Hecatonchire: Towards Multi-host Virtual Machines by Server Disaggregation

Petter Svärd[1], Benoit Hudzia[2], Johan Tordsson[1], and Erik Elmroth[1]

[1] Dept of Computing Science, Umeå University, Sweden
[2] Stratoscale, Belfast, UK

Abstract. Horizontal elasticity through scale-out is the current dogma for scaling cloud applications but requires a particular application architecture. Vertical elasticity is transparent to applications but less used as scale-up is limited by the size of a single physical server. In this paper, we propose a novel approach, server disaggregation, that aggregates memory, compute and I/O resources from multiple physical machines in resource pools. From these pools, virtual machines can be seamlessly provisioned with the right amount of resources for each application and more resources can be added to vertically scale a virtual machine as needed, regardless of the bound of any single physical machine. We present our proposed architecture and implement key functionality such as transparent memory scale-out and cloud management integration. Our approach is validated by a demonstration using benchmarks and a real-world big-data application and results indicate a low overhead in using memory scale-out in both test cases.

1 Introduction

Large peta-byte, and soon exa-byte, data collections are becoming more common [17], with data emanating from transactional enterprise applications, energy grids, social web services, weather sensors or mobile devices. To work upon these large data sets, large amounts of scalable computing resources are required. Today's cloud is designed to provide scalability via the main two scaling methods: horizontal and vertical elasticity. Horizontal elasticity involves allocating more Virtual Machines, *VMs*, to run an application while vertical on the other hand means adding more resources like CPU and memory to an existing VM.

Vertical elasticity is well suited to scale resource demanding business-critical applications such as large databases, ERP systems and big data analytics. It should therefore be a part of cloud platforms in order to enable applications and infrastructure to work together to provide the scalability they need. However, current virtualization technologies are ill-equipped to deliver vertical elasticity as they were primarily built for sharing of individual servers.

In this contribution we introduce the concept of Server Disaggregation to address these shortcomings by enabling cloud infrastructure to lift the physical limitations traditionally associated with memory, compute and I/O resources.

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part II, LNCS 8806, pp. 519–529, 2014.

Server Disaggregation allows cloud platforms to aggregate and manipulate resources more freely, for example scaling up by adding more hardware resources to a VM, regardless of the limitations of the server where it is deployed. As computer prices drop and performance continues to increase, low cost *commodity* systems are the perfect fit for the Server Disaggregation approach as they be configured in large clusters to aggregate computing power. In the paper we present the Hecatonchire, or *Heca* for short, approach to Server Disaggregation and a part-implementation of the concept, namely scale-out of memory and a proof-of-concept integration of memory scale-out into OpenStack. We validate our findings by means of a performance study of memory scale-out for a benchmark appliction and the SAP HANA in-memory database.

2 Vision and General Approach

The goal of the Heca project is to provide a true utility service by disassociating servers from their core resources and relaxing the coupling between VMs and their physical hosts, thereby creating a radically new delivery model for IaaS platforms. Today, CPU development no longer follows Moores law [7] and instead, the industry has moved towards parallelism with processors featuring more cores. The same applies to RAM and disk where, relative to CPU performance, disk performance has actually become slower over the past 30 years [15]. In contrast, network bandwidth continues to increase rapidly. Interfaces such as Infiniband provide interconnect speeds that are approaching internal bus speeds [15] and techniques like Remote Direct Memory Access, RDMA, enable fast access to remote memory. This means that the performance overhead for using resources on remote servers is decreasing.

In a Heca-enabled datacenter, a VM can use resources from multiple servers. Aggregated Memory, compute, and I/O resources are made available in separate pools from which a VM can dynamically consume the aggregated resources to meet changes in application requirements at runtime. This effectively frees the cloud system from some of the constraints of the underlying physical infrastructure and also means that larger VMs than can fit on a single server can be provisioned.

2.1 Server Disaggregation

Server Disaggregation constitutes a major shift in the evolution of data centers and serves as a key enabler for providing a complete scaling solution for platforms on the cloud. Compared to traditional IaaS platforms, the technique has several potential benefits, which we enumerate in this section.

