File Distribution Efficiencies: cfengine vs. rsync

Andrew Mayhew – Logictier, Inc.

ABSTRACT

This papers reports on a preliminary investigation of the performance of cfengine and rsync for file distribution for the purposes of system maintenance. It focuses on two aspects of file distribution: the transfer during an initial copy from a master server to a client, and the file verification of the client's files against the server's files. It is shown that for larger file transfers rsync performs better, while cfengine manages smaller transfers better.

Introduction

In originally implementing cfengine (v1.5.4) for host management on our network, we ran across several problems with the file transfer protocol [smith]. Due to faults which would halt transfers mid-session we decided that a replacement method for synchronizing applications and configurations between the servers and individual nodes was needed. We did not, however, wish to completely get rid of cfengine, because we still found its other features for system immunization useful. As such, we implemented an rsync over SSH system controlled and run from each particular host's cfengine agent. While this has worked quite well, this kludge introduces some possible instabilities and quirks in operations.

As cfengine has matured, many of the problems and bugs that we originally faced have been addressed. In reviewing our file synchronization method against current versions of cfengine, we decided to more systematically test various copy methods available to see which was really most suitable for our needs. While there are several other file-copying systems available, our testing is focused on cfengine and rsync and how various levels of securing the network affect transfer speeds.

Testing Methodology

The repository server was a Sun E450 (four 440 MHz UltraSparc CPU's and 2 GB of RAM) running Solaris 7. The clients being served in these tests were a Dell PowerEdge 2450 (two 700 MHz PIII CPU's and 512 MB of RAM) running RedHat Linux 6.2, and a Sun Netra T1 (one 440 MHz UltraSparc CPU and 1 GB of RAM) running Solaris 7. In all cases, the machines were running near completely idle with only a minimum number of required processes running in order to obviate any possible interaction issues. All hosts ran 100 mbit full-duplex Ethernet connected to a single switch isolated from the rest of our network, to eliminate any possible networking issues. Each test run was done in a serial fashion so that only one client was communicating with the server at a given time.

The versions of the file transfer applications used at the time were cfengine v1.6.3 and rsync v2.3.1. Cfengine was additionally patched to be able to directly call any external copy method desired by the user [patch]. A typical rsync incantation and similar cfengine configuration are shown in Listing 1 for example purposes.

There were four different copy configurations run with four different-sized transfers. The copy configurations were:

- **Baseline cfengine copy**: This uses the native non-encrypted cfengine copy method.
- **Tunneled cfengine copy**: Native cfengine copy method run over an SSH tunnel.
- **3DES cfengine copy**: Native encrypted cfengine copy method.

```
rsync -qrplogDze "ssh -l cfengine -i .identity -o StrictHostKeyChecking=no" \
                        server:/master/repository /local/destination

copy:
        /master/repository      dest=/local/destination
               mode=0444
               secure=true
               recurse=inf
               server=$(cfd_master)
               ignore=CVS
               backup=false
               type=md5
               purge=false
```

Listing 1: Typical rsync invocation.

- **External rsync copy**: Externally called from cfengine rsync copy with SSH as the rsync transport.

The four different-sized transfers were as follows:

- Small: 128K, 22 files, 1 directory
- Medium: 2096K, 51 files, 2 directories
- Large: 209096K, 4726 files, 454 directories
- Larger: 258388K, 5646 files, 528 directories

The above configurations were run five times on each machine at each transfer size with debugging first turned on and then again with debugging turned off, to produce 800 data points (4 transfer setups * 4 different-sized transfers * 5 runs with debugging on * 5 runs with debugging off * 2 client machines). Between runs, the copied files were removed. The whole test cycle was then repeated to test file verification speeds, for an additional 800 data points [raw]. File verification consisted of the process of subsequent runs to check the files against the server for changes, either locally or on the server. Our tests for verification were performed with no changes to the files on either client or server.

