ZFS replication vs rsync

First, a disclaimer. While I have decades of experience with rsync, I'm fairly new with zfs, so do take the following with that caveat.

There are two main ways I use to back up a ZFS file system; zfs send/recv (aka 'replication') and rsync, with zfs replication being hands down the best way to do it if backing up a zfs file system to another one. NOTE: send/recv is only available if both systems use ZFS.

That is not to say that ZFS replication is the best solution for all cases, but I'd say it should be the default when you are planning your backup strategy.

rsync

rsync is very old and stable technology, developed by Andrew Triggell (of Samba fame) in 1996, and it is under continuous development by a team of developers with the most recent (Preview) release, as of this writing, in 2023. It is available on most operating systems (Linux, *BSD, macOS and even Microsoft Windows).

rsync is file based, meaning it looks for files which are different in source and target, updating, adding, and optionally deleting files on the target to keep the two systems synchronized. rsync is optimized for slow systems, allowing transfer over multiple protocols.

The rsync executable must exist on both source and target systems.

Operation

rsync first collects a list of source files and requests a list of files on the target, along with metadata about the files. It then compares the metadata to determine if a file has changed (generally the timestamp it was last updated). It also has the ability to compare checksums between the target and destination to determine if a file has been updated. Generating and comparing this metadata can require a large amount of memory and cpu on larger file systems.

Once rsync has determined a file needs to be transferred, it will send the file to the target. The target will write the data to a temporary file until the file has completely transferred and compared (via checksum?) to verify it is valid. It will then unlink (delete) the original file and rename the temp file to the original name. NOTE: this is very important if you have hard links as it will only modify the target file; it will not modify the other hard links (but see the hard link parameter).

Details

- rsync is file based, meaning if metadata about a file is changed (such as directory location, timestamp), it will be treated as a new or modified file even if the contents stay the same. In some worst case scenarios, if a user renames a directory, rsync will decide that the original directory has been deleted, and the new directory needs to be copied in full.

- rsync is efficient. scp, cp and rcp will be much slower.
- rsync can be bidirectional. It can keep the source and target synchronized. NOTE: it does not handle file deletions very well, so you might want to look at [https://www.cis.upenn.edu/~bcpierce/unison/index.html|Unison] if you need bidirectional functionality.
- rsync is resilient. If an rsync job is interrupted for some reason, simply running the same command will pick up where it was when it stopped, with very minimal overhead. Adding the –partial flag means even if a large file transfer was interrupted, it will pick up where it left off, only copying the remainder of the file.
- With the proper flags, modifying a large file in some limited cases can result in only the changes to the file being copied. The source and target files can be broken into blocks and, only the modified blocks are sent. Note that this requires additional memory and processor, and doing something like inserting a single character in a file (so all blocks are shifted by one character) will result in the entire file being recopied.
- rsync can give a progress report while copying (–progress or -v or –verbose), and a detailed status report when complete (–stats).
- rsync has a “dry run” flag (–dry-run or -n) which will give details of what would be transferred with a given set of flags
- On file systems with hard links, rsync can be combined with a strategy where multiple snapshots of the same file system are stored in an efficient manner.

Example

```
# example of bash code to make daily backups, keeping a copy of old ones
# in versions/date.
rsync -av --delete /my/important/files backup-server:/my/backup/recent/
ssh backup-server 'mkdir /my/versions/`date +%Y-%m-%d` ; cp -alv
/my/backup/recent/* /my/versions/`date +%Y-%m-%d`'
```

ZFS Replication

ZFS replication is dependent on zfs snapshots. zfs snapshots store only the blocks which have changed since they were created, thus zfs send/recv will only copy the blocks (not files) which have changed on a system.

Because both systems are using zfs, and we are using snapshots, there is very little overhead in the process. After the first run (which copies everything), the changes from the previous run are immediately sent. This is much, much faster than rsync.

Operation

zfs replication sends a zfs snapshot to the target machine, so the first step is to create a zfs snapshot. Then, execute the 'zfs send' command on the source machine and the zfs recv command on the target machine. This is generally done in one statement, with the zfs recv executed over ssh

Details

- Initial copy can be performed over 'sneakernet', but must be done using zfs send/recv to the drive to be copied. Thus, the drive must be large enough to hold the entire file system to be replicated.
- Both source and target must be zfs file systems.
- Since only the modified file system blocks are sent, metadata changes do not affect speed. Renaming or moving files only requires the metadata to be copied.
- Snapshots, by definition, keep a history of your filesystem. Since the snapshots are synchronized on the source and target, both will have a history of the file system.
- It is resilient so long as the snapshots are maintained. If a send/recv fails, the command can be executed again to synchronize the snapshots. I am not sure if it requires a full copy or not.
- If snapshots are corrupted, or deleted in the wrong order on either the source or the target, it can sometimes require a fresh full system replication, so deletion/corruption of a snapshot on either system can cause replication to fail disastrously
- zfs send/recv will handle zfs volumes (block devices) in addition to datasets (file systems) in the same, efficient manner.
- zfs send/recv does not report progress, but it can use filters (my favorite is pv under Unix) (NOTE: there is a way to get a report, but it is not as robust as rsync's)

Example

```
# create a snapshot (named snap1) of zpool tank, dataset root_filesystem, and all datasets under it
zfs snapshot -r tank/root_filesystem@snap1
# do a dry run to estimate how large the replication will be
zfs send -rnv tank/root_filesystem@snap1
# send snapshot of tank/root_filesystem@snap1 to backupserver (under backup/otherroot), with pv showing progress
zfs send tank/root_filesystem@snap1 | pv | ssh backupserver 'zfs recv backup/otherroot'
```

NOTE

Since zfs replication requires adding new snapshots and removing old ones, several scripts have been developed to assist in managing this task.

* zrepl - https://zrepl.github.io/ * zrep - http://www.bolthole.com/solaris/zrep/