The following paper was originally presented at the
Ninth System Administration Conference (LISA '95)
Monterey, California, September 18-22, 1995

OpenDist - Incremental Software Distribution

Peter W. Osel and Wilfried Gnsheimer
Siemens AG, Mnchen, Germany

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

OpenDist – Incremental Software Distribution

Peter W. Osel and Wilfried Gänsheimer – Siemens AG, München, Germany

ABSTRACT

OpenDist provides efficient procedures and tools to synchronize our software file servers. This simple goal becomes challenging because of the size and complexity of supported software, the diversity of platforms, and because of network constraints.

Our current solution is based on *rdist*(1) [1]. However, it is not possible anymore to synchronize file servers nightly, because it takes several days just to compare distant servers.

We have analyzed the update process to find bottlenecks in the current solution. We measured the effects of network bandwidth and latency on *rdist*. We created statistics on the number of files and file sizes within all software packages.

We found that not only the line speed, but also the line delay contributes substantially to the overall update time. Our measurements revealed that adding a compression mode to *rdist* would not have solved our problem, so we decided to look for a new solution.

We have compiled a list of requirements for evaluating software distribution solutions. Based on these requirements, we evaluated both commercial and freely available tools. None of the tools fulfilled our most important requirements, so we implemented our own solution.

In the following we will describe the overall architecture of the toolset and present performance figures for the distribution engine that replaces *rdist*. The results of the prototype implementation are promising. We conclude with a description of the next steps for enhancing the OpenDist toolset.

Our Environment

The CAD Support Group of the Semiconductor Division of Siemens AG installs, integrates and distributes all software needed to develop Integrated Circuits. We have development sites in Germany (München and Düsseldorf), Austria (Villach), the United States (Cupertino, CA), and Singapore. The development sites are connected by leased lines with a speed of 64 to 128 kBit/s. At each site, a central file server stores all software. Client workstations mount software from these servers. Software is installed and integrated in München and distributed to all other development sites. System administrators of the development sites initiate the transfer on the master server in München.

The CAD Support Group takes care of the CAD software and tools, only. A separate department is responsible for system administration, i.e., maintenance of the operating system and system tools, backups, etc.

Our software distribution problem differs in many ways from the one solved by traditional software distribution tools. Most software distribution tools we looked at are designed to distribute a moderate number of fairly static software packages of moderate size to many clients.

In contrast, we have to synchronize few file servers (under a dozen), which store many (about 200) packages of sizes ranging from tiny (a couple of kilobytes) to huge (1.8 GBytes). The total size of the software we store is currently 25 GBytes, 10-15 GBytes are currently being kept up-to-date at all sites. Many packages are changed each day. A change might update only a single file of a few bytes or could change up to 50,000 files for a total of 1 GBytes per day. Every month about 10 % of the software change. Most changes are small, but many files are constantly updated. The installation of a huge patch or a new software package changes many files at once.

There is no separate installation- or test-server, all changes are applied to the systems while our clients are using them. The changes are tested in München and, ideally, copied to all slave file servers within one day. Synchronizing or cloning file servers is the best way to describe our setup.

Our Current Solution

Our current software distribution process uses *rdist*(1) to find changed files and to update slave software servers. It is no longer possible to compare two software servers in one night. A complete check of all software packages on the slave file server in Singapore would take several days which is not acceptable nor feasible. During that time, software packages would be in inconsistent states, and changes of the master software server could take

up to a week to be transferred to the slave file server. Though it is possible to apply different update schedules – updating small packages daily, some weekly – the setup is not satisfactory. With an ever-increasing number of software packages and an ever-growing size of each software package, the distribution process using *rdist* is not acceptable any more.

Searching The Bottleneck

We have analyzed the update process to find bottlenecks in our current solution. We analyzed our lines and measured bandwidth, latency and compression rate (all leased lines are equipped with datamizers – devices that compress all traffic). We created statistics on the number of files and their size for more than 200 software and data packages. Commercial software packages, technology data and cell libraries, as well as many free packages like X11 and gnu tools were analyzed. We were also interested in the compression rate and time of software packages and how much the compression rate differs when software packages are compressed file by file or as a complete archive. We analyzed where *rdist* spends its time during updates. Compared to the installed software, our change rate is small, so finding changed files must be efficient. Changes can be rather huge, so the transmission of changed files must be efficient, too.

