Parallel Python documentation

Â Module API
Â Quick start guide, SMP
Â Quick start guide, clusters
Â Quick start guide, clusters with auto-discovery
Â Advanced guide, clusters
Â Command line arguments, ppserver.py
Â Security and secret key
Â ppserver.py stats and PID file example
Â PP FAQ

Â Â pp 1.6.3 module API

Â
class Server Â Â Â ParallelÂ PythonÂ SMPÂ executionÂ serverÂ class
Â Â Methods defined here:
__init__(self, ncpus='autodetect', ppservers=(), secret=None, restart=False, proto=2,
socket_timeout=3600)CreatesÂ ServerÂ instance
Â Â Â ncpusÂ -Â theÂ numberÂ ofÂ workerÂ processesÂ toÂ startÂ onÂ theÂ localÂ
Â Â Â Â Â Â Â Â Â computer,Â ifÂ parameterÂ isÂ omittedÂ itÂ willÂ beÂ setÂ toÂ
Â Â Â Â Â Â Â Â Â theÂ numberÂ ofÂ processorsÂ inÂ theÂ system
ppserversÂ -Â listÂ ofÂ activeÂ parallelÂ pythonÂ executionÂ serversÂ
Â Â Â Â Â Â Â Â toÂ connectÂ with
secretÂ -Â passphraseÂ forÂ networkÂ connections,Â ifÂ omittedÂ aÂ default
Â Â Â Â Â Â Â Â Â passphraseÂ willÂ beÂ used.Â It'sÂ highlyÂ recommendedÂ toÂ useÂ aÂ
Â Â Â Â Â Â Â Â customÂ passphraseÂ forÂ allÂ networkÂ connections.
restartÂ - whether toÂ restartÂ workerÂ processÂ afterÂ eachÂ taskÂ completionÂ
protoÂ -Â protocolÂ numberÂ forÂ pickleÂ modulesocket_timeout - socket timeout in seconds which is also the maximum
Â Â Â Â Â Â Â time a remote job could be executed. Increase this value
Â Â Â Â Â Â Â if you have long running jobs or decrease if connectivity
Â Â Â Â Â Â Â to remote ppservers is often lost.
Â
WithÂ ncpusÂ =Â 1Â allÂ tasksÂ areÂ executedÂ consequently
ForÂ theÂ bestÃ performanceÃ eitherÂ useÂ theÂ defaultÂ "autodetect"Â value
orÂ setÂ ncpusÂ toÂ theÂ totalÂ numberÂ ofÂ processorsÂ inÂ theÂ systemdestroy(self)KillsÂ ppworkersÂ andÂ closesÂ open
get_active_nodes(self)ReturnsÂ activeÂ nodesÂ asÂ aÂ dictionaryÂ
[keysÂ -Â nodes,Â valuesÂ -Â ncpus] get_ncpus(self)ReturnsÂ theÂ numberÂ ofÂ localÂ workerÂ processesÂ (ppworkers)
get_stats(self)ReturnsÂ jobÂ executionÂ statisticsÂ asÂ aÂ dictionaryprint_stats(self)PrintsÂ jobÂ executionÂ statistics.Â UsefulÂ
nchmarkingÂ onÂ
clusters set_ncpus(self, ncpus='autodetect')SetsÂ theÂ numberÂ ofÂ localÂ workerÂ processesÂ (ppworkers)
Â
ncpusÂ -Â theÂ numberÂ ofÂ workerÂ processes,Â ifÂ parammeterÂ isÂ omitted
Â Â Â Â Â Â Â Â itÂ willÂ beÂ setÂ toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system submit(self, func, args=(), depfuncs=()
callback=None, callbackargs=(), group='default', globals=None)SubmitsÂ functionÂ toÃ theÂ executionÂ queue
Â
funcÂ -Â functionÂ toÂ beÂ executed
argsÂ -Â tupleÂ withÂ argumentsÂ ofÂ theÂ 'func'
depfuncsÂ -Â tupleÂ withÂ functionsÂ whichÂ mightÂ beÂ calledÂ fromÂ 'func'
modulesÂ -Â tupleÂ withÂ moduleÂ namesÂ toÂ import
callbackÂ -Â callbackÂ functionÂ whichÂ willÂ beÂ calledÂ withÂ argumentÂ
Â Â Â Â Â Â Â Â listÂ equalÂ toÂ callbackargs+(result,)Â
Â Â Â Â Â Â Â Â asÂ soonÂ asÂ calculationÂ isÂ done
callbackargsÂ -Â additionalÂ argumentsÂ forÂ callbackÂ function
groupÂ -Â jobÂ group,Â isÂ usedÂ whenÂ wait(group)Â isÂ calledÂ toÂ waitÂ for
jobsÂ inÂ aÂ givenÂ groupÂ toÂ finish
globalsÂ -Â dictionaryÂ fromÂ whichÂ allÂ modules,Â functionsÂ andÂ classes
willÂ beÂ imported,Â forÂ instance:Â globals=globals() wait(self, group=None)WaitsÂ forÂ allÂ jobsÂ inÂ aÂ givenÂ groupÂ toÂ
IfÂ groupÂ isÂ omittedÂ waitsÂ forÂ allÂ jobsÂ toÂ finish default_port = 60000 default_secret = 'epo20pdosl;dksldkmm'

