

paexec – distributes tasks over network or CPUs

Aleksey Cheusov
vle@gmx.net

Minsk, Belarus, 2013

Problem

- ▶ Huge amount of data to process
- ▶ Typical desktop machines have more than one CPU
- ▶ Unlimited resources are available on the Internet
- ▶ Heterogeneous environment (*BSD, Linux, Windows...)

Solution

Usage

```
paexec [OPTIONS] \  
  -n 'machines or CPUs' \  
  -t 'transport program' \  
  -c 'calculator' < tasks
```

example

```
ls -l *.wav | \  
paexec -x -c 'flac -s' -n +4 > /dev/null
```

example

```
paexec \  
  -n 'host1 host2 host3' \  
  -t /usr/bin/ssh \  
  -c ~/bin/toupper < tasks
```

Example 1: toupper

Our goal is to convert strings to upper case in parallel

~/bin/toupper

```
#!/usr/bin/awk -f
{
    print " ", toupper($0)
    print ""    # empty line -- end-of-task marker!
    fflush()    # We must flush stdout!
}
```

~/tmp/tasks

```
apple
bananas
orange
```

Example 1: toupper

~/bin/toupper script will be run only once on remote servers “server1” and “server2”. It takes tasks from stdin (one task per line). Transport program is ssh(1).

paexec invocation

```
$ paexec -t ssh -c ~/bin/toupper \  
-n 'server1 server2' < tasks > results  
$ cat results  
BANANAS  
ORANGE  
APPLE  
$
```

Example 1: toupper

Options **-l** and **-r** add the task number and server where this task is processed to stdout.

```
paexec -lr invocation  
$ paexec -lr -t ssh -c ~/bin/toupper \  
  -n 'server1 server2' < tasks > results  
$ cat results  
server2 2  BANANAS  
server2 3  ORANGE  
server1 1  APPLE  
$
```

Example 1: toupper

The same as above but four instances of `~/bin/toupper` are ran locally.

paexec invocation

```
$ paexec -n +4 -c ~/bin/toupper
```

```
< tasks > results
```

```
$ cat results
```

```
BANANAS
```

```
ORANGE
```

```
APPLE
```

```
$
```

Example 1: toupper

The same as above but without ~/bin/toupper. In this example we run AWK program for each individual task. At the same time we still make only one ssh connection to each server regardless of a number of tasks given on input.

paexec invocation

```
$ paexec -x -t ssh -n 'server1 server2' \  
-c "awk 'BEGIN {print toupper(ARGV[1])}'" " \  
    < tasks > results  
$ cat results  
ORANGE  
BANANAS  
APPLE  
$
```

Example 1: toupper

If we want to "shquote" less one can use option **-C** instead of **-c** and specify command after options.

paexec invocation

```
$ paexec -x -C -t ssh -n 'server1 server2' \  
awk 'BEGIN {print toupper(ARGV[1])}' \  
    < tasks > results  
$ cat results  
ORANGE  
BANANAS  
APPLE  
$
```

Example 1: toupper

With options **-z** or **-Z** we can easily run our tasks on unreliable hosts. If we cannot connect or lost connection to the server, paexec will redistribute failed tasks to another server.

paexec invocation

```
$ paexec -Z240 -x -t ssh \  
-n 'server1 badhostname server2' \  
-c "awk 'BEGIN {print toupper(ARGV[1])}' " \  
    < tasks > results
```

```
ssh: Could not resolve hostname badhostname:  
No address associated with hostname  
badhostname 1 fatal
```

```
$ cat results  
ORANGE  
BANANAS  
APPLE  
$
```

Example 2: parallel banner(1)

what is banner(1)?

```
$ banner -f @ NetBSD
```

```
@           @
@@          @  @@@@@@@@  @@@@@@  @@@@@@@@  @@@@@@@@
@ @         @  @          @          @          @
@  @        @  @          @          @          @
@   @       @  @@@@@@    @          @          @
@    @      @  @         @          @          @
@     @     @  @         @          @          @
@      @@    @  @@@@@@@@  @          @          @
@       @    @  @@@@@@@@  @@@@@@    @@@@@@@@
```

```
$
```

Example 2: parallel banner(1)

`~/bin/pbanner` is wrapper for `banner(1)` for reading tasks line by line. Magic line is used instead empty line as an end-of-task marker.

