Partition Repositories for Partition Cloning — OS Independent Software Maintenance in Large Clusters of PCs

Felix Rauch, Christian Kurmann, and Thomas M. Stricker
Laboratory for Computer Systems
ETH - Swiss Institute of Technology
CH-8092 Zürich, Switzerland
{rauch,kurmann,tomstr}@inf.ethz.ch

Abstract

As a novel approach to software maintenance in large clusters of PCs requiring multiple OS installations we implemented partition cloning and partition repositories as well as a set of OS independent tools for software maintenance using entire partitions, thus providing a clean abstraction of all operating system configuration state. We identify the evolution of software installations (different releases) and the customization of installed systems (different machines) as two orthogonal axes. Using this analysis we devise partition repositories as an efficient, incremental storage scheme to maintain all necessary partition images for versatile, large clusters of PCs.

We evaluate our approach with a release history of sample images used in the Patagonia multi-purpose clusters at ETH Zürich including several Linux, Windows NT and Oberon images. The study includes quantitative data that shows the viability of the OS independent approach of working with entire partitions and investigates some relevant tradeoffs: e.g., between difference granularity and compression block size. For a 2 GByte Windows NT partition our repository system enables the storage of nearly a dozen generational images or several dozens of customized images within the storage budget of twice the image size. The partitions can be replicated and transferred to a large number of PCs with our Dolly cloning tool. At present, our system is a modular university prototype based entirely on open source software, and most parts of it are in daily use to maintain our CoPs and Patagonia clusters at ETH.

1 Introduction

Clusters of PCs are an emerging low cost hardware platform for a variety of applications that require supercomputing performance in the amount of computation involved as well as in the amount of data to be communicated between the multiple nodes of a distributed system. In our attempt to broaden the use of PC clusters from scientific to corporate computing we identify three types of operation for such clusters: (1) A first kind of cluster of PCs is used for research and development in science or engineering. Those clusters are built from up to hundreds of rack-mounted PCs, typically stored in a cooled machine room and interconnected with a high speed system area network (SAN). (2) A second kind of "cluster" could comprise hundreds of PCs that large corporations dispatch to their employees' desks to give them personal computing power and access to all important information needed to do business. Such installations are not yet widely viewed as a PC cluster or high performance computing facility since the resources in a fleet of PCs are not centrally managed. Future applications will certainly tie large numbers of PCs together to deliver the power of supercomputers. Possible applications include multimedia support for collaboration (e.g. presentation cast, virtual worlds and teleconferencing) or distributed data mining. (3) A third kind of clusters is installed in all training and education environments. Those classroom PCs are used as workstations for college education and corporate training.

Together all three kind of "clusters" have broad requirements for installed software, maintenance concept and usage modes. Normally those requirements cannot be satisfied with a single operating system (OS) type or single OS configuration. In the Patagonia multi-purpose cluster computing project at ETH we learned that the desired flexibility can be achieved with multi-boot, involving several bootable OS installations (i.e. Windows NT, Linux and System Oberon) all requiring different file systems, security settings and software maintenance modes ([4]). This paper deals with the resulting software maintenance problem, proposes a systematic view of OS configuration state and a solution for maintenance based on cloning and partition repositories.

1.1 The Problem of Software Maintenance

The different usage modes (production, training, experimentation) and the different operating systems (Linux, NT, Oberon) in some PC clusters greatly complicate software maintenance. Typically software maintenance for large cluster of PCs or a corporate fleet of PCs is done with OS specific tools, e.g. a variety of utilities specifically written for Windows NT. The three different OSes in our cluster run on three different release schedules. The compatibility of OS dependent tools with future releases would be extremely hard to guarantee resulting in a unstable system. Furthermore we have to deal with increasingly error-prone software and complicated installation processes that can in fact have unexpected influence on so-called preference settings or on other parts of software configuration state. De-installation of software is even worse and some systems can never be brought back to their initial state without a re-installation from the beginning.

To overcome the drawbacks and deficiencies of highly specialized and complex maintenance tools on the market, we raise some fundamental questions and require that our maintenance tools remain completely OS independent. Such a viewpoint will mandate and enforce that there are clean abstractions of a partition with an installed software system in terms of bootable partition, visible partitions and configuration state of a partition. All maintenance operations such as archive, restore, upgrade or replication of releases and personalization or localization (i.e. single licence installations or custom drivers) must be achievable without knowledge of the file systems or the configuration files of the system software installation. Last but not least the method of storing and archiving software installations should be as efficient as possible, i.e. there can be no waste of storage space in a partition archive.

For the improved cluster maintenance mechanisms investigated in this paper we use a minimal Linux ([3]) installation on all the cluster nodes to control all setup operations. This maintenance OS can be booted remotely for maintenance tasks. While the concepts of our approach by themselves are remarkably simple, the viability is determined entirely by performance aspects of software replication and the storage efficiency of keeping a fair number of incremental OS images in our partition repository and fast replication across the network. The optimal setup and the performance of software replication (cloning) in large clusters of PCs has been discussed in [5].