Â
class Template Â Â Â TemplateÂ class
Â Â Methods defined here:

__init__(self, job_server, func, depfuncs=(), modules=(), callback=None, callbackargs=(), group='default', globals=None)CreatesÂ TemplateÂ instance
Â
jobs_serverÂ -Â ppÂ serverÂ forÂ submittingÂ jobs
funcÃ -Â functionÂ toÂ beÂ executed
depfuncsÂ -Â tupleÂ withÂ functionsÂ whichÂ mightÂ beÂ calledÂ fromÂ 'func'
modulesÂ -Â tupleÂ withÂ moduleÂ namesÂ toÂ import
callbackÂ -Â callbackÂ functionÂ whichÂ willÂ beÂ calledÂ withÂ argumentÂ
Â Â Â Â Â Â Â listÂ equalÂ toÂ callbackargs+(result,)Â
Â Â Â Â Â Â Â Â asÂ soonÂ asÂ calculationÂ isÂ done
callbackargsÂ -Â additionalÂ argumentsÂ forÂ callbackÂ function
groupÂ -Â jobÂ group,Â isÂ usedÂ whenÂ wait(group)Â isÂ calledÂ toÂ waitÂ for
jobsÂ inÂ aÂ givenÂ groupÂ toÂ finish
globalsÂ -Â dictionaryÂ fromÂ whichÂ allÂ modules,Â functionsÂ andÂ classes
willÂ beÂ imported,Â forÂ instance:Â globals=globals() submit(self, *args)SubmitsÂ functionÂ withÂ *argÂ argumentsÂ toÂ theÂ executionÂ queue

Â
Data Â Â Â Â Â Â Â copyright = 'Copyright (c) 2005-2012 Vitalii Vanovschi. All rights reserved'
version = '1.6.3' Â Quick start guide, SMP

1) Import pp module:

Â Â Â import pp

2) Start pp execution server with the number of workers set toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system

Â Â Â job_server = pp.Server()Â

3) Submit all the tasks for parallel execution:

Â Â Â f1 = job_server.submit(func1, args1, depfuncs1, modules1)

Â Â Â f2 = job_server.submit(func1, args2, depfuncs1, modules1)

Â Â Â f3 = job_server.submit(func2, args3, depfuncs2, modules2)

Â Â ...etc...

4) Retrieve the results as needed:

Â Â Â r1 = f1()

Â Â Â r2 = f2()

Â Â Â r3 = f3()Â

Â Â Â ...etc...

Â To find out how to achieve efficient parallelization with pp please take a look at examples Â Quick start guide, clustersÂ

On the nodes

1) Start parallel python execution server on all your remote computational nodes:

Â Â Â node-1> ./ppserver.py

Â Â Â node-2> ./ppserver.py

Â Â Â node-3> ./ppserver.py

On the client

2) Import pp module:

Â Â Â import pp

3)Â Create a list of all the nodes in your cluster (computers where you've run ppserver.py)

Â Â Â ppservers=("node-1", "node-2", "node-3")

4) Start pp execution server with the number of workers set toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system and list of ppservers to connect with :

Â Â Â job_server = pp.Server(ppservers=ppservers)Â

5) Submit all the tasks for parallel execution:

Â Â Â f1 = job_server.submit(func1, args1, depfuncs1, modules1)

Â Â Â f2 = job_server.submit(func1, args2, depfuncs1, modules1)

Â Â Â f3 = job_server.submit(func2, args3, depfuncs2, modules2)

Â Â ...etc...

