Coopengo: How Tryton is customized to empower Coog

Jean Cavallo
Ali Kefia

Oct 18, 2016

Our story with Tryton

- Tryton is a very good platform for developers
  - Focus on vertical from the first day
  - Good practices for developers (easy to drive projects)
  - Modularity is helpful to separate concerns (different business lines)
  - So many helpful features (internationalization, rights, etc.)

- But Tryton is sometimes hard to operate
  - No practices to deploy / monitor / scale
  - No simple process to analyze performance issues
  - Hard to understand low level code (getters, python-sql, etc.)

- Coopengo experienced difficulties last year to ramp up Coog as a main backoffice application for a team of 50 users

But of course we still love Tryton

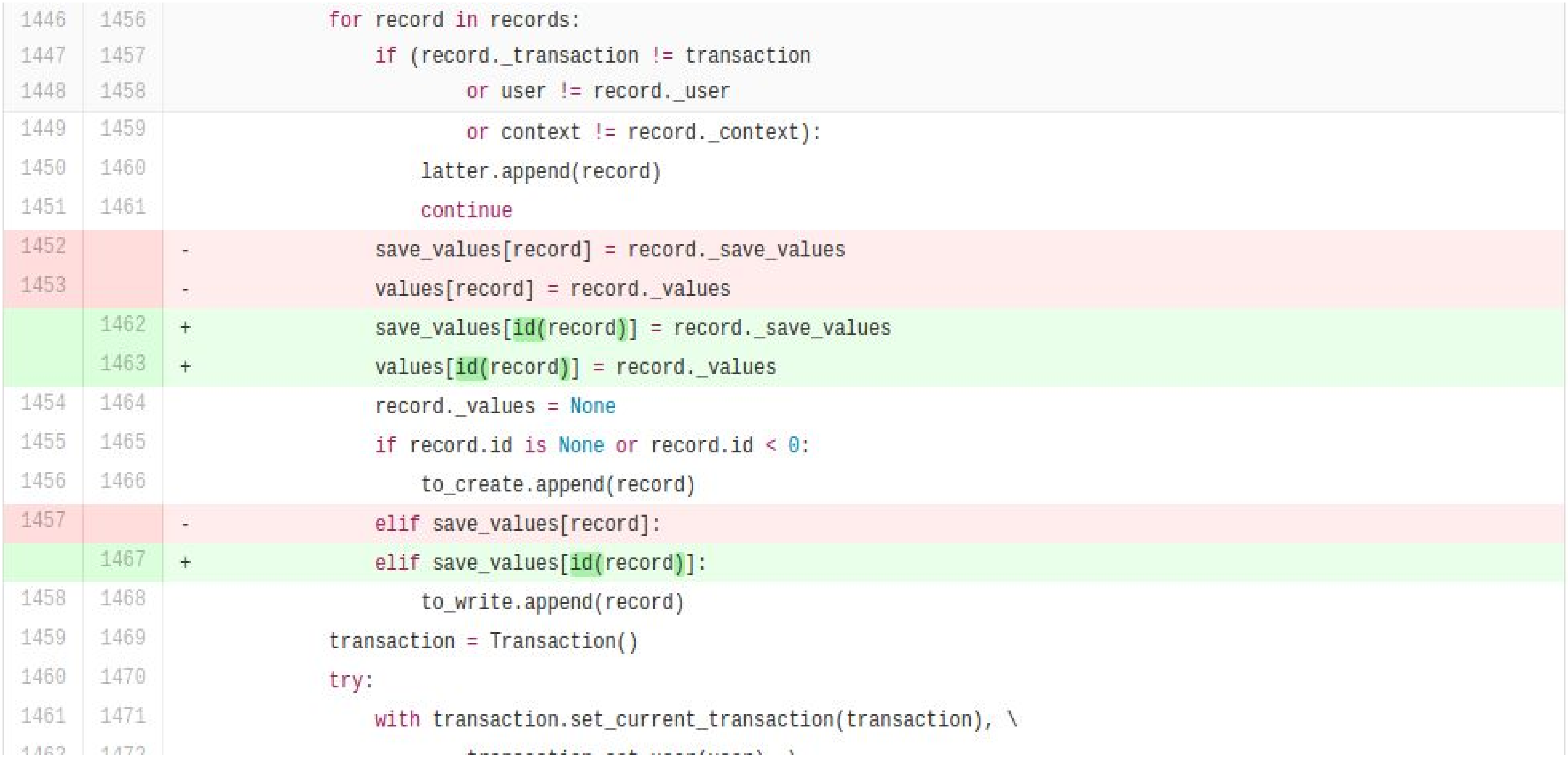
Problems we faced

- Limited to a single process server (3.8)