Partitions are distributed out of our repository by a simple tool called *Dolly*, comprising a small server and a very thin client of roughly 1000 lines of code. The client permits any chosen distribution method (full image, compressed image, incremental to the data) for the bulk of data. Dolly links the machines together in a virtual TCP multi-drop chain and is able to distribute data to all disks in the cluster in a short time over Classic, Fast or Gigabit Ethernet independent of the number of nodes. This gives us full control over how the partitions are distributed and installed on a cluster.

While the archiving of all recently installed OS images offers the advantage of going back in time and being able to correct errors made during an installation processes, it could require a tremendous amount of storage space to store the images, if done naively. Therefore we created *partition repositories*, a technique to archive and restore software installations as partitions with a full base image and incremental changes. Most commercial tools can replicate partition images by ignoring the specifics of the different OSes or store configured OS installations incrementally using the different file systems of the target OS, but so far we have not encountered one that can do both with reasonable efficiency. The partition repository proposed and implemented works in an incremental and completely OS independent manner and can store the different steps of a system installation efficiently. In the evaluation section we present empirical evidence that this approach is fully usable and we quantify the size of the different partition images and their incremental changes based on the example installations encountered in our Patagonia Cluster at ETH Zürich. The partition repository maintenance system runs with Linux and does not depend on any proprietary knowledge of the target operating system or file system. It is simple and built completely from open source software, which makes it perfectly suited for a distribution under an open source license itself.

The rest of this paper is organized as follows: In Section 2 we discuss related work, in Section 3 we introduce the notions of temporal- and spacial-differences in software maintenance. Section 4 describes the implementation details of the partition repository. In Section 5 we quantitatively evaluate our novel approach to software maintenance and in Section 6 we conclude about its significance to large clusters of PCs.

2 Related Work and Limitation of Commercial Products

One of the major challenges for educational system administrators and managers of corporate PC fleets is the constant need to maintain consistency in their distributed system installation, as users or students are continuously changing configurations or adding programs. As educational clusters often comprise a large number of identical machines, disk cloning of a master machine has been used for years but was always used in conjuction with OS specific tools. The brute force method of disk cloning is gaining more and more acceptance in the business world as a technique for software distribution and maintenance. The principle remaining problem with cloning maintenance is

the large amount of data required to store the different OS installations and incremental versions, especially once multiple operating systems are used on the same set of machines in a cluster. This problem is addressed in our paper.

A previous study in this area by Hutchinson et. al. [2] compares the speed of physical vs. logical backups of secondary storage devices to tertiary storage. The authors use a different terminology and misleadingly denote logical backups (i.e. filewise backups) as operating system independent, because the files could be restored on any operating system. Our naming is different: We call physical backups (i.e. blockwise backups) operating system independent, as no knowledge of the underlying file system structure is needed to process backup and restore operations. In contrast to logical backups, physical backups therefore work for any file system and any OS image installed in that partition.

The basic idea of disk or partition cloning is not new and in fact there are already a few successful products available such as *Norton Ghost* [1] ([1]), *ImageCast* [2] or *DriveImagePro* [3]. All these tools are capable of replicating a whole disk or individual partitions and of generating compressed image files to store the partition data. Unfortunately those commercial tools are largely operating system and file system dependent. They use knowledge of the installed operating systems and file systems to provide additional services such as resizing partitions, installing individual software packages and performing post-clone customizations such as e.g., the change of TCP/IP settings. Their operation is therefore limited to the OS versions supported by the tool. Some of the tools do include some functionality of blind disk copy operation, but those are unable to work with efficient storage techniques.

Other tools developed and used by system administrators use similar techniques: The tools described in [6] and [10] use full partition images or `dump` files for backup and restore. The approach with `dump` uses less storage space as it just stores files, while the partition image superfluously comprises unused parts of the partitions. Localizations are performed with shell scripts or by copying modified files from a second local hard disk drive. A more advanced approach presented in [8] uses a revision control system for file systems (*fsrcs*). It stores changed files for each revision of the system installation in a file tree. This approach also saves disk space but requires knowledge about the underlying file system to access the files.

To the best of our knowledge our approach with cloning installations out of an incremental partition repository remains the only solution that is *truly* operating system independent by working with raw disk partition installations. Images are stored in a block repository, where compressed raw disk blocks are administered. Modern operating systems can be setup for automatic installation and customization, e.g. network settings can be initialized by DHCP based on the unique address in the Ethernet adapter. Further post cloning configurations (as e.g. setting a new SID for Windows NT or individual license keys for application software) can be performed by scripts at startups.

