Dynamic Resources Management of Virtual Appliances on a Computational Cluster

Alexander A. Moskovsky[1], Artem Y. Pervin[1], and Bruce J. Walker[2]

[1] Program System Institute, Russian Academy of Sciences, Pereslavl, s. Botik, Russia
[2] Hewlett-Packard Laboratories, Palo Alto 1501 Page Mill Rd, US
moskov@phys069b-2.chem.msu.ru, ArtemPervin@gmail.com,
bruce.walker@hp.com

Abstract. Virtual machine (VM) technology offers increased flexibility in resource provisioning. Load for applications typically varies over time, justifying the need for dynamic resource allocation/relinquish — exactly what VM technology allows. An approach for automated, dynamic resource management of applications running on a computational cluster has been devised. The job of the framework is to maintain a certain service level of application within tolerable limits. To do this the framework is able to dynamically vary resources available to the application. To facilitate performance optimization an application performance profile can be created using stress-testing tools. A software toolkit that allows running single and multiple VM applications has been created. Sample services (including both computing oriented and web oriented) have been tested and performance-resource dependences studied. We present an ongoing work on dynamic resource allocation, involving optimal control and optimization methods.

Keywords: virtual environment management, service level agreements, virtual machines cluster, virtual appliances.

1 Introduction

The research goal is to develop methods and tools to deal with multiple applications sharing a computational cluster in a virtualized environment. With regular clusters, the number of nodes occupied by a particular application serves as the main resource consumption metric. In addition to this the virtual machine (VM) technology offers several new capabilities bare iron environments couldn't feasibly offer. In particular using `Xen` one can:

1. Suspend a VM to disk and resume it later, allowing one to utilize the resources for a higher priority service.
2. Pause a VM, leaving it in memory, to allow running some other application/VM.
3. Live migration of a VM from one host to another for any one of a variety of reasons (e.g. consolidate resources to allow something new to run).
4. Start a VM with some number of virtual CPUs and then add or remove CPUs on the fly based on need and the priority of other VM.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 33–42, 2009.

5. Start a VM with some fraction of each CPU it has allocated and then grow or shrink this fraction dynamically.
6. Start a VM with some amount of memory and then grow or shrink this footprint dynamically.
7. Start a VM with some I/O capacity and grow or shrink this capacity on the fly.

This drastic increase in flexibility allows not only "carve to order" (dedicate only as much resource to the task as needed) but also allows shared "over provisioning" instead of individual application "over provisioning". This should allow IT managers to provide considerably more services within the same infrastructure, particularly if one can automatically reallocate the resources based on their needs. As well, automation saves human time (and associated cost), human reaction can be too slow. At the same time, performance overhead of virtualization can be kept relatively low [1, 2].

Related works are numerous in the both industry and academia. `Amazon`s EC3`, `3Tera`s Applogic` are widely known industrial projects, which are capable of hosting on-line services in customized VMs. `Cluster-on-Demand` [3] provides a toolkit to create virtual clusters of VMs on top of physical machines. `Virtual Workspaces` [1] are utilizing VMs to isolate applications from hosts in a grid environment (starting grid jobs inside VM), leveraging and extending `Globus Toolkit`s` resource management [4]. `SoftUDC` [5] is a utility computing platform, which virtualizes CPU, storage and network resources for applications running on a cluster. Resource sharing techniques in virtualized environments are sometimes borrowed from economics. `Shirako` [6] offers a mechanism were the application and the framework can negotiate and contract on resource leases. `Tycoon` [7] is an example of auction-based modeling applied to resource scheduling.

We believe that one step further is feasible: an automatic resource manager can be aware of how valuable given portions of the CPU, memory or some other resource are to a given application under the current load. With this information one can more accurately decide how to deploy incremental resources and when to request resources back from applications. In this work the software has been developed to conduct experiments with various resource allocation schemas. The following assumptions are considered in these schemas:

- Each application has a set of key parameters that define a quality of service for end-users, such as a response time for a web site application. These parameters can be measured at application run time.
- With the help of VM technology, various resources can be allocated dynamically with very fine granularity and thus they can be treated as continuous variables.

