

Nova: An Approach to On-Demand Virtual Execution Environments for Grids

Srikanth Sundarrajan, Hariprasad Nellitheertha, Subhabrata Bhattacharya, Neel Arurkar
Software Engineering and Technology Laboratory, Infosys Technologies Ltd., India
{srikanth_sundarrajan, hariprasad_v01, subhabrata_b, neeln_arurkar}@infosys.com

Abstract

This paper attempts to reduce the overheads of dynamically creating and destroying the virtual environments for secure job execution. It broaches a grid architecture which we call Nova, consisting of extremely minuscule, pre-created virtual machines whose configurations could be altered with respect to the application executed within it. The benefits of the architecture are supported by experimental claims.

1. Introduction

Inspiring work [1] has already been done on using virtual machines (refer following section) [2] in Grid environment. The use of virtual machines (VMs) within grids makes efficient provisioning of computer resources feasible, adhering to stringent high-level policy parameters. Furthermore, executing un-trusted code in a secure environment is an issue quite well addressed by VMs. In dynamic environments like a grid, a job's successful execution depends on factors such as availability of computation power, storage resources and communication bandwidth. In case of a resource outage, job migration along-with its execution context, can be readily done with the use of VMs. The following sections of the paper provide further insight on the specific aspects of virtualization pertinent to our case.

Virtualization of physical computing systems is a relatively old concept [4], .It is now achieved by efficient partitioning of hardware and software, or by partial/total emulation of hardware etc., yielding to a subset of logically partitioned entities, called Virtual Machines. These VMs replicate the necessary hardware environment required to host operating systems. Hence through VMs, a variety of applications demanding different hardware environments could be executed on the same physical machine. This in turn attributes for the co-existence of various versions of libraries and

application codes. VMs behave as containers for executing un-trusted applications called sandboxes [3]. Sandboxes are environments that support differentiated service and impose irrevocable restrictions on resource usage. Ideally a sandboxing solution mandates high degree of isolation from job to host and job to another job. In both these cases, a sandbox ensures that the former is protected from the latter, thus making entities with conflicting attributes coexist in the same physical environment.

Integrating the above features, introduces virtual execution environments (VEE) [7]. Jobs that are executed in grid, constituting of VEEs are truly isolated from other external factors that hinder the performance of their execution. However, setting up such environments in grid involves a lot of complexities. We discuss a novel technique to minimize the complexities involved in instantiating VEEs in this paper.

2. Related Work

The use of Virtual Machines to create sandboxed execution environment for Compute Grid has been advocated earlier [1][7][9]. Figueiredo et al [1] in their work have suggested the use of virtual machine in grid execute nodes and have proposed an architecture where virtual machine instance are created from images, stored in a repository. Keahey et al have also suggested a similar approach to provide a sandbox for grid jobs through virtual machines. The main drawbacks with these approaches are performance and job instantiation/tear down overheads. Also the use of a solution based on shared kernel virtualization albeit better in terms of performance and instantiation overheads are unsuitable for legacy re-hosting due to the fact that all the virtual partitions share a single kernel image.

Our solution broaches the idea of pre-creating virtual machines based on para-virtualization such as Xen, with very low resource commitment and

increasing the resources levels at the time of job execution to provide a low instantiation overhead and also low execution overhead.

3. Nova: Our Approach

Our approach addresses some of the problems that have been mentioned in the previous section. The goals of the proposed solution are

- To reduce the time required to get a “working” virtual machine to as little as possible.
- To ensure that the virtual machine allocated to the Grid job has the necessary resources, both hardware and software, to perform the job.
- To perform effective clean-up of the virtual machine once the job is complete.
- To ensure that the effect of a completed job does not spill over to another job (and virtual machine)

3.1 Overview of the solution

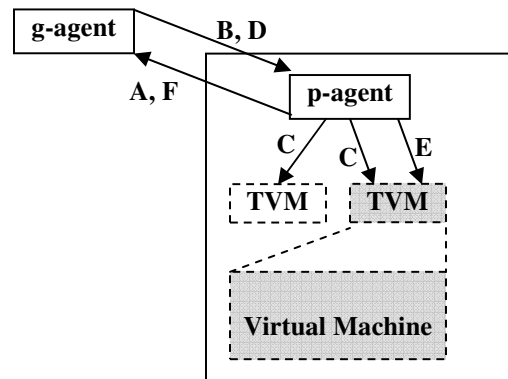
Nova addresses the above goals by creating, in advance, virtual machines with configurations that consume very little resources. We call such a virtual machine as a “Tiny VM”. We have used the open source Xen virtualization tool to create these virtual machines and in the terminology of Xen, tiny VMs are domain Us which consume very little resource. For example, one such configuration could be 32MB of RAM, 5% of CPU bandwidth and a basic mount partition.