[1] Norton Ghost©, Symantec, *http://www.symantec.com/*

[2] ImageCast©, Innovative Software Ltd., *http://www.innovativesoftware.com/*

[3] DriveImagePro©, PowerQuest, *http://www.powerquest.com/*

2.1 Partition Cloning for Efficient Distribution

This section contains the most important details about the cloning system that is closely related to our experimental investigation of partition repositories. An analytic model predicting cloning performance as well as performance measurements on the implemented system is described in [5].

First we need to define the terminology more precisely: The process of storing the contents of a disk or partition to an image file is called a *backup*. The action in reverse, writing an image to a partition is called a *restore*. In our definition, a restore includes the installation on a previously empty, unused or broken partition, as well as the upgrade of an image on an existing installation. The process of backing up an image from one machine and then (later or at the same time) restoring to one or many other machines is called *cloning*.

A somewhat simplistic approach relies on a standard networked file system (e.g. NFS) to backup a whole disk or partition[4]. The machine with the partition to be backed up simply copies its content to an NFS file system exported from a server holding the collection of images. The image can be compressed on the client side to save some storage space. For a restore, all the clients involved will read the (compressed) partition-image simultaneously over the network from the NFS server, possibly uncompress it and write the data to the local disk. This approach is simple, highly robust and most operating systems include all software functions needed. The method has the disadvantages that it does not scale beyond very few clients and that it does not have much room for improvements as each image is stored in one file and served from 1 server to n clients.

In a more sophisticated approach, the backup process can be done exactly as in the first approach or alternatively the master machine to be cloned can also be used directly as server to the clients. The second step is different: The distribution of the partition image for the restore operation is accomplished using a virtual TCP multi-drop chain with the dedicated client. The advantages of this approach are scalability and flexibility: The overall system performance is

[4] The terms *disk* and *partition* can be used interchangeably in this context. While this is not strictly technically correct, they are handled in exactly the same way in our system. Therefore we will use the term *partition* from now on.

limited solely by the performance of a client, not the server, and therefore scales perfectly even for a large number of clients.

In the basic *Dolly* partition cloning framework the partitions are best transfered as a sequence of uncompressed consecutive blocks. In the next section, we argue that it is worthwhile to change this to a sequence of (not necessarily consecutive) blocks and a paired array of disk block numbers and image blocks. With some more advanced schemes, we manage to transfer only unique blocks, saving much storage space and work in incremental installations. During the search for the best data representation we discovered that there is a systematic structure to the various partition images that are used to maintain a cluster installation. The next section analyzes this structure.

3 The Characteristics of OS Installations and Maintenance

Software installations are *not* done in one single, atomic step without ever changing them again. The installation and maintenance of an OS remains an evolutionary and rather incremental process. Because of the changing nature and complexity of todays OS installations, it would be desirable to have some sort of version control for the incremental steps as this is the case with software development. For software development it is a common practice that developers check out a stable part of the software being developed from a repository, improve or change it and check it back into the repository. If at a later time a bug is found, all old versions are still available, as every single version can be fully retrieved from the repository as a snapshot. Since there can be a potentially large number of incremental steps in the lifetime of a software development project, it is not efficient to save all full revisions of the software in the repository. The repository therefore stores *only the differences* between each version. As we will show, the same can be done with software installations in hard disk partitions. In Figure 1 we depict the typical structural relationships of an OS partition image. We explain that a cluster software installation experiences changes (due to version upgrades) along a temporal axis during its lifetime as well as changes along a spacial axis when replicated into a large number of PCs in a cluster and adapted to different clients and their hardware.

3.1 Installations and Upgrades: Temporal Differences

A common first step in OS installation and maintenance is to install the basic OS including the kernel and only the most common programs needed to run the system. In further steps more patches, service packs and additional software packages are installed. Later in the life cycle new patches need to be applied or users demand the installation of new or the upgrading of existing software packages. This process is error-prone: Software-parts might be installed in an incorrect order or might not fit together, or a configuration option might be badly chosen. For such cases it is extremely helpful if single installation steps can be completely reversed by reverting back to the last working version of the installation without having to start from the beginning again.

Sometimes software is only temporarily installed on a cluster, e.g. for special classes, courses, experiments or tests, and removed soon after installation. Most packages provide de-installing options, but often these are not capable of inverting every change done to the system during the installation of the package. These small remaining changes result in a so-called *software rot* which makes the system as a whole unstable and increasingly difficult to maintain.

The OS installation is therefore changing (and hopefully improving in quality and security) all the time, but only to the price of high complexity.

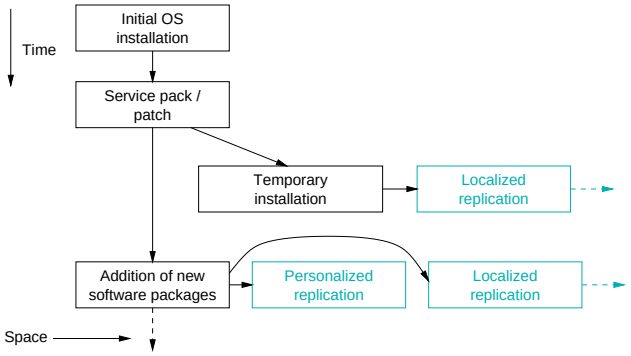

Figure 1. Diagram showing the evolutionary steps of a typical software installation. As software is added, changed or patched, the installation evolves over time (downwards). Localized and personalized replications of the installation on different machines are separated horizontally. An OS installation and maintenance tool should capture changes in both directions.