The Benchmark

We wrote a benchmark suite that measures the elapse time needed to perform typical software distribution operations such as installing, comparing, deleting, and updating files of different sizes, installing symbolic and hard links. All operations were executed many thousand times to equalize differences of the link performance.

The benchmark measures *ping* (1), *rcp* (1), and *rdist* (1) performance and times. Each *rdist* test runs on a directory with an appropriate number of random files of the same size. Each test contains an add, check, update and delete sequence. The file size is increasing from 1 Byte to 1 MBytes. Thus the effect of transfer rate and *rdist* protocol can be separated. The *rdist* part of the benchmark source tree contains approximately 5,000 files. This sums up to 10,000 transferred files, 5,000 check actions, 5,000 delete actions and 30 MBytes transferred data per test run. *rcp* (1) times are measured for a text, a binary and a compressed file of 1 MBytes each. This shows the achieved on-line compression.

The leased lines (except the dialup ISDN link) are shared by many users. So it is not astonishing that the benchmark results varied a lot, sometimes by more than a factor of three. To make our benchmark of the line performance more comparable, we calculated the average value for the best results of several runs of the benchmark. Some of the small numbers are within the magnitude of time resolution and must be interpreted cautiously.

The Results

Size and Composition

Software packages vary substantially in size and composition of file types, however bigger packages don't necessarily have bigger files, they have a few huge files, but the average file size is more or less independent of the total size of the package (Diagram 1).

	LAN	MÜNCHEN	DÜSSELDORF	VILLACH	CUPERTINO	SINGAPORE
Line Type	Ethernet	ISDN	X.25	leased	X.25	leased
Nominal Line Speed [kBit/s]	10,000	64	64	128	64	64
Transfer rate [kByte/s]	90-100	6-7	4-5	7-12	2-3	3-4
Ping Response Time [ms]	<1	33-88	188-372	81-311	530-1083	617-1375
rdist file create [s]	0.2	0.2	1.2	0.6	2.1	4.5
rdist file check [s]	0.02	0.06	0.5	0.2	1.	2.1
rdist file delete [s]	0.1	0.13	0.5	0.3	1.	2.3
10 kBytes transfer rate [kByte/s]	-	5.9	2.5	4.9	1.5	1.4
Run Benchmark [h]	1	2.5	7	4	12	24
rdist check SW subset [h]	-	-	16	8	-	>80[2]
OpenDist check SW subset [h][3]	0.5	-	2[1]	.5	.75	.75
rdist check all SW [h][2]	3	-	69	27	140	290
OpenDist check all SW [h]	1.5	-	5[1]	1.5	2	3

[1]Increased time, because software pools in Düsseldorf are accessed via NFS not UFS.

[2]Estimated.

[3]This subset consists of technology data and is changed and distributed daily. The subset contains approximately 150,000 files with a total of 1.1 GBytes.

Table 1: Line Characteristics

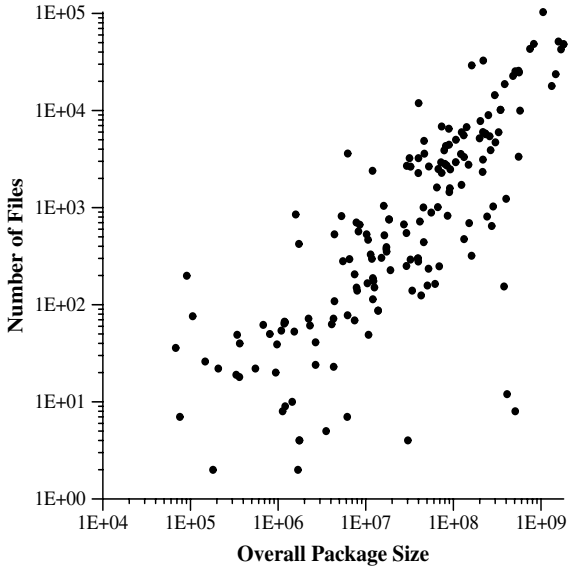

Diagram 1: Package Size vs. File Count

Compression Factor

The average compression factor of our software packages is three. Most of our software packages were compressed by this factor, though we observed compression factors between two and five.

