Batch Job Profiling and Adaptive Profile Enforcement for Virtualized Environments

Yolanda Becerra, David Carrera and Eduard Ayguadé
Technical University of Catalonia - Barcelona Supercomputing Center
Barcelona, Spain
{yolandab, dcarrera, eduard}@ac.upc.edu

Abstract—**Data center management is driven by high-level performance goals, and it is the responsibility of a management middleware to ensure that those goals are met using dynamic resource allocation. The performance delivered by the heterogeneous set of applications running in a virtualized enterprise data center must be predicted to make resource allocation decisions. For some of these applications, it is required to produce accurate profiles based on previous executions: that is the case of batch jobs. In this paper we propose a methodology to produce resource consumption profiles for batch applications running inside of virtual machines and a technique to enforce and adapt the profiles to actual execution conditions and application performance. For this purpose we have developed a testing prototype. The enforcement technique observes the fact that management middleware usually run in control cycles in which the system can be reconfigured, what imposes a tradeoff between the accuracy of the profiles and their applicability in real deployments. The novel contribution of this work is the study of the tradeoff between accuracy and applicability of workload profiles, what is a necessary step to enable existing management middleware with the performance prediction mechanisms required to perform effective dynamic resource allocation.**

I. Introduction

Server consolidation and virtualization strategy has lead to the appearance of increasingly larger virtualized data centers in which companies and organizations rely to run their applications and deliver services to customers. Such a complex scenario, involving a large number of virtual machines (VMs) running a heterogeneous set of applications, requires advanced management techniques to deliver maximum performance through dynamic resource allocation. These techniques are usually provided by an advanced management middleware.

The overall performance of a data center is defined by the particular performance delivered by each one of the applications deployed on it. The management middleware continually adjusts the performance of the data center by deciding the amount of resources allocated to each virtual machine in the system. This decision must be made based on the expected performance that each application will offer when a particular resource allocation is decided. Thus, the performance delivered by the applications that run in an enterprise data center must be predicted at the time that resource allocation decisions are made. For some applications, the performance prediction is based on the observed workload patterns; that is the case of OLTP workloads. For other applications, it is required to produce accurate profiles based on previous executions; that is the case of batch jobs.

Although workload profiling is an already studied topic in the literature, running workloads inside of VMs introduces a number of new issues to be addressed, specially because the flexibility found in resource provisioning for virtualized environments is taken one step forward. Now, resources can be allocated in an unprecedented fine grain level, and not only limited to CPU and memory consumption, but also covering network and disk bandwidth. That forces management middleware to make decisions regarding not only the amount of resources to allocate to one VM based on the desired performance level, but also in how different resources are consumed by the applications. For example, allocating a large amount of memory for a CPU-bound application can result in resource underutilization if both memory and CPU are not allocated in the correct proportion for the application to make significant progress.

Our current work focuses on single-threaded batch jobs that periodically run with similar behavior between executions in a large and virtualized enterprise data center. Examples of applications that present such a behavior are document indexing applications, that periodically run over similar inputs that grow slowly in comparison to the total volume of data; and numerical applications that perform some kind of statistical calculation for a limited period of time. It is out of the scope of our current work the analysis of black-box VMs (VMs running unknown workloads). Anyway, the techniques presented in this paper as well as some of the conclusions, are also useful for such an environment that is part of our future work.

In this paper we propose a methodology to produce resource consumption profiles and a technique to enforce and adapt the profiles to both actual execution conditions and application performance. We validate our proposal through execution in a real system enabled with state-of-the-art virtualization technology (VMWare ESX 3.5 [9]). For this purpose we have developed a testing prototype that allows real-time monitoring of the application performance and resource consumption and enforcement of previously generated profiles. An extended version of this paper can be found in [2].

The novel contribution of this work is the study of the tradeoff between accuracy and applicability of workload profiles, what is a necessary step to enable existing management middleware with the performance prediction mechanisms required to perform effective dynamic resource allocation.