- Too much server calls (heavy activity)
  - Surely due to our model (on_change, function fields)
  - But also sometimes for hidden configuration (storing column width in server)

- Some server calls take long time
  - Heavy processing for rating
  - Many intermediate records to save (design problems)
  - Some low level bottlenecks discovered

Problems we faced (example)



```diff
         for record in records:
             if (record._transaction != transaction
                     or user != record._user
                     or context != record._context):
                 latter.append(record)
                 continue
-            save_values[record] = record._save_values
-            values[record] = record._values
+            save_values[id(record)] = record._save_values
+            values[id(record)] = record._values
             record._values = None
             if record.id is None or record.id < 0:
                 to_create.append(record)
-            elif save_values[record]:
+            elif save_values[id(record)]:
                 to_write.append(record)
         transaction = Transaction()
         try:
             with transaction.set_current_transaction(transaction), \
                     transaction.set_user(user), \
```

Solutions

- Tools
  - debug module
  - performance analyzer

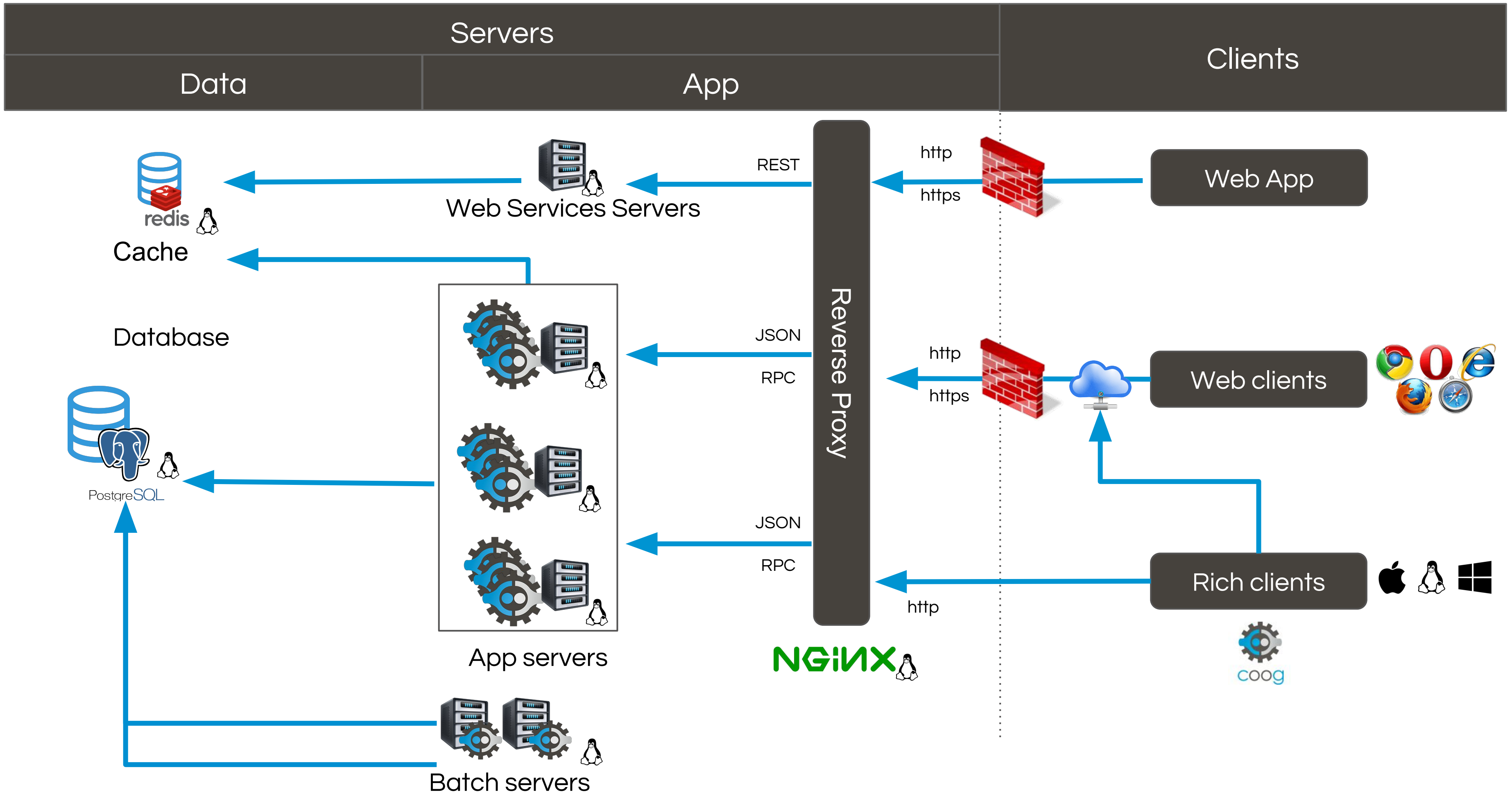
- Deployment Architecture
  - Redis as a shared cache
  - Nginx as a load balancer