Results

For the actual transfer of data, the various methods ordered themselves out fairly evenly. While rsync

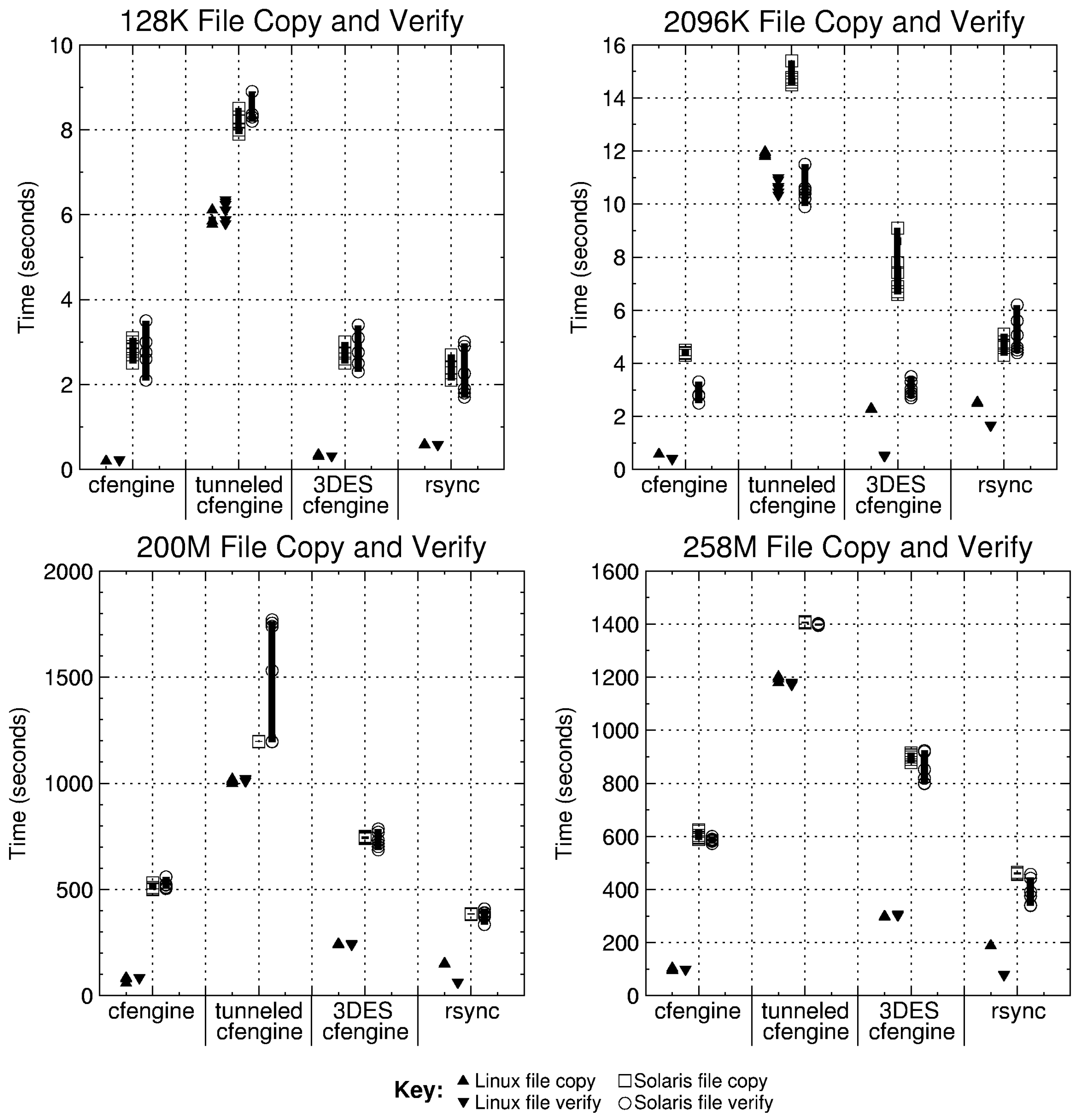

Figure 1: Comparison of file transfer performance.

ran as fast as, or faster than, cfengine on Solaris, under Linux rsync was slower than both cfengine methods until the large transfers, where rsync out-paced 3DES cfengine.