With these assumptions, the problem for managing the application's quality of service can be treated as an optimal control problem with continuous variables. This makes this work considerably different from other researches in QoS management [8-10]. In our work applications are considered as black-boxes and management framework can utilize powerful optimal control theory methods to implement efficient optimization mechanisms for resource allocation.

In order to make use of theoretical basis, we advocate the *service level* abstraction to enable automated decision-making about which resources are necessary for an

application to run. The runtime framework can receive sensor data about current application state and use application *performance model* to make decisions on how to maintain the service level for this application. In case of resource shortage, the framework can take away resources from less important (for end users) applications. Performance profiles can be either measured before runtime or generated on the fly.

The goal is to support a variety of types of single threaded and parallel services (virtual appliances) with desired performance characteristics under dynamically changing load conditions. This support implies optimal use of the resources so the greatest number of services can be launched on a given amount of hardware.

To accomplish this goal we first needed an infrastructure that allowed deployment, monitoring and resource re-allocation for virtual appliances. The next section of this paper describes the Virtual Services toolkit developed to provide such infrastructure.

Next the performance profile or models on the services are required. This information would not only describe the ideal resource mix (memory, number of virtual CPUs, CPU share value and I/O capacity) for a given load but would also describe the effect of adding or losing various resources. Section 4 outlines the concept of performance profiles. Finally we describe an arbiter that can take the performance profiles and the runtime information and re-allocate resources to optimize the effect of those resources.

2 Virtual Services Software

To accomplish the research goals a simple system was implemented that allows us to roll-out *virtual services* — parallel virtual appliances. Virtual appliances can be either online services (like web-site) or high-performance parallel applications. Other applications may include online gaming services, data processing, and transformation or retrieval applications.

Consider the components and interaction of those components in the fig. 1.

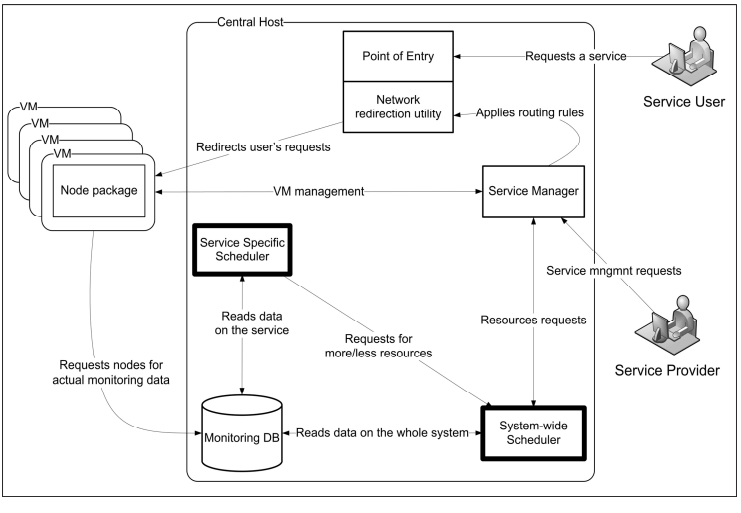

Fig. 1. Virtual service software components

A typical use-case of the system involves a Service Provider (an administrative user), who starts a service. The Service Users access it via the *point of entry* — an IP address accompanied with a port.

Capabilities of the software kit include, but not limited to:

- Instantiate/kill a service. Service instantiation involves launching one or more VMs with appropriate disk images and setting up of network traffic redirection rules and forming of virtual network for VMs of the service.
- Allocate/free the resources for the service. Resources request may originate from a human (Service Provider) or from the application-specific resource provisioning component which is known as *service specific scheduler*.
- Redirect network traffic. This is necessary to enable users' access to the virtualized application. It's also can be used for load-balancing.

