined the EMC Cloud Infrastructure group after spending the past several years working in the financial services industry, most recently at Goldman Sachs where he was a Senior Storage Strategist. Prior to this, he worked at Merrill Lynch where he focused on a variety of low-latency and utility computing problems.

Fabrizio Petrini *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (fpetrini@us.ibm.com)*. Dr. Petrini is a Senior Researcher of the Cell Solution Department of the IBM T. J. Watson Research Laboratory. His research interests include various aspects of multicore processors and supercomputers, including high-performance interconnection networks and network interfaces, fault tolerance, job scheduling algorithms, parallel architectures, operating systems, and parallel programming languages. He is associate editor of *IEEE Transactions on Parallel and Distributed Processing*.

Michael D. Day *IBM Systems and Technology Group, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (mdday@us.ibm.com)*. Mr. Day is a Virtualization Architect in the IBM Linux Technology Center, as well as Distinguished Engineer. For the past four years, he has worked on open-source virtualization technology including the Xen** and Linux** Kernel-based Virtual Machine (KVM) hypervisors. He has more than 20 years of experience as a programmer of operating systems, networking protocols, systems management technology, and hypervisor.

Muli Ben-Yehuda *IBM Haifa Research Laboratory, University Campus, Haifa 31905, Israel (muli@il.ibm.com)*. Mr. Ben-Yehuda is a Systems Researcher at the IBM Haifa Research Laboratory where he was recently named an IBM Master Inventor. His research interests include I/O for virtualized systems, I/O memory management units (IOMMUs), smart I/O devices, and various novel uses for virtual machines. He has contributed to numerous operating systems and hypervisors, including the Linux kernel, the Xen virtual machine monitor, and the Linux Kernel-based Virtual Machine (KVM).

Steven W. Hunter *IBM Research Division, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (hunters@us.ibm.com)*. Dr. Hunter is an IBM Distinguished Engineer in IBM Research where his focus has been on architecture and design for servers, network and clustering systems, and various network technologies, including those involving the IBM BladeCenter* and next-generation computing systems. He is a licensed Professional Engineer, a Senior Member of the IEEE, and an adjunct professor at North Carolina State University.

Uri Cummings *Fulcrum Microsystems, 26630 Agoura Road, Calabasas, California 91302 (uri@fulcrummicro.com)*. Dr. Cummings is the Chief Technology Officer at Fulcrum Microsystems. In January 2000, Dr. Cummings co-founded Fulcrum to commercialize research in high-performance VLSI design, and he became the founding CEO. In April 2001, Dr. Cummings recruited the current CEO of Fulcrum, and focused on product development. He managed the first commercial chip development of the company, a multi-gigabit switch chip, and numerous other validation chips. He now leads the technology and architecture direction at Fulcrum, and defined the 10-Gb Ethernet switch chips of Fulcrum.