Superior Scalability. For large memory workloads, vertical elasticity is often the most suitable scaling approach. However, there are limits on maximum memory size for commodity hardware and typically a large memory size has to be traded for reduced memory bandwidth, e.g., lower frequency DIMMs. Very often, an additional storage hierarchy that relies on SSDs or disks as a temporary

data store is introduced, with severe impact on performance. In contrast, distributed memory aggregation over high-speed interconnects across servers provides a cost-effective, high-performance, alternative as it enables applications to leverage the memory of multiple systems. Server Disaggregation thus combines a cost-effective virtual x86 platform running on commodity hardware with a large shared memory thereby enabling provisioning of resource-intensive VMs.

Improved Resource Utilization. Scheduling of VMs to achieve maximum hardware utilization is known to be an NP-hard problem [18], e.g., provisioning a lot of memory-bound VMs can lead to underutilized CPUs, etc. Using resource aggregation technology, VMs can be deployed independent of single server boundaries, to simplify scheduling and improve resource utilization. Also, fewer but larger nodes mean reduced cluster complexity and reduced fragmentation of the resources. For example, financial organizations run up to thousands of simulations at once, and a common deployment involves hundreds of servers, where each node is running a simulation application at 80% utilization. By using resource aggregation to create fewer larger nodes, every four aggregated systems can run another copy of the application, in theory approaching 100% utilization.

Better Performance. When I/O, computing and memory resources are separated into purpose-built nodes, servers can be better optimized to the requirements of the hosted applications. For compute-intense workloads, proprietary shared-memory systems have traditionally been used. Systems such as the SGI Ultraviolet [14] or the Cray XMT [5] come with significantly larger memory sizes but they are comparatively expensive. Aggregation technology benefits from the local memory bandwidth across servers, as opposed to traditional SMP [16] or, to a lesser extent, NUMA architecture, where memory bandwidth decreases as the machine scales out. Solutions based on resource aggregation can thus show close-to-linear memory bandwidth scaling, thereby delivering excellent performance in particular for many-threaded applications, e.g., graph analysis, or memory bandwidth bound ones, such as computational fluid dynamics simulations.

Easier Use and Administration. Traditionally, using distributed memory across several servers requires that the application is developed for an explicit memory distribution model which require highly skilled, domain-aware software developers using custom software libraries [11]. Having a single virtual system to manage is also simpler compared to the complexities involved in managing a cluster with respect to software installation and synchronization. Furthermore, aggregation technology also simplifies the I/O architecture by consolidating each individual server's network and storage interfaces. The administrator gets fewer I/O devices to manage leading to increased availability, higher utilization, better resiliency, and runtime scalability of I/O resources.

Improved Economics. Thanks to improved scalability, hardware utilization and performance and simplified administration, aggregation technologies show great potential for cost savings in data center operations. Server Disaggregation also provides a cost-effective x86 alternative to expensive and proprietary shared memory systems.

3 Heca Architecture and Implementation

The Heca architecture, outlined in Figure 1, decouples virtual resource management from physical resources by providing the capability to mediate between applications and servers in real-time. This decoupling is achieved by aggregating and managing server resources in a datacenter. Each resource type is exposed to the overall cloud platform via an independent mediation layer that arbitrates the allocation of resources between multiple applications, creating a distributed and shared physical resources layer. The architecture is composed of three layers,

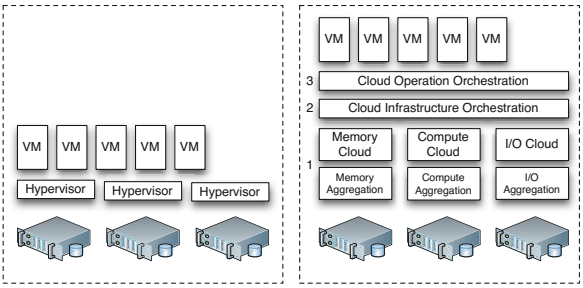

Fig. 1. Traditional (left) vs Heca (right) Virtualization

the *Cloud Resource Aggregation* layer, marked as 1 in Figure 1, provides access to and management for the aggregated resources, i.e. Memory Cloud, Compute Cloud and I/O Cloud. The *Cloud Infrastructure Orchestration* layer, marked as 2, provides the ability to compose logical virtual servers with a level of service assurance that guarantees resources and performance provided by the resource aggregation layer. It also exposes extended features enabled by the decoupled resource layers. The *Cloud Operation Orchestration* layer, marked as 3, provides service life cycle management. It enables provisioning of self-configuring, self-healing, self-optimizing services that can be composed to create self-managed business workflows that are independent of the physical infrastructure.