The resource management flexibility is crucial for the research. At the same time, the ability to run generic algorithms on a system-wide layer is also important. Bearing this in mind, a two-layer resource scheduling mechanism had been designed to isolate concerns of system and service layers. On the lower layer a pluggable, service-specific scheduler makes resource distribution decision locally, considering monitoring information. On the top layer the *system-wide scheduler* takes into account agreements between Service Provider and a System Administrator, such as the Service Provider's priority, to provide globally optimal distribution of the resources between the services. In addition, the system-wide scheduler is able to query an application performance model, when it's necessary to evaluate variants of resource allocation. The schedulers are able to communicate with each other and access the monitoring data they need. The framework automatically maintains a certain resources quantity available to the application, in order to keep agreed service level. If a cluster node fails with a VM running on it, the framework will re-instantiate the VM on available free nodes using technique similar to high-availability clustering solutions. The Ganglia monitoring system is used to implement a heartbeat mechanism.

3 Service Examples

- **WebMapServer**
  This application [11] allows querying different information from geographical maps. In our tests the data on Itasca County, MN was used. It was derived, for the most part, from USGS (US Geological Survey) 1:24,000 quadrangles. The page displayed to the user includes a custom generated map in GIF format.

- **X-Com**
  The computational service is based on the `X-Com` utility. X-Com is a meta-computing framework, developed at Moscow State University [12]. The X-Com concept is similar to the `Condor` [13]. However, the implementation is simpler, light-weight, easy to install and use, yet applicable in wide variety of computational environments.

- **Virtual Cluster**
  The virtual cluster service starts the necessary number of VMs with the network connectivity support between each of them. Start of this service results in a set of the nodes that is called a *virtual cluster*. The network support is provided with the help of a bridging mechanism. It allows including a VM into the virtual cluster absolutely transparently for the user. The users may interact with the virtual cluster node without any extra effort as if it was a normal host.

A number of computational experiments were conducted on the virtual cluster. It was shown that in such environment one can successfully run LAM MPI applications that operate on several nodes concurrently . We have also run experiments with launching of parallel programs created with the help of rapid parallel application development tools such as `OpenTS` [14].

4 Performance Profile

An application performance profile represents the dependency between the amount of resources provided to the application, the user activity being generated on this application and the quality of service this application provides to the users. The amount of resource can be expressed in either absolute values (1GB of RAM) or in relative form (53% of CPU). The user activity is specific for different application classes. For example, for the web-site the user activity is a number of concurrent requests to a particular page of the web-site (that is, request rate). Finally, the quality of service can be measured as difference between desirable (or target) state of the service and its current state. The service level concept is discussed in details below.

This dependency can be expressed in tabular form. Tabular form describes some typical use cases of the application at different levels of user activity and the interpolation or extrapolation is used to obtain the values that are not provided in the table. The data for the tables can be collected with the help of stress-test utilities, such as `httperf`. We believe that this , given sufficient data collected in tables, may be appropriate for resources provisioning.

A number of experiments with various user activity and amount of resources dedicated to the service were conducted. Currently the following VM parameters are available to tune: amount of memory, number of virtual processors allocated by VM (VCPUs) and CPU cap, defining the maximum processor's time share the VM is allowed to occupy. The user load in each test run was chosen in a way to maximize the usage of resources provided to the service and at the same time keeping the network errors (such as request timeouts) at minimal levels. These user load values are referred as *inflexion points* — the highest load the service is capable of handling well.

The WebMapServer performance profiling demonstrated an insensitivity of this application to the amount of memory: only tiny variations of request rate were observed (fig. 2). Increasing VCPUs doesn't improve performance and eventually degraded performance (fig. 3). The application is apparently single threaded so we focused on the CPU cap parameter in the tests as the most significant VM parameter. The dependence of the maximum request rate from the CPU share is almost linear.