1066-6192/09 $25.00 © 2009 IEEE
DOI 10.1109/PDP.2009.55

IEEE computer society

II. Profiling and resource management infrastructure

A. System architecture

In this section we introduce the architecture of the prototype system developed to produce application profiles as well as to enforce these profiles in applications' following executions.

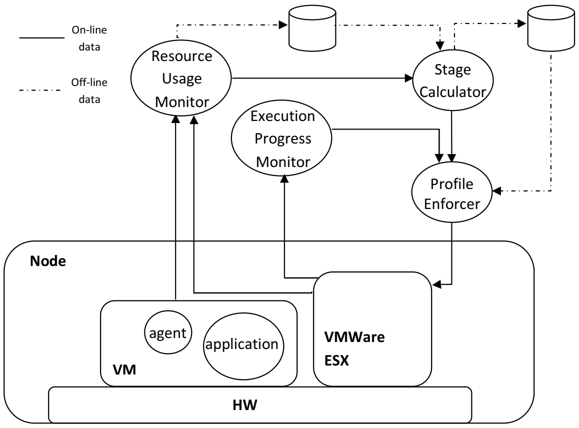

Fig. 1. Architecture of the system

This prototype consists of four main components (see figure 1), following described:

- Monitor: this process feeds the profiling architecture with a complete description of the resource consumption of an application. Our monitor communicates both with the VMWare ESX, to obtain performance counters [10], and with the guest operating system, to collect information regarding memory consumption that is only visible from inside the guest OS. In our system we assume that one single application is running inside each VM. The accuracy of this data is limited by the minimum sample length of VMWare ESX (20 seconds), which computes the average resource consumption over each sample.
- Stage calculator: this process calculates execution stages for the application, based on the resource usage reported by the monitor. Each stage is assigned a relative starting time (defined by the work progress to be done by the application before starting the stage) and characterized by the average level of consumption for each resource.
- Profile enforcer and Execution progress monitor: the stage calculator provides the profile enforcer process with the application profile. This process communicates with VMWare ESX to change the VM configuration according to the current stage definition. We have to consider that running an application with a resource allocation different than at profile-generation time, can change the resource usage of the application and thus, can change the actual length of each stage. In order to adjust stage starting time, we monitor the execution progress of the application (based on CPU consumed cycles).

Our prototype can also operate using off-line data stored in files. This way, we can re-enforce previously generated profiles.

B. Stage-detection algorithm

In this section we describe the methodology used to create workload profiles as well as the formal definition of profile. In the scope of our work, a workload profile is defined as a sequence of stages $WP = (S_0, \ldots, S_i)$, each one characterized by the amount of resources consumed by the workload in a particular period of time.

Let the resource consumption of the application on a given time t be $R_t = (C_t, M_t, D_t, N_t)$, where each component represents the amount of CPU (C_t), memory (M_t), disk bandwidth (D_t) and network bandwidth (N_t) consumed by the application. Then, a stage S_i is represented by the average resource consumption in a given period of time ($t_0 - t_n$):

$$\begin{aligned} S_i \;=\; \overline{R_{S_i}} &= \{avg(R_{t_0}, \ldots, R_{t_n})\} \qquad (1) \\ &= \{avg(C_{t_0}, \ldots, C_{t_n}), avg(M_{t_0}, \ldots, M_{t_n}), \\ &\quad avg(D_{t_0}, \ldots, D_{t_n}), avg(N_{t_0}, \ldots, N_{t_n})\} \end{aligned}$$

And we define the standard deviation of the stage as:

$$\begin{aligned} \sigma_{S_i} &= (\sigma(C_{t_0}, \ldots, C_{t_n}), \sigma(M_{t_0}, \ldots, M_{t_n}), \qquad (2) \\ &\quad \sigma(D_{t_0}, \ldots, D_{t_n}), \sigma(N_{t_0}, \ldots, N_{t_n})) \end{aligned}$$

Let current time be t_{n+1} and let the tuple representing the current resource consumption of an application be $R_{t_{n+1}}$. At this point, the stage calculator has to decide if $R_{t_{n+1}}$ is part of last stage in the workload profiling (S_i) or if it is necessary to add a new stage (S_{i+1}) to it. The condition that has to accomplish $R_{t_{n+1}}$ to be part of S_i is:

$$distance(\overline{R_{S_i}}, R_{t_n+1}) < th * \sigma_{S_i} \qquad (3)$$

Where th is a threshold determined experimentally.