Solutions - Tools

- Debug Module
  - record exploration, arbitrary code evaluation
  - model introspection
  - utilities (PYSON conversion…)
  - editor hooks

- Perf-Analyzer
  - logs rpc calls and db accesses per session
  - extra logs on db (sql on specific db calls - > 1 sec)
  - extra logs on rpc call (profile the call from dispatcher)
  - based on user (production ready with minimum overhead)

Solutions - Deployment Architecture

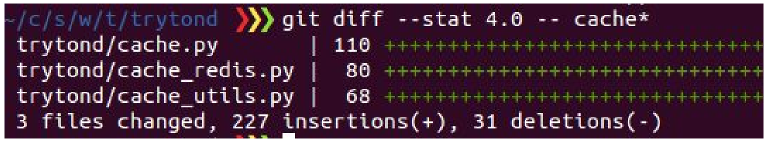


Solutions - Deployment Architecture

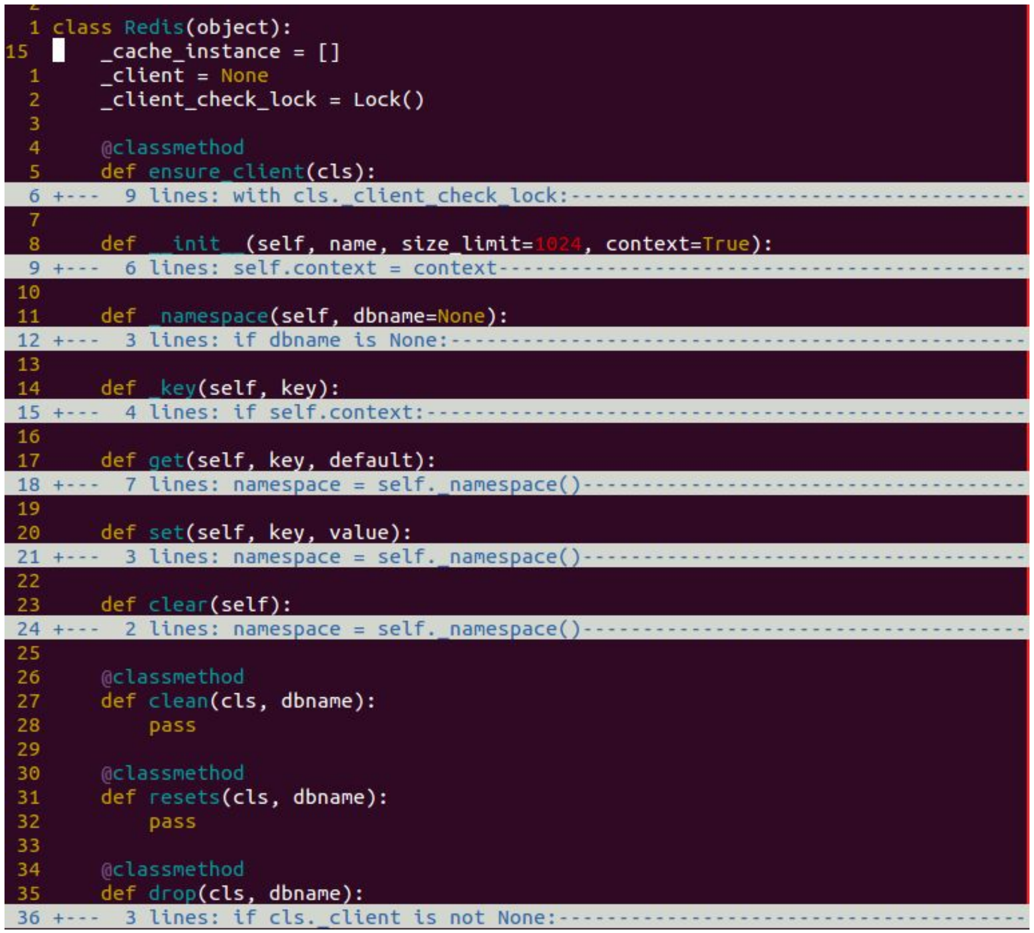
- Redis as a default cache for trytond
  - same API as trytond/cache, implem from config
  - msgpack to serialize
  - some issues with non serializable data (rng)
  - modular tryton cache management to avoid fork

