Optimizing Grid Application Setup using Operating System Mobility

Jacob Gorm Hansen & Eric Jul

Danish Center for Grid Computing
University of Copenhagen
Denmark

Abstract. This paper is about optimizing Grid application setup by allowing a user to configure a Grid application on her own PC and thereafter migrating the entire application onto the Grid where the Grid system replicates the application including the operating system instance onto the assigned nodes within the Grid. We use this strategy to develop a new Minimal Configuration Grid model (MCG). Configuration is simplified because the user does it on her own machine. Cluster administration is simplified because only a minimal software base is required: No OS nor any shared libraries need be present; they are simply migrated to each node. We have built a prototype MCG based on Intel x86 and the Xen Virtual Machine Monitor.

1 Introduction

Current Grid systems rely on static allocation of jobs to clusters. A Grid user submits a job by proving an application program and a data set which typically are subsequently run on some cluster in the Grid. With such a scheme, there are numerous configuration problems, e.g., the cluster machines must run one of a specific set of operating systems and must have certain libraries present to run the user's specific application. On the cluster side, the local cluster administrator must configure and maintain the OS and the numerous libraries. Besides these configuration problems, current Grid system most often assign jobs to clusters non-preemptively, so if, e.g., scheduling a job requiring 64 nodes on a 256 node cluster, it may be necessary to let some nodes idle until there are 64 nodes available.

In this paper, we address the problems of configuration and non-preemption in Grid systems by introducing a much more dynamic mechanism, that of OS migration, to allow more optimal Grid application setup and dynamic reconfiguration. Our intent is to make job configuration easier for the Grid application writer by allowing the writer to freely configure not only the Grid application itself, but also the writer's choice of operating system. We also intend to reduce the software base necessary on each cluster node thus reducing the maintenance task of the cluster administrator and, at the same time, increasing security by reducing the trusted software base necessary.

Our solution uses virtual machines thus allowing the concurrent execution of several independent Grid application jobs while enhancing security by providing isolation between jobs and the possibility of running multiple independent applications and even independent operating systems on a single machine. We use our implementation of a self-migrating version of Linux, running on top of the Xen Virtual Machine Monitor [1].

We have built a prototype, called Minimal Configuration Grid (MCG), to demonstrate the viability of our ideas. The model includes a low-level token mechanism for secure distribution of usage rights.

1.1 Background

Modern Grid facilities comprise a set of well-provisioned and well-connected machines that are offered to Grid users. In many cases, an entire cluster is dedicated to a single Grid application. There have been recent proposals to use Virtual Machine technology to enable the sharing of Grid resources [2], providing each user with the illusion of having her the machines to herself.

While the motivation for process migration has been transparent balancing of load across cluster-nodes, the techniques necessary for implementing OS migration introduce new possibilities such as network forking where a running OS replicates itself onto a new node on the Grid. Such network forking is greatly eased by the presence of OS migration facilities and is possible with some attention, e.g., to consistency issues.

By migrating entire OS instances we can, in effect, introduce preemption into Grid scheduling systems that previously had none.

1.2 Virtual Machines and Migration

A Virtual Machine Monitor (VMM) is a thin layer interposed between one (or more) OSes such that each OS has its own (virtual) machine which appears to the OS as the actual machine.

Process migration, a hot topic in systems research during the 1980s [3, 4, 5, 6, 7], has seen very little use for real-world applications. [8] surveys the possible reasons for this, e.g., the problem of *residual dependencies* which means that a migrated process still retains dependencies on the machine from where it migrated. In contrast, OS migration does not suffer from the residual dependency problem, because all state is encapsulated within the migrating OS image.

Our migration facility is built on top of the Xen VMM and works as follows: Each virtual machine runs a version of Linux that has migration facilities added and that can migrate *itself* to another machine, we call this facility self-propelled migration [9].

1.3 Minimal Configuration Grid

Our approach is to use the concept of Operating System Migration in the context of Grid systems to build our new model of Grid computing, called *Minimum Configuration Grid*.

The basic idea is for a user to configure the user's own favorite version of an operating system and make the user's Grid application run on the OS. The user submits the job directly to a Grid-node as packaged OS image, or migrates *entire* running operating system instance including the user's application to the node.

Furthermore, the Grid can use the migration facility to replicate the OS instance so that the job continues to execute on a number of nodes.

Our results show that, for realistic usage scenarios, it is entirely possible to migrate a production-class operating system between machines on a LAN, with minimal service disruption, typically incurring interruption times down into the $50 - 100ms$ range.

1.4 Contributions

The contributions of this paper are:

– A model, Minimum Configuration Grid, that provides an easy and effective way of handling configuration problems both for Grid users and Grid Cluster administrators.

