VM Economics for Java Cloud Computing

An Adaptive and Resource-Aware Java Runtime with *Quality-of-Execution*

José Simão
Instituto Superior Técnico
INESC-ID Lisboa
Lisbon, Portugal
Email: jsimao@cc.isel.ipl.pt

Luís Veiga
Instituto Superior Técnico
INESC-ID Lisboa
Lisbon, Portugal
Email: luis.veiga@inesc-id.pt

***Abstract*—Resource management in Cloud Computing has been dominated by system-level virtual machines to enable the management of resources using a coarse grained approach, largely in a manner independent from the applications running on these infrastructures. However, in such environments, although different types of applications can be running, the resources are delivered equally to each one, missing the opportunity to manage the available resources in a more efficient and application driven way. So, as more applications target managed runtimes, high level virtualization is a relevant abstraction layer that has not been properly explored to enhance resource usage, control, and effectiveness.**

We propose a VM economics model to manage cloud infrastructures, governed by a quality-of-execution (QoE) metric and implemented by an extended virtual machine. The *Adaptive and Resource-Aware Java Virtual Machine* (ARA-JVM**) is a cluster-enabled virtual execution environment with the ability to monitor base mechanisms (e.g. thread scheduling, garbage collection, memory or network consumptions) to assess application's performance and reconfigure these mechanisms in runtime according to previously defined resource allocation policies. Reconfiguration is driven by incremental gains in *quality-of-execution* (QoE), used by the VM economics model to balance relative resource savings and perceived performance degradation.**

Our work in progress, aims to allow cloud providers to exchange resource slices among virtual machines, continually addressing where those resources are required, while being able to determine where the reduction will be more economically effective, i.e., will contribute in lesser extent to performance degradation.

I. Introduction

In recent years, the use of Grids, Utility and Cloud Computing, shows that these are approaches with growing interest, as well as scientific and commercial impact. At the same time, managed object oriented languages (e.g., Java, C#) are becoming increasingly relevant in the development of large scale solutions, leveraging the benefits of a virtual execution environment to provide modular, reconfigurable and robust solutions.

The fusion of these two topics is a very active research area, and solutions have been proposed to federate Java virtual machines (either extended VMs or supported on middleware), aiming to provide a single system image [1]. If this image has elasticity in the sense that resources are made available proportionally to the effective need, and if these resources are accounted/charged as they are used, we can provide an object oriented virtual machine (OO-VM) across the cluster, as an utility. If these changes are made dynamically (instead of explicitly by their users) we will have an adaptive and resource-aware virtual machine, that can be offered as a value-added Platform-as-a-Service (PaaS).

Therefore, such a cluster-enabled managed environment can adapt itself to the execution of applications, from multiple tenants, with different (and sometimes dynamically changing) requirements in regard to their quality-of-execution (QoE). QoE aims at capturing the adequacy and efficiency of the resources provided to an application according to its needs. It can be inferred coarsely from application execution time for medium running applications, or request execution times for more service driven ones such as those web-based, or from critical situations such as thrashing or starvation. Also, it can be derived with more fine-grain from incremental indicators of application progress, such as execution phase detection [2], memory pages updates, amount of input processed, disk and network output generated. In our ongoing work, we are still focusing only on application execution times.

QoE can be used to drive a VM economics model, where the goal is to incrementally obtain gains in QoE for VMs running applications requiring more resources or from more privileged tenants. This, while balancing the relative resource savings drawn from other tenants' VMs with perceived performance degradation. To achieve this goal, we must be able to positively discriminate certain applications, and for this, on others we may need to restrict resource usage, imposing limits to their consumption, regardless some performance penalties (that should also be mitigated). Additionally, we can reconfigure the mechanisms and algorithms that support their execution environment (or even engaging available alternatives to these mechanisms/algorithms) [3]. In any case, these changes should be transparent to the developer and specially to the application's user.