3.1 Transparent Memory Scale-out

To enable *transparent memory scale-out*, Heca makes it possible for a VM to allocate memory on multiple servers. The server that hosts the VM to be scaled-up is termed a *memory demander*. The application is transparently scaled vertically by using memory provided by other hosts in the cluster, denoted *memory sponsors*. Figure 2 depicts memory scale-out, with a memory demander running an application, and several memory sponsors. The memory sponsors are VMs whose sole purpose is to provide memory to its demanders. Note that a server can host both memory sponsors and demanders at the same time.

All hosts run a modified Linux kernel, including a Heca kernel module, and also include a modified version of the QEMU hypervisor. On a higher level the kernel module fits in Layer 1 in Figure 1, while the modified hypervisor belongs

to Layer 2 in the same figure. The kernel module handles the operations during scale-out and the transfer of memory content to and from remote hosts. The hypervisor enables full transparency as it communicates cluster setup to the kernel module, and applications run unchanged on top of it. It also generates specialized system calls, *ioctl*s, to the kernel module, passing relevant parameters needed to set up the memory scale-out. The behavior of the kernel module differs between memory sponsors and demanders. On the memory demander, the VM's RAM is partitioned into address ranges. Each address range is registered as sponsored by a memory sponsor. Appropriate page table entries, *PTEs*, are put in place. Each memory sponsor allocates enough memory in its VM to sponsor one address range on the memory demander. Besides that, memory sponsors can continue to operate as usual.

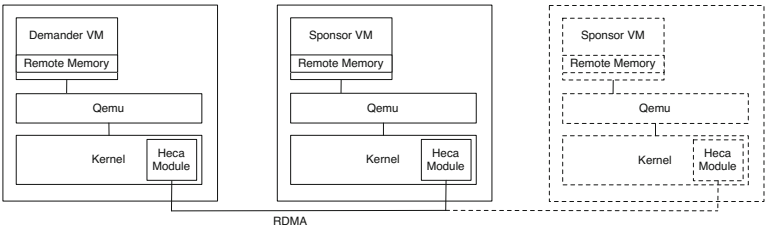

Fig. 2. High-level architecture of a memory scale-out

The partitioning of memory into address spaces is determined by the parameters passed to the VMs during provisioning. Therefore, when setting-up the memory scale-out each address range is created in accordance with a corresponding amount of physical memory provided by a memory sponsor. When the VM faults on an address, the kernel identifies the modified PTE and passes execution to the kernel module. The module requests the memory page from the memory sponsor, and the page fault is resolved. If the kernel later decides to swap out the page, its contents are re-sent to the memory sponsor, and the PTE is updated. Our solution achieves transparency as the application runs in a VM, unaware of the scale-out operation. Also, application performance is good as most memory operations are carried out in kernel space, beneath the I/O layer. Use of a virtual stack also enables integration with cloud platforms such as OpenStack [12], managing a cluster of VMs. This simple approach reflects a trade-off however, and on the downside, it binds the approach to a virtualization stack, in our case KVM.

Resilience and Fault-Tolerance. The Heca approach can provide resilience by preparing memory sponsors with twice the available memory, compared to the requirements of the memory demander. Arguments are passed to the VMs reflecting that each address space is sponsored by two memory sponsors. The hypervisors pass that information on to the kernel module. When the kernel module faults on an address, it sends the request to both memory sponsors,

sponsoring its address space. The first arriving response is used. When the kernel module swaps a page out, it sends it to both memory sponsors, and waits for validation that both of them stored the content, before discarding the page. The biggest advantage of this approach is zero-downtime failover of memory sponsors. If one sponsor fails, the other sponsor continues responding to the memory demander's requests for memory content. Furthermore, the memory demander can identify the fault (trusting the remote kernel module, and the underlying networking fabric), and disconnect from the sponsor. Another host can later join the system, taking up the role of the failed sponsor.

However, there are a few disadvantages with this approach. First of all, it consumes twice the amount of memory, compared to a non-resilient scheme. Our mirroring approach also doubles the required bandwidth, an increase that previous generations of networking fabrics could not support [8]. However, today's fabrics can handle much higher loads. With bandwidths exceeding 100 Gb/s this would require the application to be very memory intensive, swapping more than 50 Gb/s, yet it is theoretically possible. In this context we highlight that even the most memory-intensive applications are practically bound by memory bus capacities. Infiniband capacities have rapidly multiplied in the last decade, while the maximum memory bandwidth for Intel Xeon server series chipsets have only increased by a factor of 8 over this period, from 6.4 Gbps to 51.2 Gbps [4]. If this trend persists, the potential bottleneck might be further mitigated and even eliminated in most practical scenarios.

