National Software Reference Library Distributed Hashing With Sexeger Improvements

Douglas White
Software Diagnostics and Conformance Testing Division
National Institute of Standards and Technology
Email: nsrl@nist.gov

Abstract

The National Software Reference Library (NSRL) is designed to collect software from various sources and incorporate file profiles computed from this software into a Reference Data Set (RDS) of information. The RDS can be used by law enforcement, government, and industry organizations to review files on a computer by matching file profiles in the RDS. This will help alleviate much of the effort involved in determining which files are important as evidence on computers or file systems that have been seized as part of criminal investigations [1]. The harvesting of file profiles from software is accomplished by multiple computers in a loose parallel computing environment. The implementation of the environment is discussed, with attention to performance improvements using reverse regular expressions known as "sexeger".

Introduction

When software destined for the National Software Reference Library arrives in our library room, it is placed on a "new arrival" shelf. The software is taken from the new arrival shelf and the information such as application name, version, manufacturer, etc. is entered into our database, and a unique identifier is allocated to the software package, as well as identifiers for each piece of media in the package. A label is placed on the box or case, and media are marked with indelible ink. Once labeled, the software may then be placed on a shelf in the actual library, or while it is presently off the shelf having been labeled, batching can occur.

The batching process takes files from the original media and places them on our file server. Once batched, file profiles are constructed for every file on the media. These profiles are comprised of information that allow unique identification of every file. Part of the profile information is a set of hash values [2] [3] that are computed condensed representations of a file [4] [5]. The hashes are the most popularly used features of the NSRL RDS and this paper focusses on improvements in the hashing process verification time by using reversed regular expressions.

Environment

We made the NSRL hashing process modular, and we use a parallel method of hash calculation. After software media have been batched an array of computers perform the hash calculations. There are six 166 MHz Windows 98 computers that perform the distributed hashing work. We call these six machines our hashing "constellation."

This term was chosen because the term "cluster" is widely used to designate a parallel configuration of computers much more integrated than is found in our project.

Distribution of Hashing Work

We have a linux computer on our network that is the master of the distributed hashing process. It uses cron to run a Perl script every hour [6]. The master Perl script connects to a database server and runs a query to find out if a piece of media has been batched and requires hashing. If so, the master script starts a hashing process (also a Perl script) on an available constellation computer.

The heart of the NSRL hashing process is an executable called WALKER.EXE. This is a compiled C implementation of the CRC32, MD4, MD5 and SHA-1 algorithms. The hashing process applies WALKER.EXE to a list of files to generate hash codes for each file and merges the WALKER.EXE output with the list of files, checking for anomalies. This merge allows a precise match of hashes to the location of the original file. Any error messages, format inconsistencies, etc. are identified, logged and reconciled if possible. Valid data will continue through the hashing process steps, invalid data will result in messages to administrators notifying an incomplete data set from this media. We want the process to continue with as much valid data as possible and we will investigate invalid data later.

Many of the consistency checks use reversed regular expressions known as sexeger, popularized by Jeff Pinyan (japhy) [7] [8]. When we implemented sexeger, we noted an average 55% reduction in time used to perform consistency checks, and a 32% reduction in time overall in the environment when we implemented sexeger. This choice was probably the best improvement we made in our coding of the environment.

The bulk of the sexeger code is used in finding and comparing filename extension strings (.tar, .zip, .jpeg) at the end of long strings and in traversing absolute directory path strings in reverse to identify parent-child relationships in media structures.

Performance Metrics

We retained the timing data from the automated hashes of 92 pieces of media using a previous consistency check algorithm, and we rehashed those 92 pieces of media using a new algorithm that included reversed regular expressions, or sexeger. During early implementation, timing was taken in increments of seconds; for consistency this was repeated on later runs. Here are the timing results for the old and new algorithms:

<u>Old Consistency Checks:</u>

Hashes performed: 812049
Average number of hashes per media: 8826
Effective hashes/s: 74

Code section	Seconds	% of run time
initialization:	18	0%
walker hash:	11010	7%
consistency:	80044	56%
find/extract:	49124	34%
total time:	140984 (39h)	100%

The times above clearly identify the consistency checks as the initial location for attempting performance tuning.