3.2 Replication and Local Configuration: Spacial Differences

Some software packages also require local changes or adaptions on the machines, such as license keys for commercial software or changes to the local configuration files or the Microsoft Registry Database. These are only small changes between the installations, but it might nevertheless be worthwhile to store those spacial differences: When a

hard disk fails, the exact images for the partitions of that machine can be restored and no further configuration is required.

Since the original OS image and its copies on the disks are bit-wise identical, some localizing configuration steps must be taken to make them operational. In most cases a freshly cloned OS cannot be brought to life with a simple booting process but requires some customization. We use a DHCP server on the same Ethernet segment to assign IP addresses and machine names based on the unique Ethernet MAC address built into the primary network interface of each PC. Additional scripts are performed at boot time to initialize further settings such as setting a unique SID, logging onto a domain controller for Windows NT or selecting the correct driver for the graphics card installed.

Another possibility for machine specific changes would be to use *scripted installs*. With this approach, a script does the local configurations and software installations automatically after the initial OS is installed. Scripted installs have some disadvantages however: (1) They are OS dependent as the OS must support scripting languages and be configurable by scripts. (2) They are slow since the file system must be used. ([2] describes how the direct physical access of the disk is much faster than using the logical file system, while [5] shows that fast network-based installations on clusters are in fact possible with raw disk accesses). (3) The removal of software packages with a script is not as clean as restoring exactly the same installation. Most OS manufacturers recommend scripted installs from an original CD ROM distribution and force the user to become OS dependent. Our techniques are truly OS independent and will therefore work for future releases of Linux and Windows. Once some form of auto-configuration at startup time can be worked out a replication to hundreds of PCs is easily possible. However, auto-configuration and fully floating licenses are not always possible. With future operating systems it is well conceivable that a fully automatic configuration can still not be worked out and manual intervention becomes necessary. Therefore the partition repository technique allows the storage of fully localized images for each machine, and due to their efficient management of differences, can do so without exceeding the storage budget. Since partition repositories are oblivious to the file system it does not matter if the application stores its customized license key in a file, in the executable or in an entry of the central Microsoft Registry Database. The disk block based incremental imaging techniques will correctly apply any changes upon a restore.

Abstraction of the installation state and a strict OS independence seems to be the only viable approach to this maintenance problem since it is very hard to obtain proper documentation about the configuration state of OSes and application programs.

4 Partition Repositories for Incremental Maintenance

The advantages of working with partitions in an OS independent way have been outlined in the previous sections. Archiving localized and fully configured images and entire maintenance histories of a large cluster of PCs as raw image files requires a lot of storage space. Therefore most commercial tools revert to incremental storage in a file system or other dependent schemes for image maintenances.

We implemented a much different scheme for the storage of all information required to do software installation in a cluster. The implementation required many tradeoffs and design decisions, and therefore we need to quantify the storage requirement and the savings of optimized storage techniques such as the deletion of zeroed blocks, image compression and an incremental storage method called *partition repository* with a few typical example installations.

4.1 Implementation of Partition Repositories

Partition repositories work with partition images exclusively (no knowledge of the different file systems is desired or required) and therefore the method is completely OS independent. Our software maintenance system will work with any future OS version and any file system that is or will be used on our cluster of PCs. The system works at partition level and at disk device level. At disk device level there are no restrictions on the data layout of a disk and even non-standard formats, such as Oracle data disks, can be replicated. At the partition level the system works with the standard partitioning system of the cluster platform (i.e. the partition structure of the Intel, Sun or PowerPC platform). Our system relies on the partition access facilities provided by the Linux maintenance OS.

4.1.1 Optimized Storage of Full Partition Images

The most simple and obvious approach to manage different versions of installations is to store the partition image of each new or upgraded OS installation in a file on a server. To save valuable disk space, the images can be compressed with a generic data compression algorithm such as Huffman or Lempel Ziv encoding — our tool uses *GNU zip*. According to our experience, partition images of moderately filled partitions can be compressed to roughly 50% of their original size, depending on the kind of data installed and the amount of free space in the partition. We occasionally use various file system dependent tools to “wipe” free space or to fill empty blocks with zeros after a complex installation. Such tools help to achieve better compression. As there is exactly one image-file per installation, archives and restores are simple to manage. On the other hand, a lot of disk space

is wasted as most of the information in two incremental installations is identical.

