Hecatonchire: Enabling Multi-Host Virtual Machines by Resource Aggregation and Pooling

Petter Svärd
Dept of Comp. Science Umeå University, Sweden

Benoit Hudzia
Stratoscale, Belfast, UK

Johan Tordsson and Erik Elmroth
Dept of Comp. Science Umeå University, Sweden

Abstract

Vertical elasticity, or scale-up of individual virtual machines is hard to perform in today's cloud environments due to limitations in the amount of hardware resources available in single servers. We propose a novel approach that allows aggregation of memory, compute and I/O resources from multiple physical machines in resource pools which in turn are used to seamlessly provision virtual machines with the right amount of resources. We present our architecture and highlight key functionality such as transparent and resilient memory aggregation and fast live migration. Our approach is validated by a demonstration using benchmarks and a real-world big-data application. Performance results indicate a very low overhead in using aggregated memory as well as a significant improvement in live migration performance.

1 Introduction

It has been claimed [8], that by the end of 2013 more than 75% of x86 server workloads would have been virtualized. However, as it stands for business critical applications, this number is only 20%. While many valid reasons have been proposed to justify the reluctance of virtualization and cloud platform adoption by small to medium enterprises and large corporations, the ability of a system to scale well is one of the major pain points limiting a fluid transition to cloud ecosystems [7]. With data emanating from transactional enterprise applications, energy grids, social web services, weather sensors or mobile devices, large peta-byte, and soon exa-byte, data collections are becoming more common [14] and to work upon these large data sets, large amounts of vertically scalable computing resources are required.

Todays cloud is designed to provide conceptually infinite scalability via the main two scaling methods: horizontal and vertical scaling, also know as *scaling out* and *scaling up*. Scaling out involves adding more nodes to the infrastructure while scaling up on the other hand means adding more resources like CPU and memory to an existing host. Scaling up is very well suited to resource demanding business-critical applications such as large databases, ERP systems, big data analytics, Java-based applications as well as academic workloads and research based on complex technical simulations. However current virtualization technologies are ill-equipped to deliver scale-up of such features as they were primarily built for the scale-out scenario. This means that moving to the so-called *second wave* of virtualization, i.e. providing the agility of virtualization to business-critical applications is not feasible using today's cloud infrastructure.

In this contribution we present our Hecatonchire, or *Heca* for short, approach which address these shortcomings by enabling cloud infrastructure to lift the physical limitations traditionally associated with memory, compute and I/O resources. This allows cloud applications to aggregate and manipulate them at will, for example scaling up by adding more hardware resources to a Virtual Machine, *VM*, regardless of the limitations of the server where it is deployed. This paper presents the Heca architecture and a validation study of two key concepts, memory scale-out and fast live VM migration.

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. In a Heca-enabled datacenter, a VM can use resources from multiple servers. Memory, compute, and I/O resources are made available in pools from which a VM can dynamically consume and aggregate resources to meet changes in application requirements at runtime. This effectively frees the cloud system from the constraints of the underlying physical

infrastructure and enables larger VMs than can fit on a single server and thus significantly reduces the barrier of entry for using distributed and remote resources.

2.1 Resource Aggregation and Pooling

The Heca vision relies heavily on the concept of resource aggregation and pooling. Compared to traditional IaaS platforms, these techniques can provide several benefits, which we enumerate in this section.

Superior scalability: For large memory workloads, the scale-up approach is often the most suitable. 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 server nodes provides a cost-effective, high-performance, alternative as it enables applications to leverage the memory of multiple systems. Resource aggregation and pooling thus combines a cost-effective virtual x86 platform running on commodity hardware with a large shared memory thereby enabling provisioning of resource-intensive applications.

Improved resource utilization: Scheduling of VMs to achieve maximum hardware utilization is known to be an NP-hard problem [15], 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: Because I/O, computing and memory resources are separated into purpose-built nodes, these 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 SGIs Ultraviolet [12] or the Cray XMT [6] 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 [13] 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, such as computational fluid dynamics simulations. Live VM migration [1] also benefits from resource aggregation [2]. Remote memory pages do not need to be migrated leading to a significantly shorter migration time and if the VM's entire memory is externalized, memory migration is decoupled from compute migration.