<u>New Consistency Checks (with sexeger):</u>

Hashes performed: 812049
Average number of hashes per media: 8826
Effective hashes/s: 75

Code section	Seconds	% of run time
initialization:	18	0%
walker hash:	10795	11%
consistency:	35830	37%
find/extract:	50127	51%
total time:	96770 (27h)	100%

After implementing sexeger, a clear 55% reduction in consistency check time and a 32% overall execution time reduction was observed.

During the months of April and May 2002, we rehashed the entire NSRL software collection. The new hashes calculated were verified against the previous RDS to ensure that the new algorithms were not perturbing our results. The rehash allowed us to view the process end-to-end under reasonable stress. In six weeks, we batched and hashed 1583 pieces of media. This period included days where 70 CDs were batched in a 9 hour period, and it included days when all staff were on travel. On average, slightly over 1,000,000 files were hashed each week. Here are the timing results over the entire collection:

Total Collection:

Hashes peformed: 6423307
Average number of hashes per media:4057
Effective hashes/s: 34

Code section	Seconds	% of run time
initialization:	306	0%
walker hash:	184724	11%
consistency:	620288	37%
find/extract:	872811	52%
total time:	1656605 (460h)	100%

Conclusions

Measurement of time spent in major sections of the hashing execution allowed us to identify the worst performance. Implementation of counter-intuitive algorithms brought considerable time savings to the process. We found that the modularity resulted in an eventual ten-fold gain in productivity. During the months of April and May 2002, the environment showed the ability to hash and verify 1,000,000 files each week. In the current configuration, when five staff batched during the workday, the six 166 MHz constellation PCs hashed slightly faster than the work accumulated.

There are several improvements we can make to increase productivity. An increase is needed because the process is very close to saturation, and we have seen that software released on DVDs (more popular now) will extend the average time a constellation CPU is performing one task. A finer unit of timing is needed in benchmarking in the future, probably using the Benchmark and Time::HiRes modules. We will be seeking improvements in the archive extraction code and in the database manipulation code. We will be evaluating the File::MMagic module for internal verification of file types.

Bibliography

[1] Fisher, G. (2001) NIST ITL Bulletin, "Computer Forensics Guidance".
https://www.nsrl.nist.gov/itlbulletin.html

[2] Rivest, R.L. (1992) IETF RFC1321, "The MD5 Message Digest Algorithm".
http://www.ietf.org/rfc/rfc1321.txt

[3] U.S. Department of Commerce. (1995) NIST FIPS 180-1, "Secure Hash Standard".
https://www.itl.nist.gov/fipspubs/fip180-1.htm

[4] Boland, T. and Fisher, G. (2000) "Selection Of Hashing Algorithms".
https://www.nsrl.nist.gov/documents/hash-selection.doc

[5] Pieprzyk J. and Sadeghiyan, B. (1993) "Design of Hashing Algorithms".
ISBN 0-387-57500-6

[6] Vixie, P. (1993) Cron man page.
http://unixhelp.ed.ac.uk/CGI/man-cgi?file

[7] Pinyan, J.(2000), "sexeger".
http://www.perlmonks.org/index.pl?node=sexeger

[8] Sergeant, P. (2001) "Reversing Regular Expressions".
http://www.perl.com/pub/a/2001/05/01/expressions.html

Biography

Douglas White has worked at the National Institute of Standards and Technology since 1987. His experience has covered distributed systems, distributed databases and telecommunication protocols. He has written programs in many areas, including real time biomonitoring, real time video processing, web site/database integration, system administration scripts and network monitoring scripts. He holds both a B.A and M.S. in computer science from Hood College.

Slides as presented at YAPC::Europe::2002 .