Reducing Storage Requirements of Snapshot Backups based on *rsync* utility

Syed Zahed K.*, P. Sabitha Rani*, U.Vijaya Saradhi† and Anupama Potluri*
*Department of Computer and Information Sciences, University of Hyderabad
sdzahed@gmail.com, sabitha.reddi@gmail.com, apcs@uohyd.ernet.in
†Scaleable Storage, saradhi@scaleablestorage.com

***Abstract*—Backing up data is a necessary requirement typically enforced by government too for its redundancy and availability. Disks have already emerged as one of the front runners for fast and efficient backup, given the way storage cost has reduced with the advent of high capacity SATA drives. *rsync* has established itself as a reliable backup tool in small and medium scale business units. This paper discusses techniques on top of the *rsync* algorithm which can increase the storage capacity significantly by eliminating duplicate data in a file and avoiding duplication of information in storing file system backed up data. The technique uses the signatures generated by *rsync* to eliminate duplicate data in the file across backups thus helping in lesser storage costs. The technique has the potential to be extended across the entire data blocks of the file system for better efficiencies and can be made as the cheapest de-dup available in the field. The storage savings are very high for loads which do less percentage of overwrites and higher percentage of append writes.**

I. Introduction

Modern organizational applications generate large amounts of electronic data, and the trend is towards a continued increase in data storage requirements. Thus, there is an ever-increasing need for tools that can reduce storage requirements. *rsync* [3] has been extended to create automated snapshot backups on Linux [1]. The snapshot backup works as follows: *rsync* has three processes that work together - the sender, generator and receiver. The generator, takes the basis file, i.e., the previous version, divides it into blocks and sends the signatures of the blocks to the sender. The sender returns only tokens for the matching blocks and literal data for non-matching parts. The receiver uses the tokens to copy blocks from the basis file into the right position and copies the literal data sent to construct the new snapshot backup version. In this paper, we propose an enhancement to *rsync* that leverages the existing algorithm to maintain pointers to blocks of identical data present in previous versions of the backup, rather than locally copy them as traditional *rsync* does. We chose *rsync* for the basic algorithm as it is quite amenable to modifications for de-duplication and is open-source allowing us to come up with a tool that is cheap.

II. Proposed Modifications to *rsync* Algorithms

In order to be more space efficient when doing snapshot style periodic backups, *rsync* should be enhanced to maintain pointers or tokens to the identical blocks found in previous versions of the snapshot backups. To achieve this, we need to modify the processes involved in the generation of checksums (the *generator*) and reconstruction of files (the *receiver*). The sender process needs no modifications whatsoever.

Pointers or Tokens are logical references stored in the extended attributes of a file which point to blocks of identical data in a previous backup version of the same file. A file in such a system consists of literal new blocks of data and tokens to blocks of data found in previous versions of the file. Each pointer must contain the following information : position in the new file of this pointer, position of the matching block in the old file and its fast and strong signatures. We store this information in the extended attributes of the file such that they are easily accessed. Every backup has pointers to only its immediate predecessor in the hierarchy. However, the previous backup may point to blocks of data found in its predecessor and so on. Thus, if a block of data was never modified in the file, the pointers need to be followed all the way to the first version of the backup. This method has the advantage of limiting the direct dependency between the backups.

The *restore* operation to recover a particular version of the backup is a recursive process that follows the pointers backward until the literal data of the block is found and copied into the file being restored. The *delete* operation requires that the later version of the backup is restored by following pointers from the version being deleted. Once the later version is successfully restored, then, the actual deletion is done. If the file being deleted is the latest version, then, it can be directly deleted with no additional processing needed.