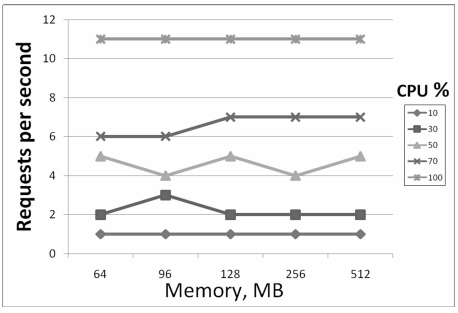

Fig. 2. Maximum requests/sec served by WebMapServer, at different amount of memory and Xen CPU caps

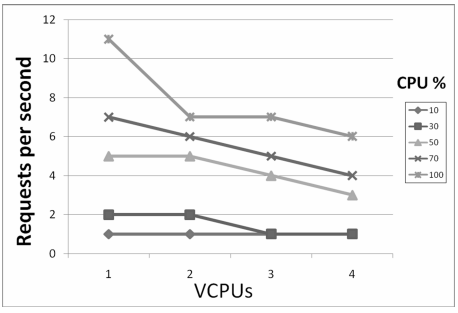

Fig. 3. Maximum requests/sec rate, served by WebMapServer, vs. VCPU of Xen at different CPU caps specified

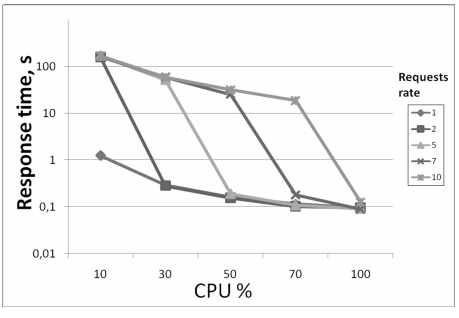

Fig. 4. Response time of WebMapServer vs. CPU cap specified in Xen configuration, under different load (requests per second)

One may see in fig. 4 that under 10 requests per second (the most right line), quality of service is quite sensitive to the CPU share provided to Xen: only above 70% share is response time tolerable. At the same time, 1 request per second can be served even with 10% CPU cap at reasonable response time (1 second).

The performance profiles can be used to evaluate variants of resource allocation without actually affecting the performance of applications running in the framework.

5 Service Level Agreements

Consider the situation when a web-site owner wishes to maintain service response time below a certain threshold (e.g. 1 second). If the web site is experiencing a spike of user activity, additional computing resources are required to keep the quality of service at a reasonable level. In this paper, this level is referred as a *target level*. It is natural to use an automated mechanism to maintain the target level since human intervention can be too slow and unreliable. One possible control schema is illustrated in the fig. 5 below. In order to re-use the optimization algorithms in the *Optimizer*, which acts on the system-wide layer, it is necessary to provide translation from application-specific parameters

(such as response time) to abstract service level value [15], which is done with the help of a *service level function*.

The idea of abstract service level function is the following. The function takes a *measured parameter* as an argument and results in a service level value that is in the 0% - 100% interval. Next, these values are passed to the Optimizer, who will find optimal resource allocation, maximizing service levels of the services by varying resources available to the services.

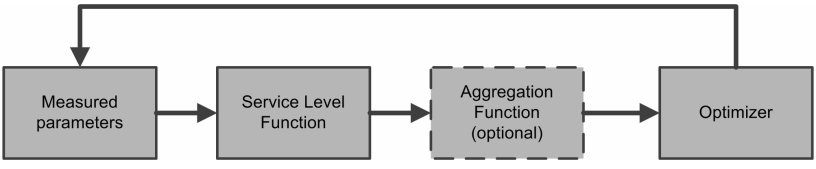

Fig. 5. Control Chain

The service level function is defined in a unique way for each service as the measured parameters and their target ranges differ from one service to another. Thus for the web-site service where the target service level is defined as the average response time below some reasonable value, the measured parameter should capture (at least) current response time. For the computational service with the deadline defined to finish the calculation, measured parameter describes an average calculation speed. In this case the optimal calculation speed will allow reaching the deadline without utilizing extra resources.

Since the service level functions are continuous, the powerful continuous optimization math algorithms can be applied in the Optimizer. In the general case the service level function is switch-shaped curve of the form:

$$s(x) = \frac{w * a * (b - x)}{\sqrt{1 + (a * (b - x))^2}} + w \tag{1}$$

where x — measured parameter (e.g. response time), s — service level, a, b and w — are parameters, allowing one to tune the curve's shape to fit application needs.

We suggest setting the value of 50% as the lowest tolerable level for any service in the framework. The Optimizer should try to keep all service levels above 50%. The value of 50 is selected because the changes on the service level function curve are the steepest in this area and hence optimization algorithms will be most sensitive to changes around this last tolerable value.