Our research work plan addresses the extension of high-level language virtual machines (e.g., Java VMs such as Jikes RVM [4] and OpenJDK) to operate more flexibly and efficiently in multi-tenancy scenarios such as those of cloud computing infrastructures. This entails three sets of requirements:

1) Enhancing current VMs with capabilities to accurately monitor resource usage and enforce constrains in re-

source management mechanisms;
2) Empower VMs with elasticity and horizontal scaling allowing VM runtimes to dynamically and transparently span several physical machines (or system-level virtual machines);
3) Enable overall resource management to be driven by finer-grained transfer of resources among tenants, trying to reconcile QoE parameters and instant resource consumption.

We propose an *Adaptive and Resource-Aware Java Virtual Machine* (ARA-JVM), a cluster-enabled virtual execution environment with the ability to monitor base mechanisms (e.g. thread scheduling, garbage collection, memory or network consumptions) to assess application's performance. Armed with this information, *ARA-JVM* decides how to reallocate (and if needed reconfigure) such mechanisms in runtime according to previously defined resource allocation policies. At a lower-level, the cluster-enabled runtime must have the ability to monitor base mechanisms to assess application's performance and the ability to reconfigure these mechanisms in run-time. At a higher-level, we drive resource adaptation according to a VM economics model based on aiming overall quality-of-execution (QoE) through resource efficiency, e.g., expressed in previously defined resource allocation policies and priorities.

ARA-JVM operates by continuously awarding resources to those tenants requiring or entitled to more, but while incrementally drawing them from the tenants where resource scarcity will hurt performance the least. In essence, putting resources where they can do the most good to applications and the cloud infrastructure provider, taking them from where they can do the least harm to applications.

II. Related Work

Adaptability is a vertical activity in current systems stack. System-wide VMs, high level language VMs and special propose middleware can all make decisions based on different profiling information, adapting some of their internal mechanisms to improve the system performance in a certain metric.

These three levels of virtualization have different distances to the machine-level resources, with a increasing distance from system-wide VMs to special purpose middleware. The dual of this relation is the transparency of the adaptation process from the application perspective. A system-wide VM aims to distribute resources with fairness, regardless of the application patterns or workload. On the other end is the middleware approach where applications use a special purpose programming interface to specify their consumption restrictions. As more applications target high level language VMs, including the ones running on cloud data centers, this virtualization level, which our work encompasses, has the potential to influence high impact resources (akin to system-wide VMs), using application's metrics (akin to the middleware approach) but still with full transparency.

In this section we survey work related to these three virtualization levels, focusing on adaptations whose goal is to improve the application performance by adapting the infrastructure mechanisms.

a) System Virtual Machines: Shao et al. [5] adapts the VCPU mapping of Xen [6] based on runtime information collected by a monitor that must be running inside each guest's operative system. They adjust the numbers of VCPUs to meet the real needs of each guest. Decisions are made based on two metrics: the average VCPU utilization rate and the parallel level. The parallel level mainly depends on the length of each VCPU's run queue. The adaptation process uses an addictive increase and subtractive decrease (AISD) strategy. Shao et al. focus their work on specific native applications. We believe our approach has the potential to influence a growing number of applications that run on high level virtual machines and whose performance is also heavily dependent on memory management.

In [7], Sharma et al. present a way to dynamically provision virtual servers in a cloud provider, based on pricing models. They target application owners (i.e. suppliers of services to end users) who want to select the best configuration to support their peak workloads (i.e. maximum number of requests per second successfully handled), minimizing the cost for the application's owner. Sharma's work uses different mechanisms to guarantee the provisioning of resources, which include: readjusting CPU, memory caps and migration. To make good decisions they need to know, for each application, what is the peak supported by each provider's configuration, which is dependent on real workloads. Furthermore, because changes to virtual servers configuration is driven by the number of requests per second, it can miss the effective computation power needed by each request.

b) High Level Virtual Machines: High level virtual machines have been augmented or designed from scratch to integrate resource accounting [8], [9]. MVM [9] is based on the Hotspot virtual machine. It supports isolated computations, akin to address spaces, to be made in the same instance of the VM. This abstraction is called *isolate*. Another distinguishing characteristic is the capacity to impose constrains regarding consumption of *isolates*. The resource management done in MVM is related to the Java Specification Request 284 [10]. Our work builds on this JSR and uses a widely accessible VM (MVM only runs on Solaris on top of SPARC's hardware). The work in [8] enables precise memory and CPU accounting. Nevertheless they do not provide an integrated interface to determine the resource consumption policy, which may involve VM, system or class library resources.