Thus, the function to update the workload profiling is:

$$update(WP) = \begin{cases} S_i \leftarrow S_i \cup \{R_{t_{n+1}}\} & \text{if (3) is true} \\ \\ S_{i+1} = \{R_{t_{n+1}}\} & \\ WP = (S_0, \ldots, S_i, S_{i+1}) & \text{otherwise} \end{cases}$$

Thus the stage length is considered to be variable and depends on the particular resource usage made by each application at each moment.

We have to consider that the profiles produced are supposed to be enforced by some management middleware [3] that usually operates in fixed-length control cycles. Thus, each stage should represent at least the behavior of a time period equivalent to one control cycle of the middleware. For this reason, we define a minimum stage length value (MSL henceforth) that represents the minimum time that any stage must last. Thus each resource consumption tuple contains the information about the resource consumption of a MSL period time: $R_t = \overline{(R_{t_0}, \ldots, R_{t_{MSL}})}$.

The minimum value for the MSL is limited by the minimum interval sample of the monitor (which is limited by the VMWare ESX sample period of 20 seconds). In order to decide which MSL is the most suitable for each profile generation we have to evaluate the trade-off between accuracy and performance. The largest the MSL value is the least accurate the profile is (see section III-B).

C. Profile-enforcement technique

Once the profile is generated we use the information about the execution stages to enforce the resource provisioning for the application. For each generated stage, the profile enforcer reconfigures the VM as specified in the stage information.

In most virtualization platforms, resources can be allocated by defining either absolute values or share values. We support both approaches in our system.

As far as an application profile is generated under certain levels of resource provisioning that may not be repeated in following executions of the same application, a profile enforcement technique must be able to adapt the stage length to the new execution conditions on real time. For instance, if an application is running more slowly than during the execution used to generate the profile, the system will have to stretch the stages in order to perform the VM reconfiguration at the time it may be required.

In our system, the CPU resource is taken as the driving resource to produce such an effect of stage-length adaptation. The system continuously monitors the CPU consumption during the execution of the application, compares the progress made by the application to the profile-generation execution, and decides to stretch or shrink stages length accordingly.

III. Experiments

In our experiments we use two different applications representative of the scenario we aim in this work: Apache Lucene [1] and the JavaGrande Montecarlo [5] simulation (class B). Apache Lucene is an open-source text search engine library written entirely in Java. In our experiments, we have run the example indexing application provided with the Lucene library to index a large set (800 MB) of files previously deployed in the filesystem. The JavaGrande montecarlo simulation is a financial simulation, using Monte Carlo techniques to price products. Each application is run inside a virtual container and is considered to be running alone in the VM. Workload profiling is done over the entire VM in which the application runs, as we consider that all the VM activity is due to the application behavior.

We used VMWare ESX 3.5 [9] as the virtualization technology. The machine we used as the host is a 8-way 2.6Ghz Intel Xeon (enabled with Intel VT [4]) system with 16GB of RAM and 4 Ultra320 SCSI disks. The guest VMs were running 64-bit Debian GNU/Linunx 4.0 with kernel version 2.6.18-6 for AMD64 architectures.

A. Profile creation: data aggregation methodology and risks

In this experiment we illustrate the behavior of our stage calculation technique. Figure 2 shows the results obtained from two different execution of the JavaGrande montecarlo benchmark, that should exactly produce the same behavior. For the first execution (shown in top row), the VM is set with a memory cap of 768MB, which is enough memory to hold the working set of the application that is around 600MB; for the second run, the VM is slightly under-provisioned with a memory cap of 512MB. In both executions the VM is set with a CPU cap of 2000Mhz. We show data for memory consumption over time, indicated with a solid line (CPU data can be found in [2]). In the same figure we include the level of resources that must be provisioned according to our profiling infrastructure and stage-detection mechanism. We consider three different MSL values: 20s, 400s and 1600s. Looking at the results it can be observed that the higher the MSL value is, the less accurate the profiles result. Notice that for an MSL of 20s, the value calculated for each stage is mostly overlapped with the profile value. The reason for this is that, since the mechanism is based on averaging techniques, when the resource consumption is strongly varying the relative error between the stage-calculated value and the actual resource consumption level observed increases. From our experiments, we observed that using stage lengths in the sub-minute range is impracticable. At the same time, it is of common use to run management middleware in periodic control-length cycles, and these cycles tend to be in the order of the minutes [3].