– A novel Linux/Xen-based implementation of self-propelled replication.

– A low-level token mechanism that allows secure distribution of usage rights.

– Experimental verification that our proposed replication techniques can be effectively and efficiently applied to Grid Computing.

2 Background

2.1 Preemption

With Grid-computing growing in popularity, the limitations of the current generation of job-control software are becoming apparent. The lack of preemptability (e.g., the ability to seamlessly move a running process to either disk or another machine without having to restart it) quickly leads to suboptimal load configurations, because it is impossible to correctly guess in advance the behavior of submitted jobs.

2.2 Configuration

Vast amounts of time is spent on the Grid Community trying to agree on standard configurations for computing nodes, so that jobs may be submitted across clusters at different institutions. Because such standards have to describe both hardware as well as complex software configurations (type of OS, kernel modules for access to network file systems, installed shared libraries, user ids and so forth) and interfaces, deviations are likely. Even if a single software configuration were to be mandated across all institutions, making sure all nodes everywhere are always up to date with the current version of the standard seems like an impossibility.

With OS migration, the need for a common standard is not removed, but because an operating system for a Grid node expects little more than access to a network interface, access to a number of raw disk blocks and memory pages, this interface is greatly simplified. All that needs to be specified is a protocol for bootstrapping and migrating operating systems, with the rest of the choices being left up to the user. The operating system is then viewed as simply a component of the user's application, in line with a shared library. Because the operating system, in contrast to the normal user process, is entirely self-contained, and because all access to external resources is already abstracted by standardised and fault-tolerant protocols, such as TCP/IP, NFS, or iSCSI, the residual dependency problem of traditional process migration systems is likewise solved.

2.3 Virtual Machine Monitors

The Xen [1] Virtual Machine Monitor is a x86-based VMM that allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion without sacrificing either performance or functionality. Xen provides an idealized virtual machine abstraction that allows OSes such as Linux, BSD, and Windows XP to be ported to it with minimal effort.

In our Minimal Configuration Grid, Xen is used for enforcing isolation between different user applications sharing the same physical hardware.

3 Migration Architecture

Given a VMM-based system and, for example, a Linux guest OS instance running on top of the Xen hypervisor, and a wish to migrate the Linux guest OS to another machine with a similar VMM, we have a number of choices of how to proceed. Our goal is simple; somehow copy the contents of entire virtual machine containing the Linux instance to an empty virtual machine at the destination host, minimising hiccup-time, and with no overlap of execution when viewed from the outside, and in a fail-safe way that will allow us to restore execution on the originating machine, if there is a network partition or other error preventing the migration.

Because live OS migration demands detailed introspection and control of both address space layout and exception handling of the migrating OS, this functionality can only be placed either inside the VMM, or in the guest OS. We have chosen to use an unmodified version of Xen, so that the entire migration is implemented in the guest-OS which in our case is Linux. A typical OS has all the facilities needed for migration, e.g., a TCP/IP stack, fine-grained control of virtual memory, etc. Therefore, it is possible for a guest operating system to perform its own migration. This also simplifies the VMM because it no longer needs to support a full set of migration facilities, for instance it does not need to implement a TCP/IP stack. This has implications for security, as a simpler functionality and smaller implementation of the VMM will make it easier to verify for correctness.

3.1 Self-propelled Migration

Traditional process migration has focused on transparent migration, that is, migrating processes that are unaware that their are being migrated, in order to allow any process to be migrated. For OS migration, such transparency is not necessarily an advantage. In practice only a few different operating systems are needed by most popular applications, and it may be realistic to modify each of these to support their migration. We therefore propose the concept of *self-propelled OS mobility* as a viable alternative to transparent OS mobility. Self-propelled OS mobility means that the OS not only participates in its own migration, it actually performs the migration itself. There are a number of benefits to self-propelled migration:

- The operating system has close to perfect knowledge of what constitutes its own state including the state of the processes that it runs.
- Much effort has already been put into making popular operating systems efficient at handling the types of operations that are involved in migration, e.g., memory management handling and transmission of large amounts of data over the network.
- The operating system can make better informed decisions during migration, e.g. by rate-limiting processes with large working sets.
- The opportunities for performing other operations such as checkpointing and forking are much better. The operating system can make the often trivial, but essential changes to, e.g., a forked version of itself.

One disadvantage of self-propelled migration is that each OS must be extended to support it; however, once an OS has been extended with self-migration support, the implementation can be ported across multiple VMMs.