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 [9]. 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.

3 Heca Architecture and Implementation

The Heca cloud 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 both local resources and remote resources available over the network. 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, 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 resources layers. The *Cloud Operation Orchestration* layer, marked as 3, provides service life cycle management. It en-

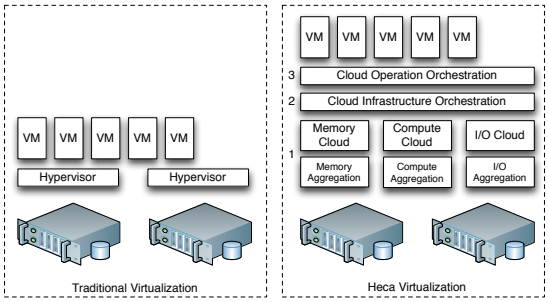

Figure 1: Traditional vs Heca Virtualization.

ables 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 is based on a cluster of servers. The server that hosts the application to be scaled-up is termed a *memory demander*. The application is transparently scaled-up by using memory provided by other hosts in the cluster, denoted *memory sponsors*. Figure 2 depicts the high-level architecture of a cluster, with a memory demander running an application, and several memory sponsors. Note that a node can both be a memory sponsor and a demander at the same time. All hosts run a modified Linux kernel version, with

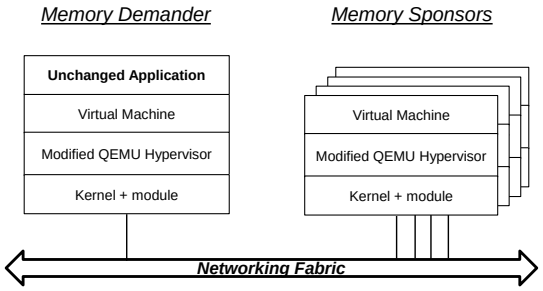

Figure 2: High-level architecture of a memory scale-out.

a specialized Heca module, and also include a modified version of the QEMU hypervisor. The kernel module handles the routine operations during scale-out and 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 one 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. Other than that, memory sponsors can continue to operate as usual.

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 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 a new PTE is put in its place. This simple approach reflects a trade-off. Or solution achieves transparency as the application runs in a VM, unaware of the scale-out operation, and application performance is good as most routine actions 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 [10], managing a cluster of VMs. On the downside, it binds our approach to a particular virtualization stack, in our case KVM.

As externalising memory increases the risk of failure we devised and implemented a scheme to enable memory fault-resilience where all remote memory is mirrored across two or more memory sponsors. Hence, a failure in one sponsor does not necessarily cause loss of data. Furthermore, such failures does not cause any delay or downtime in the running applications [5], at the expense of using more memory.

3.2 Cloud Management Integration

To simplify the use of memory scale-out we have integrated automatic splitting of VMs into sponsors and a demander in OpenStack. However, in the proof-of-concept implementation, only one sponsor per demander is supported. To enable this feature, an extra flag "heca=enabled" is set 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 a sponsor and demander, extra information must be passed to the hy-

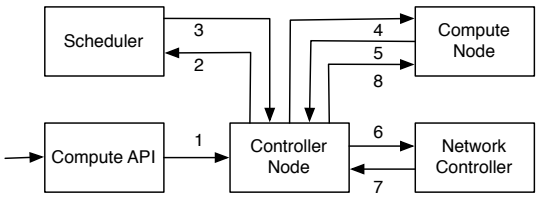

Figure 3: OpenStack deployment.

pervisor. These parameter include a heca mode, sponsor or demander, two heca process identifiers, TCP ports for control and memory transfer, RDMA IP addresses for both demander and sponsor as well as the start address and size of the shared memory region which is given by how much more than the maximum of free RAM on any host the VM requests. 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 a problem as the sponsor and demander both need each others RDMA IP addresses at the time of creation. Our workaround is to look up the available RAM on the compute nodes during instance creation. If the VM is too large to fit on any host, OpenStack splits the VM into a sponsor and a demander. Scheduler hints are then used to place the demander on the host with the most RAM available and the sponsor on the host with the second-most amount of available RAM. This is a work-around solution, and in future versions of the Heca kernel, this mutual dependency will be removed. 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 which passes the Heca parameters to the qemu-kvm hypervisor.