- Nginx as load balancer and reverse proxy
  - works with Tryton 3.8
  - on Tryton >= 4.0
    - scales on different servers
    - works well with uwsg
  - alternative to SSL from Python
  - basic security rules

Deployment: Redis



```
~/c/s/w/t/trytond ❯❯❯ git diff --stat 4.0 -- cache*
 trytond/cache.py        | 110 ++++++++++++++++++++++++++++++
 trytond/cache_redis.py  |  80 ++++++++++++++++++++++++++++++
 trytond/cache_utils.py  |  68 ++++++++++++++++++++++++++++++
 3 files changed, 227 insertions(+), 31 deletions(-)
```



```
class Redis(object):
    _cache_instance = []
    _client = None
    _client_check_lock = Lock()

    @classmethod
    def ensure_client(cls):
+---  9 lines: with cls._client_check_lock:---------------------------------------

    def __init__(self, name, size_limit=1024, context=True):
+---  6 lines: self.context = context----------------------------------------------

    def _namespace(self, dbname=None):
+---  3 lines: if dbname is None:--------------------------------------------------

    def _key(self, key):
+---  4 lines: if self.context:----------------------------------------------------

    def get(self, key, default):
+---  7 lines: namespace = self._namespace()---------------------------------------

    def set(self, key, value):
+---  3 lines: namespace = self._namespace()---------------------------------------

    def clear(self):
+---  2 lines: namespace = self._namespace()---------------------------------------

    @classmethod
    def clean(cls, dbname):
        pass

    @classmethod
    def resets(cls, dbname):
        pass

    @classmethod
    def drop(cls, dbname):
+---  3 lines: if cls._client is not None:-----------------------------------------
```

Going further - More on deployment

- On Tryton 4.0, combine uwsgi and nginx to explore more possibilities
- Docker as deployment tool
  - easier when dealing with distribution specificities
  - industrialization (scripts for update, monitor, etc.)
- All managed via coog-admin
- Tryton to adopt Redis as a cache broker (and move things like session to cache)?
- Nginx module for Tryton (log called method from JSON)

Going further - Web

- Purpose: nodejs technology for middleware (concurrency, active community, etc.)
- Started as a Sao code extraction
  - extracted communication / model features
  - remove some constraints : mono-session, jquery deps, etc
- Now standalone libraries
  - types: datetime convenient constructors
  - session: model description, cache, hooks on start/stop
  - model: easy API to read, get, save, etc.

Conclusion

- Within Tryton Community, we are missing
  - practices (for example django explains how it can be deployed on nginx)
  - knowledge share (a good configuration for uwsgi: threads / processes)
  - story telling (that company is using Tryton for 1000 fulltime users and it is working well)
  - modules store / market place with incentives (stars)

- Very often, you get yourself alone, you experiment things and you decide based on an individual context