From the use of the pre-copy technique in NomadBIOS [10], we know that it is possible to identify the working set of a running operating system, i.e., the pages that are frequently modified during a pre-copy delta cycle, and that this set is both considerably smaller than the full OS. We can start a migration by using pre-copy, and conclude it by using a small buffer for storing the remaining working set by means of copy-on-write. Our self-migration technique thus becomes:

1. Retract all writable page mappings and copy the full OS state to the target machine. Log page frame numbers of modified pages in the page-fault handler.
2. Retract and copy every page that was modified during the last copy.
3. Repeat above step until working set is sufficiently small.
4. Repeat once more, this time making backup copies of any modified pages upon write-fault, and keep the backups in the snapshot buffer.
5. Conclude by copying the backed up pages in the snapshot buffer to their original positions in the image on the target host.

4 Minimum Configuration Grid

4.1 Cluster Configuration

From an administrator perspective, cluster configuration is usually handled using specially crafted tools, cloning a common operating system image (for instance a popular Linux distribution) onto the harddrives of all nodes. While the process of adding a new node to the network is often fairly automated, the base installation on each node is heavy-weight, often consisting of hundreds of megabytes of persistent state, and so continued maintenance of this persistent state, including the task of keeping all nodes in sync, is cumbersome.

With our system, nodes run only a minimal operating system, with no persistent state on node harddrives. Only the Xen VMM, augmented with a simple mechanism for receiving cryptographic tokens as payment for use, and a small unprivileged bootstrapping executable, reside on each cluster node. All remaining system software and configuration data, for instance, a Linux kernel with a file system populated with shared libraries, executables and configuration files, is supplied by the end-user.

Compared to the current generation of systems, the cluster administrator is relieved of the burden of having to maintain a set of complex OS installations, including the maintenance of cluster-wide state, such as login and password databases.

4.2 Payment for Use

Our self-migration system would work fine in a system without payment. However, we can readily add a mechanism to provide payment for use. Because one of our overall goals is simplicity, we have designed our resource payment system with a simple analogy in mind, namely that of Laundromats. From a customer viewpoint, Laundromats are simple, stand-alone devices that feed on Laundromat tokens. Even though Laundromat sometime fail to deliver the expected service, most customers are still willing to use them, in spite of a small risk of token loss.

For a token-system to withstand abuse, it must be resistant to the following classes of attack:

Counterfeiting An attacker should not be allowed to spend home-minted tokens.

Double Spending It should not be possible to spend a token more than once.

There is a number of ways of preventing counterfeiting of tokens. One way is to sign all tokens with the public key of the entity minting the tokens, and have all nodes trust the signature of that entity. Another, and perhaps simpler, method is to use a one-time password scheme, based on a one-way hash function $H()$, which is what we are suggesting. The initial boot-token purchased from and signed by the token-vending service, contains the outermost value T_n in a hash-chain $H^n(T_0), H^{n-1}(T_0), \ldots, H(T_0)$. The rest of the chain is kept as a secret at the token-vending service, and these secrets are only released if the customer pays for them, e.g. using a credit-card or some other form of real-world currency.

During execution, or when attempting to expand its resource allowance, new token-values $T_n, T_{n-1}, \ldots, T_0$ are fed to the VMM on a node by means of a special system call. Token authenticity is simple to verify, by checking that the newly presented token hashes to the previously accepted one.

Because new tokens are minted for each customer-node combination, and because the use of a one-way hash function imposes an ordering of tokens, tokens can only be spent at the node for which they have been minted, and only once. The customer will not purchase an entire chain of tokens at once, but rather purchase tokens little by little, in response to demand from his application and to verification of its progress.

4.3 Security

The security of the system hinges on the isolation properties of the underlying Virtual Machine Monitor (VMM). The problem with most VMMs is, that while they may securely isolate a number of unprivileged guest VMs, it is common to have a privileged host VM running for management purposes. The host VM will usually be sufficiently powerful to pose a danger to the integrity of the guest VMs, in that it has the ability to destroy them or inspect or modify their address spaces. Apart from the VMM itself, which we need to trust, we do not wish to run any code which has privileges over other VMs if it can be avoided, which is one of the main guidelines for our various design choices.

It is clear that we do need *some* privileged functionality, in order to instantiate new VMs, but the actual network stack which is handling the arrival can be created on demand, i.e. upon receival of the first network packet of a new connection initiation. By augmenting connection setup with a cryptographic token, as described in section 4.2, and then instantiate the receiving TCP stack, we can handle most of the setup of new VMs and the receival of migrations without having to resort to privileged code. Only token verification, which is possible with fixed-length buffers and very basic cryptographic algorithms (e.g. a one-way hash function and secret shared between token vendor and Grid node), need to reside inside the VMM or in a privileged VM.