4.1.2 Storing Partition Increments (difference, blockwise)

A better and more advanced approach to capture at least some of the similarities between two software installation images is to store the full base image in a file for the first installation. Subsequent installations are then generated by comparing the updated partition on a block-by-block basis. Only blocks which actually differ from previous reference images are stored in a so called *diff-file*. With this approach, the unchanged blocks are not stored twice. Small changes in an installation can thus be archived quite efficiently.

The generation of the diff-files is simple: The old image file and the upgraded partition are both read sequentially and each block is compared with the block at the same position of the other input stream. When two blocks differ, the block is written to the diff-file. When the two blocks are identical, only the number of the block in the original image-file is written to the diff-file, thus saving the space of an entire block.

There are some cases that are not handled very well by this simple method: For some operating systems, the removal and re-installation (probably with a partial update) of software-packages is a common operation. In this case, the program itself, some configuration files as well as shared libraries, are frequently moved to other locations in the partition, but not necessarily changed. For some file systems it is a common operation to defragment the stored files, which results primarily in moving data blocks around. Only some block and directory structures on the disk need actually to be changed. Another case are modern log-structured file systems (see e.g. [7, 9]) in which even unchanged blocks are frequently moved around by the cleaner to regain space in new empty segments. These changes in location rather than content are not detected properly by the block-by-block compare.

4.1.3 Storage of a Partition Image in a Block Repository

Our most sophisticated solution, the partition block repositories, overcomes the limitation of the two previous solutions and is fully enabled to detect the relocation of unchanged data during the difference calculation. It works by comparing all blocks of the base image and incremental changes with each other and by storing only the unique blocks in a *block repository*. The result is that blocks are not stored again in the block repository if they were just moved around during an upgrade or defragmenting process by a system administrator or the cleaning process of a log-structured file system. An new differential image only contains pointers to the blocks to be found in the repository. Changed blocks not yet in the repository are stored and a pointer to them is inserted in the corresponding archive file. This approach has the additional advantage that identical blocks in the same image (such as zeroed blocks or common identical font files, libraries, graphics or dictionaries found in may software packages) will also be stored only once, resulting in improved storage efficiency.

The downside of this most advanced approach of storing software installations is that comparison of all blocks is an expensive operation, as each block of an image has to be compared with each block of the other image. However, there are some possibilities to speed up the process. Instead of cross-comparing all the blocks, we speed up the process by generating hash tables over the contents of the blocks at runtime when comparing compressed images. We then compare the hash-values in memory only. For colliding hash values, the blocks have to be compared in a second comparison pass, but the comparison of images is nevertheless much faster than it would be with a naive approach. The comparison of two uncompressed Windows NT partitions of 2 GByte each takes about 7 minutes on a high end PC and requires a few hundred MB of memory. The same comparison with compressed images took less than 9 minutes.

This operation is done only once for each archived installation step and is not required for the more frequent restores. Furthermore, it is still much smaller compared to the time it usually takes to apply and test a system installation or upgrade, and it can easily be done as a background process during the night, which makes it a reasonable effort.

5 Experimental Evaluation

For evaluating the different storage schemes we study a few partition image series of real OS installations on our Patagonia cluster of PCs. The images studied are taken from our server which runs the software maintenance system as described in Section 4.1.1. Figure 2 shows the evolutionary steps of the installations and the replications to multiple PCs in our cluster.

Table 1 lists the relevant storage requirements for a comparison of the different methods of storage and for an estimate of storage requirements to maintain all system software in a large and diverse cluster of PCs or a fleet of corporate PCs.

In Table 1 the different storage methods as described in Section 4 are compared. For some installations we have a base installation and one incremental version available; other tests are done with two increments. With the blockwise diff approach, the base image size is equal to the partition size of the installation, while the blockwise repository base is slightly smaller due to the detection of identical disk

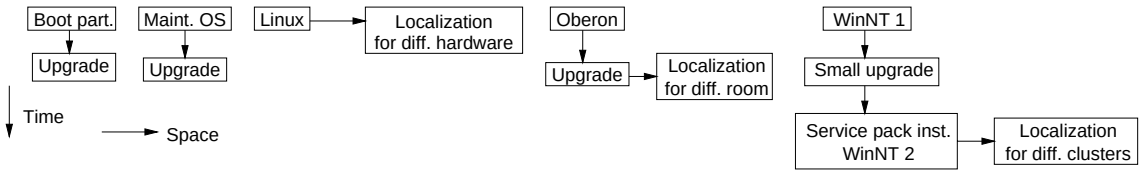

Figure 2. Evolutionary steps of the installations in Table 1 showing the temporal and spacial changes.