The Grid middleware is modified a little to communicate to the host machine indicating the resources necessary to execute a given Grid job. In response, the host machine “balloons” one of the tiny VMs to the desired configuration. The Grid middleware then hands over the job to this new machine. Upon completion of the job, results are transmitted back to the middleware and the machine is teared down. The entire process of ballooning the machine to its new configuration takes only a few milliseconds.

3.2 Implementation details

Nova can be implemented on any machine where Xen could be run. The solution involves writing two agents. One of them is a p-agent and runs on the physical machine hosting the VMs. The other is a g-agent and runs on the Grid middleware. As depicted in Fig 1, there are a number of steps in setting up a machine for accepting a Grid job. Once

a physical machine decides to donate its resources to the Grid, the p-agent announces itself to the g-agent and also mentions the number of VMs it is donating and also their capabilities. The g-agent will use this information to schedule jobs on the VMs at a later point in time. The p-agent then goes ahead and creates tiny VMs equal in number to that advertised to the g-agent. For example, if VMs of 512MB RAM, 20% CPU share and 10GB hard disk space are advertised, tiny VMs could be created with 32MB RAM, 10% CPU share and 1GB hard disk.



- A – p-agent announces availability of machine
- B – g-agent acknowledges p-agent
- C – p-agent creates tiny virtual machine
- D – g-agent indicates availability of job
- E – p-agent balloons tiny VM into a larger VM
- F – p-agent announces readiness of VM
- TVM – Tiny virtual machine

Fig 1: Expanding a Tiny VM

Once the Grid middleware finds a suitable job to run on the VM, the g-agent makes an indication to the p-agent. The p-agent, in response, balloons one of the tiny VMs into a larger configuration. Under Xen, the balloon driver could be used to increase memory allocated to the VM and the “xm” command could be used to increase the CPU parameters. The p-agent then indicates the readiness of the VM to the g-agent and the job is scheduled on the new machine. Once the job is completed and the results are returned to the Grid middleware, the p-agent tears down the VM, frees up its resources and removes any left over state information.

During the cleanup of the VM, the total number of VMs available on the machine is less than what is advertised to the middleware. This can cause scheduling problems. One solution to this is to keep an additional “unadvertised” VM in the pool

which gets added to the active VMs when a cleanup happens. Another approach could be to start the boot-up of a new virtual machine when an existing one is getting cleaned up.

4. Experiments

In this section we present an evaluation of our approach comparing the same with dynamic instantiation of Virtual Machines. We have carried out three sets of experiments to determine the time it takes for a tiny VM to expand and assume full size for a grid job to be run. Firstly, we look at time consumed in expanding memory of the tiny VM. Secondly, we examine the effects of increasing both the memory allotted to a Virtual Machine and CPU slices allotted simultaneously. Finally, we assess the performance of VM expansion considering all three aspects vis-à-vis Memory, CPU and disk. We use the results from these tests to compare them with the time it takes for a VM to start afresh on demand when the grid job arrives.

Experiments were conducted using the latest stable release of Xen 2.0.6 on Linux 2.6.11.12 running on an Intel Pentium IV 3.2GHz with 1GB RAM and 40GB storage. Xen Domain 0 was set up with 128MB and 10GB of disk space. Shell scripts were used to conduct the tests and Xen management commands were used to expand memory and CPU slices. Each iteration uses a new VM/Domain instance for the experiment and the experiments were repeated to ascertain that the time recorded were reliable and consistent. The results of the experiments are tabulated and presented in the following section.

5. Results

Our first test measures the time it takes to expand the memory allocated to a Xen domain from a minimal state (as in Tiny VM) 32MB to 64MB or higher. The ballooning option available in Xen is used to expand the memory allocated to a domain. As is evident from the table below and graph in Fig 2, the time system takes to expand a domain's memory from 32MB to 64MB or 32MB to 512MB is uniform. It is easy to conclude from the recorded results that the system is likely to take the same amount of time to expand every time and is not dependent on the size of expansion.

Table 1: Memory Ballooning

| # | Memory(MB) | Time (ms) |
|---|------------|-----------|
| 1 | 64 | 70 |

| # | Memory(MB) | Time (ms) |
|---|------------|-----------|
| 2 | 128 | 80 |
| 3 | 256 | 80 |
| 4 | 512 | 80 |

In our second set of experiments both the memory and the CPU slices were expanded and the time to complete both the operations was recorded. A tiny VM having 32 MB of RAM and with a 10% CPU slice was used as the reference configuration for each test and the memory was exponentially incremented by a factor of 2 and CPU slice increased in steps of 10%. The memory was expanded all the way to 512MB, while the CPU slice allocated up to 50%. Borrowed Virtual Time (bvt) scheduler option in Xen was used to increase the CPU slice and ballooning option was used for increasing the memory as in the earlier test.

Table 2: Memory/CPU Expansion

| # | Memory(MB) | CPU % | Time(ms) |
|---|------------|-------|----------|
| 1 | 64 | 20 | 165 |
| 2 | 128 | 30 | 167 |
| 3 | 256 | 40 | 166 |
| 4 | 512 | 50 | 168 |

As in the earlier case, the expansion time remains uniform and the expansion size does not affect the time taken for expanding the resources for a VM.