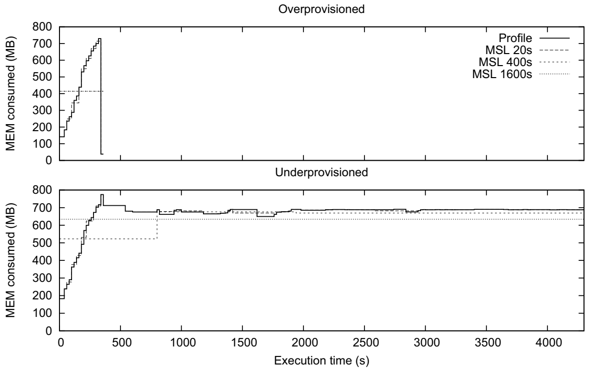

Fig. 2. Montecarlo: consumption and stage calculation

The effect of under-provisioning a VM can be seen in the same experiment. When the working set of the application does not fit in the allocated memory (lower row in figure 2), the execution time grows quickly, and due to the swapping activity of the host, the CPU-consumption measured for the guest drops from 1Ghz to a value around 200Mhz (see [2] for details). Memory under-provisioning causes the most important performance degradation, compared to the original performance observed at time of profile collection. Under-provisioning other resources result in not so dramatic effects, as it will be discussed in more detail in section IV.

B. Profile enforcement: stage adaptation

The system must be able to adapt the stage length to the current execution conditions. In this experiment we illustrate how this process takes place.

Figure 3 shows an execution scenario for the lucene application, that is CPU and disk intensive, in which the use of the disk is severely constrained for a period of time as compared to the original profile-generation execution. In the top chart it can be observed the original generated stages, with dotted line, as well as the actual CPU consumption observed at the time the profile is enforced, with solid line. Notice that the level of CPU usage is much lower than it was expected according

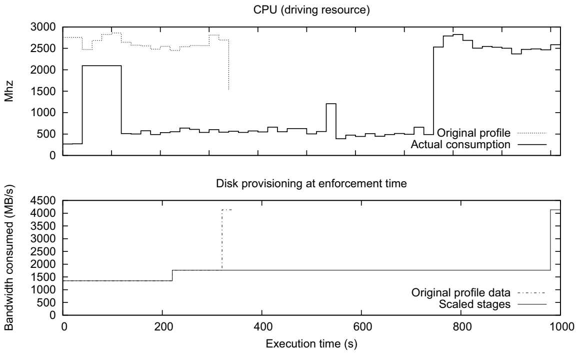

Fig. 3. Lucene: stage adaptation

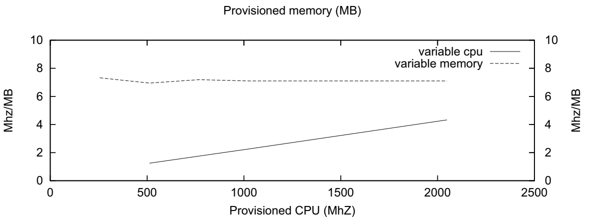

Fig. 4. Lucene performance model

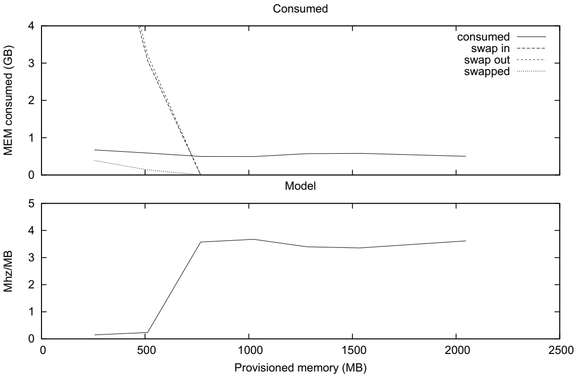

Fig. 5. Montecarlo performance model

to the application profile. This is caused by the constrained access to the disk, because both CPU and disk are consumed proportionally.