Garbage collection is known to have different performance impacts in different application [11]. Several strategies have been used to improve execution time of a program running in a high level virtual machine. Historically this improvement has been accomplished by new algorithms. Recent work takes advantage of interactions with the operative system (e.g. Hertz et al. [12] try to avoid page faults) and experiment with different configuration for a family of algorithms (e.g. Soman et al. [13] switch the GC algorithm at previously defined points or taking into account the available memory).

Singer et al. [14] discuss the economics of GC, relating heap size and number of collections with the price and demand law of micro-economics - with bigger heaps there will be less collections. This relation extends to the notion of elasticity to measure the sensitivity of the heap size to the number of GCs. They devise an heuristic based on elasticity to find a tradeoff between heap size and execution time.

In [15] the GC is auto-tuned in order to improve the performance of a MapReduce [16] Java implementation for multi-core hardware. For each relevant benchmark, machine learning techniques are used to find the best execution time for each combination of input size, heap size and number of threads in relation to a given GC algorithm (i.e. serial, parallel or concurrent). Their goal is to make a good decision about a GC policy when a new MapReduce application arrives. The decision is made locally to an instance of the JVM.

Our work is also related to memory management inside a high level virtual machine, but the definition of QoE (as presented in Section I and further detailed in Section III) expands beyond this resource and can be used in other levels of the execution stack.

c) Middleware: Duran et al. [17] uses a thin high level virtual machine to virtualize CPU and network bandwidth. Their goal is to provide an environment for resource management, that is, resource allocation and/or adaptation. Applications targeting this framework use a special purpose programming interface to specify reservations and adaptation strategies. When compared to more heavyweight approaches like system VMs, this lightweight framework can adapt more efficiently for I/O intensive applications. Though, the approach taken in Duran's work bounds the application to a given resource adaptation interface.

Although in our system the application (or the libraries they use) can also impose their own restrictions, the adaptation process is mainly driven by the underlying virtual machine without direct intervention of the applications.

III. Proposed approach

The architecture of ARA-*JVM* is presented in Figure 1. Our vision is that ARA-*JVM* will execute applications with different requisites regarding their quality-of-execution (QoE). Target applications have typically a long execution time and can spawn several execution flows to parallelize their work. This is common in the field of science supported by informatics like economics and statistics, computational biology and network protocols simulation.

ARA-*JVM* is supported by several runtime instances, each one cooperating to the sharing of resources. For an effective resource sharing, a global mechanism must be in place to make weak (e.g. change parameters) or strong (e.g. change GC algorithm, migrate running application) adaptations [3]. The first building block above the operating system in each node is a process-level managed language virtual machine, enhanced with services that are not available in regular VMs. These services include: i) the accounting of resource consumption, ii) dynamic reconfiguration of internal mechanisms, and iii) mechanisms for checkpointing, restore and migration of the whole application. These mechanisms should and must be made available at a lower-level, inside an extended VM, for reasons of control, interception and efficiency.

The second building block aggregates individual VMs, as the ones described above, to form a cluster with a distributed shared object space. It gives running applications support for a single system image semantics, across the cluster, regarding the object address space. Techniques like byte code enhancement/instrumentation or rewriting are used, so that unmodified applications can operate in a partitioned global address space, where some objects exist only as local copies and others are shared in a global heap.

Data collected from running application on top of ARA-*JVM* can be used as input to a policy decision point, where policies are evaluated in order to determine a certain rule outcome. The other purpose of collecting this data is to infer a profile for a given application. Such profiles will result in the automatic use of policies for a certain group of applications, aiming to improve their performance. The effects, positive or negative, of applying such policies are then used to confirm, or reject, the level of correlation between the profile and the applications.