Installation	Full images			Blockwise diff			Blockwise repository		
	Base	Increment	Total	Base	Increment	Total	Base	Increment	Total
Boot partition	16	16	32	16	(18kB)	16	14	(18kB)	14
Maintenance OS	205	205	410	205	0.3	205	195	0.3	195
Linux	945	945	1890	945	12	957	844	6	850
Oberon	106	106 + 106	318	106	25 + 22	153	28	10 + 2	40
WinNT 1	2048	2048 + 2048	6144	2048	35 + 430	2516	1893	22 + 280	2195
WinNT 2	2048	2048	4096	2048	29	2077	1804	17	1821

Table 1. Comparison of the three presented approaches of storing incremental installations for different production images in use on the Patagonia cluster at ETH Zürich. All numbers are in MByte except where otherwise noted.

blocks. The listed diffs are calculated between the first version and the second version of the installation or from the second version to the third one respectively. The listed installations have the following characteristics: The first two installations are a very small boot partition, where only a minor change was applied, and a maintenance OS (Linux in our case) which was slightly upgraded. The series of two subsequent Linux installations differ in a few configuration and kernel changes for slightly different hardware (amount of memory, processors, graphic and network cards), resulting in a few changed and a few relocated blocks. Oberon uses only a small portion of its partition which results in a quite small partition repository. The second upgrade is the installation of nearly the same system for slightly different hardware, resulting in mostly the same information at different locations on the disk. The fifth example is a series of Windows NT installations starting with a baseline Windows NT installation followed by service pack installs. The second Windows NT installation is for two nearly identical PC clusters where only a few default settings (such as printers and boot scripts) have been changed.

In Figure 3 the different storage techniques for Windows NT and the Linux maintenance software installation series is examined in more detail. The above methods are combined with compression of the base archive as well as the increments. The incremental archive approaches show a great potential in saving disk space without knowledge of the underlying file system.

OS Installation	Part. Size	Repository size	
		Orig. img.	Incr. after inst.
Windows NT	2048	1791	203
Linux	2048	1535	209

Table 2. Storage requirements in the repository for installing the same office package *StarOffice 5.1* on both Windows NT and Linux. The numbers are in MByte.

We then compare the storage requirements of our disk block repository when installing the same software package on different OSes with different file systems. The software package used for this experiment is Sun Microsystems' StarOffice 5.1 which is available for both Microsoft Windows NT and Linux. The results shown in Table 2 and Figure 4 show that the diskblock repository technique works comparably well on both OSes and file systems. According to the installation documentation of the software, the package requires 110–140 MByte of permanent disk space and around an additional 20 MByte during the installation. The numbers in the table are higher because our tool also captures changed entries in the directory and block handling structures of the file system.

Another common operation in software installation maintenance is upgrading or patching the base OS or ker-

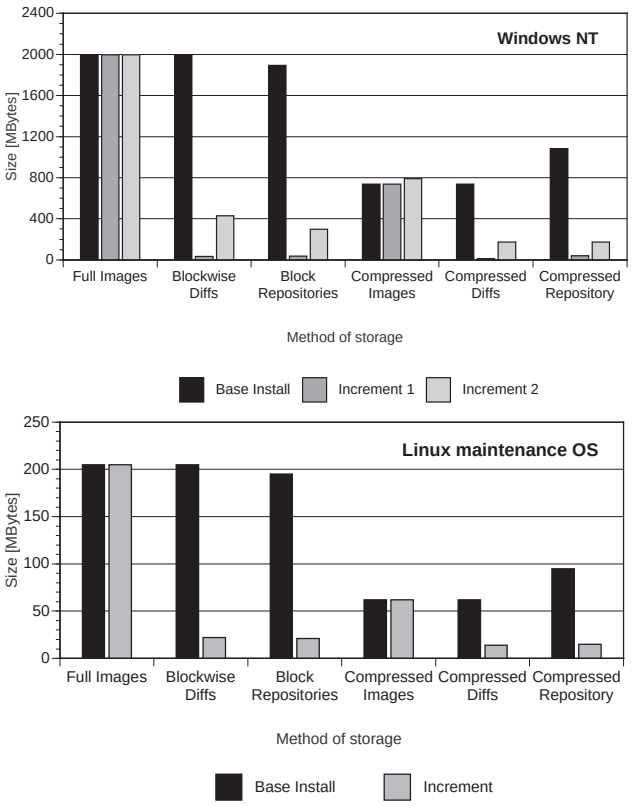

Figure 3. Storage required for a successive Windows NT and Linux update path. Three different generations of installation steps from a base install to a fully updated image are examined for Windows NT and two generations for the Linux maintenance OS respectively. The three presented methods of archival are shown, without compression applied in the left three cases and combined with compression in the right three. The blocksize used for the diffs and the repository is 1 KB.

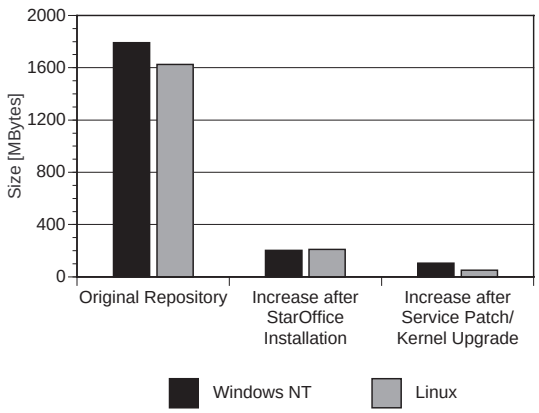

Figure 4. Storage requirements in the repository for an incremental software update, i.e. installing the same new office package *StarOffice 5.1* and for upgrading or patching the base OS or kernel respectively on both Windows NT and Linux. The original images are 2 GB large.