3.3 Fast VM live migration

The Heca-modified qemu-kvm version utilizes the Heca kernel module's fast zero-copy RDMA transfer to implement post-copy and hybrid live migration. The kernel module handle transfer of memory pages and the modified QEMU hypervisor handles set-up and control of the operation as outlined in Figure 4. Post-copy live migration defers the transfer of a VM's memory contents until after its CPU state and device state have been sent to the destination host [3]. In comparison with the classic pre-copy approach, post-copy live migration can improve several metrics including total bytes transferred and total migration time. It also provides a deterministic migration process as it only sends each page exactly once. The drawback of post-copy is that no pages are available directly after the VM is migrated, which can lead to performance degradration in the running applica-

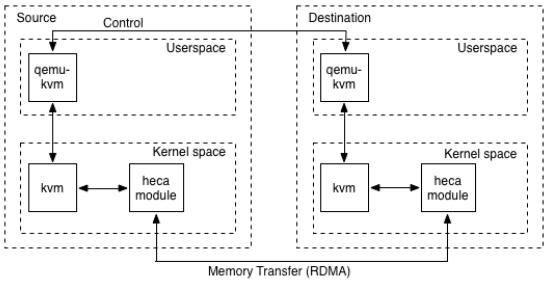

Figure 4: Heca Post-copy live migration.

tions. However, the Heca in-kernel, zero-copy approach over RDMA reduces the performance penalty of fetching remote pages, reducing this problem.

To further reduce the migration performance degradation, Heca also supports a hybrid live migration scheme, which is a special case of post-copy [3]. In hybrid migration, a short pre-copy stage is run before the main memory transfer, as seen in Figure 5. This is to mitigate the negative performance impact of page faulting by pushing a subset of pages to the destination host prior to transferring execution control to the destination host. The du-

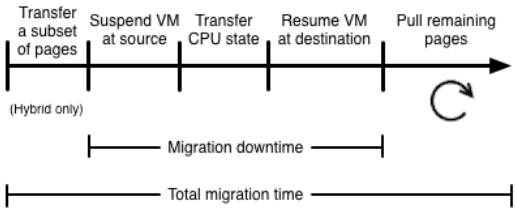

Figure 5: Post-copy and hybrid live migration.

ration of the pre-copy phase can, and should, be limited according to the required total migration time to provide the benefits of reduced page faults without an leading to an unnecessarily long migration time.

4 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 in Table 1. We ran the deployment twice, with and without memory scale-out enabled. The outcome of the two deployments are

Table 1: Testbed Description.

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

shown in Figure 6. In the first deployment, the VM is placed on the host with the most amount of RAM avail-

able. 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 using memory scale-out and placing the memory demander on Node B and a memory sponsor on Node A.

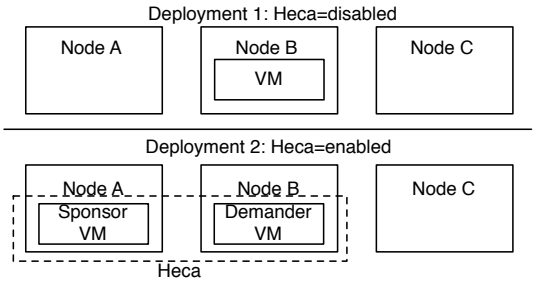

Figure 6: Deployment outcome

Remote memory performance: To evaluate the relative performance of using remote memory we compared four cases using the Linux *MBW* 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. The results from the test can be seen in Figure 7. In the baseline case, marked as *bare-metal*, the MBW tool was run non-virtualized with more than 6 GB of free RAM. In the second case, denoted by *virtualized* in Figure 7, MBW was run in an 8 GB VM, with more than 6 GB of free RAM. For the third experiment, *overprovisioned* 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 overprovisioned. 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. The other conditions were the same as in the overprovisioned case. When comparing the results, it can be seen that virtualized is 6% slower than bare-metal and memory scale-out is 6% slower than virtualized. The results of the overprovisioned case vary greatly between iterations due to swapping of memory pages.

