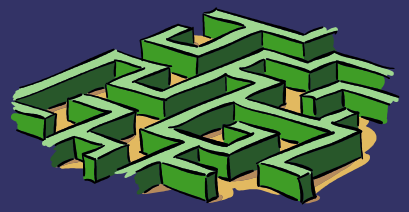
Deploying OpenERP Server with Gunicorn

Vo Minh Thu, OpenERP s.a. (@noteed)

April 11, 2012

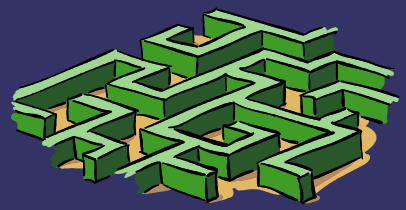


Overview

We wanted to achieve:

- Better performance on multi-core machines
- Better stability (CPU and memory consumption monitoring, less interference between a misbehaving request and other requests)
- Simplify our code base

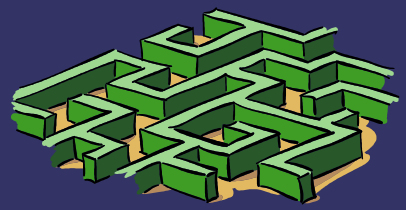
demo.openerp.com uses the new setup for some times now, and it really proves useful.



Overview (cont'd)

The puzzle pieces:

- Multi-threading and the GIL
- Stateless processes
- OpenERP is WSGI compliant (since 6.1)
- Gunicorn is a WSGI compliant HTTP server (with some nice features)



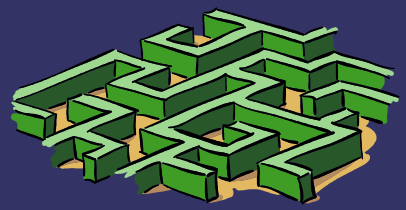
Multi-threading and the GIL

In CPython, the Global Interpreter Lock (GIL for short) prevents multiple OS threads from executing Python bytecode at the same time. This is also true on multi-core systems.

To take advantage of multiple cores, a solution is to use multiple processes instead of threads (e.g. with the standard multiprocessing library).

To have multiple instances of the OpenERP server talk to the same database, we have to make the server stateless.

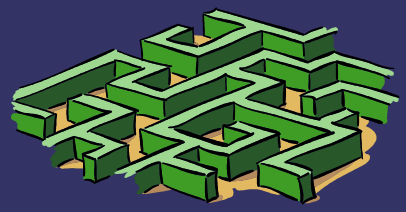
See e.g. the Python wiki wiki.python.org/moin/GlobalInterpreterLock or Understanding the GIL at www.dabeaz.com/GIL/.



State

Prior to v6.1, OpenERP server was not stateless (it is still not striclty stateless but that's ok).

osv_memory
Model registry (installing a new module updates the registry)
@tools.ormcache
Various other caches and other hidden places (e.g. Float field's digits_compute implementation)
We treat the last point as "caches" too.

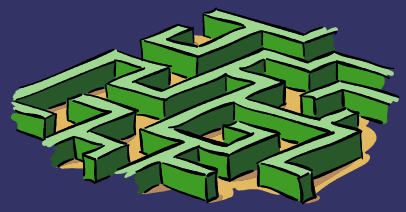


State (cont'd) - osv_memory

osv_memory are now implemented as regular osv objects (now called TransientModel and Model respectively).

TransientModel are simply deleted from the database automatically after some time, or after a numerical limit (just as osv_memory were removed from memory in the same way).

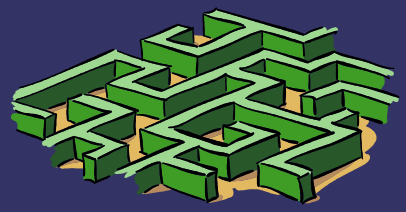
For instance, a wizard can be initiated on one process, and the next request can be served by another process, possibly on different machines.