When using *gzip*(1), you can regulate the compression speed between fast (less compression) and slow (best compression). For our software packages, increasing the compression quality reduces the compressed file size by less than 5 %, the compression time however sometimes increased by more than 200 % (Diagram 2). The default compression level of 6 is a good compromise, so we decided to use it.

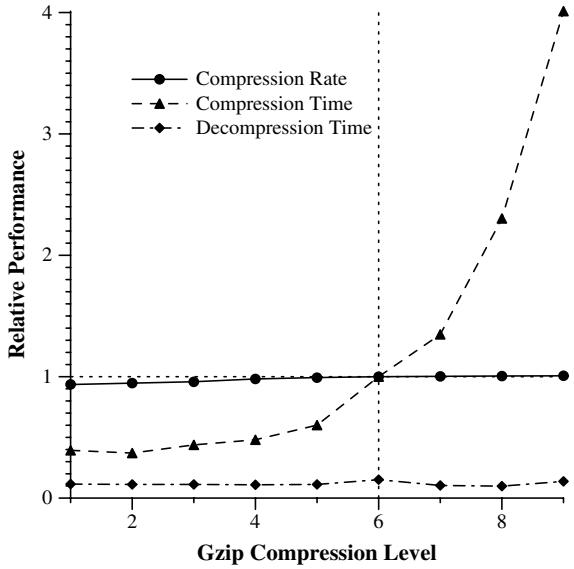

Diagram 2: Gzip compression Quality and Speed

Though our leased lines are equipped with datamizers that compress network traffic, it is worthwhile to compress archives before transmission. Datamizers increased the transmission rate of uncompressed data by 10 .. 15 %, whereas *gzip* reduced the data to a third of their original size.

Compression Rate

On a SPARCstation 10/41 (Solaris 2.4, 128 MBytes memory) *gzip* created compressed data at a rate of 65 kByte/s, many times faster than the speed of our leased lines. This figure is important to know when you want to pipeline the creation, compression, and transmission of update archives. In case the throughput of the lines is in the same order of magnitude as the *gzip* output rate, it would be advisable to decrease the compression level.

Decompression

Decompressing the archives with *gunzip*(1) is usually six times faster than compressing the data. Decompression time does not depend significantly on the compression quality chosen for compression (Diagram 2).

Compression and Archives

It is better to compress an archive of files than to archive compressed files. Compressing complete packages is significantly faster and creates smaller archives than compressing each file separately and archiving the compressed files. For example, archiving and compressing X11R6 was completed in three minutes elapse time, and the overall size was reduced by 55 %. Compressing each individual file and archiving the compressed files in a second step took five minutes elapse time and reduced the overall file size by only 45 %. All tests were performed several times on an unloaded machine. Compressing individual files and archiving them needs many more file and disk operations compared to archiving the uncompressed files and compressing the archive. Compressing several small files (or small network packets) is not as efficient as compressing the files in a single run.

Transmission and Archives

It is better to transmit an archive of files than to transmit each file individually. Depending on the file transfer protocol used, the latency of the line has a high impact on transfer rates. The smaller the files and the higher the latency, the higher is the delay caused by inefficient protocols.

The latency increases the time *rdist* needs to check or create files. If you have many files, *rdist* needs a long time to compare master and slave server. If many or all files changed (e.g. when installing a new software package), *rdist* will need much more time to transfer all files. The average file size of our software packages is 30 kBytes (Diagram 3). To our Singapore site, we need about 10 seconds (3 kByte/s) to transfer a file of this size. However, *rdist* needs more than 4 seconds to create the new file, for a total transmission time of 14

seconds (40 % increase), a 30 % decrease in transfer rate. The transfer rate for 10 kBytes files is only half of the normally achievable transfer rate (See Table 1).