Other resiliency approaches, such as RAID-5, are more conservative in memory and bandwidth requirements. Yet such approaches require a lengthy computation process to recover from a fault, in which lost data is re-built. This prevents them from ensuring zero-downtime failover. Additionally, such approaches may incur a performance penalty on the scale-out operation, as computation of parity bits is required when swapping pages out.

We highlight that this discussion does not deal with fault tolerance for the main host running the application, the memory demander. This issue is beyond the scope of this paper, as it is not a scale-out challenge, but rather a generic challenge of fault tolerance for VMs.

3.2 Cloud Management Integration

To simplify the use of memory scale-out we have integrated resource disaggregation of VMs into OpenStack. If the VM is too large to fit on any host, our modified OpenStack scheduler splits the VM into sponsors and a demander. The feature is enabled by setting a flag in an OpenStack VM flavor.

The launch of a VM instance in OpenStack starts with the cloud controller receiving a request to deploy an instance via the Compute API (Step 1 in Figure 3). The instance is given an instance ID and the message is forwarded to the scheduler which selects a suitable worker to run the instance (Steps 2 and 3) and passes the message to it (Step 4). The compute worker sends a message to the network controller to get an IP for the instance (Steps 5-8) and continues provisioning of the instance.

To provision the VMs correctly as demander and sponsors, extra information must be passed to the hypervisor. These parameters include a heca mode, *sponsor* or *demander*, two heca process identifiers, TCP ports for control and memory transfer, and RDMA IP addresses for both demander and sponsors. The start address and size of the shared memory region is also needed and is given by how much more memory the VM requests than maximum free on any host. Figure 3 illustrates how OpenStack allocates the instance before it is sent to the scheduler, which also performs the actual deployment in an asynchronous manner. This creates an issue as the sponsor and demander both need each others RDMA IP addresses at the time of creation. Our pragmatic solution is to perform a "pre-scheduling" round to determine the placement of the VMs without actually provisioning them. The instances are then sent to the scheduler again, using scheduler hints to achieve the desired placement. To pass these parameters to qemu-kvm our modified OpenStack constructs a `<qemu:commandline>` block that is added to the instance.xml file. On instance creation, instance.xml is fed to the libvirt API that passes the Heca parameters to the qemu-kvm hypervisor.

4 Experimental Demonstration of Heca functionality

To verify the memory scale-out functionality we deploy an 8 GB VM to an OpenStack cloud with a controller node and three compute nodes, see Table 1. We present performance results for two deployments, with and without memory scale-out enabled. The outcome of the two deployments are shown in Figure 4.

Table 1. Testbed Description.

Node	CPU	RAM	Free RAM	Network	Kernel
Controller	i5@3 GHz	4 GB	N/A	Gb Ethernet	Linux Heca 3.6
Compute A,B,C	i5@3 GHz	8 GB	4,3,5 GB	iWARP	Linux Heca 3.6

In the first deployment, the VM is provisioned on the host with the most amount of RAM available. As overbooking of resources is enabled in OpenStack, virtual memory is used to account for the overbooked RAM. In the second deployment, the modified OpenStack avoids using virtual memory by memory scale-out and provisions the memory demander on Node B and a memory sponsor on Node A.