Yield driven adaptation model:

Our current research work takes an infrastructure-centric approach in the sense that we want to transparently transfer resource allocation between applications running in the cluster, minimizing the perceived impact in their execution. We advocate a way for each application owner to specify that, when the application is executing in a constrained environment, the infrastructure may remove m units of a given resource, from a set of resources R, and give it to another application that can benefit from this transfer. This transfer may have a negative impact in the application who offers resources (although intended to be the minimum across the possible alternatives), while it is expected to have a positive impact in the receiving application. To assess the effectiveness of the transfer, the infrastructure must be able to measure the impact on the giver and receiver applications.

For each controlled resource, ARA-*JVM* dynamically adapts its parameters to make an efficient management of resources in the cluster. In general, there is a *yield* regarding a given resource R_j from the set of resources R, and a management strategy S_x, i.e., a return or reward from applying a given strategy to some managed resource. Given that the *yield* may be known only partially, for a given time span ts, as the application executes continually, we define it as:

$$Yield_{ts}(R, S_a, S_b) = \frac{\text{resource savings}(R, S_a, S_b)}{\text{performance degradation}(S_a, S_b)} \quad (1)$$

The *resource savings* represents the savings of a given resource when two allocation or management strategies are compared. Regarding *performance degradation*, it represents

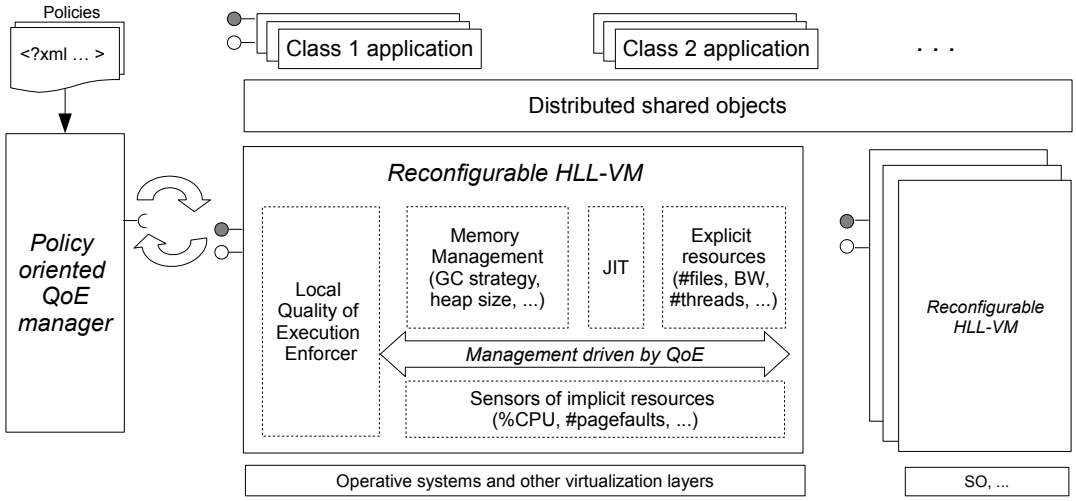


Fig. 1: Architecture of the ARA-*JVM*

the impact of the savings, given a specific performance metric. Considering the time taken to execute an application (or part of it), the performance degradation relates the execution time of the original configuration and the execution time after the resource allocation strategy has been modified.

For a given execution or evaluation period, the total *yield* is the result of summing all significant partial yields:

$$Yield(R, S_a, S_b) = \sum_{ts=0}^{n} Yield_{ts} \tag{2}$$

In addition to periodic evaluation, phase detection in managed programs [2] could also be used to trigger evaluation and adaptation. Phase detection is a well researched topic, and is typically used to drive JIT compiler optimizations. Nevertheless, these techniques could be used to change the strategy used in other components of the VM, like the garbage collector, or instruct lower virtualization layers (e.g. operative system, hypervisor) to change their policies (e.g. scheduling).