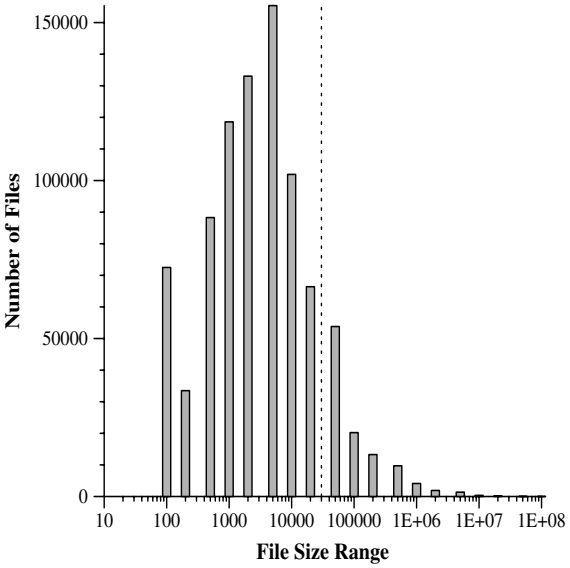

Diagram 3: File size range (All packages)

Besides avoiding protocol overhead, the transmission of archives has additional advantages. By first transferring all changed files to a holding disk, and installing changes locally on the remote server from the holding disk, the time during which the software package is in an inconsistent state is significantly reduced. Moreover, we can use the same tools to archive and roll-back changes. The installation of changes can be done asynchronously, so a system administrator at the remote site can easily postpone updates. The advantages compensate the disadvantage of needing holding disks to temporarily store the file archives.

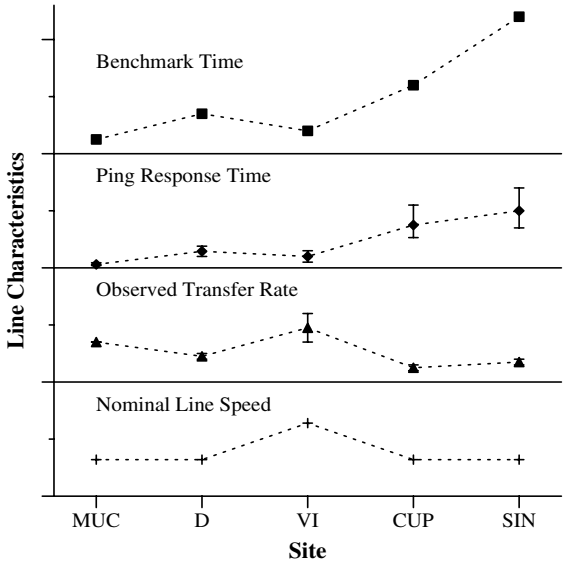

Diagram 4: Line Characteristics

rdist and Latency

Although the line speed from München to Villach and to Singapore differs by only a factor of two, the time needed to run the *rdist* benchmark differs by a factor of six (see Table 1 and Diagram 4). The ping response time (and therefore latency) has a greater impact on the time *rdist* needs to create or compare files than the line speed.

Benchmark Summary

Our measurements have revealed that the line speed is not the only bottleneck: the latency also plays an important role. *rdist* compares source and target directory file by file. Because the time for this is proportional to the latency, and because our change rate is small compared to the installed software, adding compression to *rdist* would not have solved our problem. *rdist* spent most of its time trying to figure out what to update, and not actually updating files. On the other hand, if a new version of our biggest software package is installed, we have to transmit 1.8 GBytes, so transmission must be optimized, too. The transmission of single files is another bottleneck as in our environment, the protocol overhead and transmission time are in the same order of magnitude, which reduces the average actual transfer rate by up to 30 %. For an efficient solution in our environment, files that have to be updated must be archived first and then be transmitted in one large file.

Upgrading our lines would not solve our problem, because the latency would not get small enough. It is also a very costly solution.

We found that we had to tackle two problems: making the finding of changed files more efficient, and making the transmission of data more efficient. We began to look for a new solution.

Requirements for Software Maintenance

We compiled a long list of requirements that a new solution should fulfill. Here are some of the more important ones:

Optimal Support of Incremental Distribution

We do not want to trace changes as they are applied and re-apply them at a later date on slave file servers. Changes should be found by comparing the status of the master and the slave file server. Comparison should be stateless – it should not depend on update history. Each file server is administrated by independent system administrator groups, so we don't want to rely on what we think the status is, but we rather have to check the actual status of the remote file server. We have to detect changes applied by remote administrators.

Update Programs Currently Executing

Files that are updated may not be overwritten. The old file has to be moved and unlinked, then the new file has to be moved to it's final destination.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.