`~/bin/pbanner`

```
#!/usr/bin/env sh

while read task; do
    banner -f M "$task" |
    echo "$PAEXEC_EOT" # end-of-task marker
done
```

`tasks`

```
pae
xec
```

`paexec invocation`

```
$ paexec -l -mt='SE@X-LOS0!&' -c ~/bin/pbanner \
    -n +2 < tasks > result
$
```

Example 2: parallel banner(1)

paexec(1) reads calculator's output asynchronously. So, its output is sliced.

```

$ cat result
2
2
2  M      M  MMMMMM  MMMM
2  M      M      M      M
1
1
1  MMMMM      MM      MMMMMM
1  M      M      M      M
1  M      M      M      M
1  MMMMM      MMMMMM  M
2  MM      MMMMM      M
2  MM      M      M
2  M      M      M      M
2  M      M      MMMMMM  MMMM
2
1  M      M      M      M
1  M      M      M      M
1
$
```

Sliced result

Example 2: parallel banner(1)

`paexec_reorder(1)` normalizes `paexec(1)`'s output.

```
Ordered result
$ paexec_reorder -mt='SE@X-L0S0!&' results
```

```
MMMMM      MM      MMMMMM
M      M      M      M      M
M      M      M      M      M
MMMMM      MMMMMM      M
M      M      M      M      M
M      M      M      M      M
```

```
M      M      MMMMMM      MMMM
M      M      M      M      M
      MM      MMMMMM      M
      MM      M      M      M
M      M      M      M      M
M      M      MMMMMM      MMMM
```

```
$
```

Example 2: parallel banner(1)

The same as above but using magic end-of-task marker provided by `paexec(1)`.

Ordered result

```
$ paexec -y -lc ~/bin/pbanner -n+2 < tasks | paexec_reorder -y
```

```
MMMMM      MM      MMMMMM  
M      M      M      M      M  
M      M      M      M      M  
MMMMM      MMMMMM      M  
M      M      M      M      M  
M      M      M      M      M
```

```
M      M      MMMMMM      MMMM  
M      M      M      M      M      M  
      MM      MMMMMM      M  
      MM      M      M      M  
M      M      M      M      M      M  
M      M      MMMMMM      MMMM
```

```
$
```

Example 2: parallel banner(1)

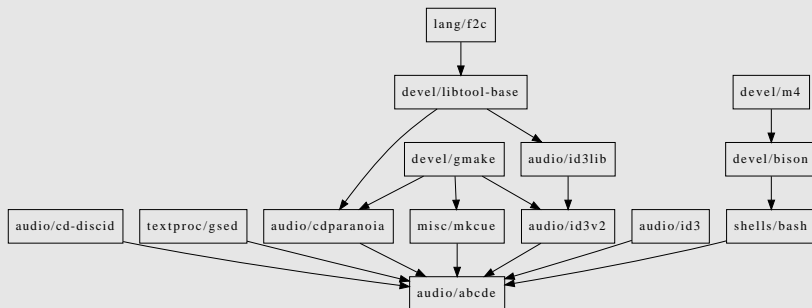
For this trivial task wrapper like `~/bin/pbanner` is not needed.
We can easily run `banner(1)` directly.

```

S Sliced result
$ paexec -l -x -c banner -n+2 < tasks
2
2
2  M      M  MMMMMM  MMMM
2  M  M  M  M      M      M
2    MM  MMMMM  M
2    MM  M      M
2  M  M  M      M      M
1
1
1  MMMMM      MM  MMMMMM
1  M      M  M  M  M
1  M      M  M      M  MMMMM
1  MMMMM  MMMMMM  M
2  M      M  MMMMMM  MMMM
2
1  M      M      M  M
1  M      M      M  MMMMMM
1
$
```

Example 3: dependency graph of tasks

`paexec(1)` is able to build tasks taking into account their “dependencies”. Here “`devel/gmake`” and others are `pkgsrc` packages. Our goal in this example is to build `pkgsrc` package `audio/abcde` and all its build-time and compile-time dependencies.



Example 3: dependency graph of tasks

‘paexec -g’ takes a dependency graph on input (in tsort(1) format). Tasks are separated by space (paexec -md=).

~/tmp/packages.to_build

```
audio/cd-discid audio/abcde
textproc/gsed audio/abcde
audio/cdparanoia audio/abcde
audio/id3v2 audio/abcde
audio/id3 audio/abcde
misc/mkcue audio/abcde
shells/bash audio/abcde
devel/libtool-base audio/cdparanoia
devel/gmake audio/cdparanoia
devel/libtool-base audio/id3lib
devel/gmake audio/id3v2
audio/id3lib audio/id3v2
devel/m4 devel/bison
lang/f2c devel/libtool-base
devel/gmake misc/mkcue
devel/bison shells/bash
```

Example 3: dependency graph of tasks

If option `-g` is applied, every task may succeed or fail. In case of failure all dependants fail recursively. For this to work we have to slightly adapt “calculator”.

```
~/bin/pkg_builder  
#!/usr/bin/awk -f  
  
{  
    print "build " $0  
    print "success" # build succeeded! (paexec -ms=)  
    print ""       # end-of-task marker  
    fflush()       # we must flush stdout  
}
```