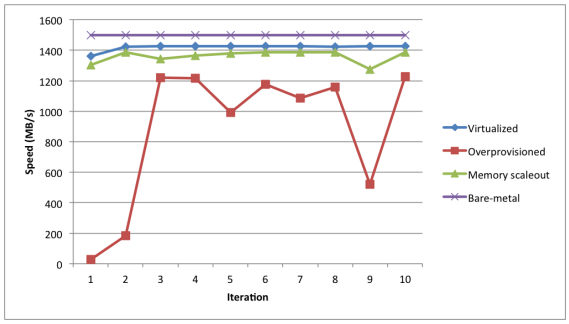

Figure 7: Memcopy speed test results.

To further evaluate the memory scale-out functionality we performed an experiment with a real-world, big data application, SAP HANA [11], an in-memory database. The application was run on a 1 TB 40 vCPU VM on a 4 socket, 10 core Intel Xeon West Mere cluster with 1TB 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 vary 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 8. In all runs, the overhead in query response time with 50% remote memory was around 3% compared to running virtualized with no remote memory.

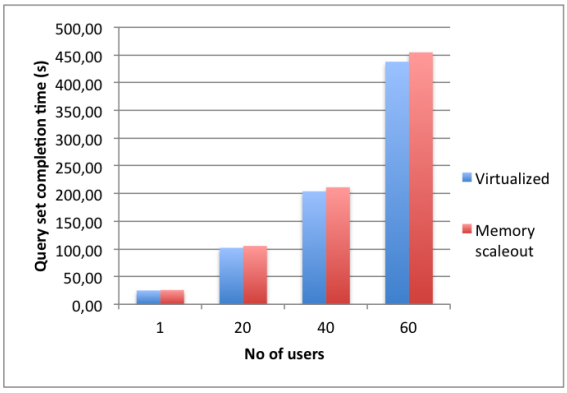

Figure 8: Overhead per query set, 50% memory remote.

To investigate the overhead when using remote memory in more detail, we performed a second test with HANA on a smaller VM, this time using 80 users but varying the amount of remote memory. We repeated the experiment four times, the first two using one memory sponsor and the last two using two sponsors. The distribution of memory between the demander and sponsors is 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 increases performance.

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%

Fast VM live migration: The performance of the Heca hybrid RDMA live migration algorithm was compared with the standard KVM pre-copy algorithm and the Yabusame [4] userspace post-copy algorithm. A synthetic benchmark tailored specifically for evaluation of live migration, *Appmembench*, was used in the evaluation. Appmenbench measures the relative application

performance between 0 and 1 where 0 means the application is frozen and 1 represents full performance. The rate at which Appmembench writes data to the VM's memory, the *dirtying rate*, can be tuned and the tests were run at rate of 10 MB/s, 100 MB/s and 1 GB/s. The migration process was started at $t = 10$ *s* and forced at $t = 70$ *s* if not completed by then. In addition to the relative application performance, the migration downtime was also measured. The results from the tests can be seen in Table 3 and Figure 9. Notably, the precopy KVM algorithms

Table 3: Migration downtime.