State (cont'd) - registry and caches

OpenERP supports dynamic reconfiguration: modules can be installed, thus creating new models, views, workflows, ... at runtime. Different OpenERP processes serving the same database must be kept in sync.

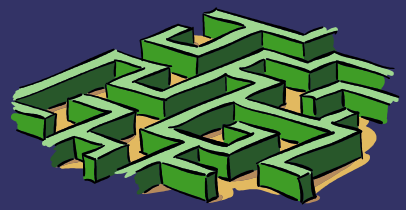
Whenever a module is installed, or a cache is invalidated, the process increments a PostgreSQL sequence. Other processes test the sequence value before handling each request. If necessary, they reload the registry or invalidate their own caches.



WSGI

The Web Server Gateway Interface is a standard (in the Python world) interface between a web application and a web server.

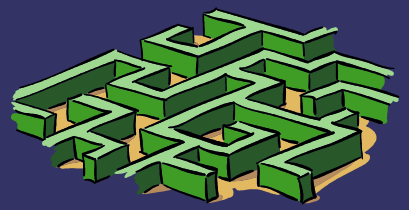
- A WSGI compliant web server translates HTTP requests back and forth and calls an application entry point (a callable)
- A WSGI compliant application can be embedded in any WSGI compliant web server
- See e.g. the Wikipedia page en.wikipedia.org/wiki/Web_Server_Gateway_Interface or wsgi.org.



OpenERP is WSGI compliant

Since OpenERP 6.1, the server is WSGI compliant

- can be called by a WSGI server (e.g. Apache + mod_wsgi)
- can embed (register and serve) other WSGI applications
- E.g. the 6.1 web client is a WSGI compliant application that is usualy embedded in the server.

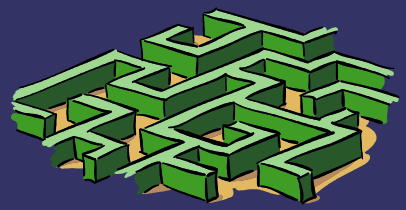


Gunicorn

What is Gunicorn?

- Gunicorn is a HTTP server
- WSGI compliant
- written in Python (port of Ruby's Unicorn server)
- multi-process
- uses the pre-fork worker model
- is flexible and configured by a Python script

  See gunicorn.org

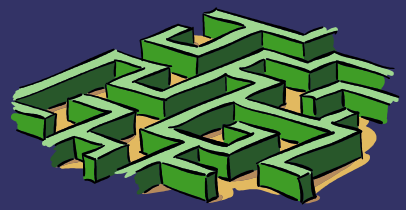


Arbitrer

In Gunicorn, the main process, called "arbiter", forks itself and manages its children: the workers.

The arbiter:

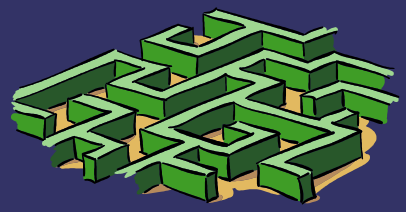
- restarts killed workers
- can increment/decrement the worker count via signals (TTIN and TTOU)
- doesn't know about HTTP
- can't inspect the HTTP request to dispatch to "more appropriate" workers (e.g. a worker with an already loaded registry for the desired database).



Workers

know about HTTP

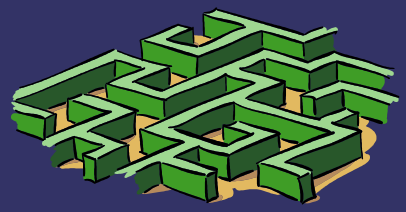
- do the WSGI translation
- call the application entry point
- can gracefully die after a configured number of requests



Configuration

Gunicorn configuration is a Python script. A sample gunicorn.conf.py file is given with the server.

Being a Python file, we can populate standard OpenERP configuration values from within the script.