Example 3: dependency graph of tasks

paexec -g invocation (no failures)

```
$ paexec -g -l -c ~/bin/pkg_builder -n 'server2 server1' \  
  -t ssh < ~/tmp/packages_to_build | paexec_reorder > result  
$ cat result  
build textproc/gsed  
success  
build devel/gmake  
success  
build misc/mkcue  
success  
build devel/m4  
success  
build devel/bison  
success  
...  
build audio/id3v2  
success  
build audio/abcde  
success  
$
```

Example 3: dependency graph of tasks

Let's suppose that “devel/gmake” fails to build.

```
~/bin/pkg_builder  
#!/usr/bin/awk -f  
  
{  
    print "build " $0  
    if ($0 == "devel/gmake")  
        print "failure" # Oh no...  
    else  
        print "success" # build succeeded!  
  
    print ""          # end-of-task marker  
    fflush()         # we must flush stdout  
}
```

Example 3: dependency graph of tasks

Package “devel/gmake” and all dependants are marked as failed.
Even if failures happen, the build continues.

paexec -g invocation (with failures)

```
$ paexec -gl -c ~/bin/pkg_builder -n 'server2 server1' \  
-t ssh < ~/tmp/packages_to_build | paexec_reorder > result  
$ cat result  
build audio/cd-discid  
success  
build audio/id3  
success  
build devel/gmake  
failure  
devel/gmake audio/cdparanoia audio/abcde audio/id3v2 misc/mkcue  
build devel/m4  
success  
build textproc/gsed  
success  
...  
$
```

Example 3: dependency graph of tasks

paexec is resistant not only to network failures but also to **unexpected** calculator **exits or crashes**.

~/bin/pkg_builder

```
#!/usr/bin/awk -f

{
    "hostname -s" | getline hostname
    print "build " $0 " on " hostname

    if (hostname == "server1" && $0 == "textproc/gsed")
        exit 139
        # Damn it, I'm dying...
        # Take a note that exit status doesn't matter.
    else
        print "success" # Yes! :-)

    print ""          # end-of-task marker
    fflush()          # we must flush stdout
}
```

Example 3: dependency graph of tasks

“textproc/gsed” failed on “server1” but then succeeded on “server2”. Every 300 seconds we try to reconnect to “server1”. Keywords “success”, “failure” and “fatal” may be changed with a help of -ms=, -mf= and -mF= options respectively.

paexec -Z300 invocation (with failure)

```
$ paexec -gl -Z300 -t ssh -c ~/bin/pkg_builder \  
    -n 'server2 server1' < ~/tmp/packages_to_build \  
    | paexec_reorder > result  
$ cat result  
build audio/cd-discid on server2  
success  
build textproc/gsed on server1  
fatal  
build textproc/gsed on server2  
success  
build audio/id3 on server2  
success  
...  
$
```

Example 4: Converting .wav files to .flac or .ogg

In trivial cases we don't need relatively complex "calculator". Running a trivial command may be enough. Below we run three .wav to .flac/.ogg convertors in parallel. If **-x** is applied, task is passed to calculator as an argument.

paexec -x invocation

```
$ ls -1 *.wav | paexec -x -c 'flac -s' -n+3 >/dev/null
$
```

paexec -x invocation

```
$ ls -1 *.wav | paexec -ixC -n+3 oggenc -Q | grep .
01-Crying_Wolf.wav
02-Autumn.wav
03-Time_Heals.wav
04-Alice_(Letting_Go).wav
05-This_Side_Of_The_Looking_Glass.wav
...
$
```

Example 5: paexec -W

If different tasks take different amount of time to process, than it makes sense to process “heavier” ones earlier in order to minimize total calculation time. For this to work one can weigh each tasks. Note that this mode enables “graph” mode automatically.

~/bin/calc

```
#!/bin/sh
# $1 -- task given on input
if test $1 = huge; then
    sleep 6
else
    sleep 1
fi

echo "task $1 done"
```

Example 5: paexec -W

This is how we run unweighted tasks. The whole process takes 8 seconds.

paexec invocation

```
$ printf 'small1\nsmall2\nsmall3\nsmall4\nsmall5\nhuge\n' |  
time -p paexec -c ~/bin/calc -n +2 -xg | grep -v success  
task small2 done  
task small1 done  
task small3 done  
task small4 done  
task small5 done  
task huge done  
real          8.04  
user          0.03  
sys           0.03  
$
```

Example 5: paexec -W

If we say paexec that the task “huge” is performed 6 times longer than others, it starts “huge” first and then others. In total we spend 6 seconds for all tasks.

paexec -W1 invocation

```
$ printf 'small1\nsmall2\nsmall3\nsmall4\nweight: huge 6\n' |  
time -p paexec -c ~/bin/calc -n +2 -x -W1 | grep -v success  
task small1 done  
task small2 done  
task small3 done  
task small4 done  
task small5 done  
task huge done  
real          6.02  
user          0.03  
sys           0.02  
$
```

For details see the manual page.

The End