OS Installation	Part. Size	Repository size Orig. img.	Incr. after patch Abs.	Perc.
Windows NT	2048	1791	106	5.9 %
Linux	2048	1535	50	3.2 %

Table 3. Storage requirements in the repository for upgrading or patching the base OS or kernel respectively. The numbers are again in MByte.

nel respectively. We thus measure typical patch operations for Windows NT and Linux. On Windows NT we upgrade from Service Pack 4 to Service Pack 5, while on Linux we patch the kernel source tree on the machine from 2.2.14 to 2.2.15, recompile and install the kernel. These two operations are not directly comparable but show typical software maintenance tasks for both OSes used in our clusters. The results are depicted in Table 3 and Figure 4.

In our next experiment we compare the installations on two identical machines which have been localized, again both for Windows NT and Linux. The Windows NT system was cloned, self-configured during the first bootup, and rebooted eventually as the new localized machine. For Linux there is no need for such a configuration step in our installation, as completely identical images can be used (we use DHCP for setting the IP-numbers and hostnames, we do not have software with local licenses and we use scripts to detect and select the right graphic card driver during bootup). The Linux numbers are derived by comparing two identical machines after they were only in light use for a few weeks. The results are shown in Table 4. The table's last column shows the number of localized partition images that can be inserted into the repository before the repository is twice as big as the uncompressed partition. This means that we can store 75 localized Windows NT installations in a repository that has only twice the size of the partition.

A shortcoming of the blockwise repository storage technique is that the fragmentation into blocks adversely af-

OS Installation	Partition Size	Repository size: Original Image	Increase after localization: Absolute	Increase after localization: Relative	# of inst. to double
Windows NT	2048	1791	27	1.5 %	75
Linux	2048	1535	9	0.5 %	227

Table 4. Storage requirements in the disk block repository for localized replications. The numbers are in MB except in the last column, which lists the number of localized replications that can be inserted in the repository before its size is twice as big as the uncompressed partition.

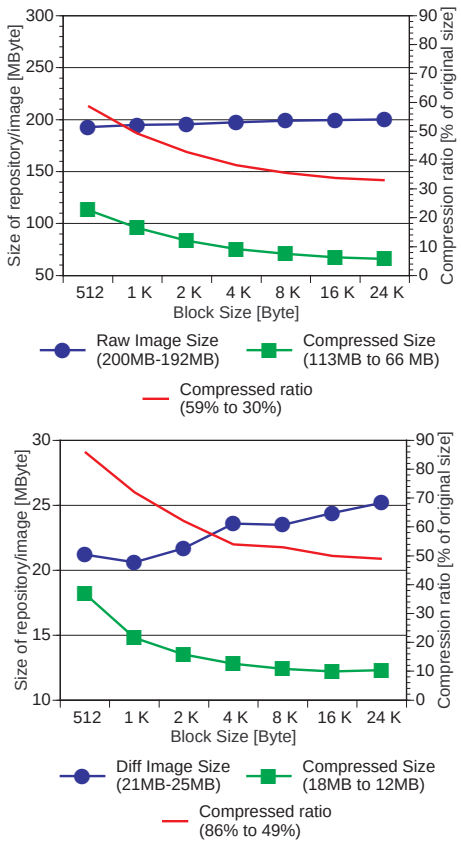

Figure 5. Comparison of disk space requirements depending on different block sizes for the comparison (diff) and compression algorithm. The 205 MByte Linux maintenance partition is inserted alone in the upper figure, while in the lower figure, the first update is also inserted. In the upper case the absolute sizes of the uncompressed and compressed repositories are compared and the compression ratio is shown. The lower figure shows the same characteristics for an incremental partition image of 25 MByte as an uncompressed and compressed repository after inserting the updated partition.

fects the compression rate since compression is now blockwise (instead of the whole image at once) so that individual blocks can later be accessed directly without first uncompressing the whole repository-file. We examined the rate in more detail by comparing the raw size as well as the increase (diff size) of the uncompressed and compressed repository, after inserting a base installation as well as an updated partition. We store the 205 MByte large maintenance Linux base partition as well as its 25 MByte incremental updated image in the repository and use varying block sizes for the gzip compression algorithm to compare (diff) and compress the blocks. The results in Figure 5 show that for increasing block sizes the uncompressed repository size increases due to the coarser granularity of difference detection because, for small changes, a larger block needs to be inserted into the repository. The compression algorithm, on the other hand, works better for larger block sizes, thereby improving the compression factor. The results indicate that one might reduce the disk space requirements considerably by using 16 KByte blocks and by optimally combining blockwise repositories to whole image compression techniques. 16 KByte seems to be optimal for the incremental image, since the compression algorithm shows little additional gains beyond 16 KByte and is offset by the loss in the accuracy of difference processing.

