Parallel Rsync (how I believe it's done)

catuion this is a work in progress.. I am writing down my notes as I go!

caution please be careful with the instructions below and think it through yourself. I will take no responsibility for any data loss as a result of this article.

rsync is sooo cool, chances are, if you need to copy some files for whatever reason from one linux machine to another or even from one directory to another, rsync has everything you need. one thing though is terribly missing: parallelism

here is, how i did it when i needed to copy 40 TB of data from one raidset to another while the server was still online serving files to everybody in the company:

Step 1: creat an incremental file list

depending on your needs, there are different options how to do that.

Option 1: rsync --dry-run

one possibly slow option is to do a dry-run of rsync with all your options you want to use and then use the file-list created by the dry-run for your rsync job.

first do the dry run:

```
rsync -aHvx --dry-run --out-format="%n" /source/ /target/ | tee
/tmp/rawfilelist
```

use rsync options like you would for a simple rsync run to copy all your files, but add the –dry-run –out-format=“%n” options. the out-format option is to make sure you get a simple list of files without the added information about symlinks and hardlinks, that you would get when this option was omitted.

now clean up the resulting file: the problem with the dry-run output is, that you also get directory names before you get the list of the contets of each directory. that's useless if we want to continue later on and run an rsync for each file. so we need to get rid of these directory paths. this will obviously lead to empty directories not being copied, we can fix that later on by running a simple single thread rsync at the end to fix things like exactly that :) so here we go.. let's clean up the filelist (you can do this inplace, but you might just want to use this line and pipe it directly into parallel further down the road)

```
cat /tmp/rawfilelist | sed -e 's/.*\/$//' | sed -e 's/sent .* bytes\/sec$//'
| sed -e 's/^total .* (DRY RUN)$//' | sed -e 's/sending incremental file
list//' | sed -e '/^$/d' > /tmp/filelist
```

Option 2: using find

after waiting too long for Option 1 to finish on a system that carried tons of backups of other systems, i tried this option:
if you have tons of files and want to skip the lengthy process of producing a file list via rsync, you can create a list of directories using find and then simply run an rsync per directory. this will give you the full parallelism at the begining but might end with a few ever lasting rsyncs if you don't dig deep enough when doing your initial directory list. still, this might save alot of time.

```
find /source/./ -maxdepth 5 -type d | perl -pe 's|^.*?/\./|\1|' >
/tmp/filelist
```

with the –maxdepth option you can set how deep you want to dive into your directory tree.. the goal is to get directories with a rather small number of files so you don't have to wait too long for the last couple of rsyncs to finish. also note the added `/./` at the end of the source path. that's important as we need this to define to which point rsync should be relative. also check out the man page of rsync, i stole the idea from there ;)

now it's time to clean up the list. we need to move all lines that contain less than the `-maxdepth` number of directories to a separate file list as these directories will need to be synced without recursion. i tried doing this with a loop that went through all lines trying to find the respective lines, but it took way too long for a rawfilelist with more than 300'000 entries, so i tried it with sed inplace and it was incredibly fast!

```
cp /tmp/rawfilelist /tmp/parentslist
cp /tmp/rawfilelist /tmp/filelist
sed -i '/^\(.*\/\)\{4\}.*$/d' /tmp/parentlist
sed -i '/^\(.*\/\)\{4\}.*$/!d' /tmp/filelist
```

make sure that the number in the sed regex is your –maxdepth number minus 1! now we need to sync the parents without recursion first before continuing to step 2

```
cat /tmp/parentlist | parallel -j 3 'shopt -s dotglob; rsync -aHvx --no-r --
relative /tmp/source/./{}/* /tmp/target/'
```

the trick here is to use the –no-r option to remove recursion of whatever rsync parameters you have specified before it. also check out the `shopt` command which results in a * matching also hidden files like `.htaccess` and so on.

Step 2: run Rsync with GNU prallel

now it's time to feed our filelist into rsync and run our parallel sync job. in order to parallelize rsync we use the GNU tool `parallel`. it will take a list of files and run a command in parallel with as many processes as are specified by the `-j` option. in the command string, it will replace `{}` with the contents of the respective line. pretty simple :)

```
cat /tmp/filelist | parallel -j 10 rsync -aHvx --relative /source/./{}
/target/
```

note how, like in the above mentioned Option 2, we use the '/./' separator in the source path to tell rsync where to start with the relative path that it transmits to the client. also make sure you actually use the `--relative` option, otherwise your targets file structure will be very flat :)
note that parallel is thoroug as far as escaping goes. there are no quotes needed even with funny directory names.

Step 3: make sure we didn't miss anything

probably the best feature about rsync is, that it resumes aborted previous jobs nicely and it can be run several times across the same source and target with no harm. so let's use this property to just fix everything we have missed or done wrong by simply running a single thread rsync in the end. now this can take some time, and I know no way around that.

```
rsync -aHvx /source/ /target/
```