Cfengine tunneled over SSH performed poorly in all tests and showed itself to be a non-valid solution to the problem. This poor performance resulted because communications between cfengine and SSH was through TCP Socket connections. This created more work for the OS's TCP stack since two sockets were opened on each machine (client and server) per connection. In addition to observing a slow initialization and poor general performance, we also found in testing that setting up the tunnel was too brittle to be used in an actual production environment. This instability was attributed to the timings required from opening the tunnel and then initiating the cfengine communication through that tunnel. This timing issue was most problematic when cfengine needed to close and then open a new connect to a cfd server. If the SSH tunnel had not closed before the new cfengine connection was attempted, then a port conflict would arise and the new connection would fail.

What was more interesting was the comparison between rsync and 3DES cfengine, since both used 3DES, from the same OpenSSL [SSL] library, for their cipher. Rsync did suffer in smaller transfers, mostly due to the initialization time required to handle the RSA authentication between hosts. 3DES Cfengine was slower for the large file transfers, mostly due to the extra checks on the client side. The process cfengine used in a transfer was to write the incoming file to a temporary location and then verify that the file was successfully saved before copying it to its final destination on the local disk. It was only a slight increase in I/O utilization in comparison to rsync, but it did slow down the overall transfer process. Those extra checks ensured that the files arrived uncorrupted, which was especially important for distributing critical binaries and data files.

For file verification, the RSA authentication used for rsync created an initial performance hit in opening the connection, which dramatically affected results for smaller file sets. Rsync's checksum algorithm [tridgell] allowed it to best cfengine in the larger file set verifications, even when cfengine was not encrypted. Part of this was because much of the I/O and computational actions are off-loaded by rsync onto the server. With the server being unloaded and having considerably more power than the client, this was a definite plus in our test environment. But this would create scaling problems as demand on the server increased. What constitutes a large demand, though, was not within the scope of our testing. For its file verification, cfengine sent the server the MD5 checksum of its local file. The server then did an MD5 check of its file (which is cached) and responded with the results of the check. While this was a more simplified check than rsync's rolling CRC's, it was just as effective for file verification. The requirement of the client to handle much of the verification load was consistent with the general cfengine design philosophy of "smart client/dumb server" and allowed the server to scale in order to handle more client requests. This did put a slightly greater load on each individual client, although, nothing so significant as to generally hinder the client from performing its primary function.

Conclusions

The results of our tests are preliminary and come from only a single set of parameters. As such, it is difficult to make any definitive conclusions, but we can see trends developing. It can be seen that rsync has advantages over cfengine for large file transport and verification. This is particularly the case where there are a large number of files which have frequent small changes that need to be kept synchronized. On the other hand, if you are interested in synchronizing a few megabytes' worth of less frequently changing data, cfengine is a better choice. This is especially true if you are going to use cfengine's other more interesting features to actually maintain the client hosts. The one advantage that rsync does have over cfengine is the ability to use RSA authentication.

As cfengine evolves, the advantages of rsync for transport and verification of file sets may diminish. The cfengine protocol in the 2.0.x versions is still in the development stages, but already gets rid of the 4096 byte block sizing issue and other transport efficiency issues. Additionally, RSA-style authentication is in the process of being implemented.

Author Information

Andrew Mayhew has a BS-CS from the University of Central Florida in 1997. He stayed in Orlando to work at various ISP's until moving to California to work for Netscape Communication. He left Netscape after the AOL purchase to work as a Senior Internet Infrastructure Engineer for Logictier, Inc. until their closing this September. He continues to live in Sunnyvale, CA and can be reached electronically at amayhew@icewire.com.

References and Further Reading

[cfengine] "Cfengine," http://www.cfengine.org/.

[SSH] "OpenSSH," http://www.openssh.org/.

[SSL] "OpenSSL," http://www.openssl.org/.

[patch] Mayhew, Andrew, "A cfengine Patch Used to Test External Copy Methods," http://icewire.com/cfengine/patches/.

[raw] Mayhew, Andrew, Test scripts, configurations, and results used for this paper, http://icewire.com/cfengine/testing/.

[rsync] "Rsync," http://rsync.samba.org/.

[smith] Smith, Gregory P., "The Perl cfd Replacement Daemon," http://perl-cfd.sourceforge.net/README.perl-cfd.html.

[tridgell] Tridgell, Andrew, and Paul Mackerras, “The rsync algorithm,” http://rsync.samba.org/rsync/tech_report/ .