Table 3: Memory/CPU/Disk Expansion

| # | Memory (MB) | CPU (%) | Mount (MB) | Time (ms) |
|---|-------------|---------|------------|-----------|
| 1 | 64 | 20 | 100 | 220 |
| 2 | 128 | 30 | 200 | 245 |
| 3 | 256 | 40 | 300 | 250 |
| 4 | 512 | 50 | 400 | 260 |

Finally, we increased the memory and CPU as in the above test cases in addition to mounting disk partitions of up to 500 MB in steps of 100MB. Virtual block devices were created and mounted for providing additional storage to VM for this test. It can be seen from the results that with the increase in the size of the disk mount, the time taken linearly increases. This is attributed to the time taken to create the virtual block devices

In addition to the above sets of experiments, the time taken to bring up a new Xen domain has also been measured. A Xen domain with 512 MB of RAM and 4 GB of disk space took an average 20.244 seconds.

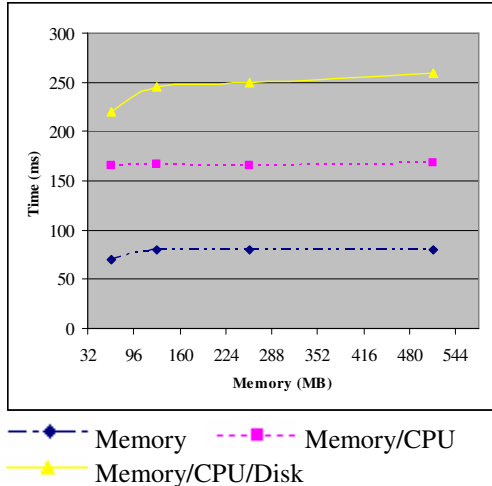


Fig 2: Comparison of expansion times

With our solution, it is possible to have the virtual machine up in order of hundreds of milliseconds as can be seen from the tabulated results. This is one hundredth of the time it takes to bring up the VM. An immediate improvement in performance can be seen for grid jobs that typically run for a few seconds. For such jobs, the overhead of starting new VMs before the jobs are run would be overkill. Our solution promises to bring it down to the order of a few hundred milliseconds.

6. Benefits

Benefits from our approach are listed below.

- Latency in getting a machine ready for accepting Grid jobs comes down drastically.
- Resources can be provisioned dynamically between virtual machines improving overall utilization.
- The cleanup of the virtual machines upon job completion ensures that no state information is left behind
- This approach ensures that promised number of virtual machines is always available.

7. Shortcomings with the approach

Following are few of the shortcomings with this approach.

- Pre-created virtual machines consume resources with no useful work being done.
- The physical machine hosting these pre-created VMs needs to be fairly resource rich.
- There could be some technical difficulties, depending on the virtual machine software being used.

8. Conclusion

We believe that as grids move to businesses, the need for an execution environment will be extensive. Increasingly, Virtual Machine will be looked upon to facilitate creation of an execution environment for the grids. But inherent complexities and issues with implementing the virtual machine for execution environment as discussed in the earlier sections may affect this adoption. Our approach of creating Tiny VMs ahead of time of a job request and expanding it to suit the arriving job on demand can go a long way in addressing some of the concerns that were highlighted in the earlier sections. The results of our experiments have been very promising and have shown that it is possible to build a grid architecture with virtual machines for sandboxing at execute nodes without affecting the overall execution time with necessary guarantees of isolation, fault tolerance, resource control etc.

9. References

- R. Figueiredo et al, "A Case for Grid Computing on Virtual Machines" Proc. 23rd Int'l Conf. Distributed Computing Systems (ICDCS), IEEE CS Press, 2003, pp. 550-559.
- A. Silberschatz et al, Operating Systems Concepts, Addison-Wesley, 1991, pp.54-55
- Amit Singh, Internet URL: www.kernelthread.com/publications/security/sandboxing.html, 2005
- A.S. Tanenbaum, A.S. Woodhull, "Operating Systems, Design and Implementation, Second Edition" pp 40-41
- Herbert Pötzl, Internet URL: <http://linux-vserver.org/Linux-VServer-Paper>, 2005
- Mendel Rosenblum, Tal Garfinkel. "Virtual Machine Monitors: Current Technology and Future Trends," Computer, vol. 38, no. 5, pp. 39-47, May, 2005
- K Keahey et al, "From Sandbox to Playground: Dynamic Virtual Environments in the Grid", Proc. 5th International Workshop on Grid, 2004
- B. Dragovic et al, "Xen and the Art of Virtualization", Proceedings of the ACM Symposium on Operating Systems Principles, Oct, 2003
- I. Krsul et al, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," Proc. IEEE/ACM Supercomputing 2004, p 7.