By tweaking the service level function parameters one is able to create soft or hard requirements to the function. Consider an example of service level function (1) with parameter a equal to 10, b — 1 and w — 0.5. This function is almost 100% below 1 second and rapidly vanishes to zero above 1.5 seconds (fig. 6). The curve parameters should be carefully chosen to fit application characteristics and its target service level. The service level concept can be easily generalized to include more parameters, e.g. percent of errors in addition to response time. In this case multidimensional function can be used.

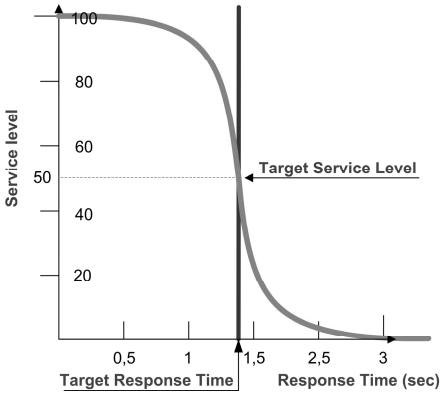

Fig. 6. Service Level Curve for the web-site service

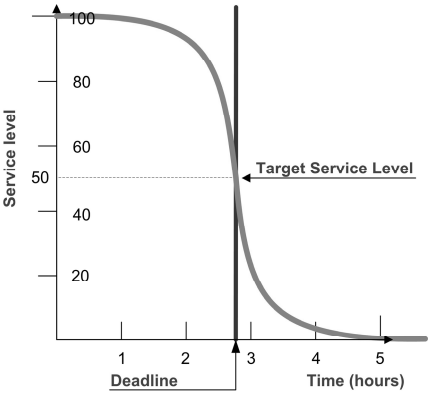

Fig. 7. Service Level Curve for the computational service

It is clear that the service level approach is not only valid for web-site applications but can be extended to other application types. It can be applied for the job queues as well. As was mentioned before, the Service Provider may specify a deadline by which all jobs in the queue should be completed (see fig 7). Given job characteristics, it is possible to evaluate calculation speed and estimate completion time. If the deadline is met under current resources the service level is 50%. It will degrade until it becomes 0%, when results of the calculation are obsolete (say, weather forecast for tomorrow completed three months later).

In real life the framework should be able to deal with multiple applications simultaneously. The aggregation of n service levels into one can be achieved by weighted multiplication:

$$\sigma(x_1 \ldots x_n) = \prod_{i=1}^{n} w_i S_i(x_i) \tag{2}$$

where the w_i are relative weights assigned to each application, S_i are application service levels and σ is the whole system service level. By maximizing σ the framework should enable tomorrows weather forecast service to be finished by afternoon at, probably, the expense of some inconvenience to web-site visitors in the morning. An array of optimization or optimal control methods can be applied in order to maximize σ - dynamic programming to name just a starting point.

6 Preliminary Implementation of Optimization Algorithm

So far, a naive one-dimensional constrained optimization is used to find the optimal resources required for a given application. Consider the example with the computational service. The algorithms below tries to figure out the minimum CPU share that still satisfies the target service level by using a binary search:

- First stage requires finding the boundaries of the range of CPU share.
- System starts with some guess on the resource necessary for the application. If the current service level is below the target level (case 1), the value of the guess CPU share will be higher than currently available to the service. Otherwise the algorithm will try to decrease the CPU share (case 2).
- This step will be repeated with increasing values of guess doubling the step until the current service level will not reach some value near the target level (higher than the target level in case 1, and lower in case 2).
- On the second stage, the algorithm uses the last 2 values obtained on the previous stage to perform a binary search in the range of these values to find an optimal CPU share distribution for the service. This step implies continuous increasing (or decreasing) CPU share available to the application and measuring of the new service level.