6 Conclusions

In our study we analyzed the problem of software maintenance in large clusters of PCs. We think that our work applies to traditional clusters for scientific computing, novel clusters for multimedial collaboration or even computers in a corporate fleet of PCs. Versatility in the use of clusters and the rapid release schedule of different operating systems in the commodity software market require a cluster maintenance tool to use simple and clean abstractions of the state comprised of system installation.

Therefore, we propose to build a software maintenance system for clusters based on the storage and the distribution of entire partition images. The system thus does not depend on any operating system or file system. In previous work we proposed to use our cloning tool *Dolly* for high

speed partition cloning and data distribution across the high speed network of large clusters of PCs. In this paper we investigated blockwise partition repositories to address the storage problems associated with the partition maintenance approach in large clusters of PCs. We clearly identify two reasons for a replicated storage of partitions: the temporal evolution of an OS installation following the release schedule for updates and the installations of additional middleware or application packages, the individual configuration of images for heterogeneous hardware environment or for network or license key configurations.

The implementation of a block-based difference scheme demonstrates that we can reduce the storage needs for keeping a full software history based on partition images drastically by storing increments of 1% up to 20% of the space that would be required to store a full partition. A comparison between a release history of Windows NT and Linux partition on our dual boot cluster does not reveal any fundamental difference between the two most popular cluster operating systems and shows that our partition repositories work for both operating systems in the same way.

We also demonstrate that storing fully customized images for the different nodes in a large cluster is relatively cheap. Since our Linux system offers complete network autoconfiguration and none of the Linux Software packages relies on individual license keys the capability for localized replication is not as important for Linux as it is for Windows NT. For Windows NT our method can store up to 75 incremental customizations in the space required by a typical 2 GB operating system installation. We hope that all future operating systems will eventually migrate to full networked autoconfiguration, but we still appreciate a maintenance system that offers a fallback solution. In our evaluation we carefully consider the interaction between difference processing that works best at a fine granularity and data compression that works best at large granularity. It appears that the best block size for block partition repositories would be around 16 KByte.

Our software maintenance system based on high speed partition cloning across the network and highly efficient storage in partition repositories is at the stage of a highly modular university prototype, and some parts of the system are already in daily use by our system administrators. It is based on simple ideas, and all software involved is available under open source.

References

[1] Symantec Corporation. Norton Ghost: Disk Cloning Technology for the Overburdened IS Professional, 1998. http://www.symantec.com/sabu/ghost.

[2] Norman C. Hutchinson, Stephen Manley, Mike Federwisch, Guy Harris, Dave Hitz, Steven Kleiman, and Sean O Malley. Logical vs. Physical File System Backup. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, New Orleans, Louisiana*, pages 239–249. The USENIX Association, February 1999.

[3] Linux Online. Linux online information. WWW site, 1999. *http://www.linux.org/*.

[4] Felix Rauch, Christian Kurmann, Blanca Maria Müller-Lagunez, and Thomas M. Stricker. Patagonia — A Dual Use Cluster of PCs for Computation and Education. In *2. Workshop Cluster Computing, Karlsruhe*, March 1999.

[5] Felix Rauch, Christian Kurmann, and Thomas M. Stricker. Partition Cast — Modelling and Optimizing the Distribution of Large Data Sets on PC Clusters. In Arndt Bode, Thomas Ludwig, Wolfgang Karl, and Roland Wismüller, editors, *Lecture Notes in Computer Science 1900, Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference Munich*, Munich, Germany, August 2000. Springer. Also available as Technical Report 343, Department of Computer Science, ETH Zürich, `http://www.inf.ethz.ch/`.

[6] Paul Riddle. Automated Upgrades in a Lab Environment. In *Proceedings of the Eighth Systems Administration Conference: (LISA VIII)*, pages 33–36. USENIX Association, September 1994.

[7] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM SIGOPS Operating Systems Review*, 25(5):1–15, 1991.

[8] Gottfried Rudorfer. Managing PC Operating Systems with a Revision Control System. In *Proceedings of the Eleventh Systems Administration Conference (LISA '97)*, pages 79–84. USENIX Association, October 1997.

[9] Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, and Carl Staelin. An Implementation of a Log-Structured File System for UNIX. In *Proceedings of the Winter 1993 USENIX Conference*, pages 307–326. The USENIX Association, 1993.

[10] Michael E. Shaddock, Michael C. Mitchell, and Helen E. Harrison. How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday. In *Proceedings of the Ninth Systems Administration Conference: (LISA IX)*, pages 59–65. USENIX Association, September 1995.