IV. On going development and results

In our current research work, we are addressing two key lines of work to incorporate the QoE metric and the VM economics model in virtual machines. Currently, we are working on:

- Having a managed language virtual machine with the capacity to monitor and restrain the use of resources, based on a dynamic policy, defined declaratively outside the VM;
- Finer-grained transfer of resources among tenants, using different strategies in managing the resource consumption decision inside the virtual machine.

Resource management inside a managed language VM:

We have chosen to extend the Jikes RVM [4], a research Java Virtual Machine, to be resource-aware. The resources that can be monitored in a virtual machine can be either specific of the runtime (e.g. number of threads, number of objects), which we call *intrinsic resources*, or be strongly dependent on the underlying operating system (e.g. CPU usage), which we call *extrinsic resources*.

To unify the management of such disparate types of resources, we have implemented JSR 284 - The Resource Management API [10] - in the context of Jikes RVM. This API was designed to be used by applications running on top of a Java VM, so that they can determine the resource consumption policy.

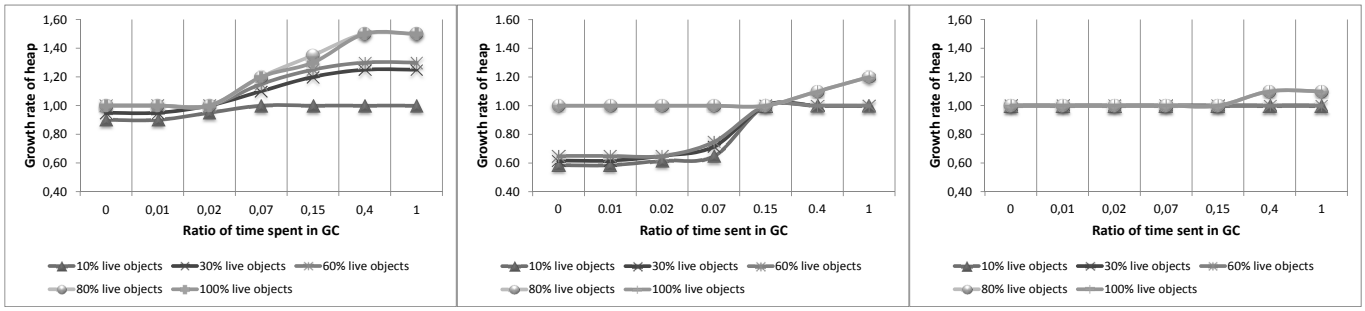
We propose to use the same principles for managing internals components of the virtual machine, transparently to the applications. So, we have defined a XML syntax to express the following policy elements: resource consumption event (e.g. garbage collection triggered, new thread created), action when the event happens, action if the event is allowed (i.e. if the resource can be consumed), action if the event is denied (e.g. change GC parameter, throw exception, change thread allocation site). Currently the policy must be loaded when the VM starts but we intend to give the possibility of changing it during runtime.

Experiments with memory management:

The research runtime Jikes RVM [4] uses a built-in matrix to determine how the heap will grow, shrink or remain unchanged, after a memory collection. The growing factor is determined by the ratio of live objects and the time spent in the last GC operation.For example, a growing rate of 1.0 will maintain the same heap size, while a growing rate of 1.2 or 0.8 will increase or decrease the heap size in 20%, respectively.

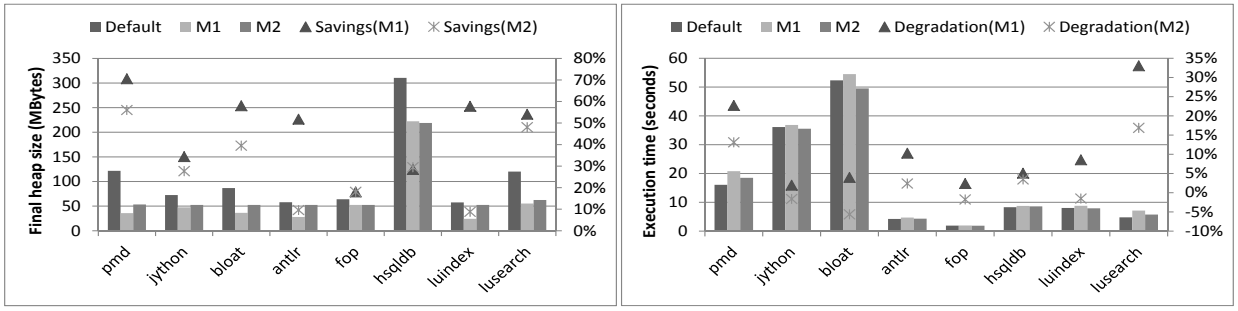
Preliminary Results:

Figure 2.a shows the default growing rates of the heap for each series of live objects. The default rates determine that the heap shrinks about 10% when the time spent in GC is low (less than 7%) when compared to regular program execution, and the ratio of live objects is also low (less than 50%). This allows for savings in memory used. On the other hand, the heap will



(a) Default matrix (b) Matrix $M1$ (c) Matrix $M2$

Fig. 2: Different heap growing rate matrices which have a different returned *yields*



(a) Resource savings (b) Performance degradation

Fig. 3: These two groups of graphics correspond to DaCapo's `large` configuration in which we used $heap_{min} = 53Mbytes$ and $heap_{max} = 315Mbytes$.

grow for about 50%, as the time spent in GC also grows and the number of live objects remains high. This growth in heap size will lead to an increase in memory used by the runtime, aiming to use less CPU because the GC will run less frequently.

To assess the benefits of our resource management economics we have setup two new heap size changing matrices, which we call *heap saving matrices*. The distinctive factor is the growth/decrease rate determined by each matrix. While matrix M_1, presented in Figure 2.b imposes a strong reduction on the heap size when memory usage and management activity is low (i.e. few live objects and short time spent on GC), matrix M_2, in Figure 2.c, enforces a constant heap size unless the program dynamics reach an high activity point (i.e. high rate of live objects and longer time spent on GC).

Figure 3 shows some preliminary results of running different applications from the DaCapo benchmarks [18] (with configuration `large`) using the default matrix and the two new previously presented. Figure 3.a (left axis) shows the final heap size after running the benchmarks with the three different matrices. Series *Savings (M1)* and *Savings (M2)* (right axis) represent the *resource savings* obtained for each of these matrices ($M1$ and $M2$) when compared to the default matrix. These savings go up to 71%. The minimum saving in this configuration is 18%. In Figure 3.b (left axis) we present the evaluation time of the benchmarks and the *performance degradation* (right axis) regarding the use of each of the ratio matrices. Degradation of execution time reaches a maximum of 35% for `lusearch` but below 10% for most of the remainder benchmarks.

From these experiments and the collected results we can see that the returned *yield* has different magnitudes across applications (e.g. `jython` 17.56, `pmd` 3.10), but has always a "positive" impact, that is, percentual resource savings always overcome percentual performance degradation, by a factor never lower than 1.85. We think these experiments demonstrate the usefulness of applying different strategies to specific applications.

V. Conclusion and Future Work

In this paper, we described the ongoing research to design an *Adaptive and Resource-Aware Java Virtual Machine* (*ARA-JVM*). Resource allocation and adaptation obeys to a VM economics model, based on aiming overall quality-of-execution (QoE) through resource efficiency, e.g., expressed in previously defined resource allocation policies. The QoE model governs cloud infrastructures to continuously exchange (more fine-grained) resource slices among virtual machines, awarding resources to those tenants requiring or entitled to more, while being able to determine where the resource reduction will be more economically effective, i.e., will contribute in lesser extent to performance degradation.

We presented the details of our adaptation mechanisms in each VM. Preliminary experimentations where done to manage memory. The benefits were evaluated, showing resources can be reverted among applications, from where they hurt performance the least (higher *yields* in our metrics), to more higher priority or requirements applications.

The more vast goal is to improve flexibility, control and efficiency of infrastructures running long applications in clusters. To this end we have some challenges to address: *i)* determine how application's phases can influence our economic model; *ii)* determine how the model can be applied to control other layers of the virtualization stack, such as the hypervisor; *iii)* enhance the model to take into account several running applications.