Once a token has been verified, a new VM is instantiated. This VM contains a very basic UIP [11]-derived TCP/IP stack, which handles receiving and final setup of the incoming OS, which will take over the new VM for its own purpose. With this approach, and under the assumption that the underlying hardware and VMM can be trusted, and with the use of the correct cryptographic protocols, it becomes possible to guarantee the integrity of guest VMs.

5 Evaluation

5.1 Bandwidth Test

We measure the overhead of our migration mechanism by looking at the throughput loss of a bandwidth metering test running inside a migrating operating system. From the resulting graph in figure 1, we see that while performance drops during migration, the migrating OS is still able to sustain almost 80% performance. This test is performed across a pair of server-class 2GHz Xeon machines, connected via Gigabit Ethernet.

5.2 Responsiveness Tests

The purpose of the two remaining tests is to assess service disruption to interactive server processes, when migrating an OS to a different host.

In these tests, we are migrating between an identical pair of workstation-class machines, with 2.4 GHz Pentium 4 CPUs. The machines are connected via switched 100Mbit Ethernet, with Intel EEPro1000 interfaces.

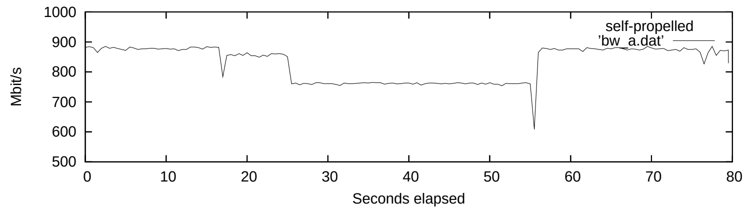

Fig. 1. iPerf bandwidth measure of a Linux VM self-migrating to an equally configured host. The first small drop in performance, at $t = 16s$, is due to migration being initiated, and the final drop, at $t = 56s$, is downtime resulting from execution switching to the new host

In figure 2 and figure 2, we see that it is possible to migrate both a streaming video server, as well as a quake 2 game server, with minimal disturbance to users. Downtime in this case is about $100ms$. Other tests run across a Gigabit network indicate that with improved network bandwidth it may be reduced below $50ms$.

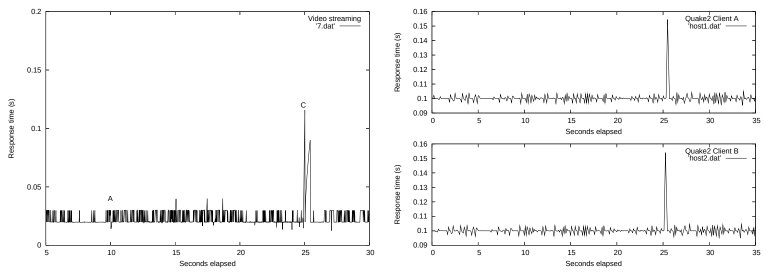

Fig. 2. Left: Downtime experienced when migrating a VLC streaming video server. Final migration occurs at $t = 25s$. Right: Downtime experienced when migrating Quake 2 interactive game server with two clients. Final migration occurs at $t = 25s$

6 Conclusions and Further Work

We have presented a new model for Grid Computing: Minimal Configuration Grid. Our model lets Grid application writers configure their application on their own machines, start up their applications, and when everything is running submit their application to the Grid. The model includes a simple mechanism for resource payment and accounting, it minimizes the trusted computing base of the cluster nodes, reducing configuration complexity and increasing security.

In future work, we plan to expand the prototype onto a 32 node cluster and measure a number of different Grid applications.

Bibliography

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19)*, pages 164–177. ACM Press, 2003.

[2] Renato J. Figueiredo, Peter A. Dinda, and Jos A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550. IEEE Computer Society, 2003.

[3] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. *ACM Trans. Comput. Syst.*, 6(1):109–133, 1988.

[4] Michael L. Powell and Barton P. Miller. Process migration in DEMOS/MP. In *Proceedings of the ninth ACM Symposium on Operating System Principles*, pages 110–119. ACM Press, 1983.

[5] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the tenth ACM Symposium on Operating System Principles*, pages 2–12. ACM Press, 1985.

[6] Fred Douglis and John K. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software - Practice and Experience*, 21(8):757–785, 1991.

[7] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372, March 1998.

[8] Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.

[9] Jacob G. Hansen and Eric Jul. Self-migration of operating systems. In *Proceedings of the 11th ACM SIGOPS European Workshop (EW 2004)*, pages 126–130, 2004.

[10] Jacob G. Hansen and Asger K. Henriksen. Nomadic operating systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark, 2002.

[11] Adam Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the first international conference on mobile applications, systems and services (MOBISYS 2003).*