6) Retrieve the results as needed:

Â Â Â r1 = f1()

Â Â Â r2 = f2()

Â Â Â r3 = f3()Â

Â Â Â ...etc...

Â To find out how to achieve efficient parallelization with pp please take a look at examples Â Quick start guide, clusters with autodiscovery

On the nodesÂ

1) Start parallel python execution server on all your remote computational nodes:

Â Â Â node-1> ./ppserver.py -a

Â Â Â node-2> ./ppserver.py -a

Â Â Â node-3> ./ppserver.py -a

On the client

2) Import pp module:

Â Â Â import pp

3)Â Set ppservers list to auto-discovery:

Â Â Â ppservers=("*",)

4) Start pp execution server with the number of workers set toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system and list of ppservers to connect with :

Â Â Â job_server = pp.Server(ppservers=ppservers)Â

5) Submit all the tasks for parallel execution:

Â Â Â f1 = job_server.submit(func1, args1, depfuncs1, modules1)

Â Â Â f2 = job_server.submit(func1, args2, depfuncs1, modules1)

Â Â Â f3 = job_server.submit(func2, args3, depfuncs2, modules2)

Â Â ...etc...

6) Retrieve the results as needed:

Â Â Â r1 = f1()

Â Â Â r2 = f2()

Â Â Â r3 = f3()Â

Â Â Â ...etc...

Â To find out how to achieve efficient parallelization with pp please take a look at examplesÂ Â Â Â Advanced guide, clustersÂ

On the nodes Â

1) Start parallel python execution server on all your remote computational nodes (listen to a given port 35000,
and local network interface only, accept only connections which know correct secret):

Â Â Â node-1> ./ppserver.py -p 35000 -i 192.168.0.101 -s "mysecret"

Â Â Â node-2> ./ppserver.py -p 35000 -i 192.168.0.102 -s "mysecret"

Â Â Â node-3> ./ppserver.py -p 35000 -i 192.168.0.103 -s "mysecret"

On the client

2) Import pp module:

Â Â Â import pp

3)Â Create a list of all the nodes in your cluster (computers where you've run ppserver.py)

Â Â Â ppservers=("node-1:35000", "node-2:35000", "node-3:35000")

4) Start pp execution server with the number of workers set toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system,
list of ppservers to connect with and secret key to authorize the connection:

Â Â Â job_server = pp.Server(ppservers=ppservers, secret="mysecret")Â

5) Submit all the tasks for parallel execution:

Â Â Â f1 = job_server.submit(func1, args1, depfuncs1, modules1)

Â Â Â f2 = job_server.submit(func1, args2, depfuncs1, modules1)

Â Â Â f3 = job_server.submit(func2, args3, depfuncs2, modules2)

Â Â ...etc...

6) Retrieve the results as needed:

Â Â Â r1 = f1()

Â Â Â r2 = f2()

Â Â Â r3 = f3()Â

Â Â Â ...etc...

Â 7) Print the execution statistics:

Â Â Â job_server.print_stats()

To find out how to achieve efficient parallelization with pp please take a look at examples Â Command line options,
ppserver.py Usage: ppserver.py [-hda] [-i interface] [-b broadcast] [-p port] [-w nworkers] [-s secret] [-t seconds]
 Options:
 -h   : this help message
 -d   : debug
 -a   : enable auto-discovery service
 -i interface   : interface to listen
 -b broadcast   : broadcast address for auto-discovery service
 -p port   : port to listen
 -w nworkers   : number of workers to start
 -s secret   : secret for authentication
 -t seconds   : timeout to exit if no connections with clients exist
 -k seconds   : socket timeout in seconds
 -P pid_file   : file to write PID to
Â Security and secret key

Â Due to the security concerns it is highly recommended to run ppserver.py with an non-trivial secret key (-s command line argument) which should be paired with the matching secret keyword of PP Server class constructor. Since PP 1.5.3 it is possible to set secret key by assigning pp_secret variable in the configuration file .pythonrc.py which should be located in the user home directory (please make this file readable and writable only by user). The key set in .pythonrc.py could be overridden by command line argument (for ppserver.py) and secret keyword (for PP Server class constructor).

Â Â ppserver.py stats and PID file example

To print job execution statistics for ppserver.py send a SIGUSR1 signal to its main process.
For instance on UNIX platform following commands will start a server and print its stats:
ppserver.pyÂ -P /tmp/ppserver.pid

kill -s SIGUSR1 `cat /tmp/ppserver.pid`