Acknowlegments*:* This work was partially supported by national funds through FCT Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113613/2009, and PEst-OE/EEI/LA0021/2011

REFERENCES

[1] W. Zhu, C.-L. Wang, and F. C. M. Lau, "JESSICA2: A distributed Java virtual machine with transparent thread migration support," *Cluster Computing, IEEE International Conference on*, vol. 0, p. 381, 2002.

[2] P. Nagpurkar, C. Krintz, M. Hind, P. F. Sweeney, and V. T. Rajan, "Online phase detection algorithms," in *Proceedings of the International Symposium on Code Generation and Optimization*, ser. CGO '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 111–123. [Online]. Available: http://dx.doi.org/10.1109/CGO.2006.26

[3] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, pp. 14:1–14:42, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[4] B. Alpern, S. Augart, S. M. Blackburn, M. Butrico, A. Cocchi, P. Cheng, J. Dolby, S. Fink, D. Grove, M. Hind, K. S. McKinley, M. Mergen, J. E. B. Moss, T. Ngo, and V. Sarkar, "The Jikes research virtual machine project: building an open-source research community," *IBM Syst. J.*, vol. 44, pp. 399–417, January 2005. [Online]. Available: http://dx.doi.org/10.1147/sj.442.0399

[5] Z. Shao, H. Jin, and Y. Li, "Virtual machine resource management for high performance computing applications," *Parallel and Distributed Processing with Applications, International Symposium on*, vol. 0, pp. 137–144, 2009.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 164–177, October 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[7] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 559–570. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2011.59

[8] G. Back and W. C. Hsieh, "The kaffeos Java runtime system," *ACM Trans. Program. Lang. Syst.*, vol. 27, pp. 583–630, July 2005. [Online]. Available: http://doi.acm.org/10.1145/1075382.1075383

[9] G. Czajkowski, S. Hahn, G. Skinner, P. Soper, and C. Bryce, "A resource management interface for the Java platform," *Softw. Pract. Exper.*, vol. 35, pp. 123–157, February 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1055953.1055955

[10] G. C. et al., *Java Specification Request 284 - Resource Consumption Management API, http://jcp.org/en/jsr/detail?id=284*, Sun Microsystems Std., 2009.

[11] F. Mao, E. Z. Zhang, and X. Shen, "Influence of program inputs on the selection of garbage collectors," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1508293.1508307

[12] M. Hertz, S. Kane, E. Keudel, T. Bai, C. Ding, X. Gu, and J. E. Bard, "Waste not, want not: resource-based garbage collection in a shared environment," in *Proceedings of the international symposium on Memory management*, ser. ISMM '11. New York, NY, USA: ACM, 2011, pp. 65–76. [Online]. Available: http://doi.acm.org/10.1145/1993478.1993487

[13] S. Soman and C. Krintz, "Application-specific garbage collection," *J. Syst. Softw.*, vol. 80, pp. 1037–1056, July 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.12.566

[14] J. Singer, R. E. Jones, G. Brown, and M. Luján, "The economics of garbage collection," *SIGPLAN Not.*, vol. 45, pp. 103–112, June 2010. [Online]. Available: http://doi.acm.org/10.1145/1837855.1806669

[15] J. Singer, G. Kovoor, G. Brown, and M. Luján, "Garbage collection auto-tuning for java mapreduce on multi-cores," in *Proceedings of the international symposium on Memory management*, ser. ISMM '11. New York, NY, USA: ACM, 2011, pp. 109–118. [Online]. Available: http://doi.acm.org/10.1145/1993478.1993495

[16] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[17] H. Duran-Limon, M. Siller, G. Blair, A. Lopez, and J. Lombera-Landa, "Using lightweight virtual machines to achieve resource adaptation in middleware," *IET Software*, vol. 5, no. 2, pp. 229–237, 2011. [Online]. Available: http://link.aip.org/link/?SEN/5/229/1

[18] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The dacapo benchmarks: Java benchmarking development and analysis," in *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 2006, pp. 169–190.