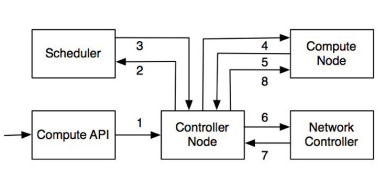

Fig. 3. OpenStack deployment

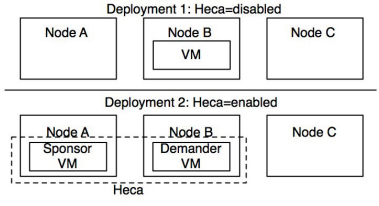

Fig. 4. Deployment outcome

Remote Memory Performance. To evaluate the relative performance of using remote memory we made four comparisons using the Linux *MBW* [9] tool that allocates two arrays and copies the first to the second using memcopy. We ran MBW with an array size of 3 GB which means it allocated 6 GB of RAM. In Figure 5, for the baseline case, marked as *bare-metal*, MBW was run non-virtualized with more than 6 GB of free RAM. In the second case, *virtualized* in Figure 5, MBW was run in an 8 GB VM, with more than 6 GB of free RAM. For the third experiment, *overcommitted* in the same figure, the second experiment was repeated but the amount of free memory on the host was restricted to 4 GB meaning that the host is overcommitted. In the fourth experiment, marked as *memory scale-out*, MBW was run on a demander-sponsor VM pair with 2 GB scaled out to the sponsor. All other conditions were identical to the overcommitted case. An overall observation is that virtualized is 6% slower than bare-metal and memory scale-out is 6% slower than virtualized. The results of the overcommitted case vary greatly between iterations due to swapping.

To further evaluate the memory scale-out functionality we performed an experiment with a real-world, big data application, SAP HANA [13], which is an in-memory database. The application was run on a 40 vCPU VM on a 4 socket, 10 core Intel Xeon West Mere cluster with 1 TB RAM, connected by a 40 Gbps Infiniband network. The experiment was performed with a set of 18 different queries against a 2.5 TB OLAP dataset. Between tests, we varied the number of simultaneous users running the query sets. The complete test was performed twice, the second time 512 GB of the VMs RAM was scaled out to a memory sponsor. The results are shown in Figure 6. In all runs, the overhead in query response time with 50% remote memory was around 3% compared to running virtualized with no remote memory.

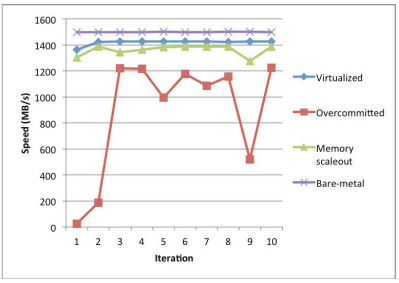

Fig. 5. Memcopy speed test results

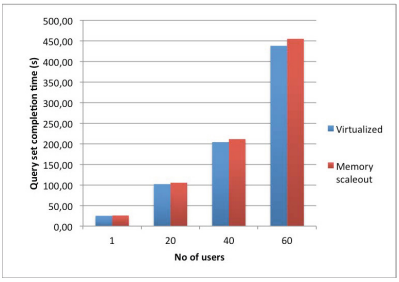

Fig. 6. Overhead per query set

Table 2. Overhead per query set with 80 HANA users

Demander : Sponsor A : Sponsor B	Overhead
1 GB : 2 GB : -	4%
1 GB : 3 GB : -	5.6%
2 GB : 1 GB : 1 GB	0.9%
1 GB : 1 GB : 1 GB	2%

To investigate the overhead when using remote memory in more detail, we present the result from a test running HANA on a smaller VM, this time with 80 users but varying the amount of remote memory. We ran the experiment with one and two sponsors, varying the distribution of memory between the demander and sponsors as shown in Table 2. The table also shows that the overhead increases with the amount of remote memory and that distributing the remote memory over several sponsors improves performance, due to the increased bandwidth.

5 Related Work

Han et al. advocate a datacenter architecture in which the resources within a server are disaggregated and the datacenter is architected as a collection of standalone resources [3]. However, they do not implement anything but rather investigate the feasibility of such an approach.

The Oracle Transcendent Memory project makes unused memory on a node available to other nodes through the use of an API [6]. The main difference from Heca is that guest OS changes are explicitly required in order to use the shared memory using the Oracle approach. Also, there is no guarantee that memory that is currently idle will not eventually be needed as the future working set size of a VM cannot be accurately predicted. Another similar approach is VMware DRS which enables managing a cluster containing many potentially-heterogeneous hosts as if it were a single pool of resources [2]. The main difference between the DRS and the Heca approaches is that the DRS approach splits a cluster into smaller groups. A number of VMs attached to a group can then share the CPU and Memory resources in the group among them. Dragojevic et al. propose their Fast Remote Memory, *FaRM*, approach which exposes remote memory over RDMA as a shared address space [1], consisting of 2 GB memory regions. In contrast to Heca, the FaRM approach does not use a virtualization stack, but the shared memory is made available trough a programming model.