This algorithm is started periodically. In that way the service level of the application is kept at target value automatically. The existing implementation is simplistic, but still applicable to various services. Thus, for example, this algorithm was used to dynamically allocate resources for the computational X-Com service during its operation. The X-Com service was launched with some deadline to finish the job and the volume of resources a priori insufficient to complete the computation on time. However using this schema, the system increased the amount of resource dedicated to the service enough to finish the job close to the deadline. Our tests demonstrated only 0.5% deviation from the deadline. Thus a 40 minutes long computation was finished only 10 to 15 seconds late. The experimental setup was the following:

- Computational service CPU share initially was 40% of one CPU, at the end was 3.25 CPUs.
- Service was supplied with a set of uniform tasks.
- The optimization algorithm was launched once every 5 minutes.
- The hardware platform was 4-node cluster, each node has 2.0 GHz Opteron 175 (Dual-core) CPU and 2 GBytes of RAM, nodes are connected with Gigabit Ethernet.

7 Conclusions

We have created the platform where the central orchestrator and service specific scheduling agents can co-operatively create and remove instances of services and increase/decrease the resources for services. Simple automation based on service level concept has been demonstrated. More importantly, however, applications were tested on this platform in order to create performance profiles for more sophisticated automation software to leverage. This approach is not tightly knit to the Virtual Services and can be harnessed in other frameworks, (e.g. Virtual Workspaces, Cluster-on-Demand) after minor modifications.

One might argue that modifications on VM container CPU share, abrupt disconnections of VM instances may be detrimental to high-performance computing applications. It is true that most of existing MPI applications will suffer: they are written

with the assumption of uniform processor performance and could not sustain disconnection of even a single process. Nevertheless, more advanced parallel applications can live with this. `MapReduce` [15] is an example of a general purpose framework that can deal with the additional complexity of such a dynamic environment. With all the efforts spent by IT community to improve high-level parallel programming tools and techniques, one can expect such applications to become more widespread.

References

1. Keahey, K., Foster, I., Freeman, F., Zhang, X., Galron, D.: Virtual Workspaces in the Grid. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 421–431. Springer, Heidelberg (2005)
2. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Paravirtualization for HPC Systems. In: Workshop on Xen in HPC Cluster and Grid Computing Environments, Sorrento (2006)
3. Moore, J., Irwin, D., Grit, L., Sprenkle, S., Chase, J.: Managing Mixed-Use Cluster with Cluster-on-Demand, Technical Report (2002)
4. Sotomayor, B.: A Resource Management Model for VM Based Virtual Workspaces, Masters Paper, University of Chicago (2007)
5. Kallahalla, M., et al.: SoftUDC: A Software-Based Data Center for Utility Computing. Computer 37(11), 38–46 (2004)
6. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: SHARP: An Architecture for Secure Resource Peering. In: 19th ACM Symposium on Operating Systems Principles (2003)
7. Lai, K., Rasmusson, L., Adar, E., Sorkin, S., Zhang, L., Huberman, B.: Tycoon: an implementation of a Distributed Market-Based Resource Allocation System. Technical Report, HP Labs, Palo Alto (2004)
8. Moroni, S., Joffre, A., Figueroa, N., Sahai, A., Chen, Y., Iyer, S.: A Game-theoretic framework for Optimal SLA/Contract Creation, HPL Tech. Report (2007)
9. Bennani, M., Menasce, D.: Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In: Second International Conference on Autonomic Computing
10. Menasce, D., Bennani, M.: Autonomic Virtualized Environment. In: International Conference on Autonomic and Autonomous Systems, p. 28
11. MapServer, `http://mapserver.gis.umn.edu/`
12. Voevodin, V.l., Filamofitskiy, M.: Supercomputer for a weekend, Open Systems. In: Otkrytie Sistemi, vol. 5, pp. 43–48 (2003) (in Russian)
13. Thain, D., Tannenbaum, T., Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience 17(2-4), 323–356 (2005)
14. Abramov, S., Adamovich, A., Inyuhin, A., Moskovsky, A., Roganov, V., Schevchuk, Y., Schevchuk, E.: The T-system with an open architecture. In: Supercomputer Systems and Applications, Minsk, pp. 18–22 (2004)
15. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: 6th Symposium on Operating System Design and Implementation, pp. 137–150 (2004)