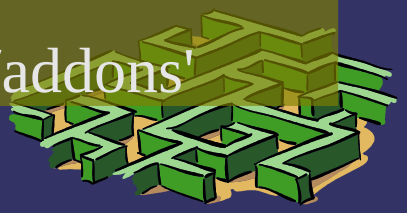
Configuration (cont'd)

```
import openerp

bind = '127.0.0.1:8069'
pidfile = '.gunicorn.pid'
workers = 4

on_starting = openerp.wsgi.core.on_starting
when_ready = openerp.wsgi.core.when_ready
pre_request = openerp.wsgi.core.pre_request
post_request = openerp.wsgi.core.post_request

conf = openerp.tools.config
conf['addons_path'] =
'/home/openerp/addons/trunk,/home/openerp/web/trunk/addons'
```



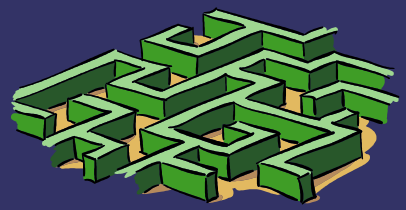
Process monitoring

Gunicorn, by restarting workers (because of a timeout, or they were killed, or they have reach some limit on the number of requests), provides a simple process monitoring solution.

It is also able to dynamically reload you application (via a HUP signal).

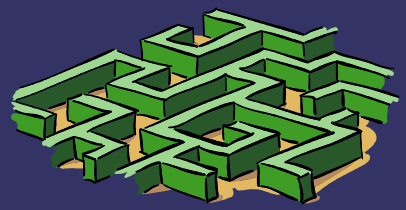
In the worker, we also establish limits on CPU and memory usage (with the resource Python library). Workers will be gracefully/hard killed when overpassing those limits.

This is used to prevent unwanted memory growth, process lock up or taking too much time.



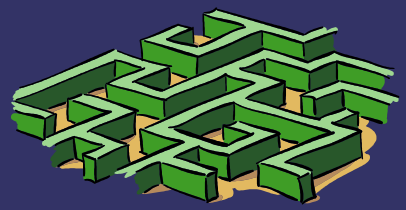
Running OpenERP with Gunicorn

```
> gunicorn openerp:wsgi.core.application -c gunicorn.conf.py
```



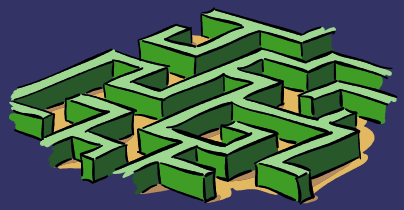
Cron jobs

- OpenERP with Gunicorn doesn't run cron jobs
- In 6.1, until recently, cron jobs were multi-threaded
- We have now a multi-process solution
- openerp-cron-worker instead of openerp-server will run a single process to execute cron jobs
- More than one such process can be run at the same time (We hope to tie soon the cron worker script and the server script into a new oe tool, already available at launch-pad.net/openerp-command.)



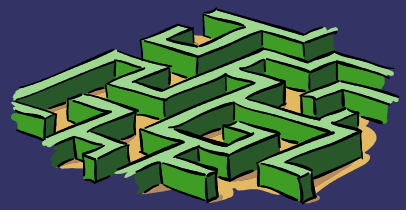
Good to know

- Each Gunicorn worker is not multi-threaded: you can have as many parallel requests as you have workers, not more.
- For requests using only CPU, you can really serve two requests in parallel (assuming you have at least two workers and two cpu or cores).
- For requests mainly using the same tables and/or rows in database, requests will be sequentialized.
- Default Gunicorn worker types are designed to handle requests proxied by Nginx (Nginx manages request buffering and keep-alive).
- The web client stores sessions on disk (so can't be deployed itself on different machines unless you make sure sessions are shared).



Conclusion

- WSGI compliance makes OpenERP simpler
- It makes it more versatile (more deployment options, new web modules, ...)
- With statelessness, multiple OpenERP processes can use the same database
- With multiple processes, we can reap the benefits of multi-core systems (overpassing the GIL)
- Using Gunicorn (and rlimit), we can easily manage multiple processes and have (simple but useful) process monitoring



Thanks!

Any questions?