Algorithm	10 Mbit/s	100 Mbit/s	1 GB/s
Precopy	364 ms	1256 ms	2765 ms
Post-copy (userspace)	400 ms	655 ms	690 ms
Heca Hybrid	94 ms	104 ms	112 ms

downtime rises considerably with increasing page dirtying rate while the Yabusame postcopy approach shows less increase and the Heca hybrid algorithm has a low, stable downtime. In Figure 9, which shows relative per-

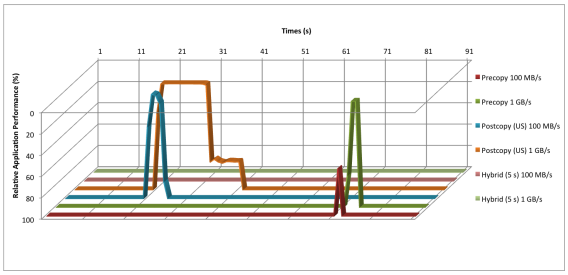

Figure 9: Performance degradation during migration.

formance degradation, the Heca hybrid algorithm shows no degradation at all while at a dirtying rate of 1 GB/s the performance degradation for the Yabusame algorithm is quite severe and lasts for more than 20 seconds. As expected, the precopy algorithm suffers from a spike of performance degradation when both the source and destination hosts are suspended.

5 Conclusion

We propose a solution to enable vertical elasticity, or scale-up, 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 was implemented in a proof-of-concept solution which was validated by an evaluation of memory scale-out and fast VM live migration. The results indicate that resource aggregation is a feasible concept and provides both improved performance and simplified administration, thus enabling a broader range of cloud applications than feasible today.

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] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *NSDI '05: 2nd Symposium on Networked Systems Design and Implementation* (2005), ACM, pp. 273–286.

[2] DESHPANDE, U., SCHLINKER, B., ADLER, E., AND GOPALAN, K. Gang migration of virtual machines using cluster-wide deduplication. In *CCGrid '13: The 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2013), ACM, pp. 394–401.

[3] HINES, M. R., AND GOPALAN, K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: The 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (2009), ACM, pp. 51–60.

[4] HIROFUCHI, T., NAKADA, H., ITOH, S., AND SEKIGUCHI, S. Reactive consolidation of virtual machines enabled by postcopy live migration. In *VTDC '11: Workshop on Virtualization Technologies in Distributed Computing* (2011), IEEE, pp. 11–18.

[5] HUDZIA, B., WALSH, S., TELL, R., ITZAK, P., AND SHRIBMAN, A. Leveraging memory mirroring for transparent memory scale-out with zero-downtime failover of remote hosts. In *ISCC '13: The 2013 IEEE Symposium on Computers and Communications* (2013), IEEE.

[6] KONECNY, P. Introducing the Cray XMT. In *CUG '07: The 2007 Cray User Group meeting* (2007).

[7] KOSSMANN, D., AND KRASKA, T. Data management in the cloud: promises, state-of-the-art, and open questions. *Datenbank-Spektrum 10*, 3 (2010), 121–129.

[8] NOMURA. Nomura VMWare Q4 report. Tech. rep., Nomura Research, 2012.

[9] NUSSLE, M., SCHERER, M., AND BRUNING, U. A resource optimized remote-memory-access architecture for low-latency communication. In *ICPP '09: The 2009 International Conference on Parallel Processing.* (2009), IEEE, pp. 220–227.

[10] OPENSTACK. OpenStack Cloud OS. `https://www.openstack.org`, Visited on 2014-01-29.

[11] SAP. SAP HANA, 2014. `http://bit.ly/GKZkDy`, Visited on 2014-01-29.

[12] SGI. Technical Advances in the SGI UVTM Architecture. `http://www.sgi.com/pdfs/4192.pdf`, Visited on 2014-01-29.

[13] TIPPARAJU, V., NIEPLOCHA, J., AND PANDA, D. Fast collective operations using shared and remote memory access protocols on clusters. In *IPDPS '03: The 2003 international Parallel and Distributed Processing Symposium* (2003), IEEE.

[14] TRELLES, O., PRINS, P., SNIR, M., AND JANSEN, R. C. Big data, but are we ready? *Nature reviews Genetics 12*, 3 (2011), 224–224.

[15] WOOD, T., SHENOY, P. J., VENKATARAMANI, A., AND YOUSIF, M. S. Black-box and gray-box strategies for virtual machine migration. In *NSDI* (2007), vol. 7, pp. 229–242.