III. Analysis

The new tool built by modifying *rsync* was evaluated under two different simulated environments based on the most popular real-life environments. One of the scenarios mimics the typical development environment in real life, where multiple developers work on the same set of files but modify different files and add/delete a few of them. Another test was conducted with a large repository of data files with a random number of create, delete and append operations to these directories and files. Even though these tests cannot be considered to be complete in the coverage of all possible scenarios, they do establish the effectiveness of the new tool. We had to develop a new tool *dirsize* since *du* calculates the blocks allocated rather than those used as well as does not take into account the size of the extended attributes. The comparison of space used, in the tables, is based on the calculation by *dirsize*.

TABLE I
RESULTS OF TESTS IN ENVIRONMENT-1

$Backup$	$Sender$		$Receiver$		$Savings$
	$TotalSize$	$XattrSize$	$TotalSize$	$XattrSize$	
$b1$	76627418	0	76663870	36452	None
$b2$	77177838	0	9371035	3089573	87.86%
$b3$	78011995	0	8783518	3186027	88.74%
$b4$	78662295	0	8418222	3237823	89.30%
$b5$	79000342	0	7727289	3289571	90.22%
$b6$	81718027	0	12638895	3224472	84.53%
$b7$	82524971	0	10651515	3372532	87.09%
$Total$	553722886	0	134254344	19436450	75.75%

TABLE II
RESULTS OF TESTS IN ENVIRONMENT-2

$Operation$	$Sender$		$Receiver$		$Savings$
	$TotalSize$	$XattrSize$	$TotalSize$	$XattrSize$	
Initial run with 500000 files	1000.26 MB	14 B	1006.46 MB	6.2 MB	None
Creates = 1% Deletes = 1% Appends=69%	1077.25 MB	9 B	546.75 MB	126.94 MB	49.25%
Creates = 3% Deletes = 2% Appends=47%	1137.27 MB	9 B	401.70 MB	136.32 MB	64.68%
$TotalBackup$	$3214.78MB$	$32B$	$1954.91MB$	$269.46MB$	39.19%

The first test environment consisted of ten different source directories of *rsync* code, representing a development team of ten developers. The code in every one of the source directories was modified in small proportions. These modifications included addition of dummy functions, modifications at random locations in the code, adding new source files etc. After these modifications were made, all the directories were compiled to generate newer set of object files. Once this was done, a backup was taken before further modifications. The results shown in Table I indicate a total saving of 75.75% for a total of one week's backup with few modifications between the backups from day-to-day.

A second set of tests were carried out to determine the storage savings with random amount of create, delete and append operations. In order to simulate such an environment, we have created a directory structure with half a million files of size $2KB$ each. Three different backups were taken by modifying the amounts of creates, deletes and appends in the directory structure. At the end of each backup the space occupied by the backed up data was compared with the original size of the data. The results of these tests are shown in Table II. The average savings in this environment were a significant 39.19%.

IV. CONCLUSIONS

In the current world scenario of high storage requirements, any tool that reduces the cost of storage media required is welcome. Efficient utilization of the existing storage space is extremely important, especially for small enterprises and universities. This paper is an attempt to develop a free, open-source snapshot backup tool, based on *rsync*, to reduce storage costs. We enhance the *rsync* algorithm to store pointers to identical blocks of data of previous backup versions as extended attributes rather than store the data itself. Simulation experiments of real-life scenarios such as a developer environment and random number of create, delete and append operations on large directory structures show a minimum of 39% savings in storage space required.

REFERENCES

[1] Mike Rubel, *Easy Automated Snapshot-Style Backups with Linux and Rsync*, http://www.mikerubel.org/computers/rsync_snapshots/
[2] *The article describing how* rsync *works and how the algorithm is implemented*, http://www.samba.org/rsync/how-rsync-works.html
[3] Dr. Andrew Tridgell, *Ph.D thesis*, http://samba.org/tridge/phd_thesis.pdf.
[4] *The detailed* rsync *technical report*, http://rsync.samba.org/tech_report/
[5] Manual Pages of *rsync* and *rsyncd.conf*.
[6] *XFS overview and internals*, http://oss.sgi.com/projects/xfs/training/