The bottom chart shows, for the same execution, the stages that were initially calculated (dashed line) as well as the actual stages applied at enforcement time (solid line). While the first stage is applied exactly as it was defined in the profile (both lines completely overlap), when the second stage is in progress it is stretch to last much longer that is was defined in the profile. As the CPU resource is clearly underutilized compared to the profile prediction, the system adapts the stage to the new applications' progress speed and thus, the profile enforcement is adapted accordingly.

In this experiment we only show stage adaptation for the disk resource due to space limitations, but notice that each stage characterizes the provisioning levels for all the resources considered in the system, so exactly the same is happening with network, CPU and memory provisioning in this case.

C. *Provisioning with shares*

We have evaluated the effect of provisioning resources through resource shares instead of absolute allocation values. Due to space limitations we can not include a detailed description of this experiment (see [2] for details). In brief, results for this experiment showed that, although resource provisioning is calculated correctly, resource shares are not perfectly guaranteed over the experiment, and the applications get different effective resources than they require. In conclusion, working with shares must be done using higher security margins to avoid severe under-provisioning of resources.

IV. Resource consumption model

In this section we show that beyond workload profiles, performance models are required for performing accurate dynamic resource allocation, allowing the management middleware to scale the resources of the VMs proportionally without changing the bounding resource for the application, what would result in a completely unpredictable performance.

In our study, as well as in other works discussed in section V, we assume that there is a linear relation between the consumption of different resources, resulting that resource allocations will have to be done proportionally for each resource. It is particularly remarkable that in our experiments we observed that while such a linear model works fine in a wide range of scenarios (see [2] for details), there is a particular scenario in which it does not work: the point at which applications start causing memory swapping due to large memory consumption.

In a first experiment we run lucene in a number of different configurations of CPU and memory allocation. First we run lucene in a number of memory allocation scenarios (from 128Mb to 2048MB), and 2048Mhz of CPU allocation. Later, we run lucene in a number of CPU allocation scenarios (from 512Mhz to 2048Mhz) and 2048MB of memory set. Then we measure the CPU and memory consumed in each experiment and calculate the ratio $\frac{CPU\ consumed}{memory\ consumed}$, what indicates how many Mhz must be allocated for each MB of memory. The resulting ratios calculated for the two sets of tests (variable CPU allocation and variable memory allocation) are shown in figure 4. Notice that from these models it can be deduced that the driving resource for the lucene application is the CPU: changing the memory allocation does not change the ratio as changing the allocated CPU capacity does. Notice also that both models are linear, predictive and progressive.

In a second experiment, we run the montecarlo application in a set of memory allocation conditions, ranging from 128Mb to 2048MB, and 2048Mhz of CPU allocation. We measure the CPU and memory consumption as well as the swapping activity, and also calculate the resource consumption ratio described above. It is important to remark that the working set of the montecarlo application is above 600MB and thus, swapping is expected for memory allocation lower than that value. Figure 5 contains the collected data, showing memory consumption and swapping activity in the top chart, and the model in the bottom chart. Notice here that when the application starts swapping, the ratio breaks its linearity and performance prediction becomes impractical due to the complex effects of memory swapping.

In conclusion, results indicate that performance prediction based on resource usage profiles is feasible when scaling resources other than memory, but challenging when changing memory allocation.