Sharing of resources is also provided by XtreemOS, which is a distributed Linux distribution that aggregates resources from compute resources in a cluster [10]. However, as the system is perceived as one single computer this approach can be cumbersome when running many applications in parallel. This means that XtreemOS is more suited for the Grid use-case.

6 Conclusion

We propose a solution to enable vertical elasticity, beyond the capacity limitations of individual servers, by aggregating CPU, memory, and I/O resources into reusable pools that can be used to provision VMs independent of limitations of the underlying hardware. The core concepts of our outlined architecture is implemented as a kernel module and a modified Qemu-KVM hypervisor integrated into OpenStack. Our approach is validated by provisioning multi-host VMs and a performing an evaluation of memory scale-out. The results indicate that our server disaggregation concept is feasible, with as little as 6% overhead compared

to single-host virtualization as well as simplified administration, thus enabling a broader range of applications to take advantage of the cloud.

Acknowledgments. The authors thank Steve Walsh and Aidan Shribman for their valuable contributions to this project as well as SAP (UK) Limited, Belfast, where much of this work was performed. Financial support has been provided in part by the Swedish Governments strategic effort eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

References

1. Dragojević, A., Narayanan, D., Hodson, O., Castro, M.: FaRM: fast remote memory. In: NSDI 2014. USENIX (2014)
2. Gulati, A., Holler, A., Ji, M., Shanmuganathan, G., Waldspurger, C., Zhu, X.: Vmware distributed resource management: Design, implementation, and lessons learned. VMware Technical Journal 1(1), 45–64 (2012)
3. Han, S., Egi, N., Panda, A., Ratnasamy, S., Shi, G., Shenker, S.: Network support for resource disaggregation in next-generation datacenters. In: HOTNETS 2013: The Twelfth ACM Workshop on Hot Topics in Networks, pp. 10:1–10:7. ACM (2013)
4. Intel. Microprocessor quick reference guide, `http://www.intel.com/pressroom/kits/quickrefyr.htm` (Visited on April 27, 2014)
5. Konecny, P.: Introducing the Cray XMT. In: CUG 2007: The 2007 Cray User Group meeting (2007)
6. Magenheimer, D., Mason, C., McCracken, D., Hackel, K.: Transcendent memory and linux. In: Proceedings of the Linux Symposium, pp. 191–200 (2009)
7. Mann, C.C.: The end of Moores law. Technology Review 103(3), 42–48 (2000)
8. Markatos, E., LeBlanc, T.: Using processor affinity in loop scheduling on shared-memory multiprocessors. IEEE Transactions on Parallel and Distributed Systems 5(4), 379–400 (1994)
9. MBW. MBW: Memory bandwidth benchmark (2010), `http://manpages.ubuntu.com/manpages/lucid/man1/mbw.1.html` (Visited on January 2, 2014)
10. Morin, C.X.: A grid operating system making your computer ready for participating in virtual organizations. In: ISORC 2007: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp. 393–402 (2007)
11. Nussle, M., Scherer, M., Bruning, U.: A resource optimized remote-memory-access architecture for low-latency communication. In: ICPP 2009: The 2009 International Conference on Parallel Processing, pp. 220–227. IEEE (2009)
12. OpenStack. OpenStack Cloud OS, `https://www.openstack.org` (Visited on April 27, 2014)
13. SAP. SAP HANA (2014), `http://bit.ly/GKZkDy` (Visited on April 27, 2014)
14. SGI. Technical Advances in the SGI UVTM Architecture, `http://www.sgi.com/pdfs/4192.pdf` (visited on April 27, 2014)

15. Subramoni, H., Koop, M., Panda, D.: Designing next generation clusters: Evaluation of InfiniBand DDR/QDR on Intel computing platforms. In: HOTI 2009: The 2009 IEEE Symposium on High Performance Interconnects, pp. 112–120 (2009)
16. Tipparaju, V., Nieplocha, J., Panda, D.: Fast collective operations using shared and remote memory access protocols on clusters. In: IPDPS 2003: The 2003 International Parallel and Distributed Processing Symposium, p. 10. IEEE (2003)
17. Trelles, O., Prins, P., Snir, M., Jansen, R.C.: Big data, but are we ready? Nature Reviews Genetics 12(3), 224–224 (2011)
18. Wood, T., Shenoy, P.J., Venkataramani, A., Yousif, M.S.: Black-box and gray-box strategies for virtual machine migration. In: NSDI, vol. 7, pp. 229–242 (2007)