V. Related work

Although there are a lot of previous work on application profiling and resource provisioning for transactional applications in virtualized execution environments ([8], [7], [11], [12]), the interest of the research community in the execution of long running jobs in virtualized environments is recent. However, we can find in the literature previous work that propose a methodology to model the resources consumption of long running applications to drive the resource provisioning decisions. To the best of our knowledge, none of those previous work neither consider all the machine resources in the execution model nor consider the effects that the execution on a environment with restricted availability of resources can have on the execution model of an application. For example, in [6] the authors propose a statistical model to predict CPU consumption based on previous consumption on both CPU and memory. However, they base the evaluation of their model on measuring the hit prediction rate on traces generated by a previous execution. Thus, they do not consider the effects that different execution conditions could have on the resource consumption. Zhang et al. propose in [13] and [14] a methodology to guide resource provisioning, detecting and classifying the execution phases of the applications based on resource consumption. The methodology proposed to detect the execution phases has two main drawbacks. First, they consider a linear relation between resources consumption, which, as we have seen in section IV, is not a realistic approach. Moreover, the authors assume that the model of resources consumption is the same for a given application during all its execution. Second, the authors validate their proposal using quite stable simulation environments and, thus they do not consider the influence on the execution model that can have an execution on a resources-restricted environment. In addition, due to the evaluation based on simulation, their work lack an evaluation of other concerns that can affect the performance on a real execution environment as, for example, the effects of data sampling granularity.

VI. Conclusions

In this paper we have summarized our experiences in the process not only of generating workload profiles for resource provisioning but also in the process of enforcing resource profiles in real systems. The focus of our work is put in the provisioning of resources for clusters of VMs running in large scale enterprise data centers.

We have motivated the need for application profiles in such an environment, usually orchestrated by a management middleware that needs some kind of performance prediction mechanism even for long running batch jobs. We have also presented the testing environment prototype we have used, as well as the results for some experiments that outline some of the difficulties and hot-spots involved in the generation and enforcement of workload profiles. Finally we have explored the construction of simple resource consumption models that will allow the management middleware to scale VMs to fit the available resources in the system and still be able to predict the performance of the applications running on them.

Our study proves that the existence of workload profiles describing resource consumption over time is not enough to drive the management middleware in their resource allocation decisions. Our work highlights the need of an adaptive profile enforcement technique for this purpose. Our proposed technique is driven on-the-fly by the observed application progress.

Acknowledgements

This work is partially supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625.

References

[1] Apache Software Foundation. *Apache Lucene. http://lucene.apache.org/.*

[2] Y. Becerra, D. Carrera, and E. Ayguad. Experiences with virtual machine profiling for batch jobs in real systems. Number UPC-DAC-RR-CAP-2008-32, 2008.

[3] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé. Managing SLAs of heterogeneous workloads using dynamic application placement. In *Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)*, 2008.

[4] Intel. *Intel Virtualization Technology http://www.intel.com/technology/platform-technology/virtualization.*

[5] Java Grande Forum. *JavaGrande Benchmarks Home Page. http://www2.epcc.ed.ac.uk/computing/research*$_{a}ctivities/java_{g}rande/$.

[6] J. Liang, K. Nahrstedt, and Y. Zhou. Adaptive multi-resource prediction in distributed resource sharing environment. In CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 293–300, Washington, USA, 2004. IEEE Computer Society.*

[7] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Dynamic estimation of cpu demand of web traffic. In valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, page 26, New York, USA, 2006. ACM.*

[8] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 239–254, New York, USA, 2002. ACM.*

[9] VMware. VMware ESX. http://www.vmware.com/products/vi/esx/.

[10] VMware. VMware Infrastructure SDK Programming Guide. http://www.vmware.com/pdf/ProgrammingGuide201.pdf.

[11] T. Wood, L. Cherkasova, K. Ozonat, , and P. Shenoy. Profiling and modeling resource utilization of virtualized applications. In Xen Summit 2008*, Boston, USA, 2008.*

[12] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource utilization of virtualized applications. In Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)*, Leuven, Belgium, 2008.*

[13] J. Zhang and R. Figueiredo. Application classification through monitoring and learning of resource consumption patterns. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, 2006.*

[14] J. Zhang, M. Yousif, R. Carpenter, and R. J. Figueiredo. Application resource demand phase analysis and prediction in support of dynamic resource provisioning. In ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 12, Washington, USA, 2007. IEEE Computer Society.*