Chrono - Development #42169

perfs, structure/algos sur les intervalles

27 avril 2020 21:43 - Frédéric Péters

Statut:	Fermé	**Début:**	27 avril 2020
Priorité:	Normal	**Echéance:**	
Assigné à:	Benjamin Dauvergne	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	Non

Description

Après #42142#note-5, on arrive donc à 20 secondes,

```
                for slot in time_period.get_time_slots(
                    base_duration=base_duration, **used_time_period_filters
                ):
                    slot.full = False
                    open_slots[agenda.id][time_period.desk_id].add(
                        slot.start_datetime, slot.end_datetime, slot
                    )
```

Sur une exécution, il y a ~7 secondes cumulées passées dans les appels à get_time_slots et ~6 dans les appels à .add().

En tout un total ~13 pour récupérer tous les slots puis 7 secondes pour la partie supprimant les zones d'exclusion (sous "# remove excluded slot"), dans cette partie c'est 6 secondes dans remove_overlap.

Bref, tout m'apparait désormais comme relevant de la structure de données et des algorithmes de gestion des intervalles.

Pour donner une idée de la tête de la structure, pp open_slots[agenda.id][desk_id].container :

```
[[],
 [],
 [],
 ... (en fait il y a 990 listes vides),
 [<Interval [2020-05-07 08:00:00+02:00, 2020-05-07 08:05:00+02:00] 7 mai 2020 08:00>],
 [<Interval [2020-05-07 08:00:00+02:00, 2020-05-07 08:05:00+02:00] 7 mai 2020 08:00>,
  <Interval [2020-05-07 08:05:00+02:00, 2020-05-07 08:10:00+02:00] 7 mai 2020 08:05>],
 [<Interval [2020-05-07 08:05:00+02:00, 2020-05-07 08:10:00+02:00] 7 mai 2020 08:05>,
  <Interval [2020-05-07 08:10:00+02:00, 2020-05-07 08:15:00+02:00] 7 mai 2020 08:10>],
 [<Interval [2020-05-07 08:10:00+02:00, 2020-05-07 08:15:00+02:00] 7 mai 2020 08:10>,
  <Interval [2020-05-07 08:15:00+02:00, 2020-05-07 08:20:00+02:00] 7 mai 2020 08:15>],
 ... (330 listes avec deux éléments)
 [<Interval [2020-05-09 17:45:00+02:00, 2020-05-09 17:50:00+02:00] 9 mai 2020 17:45>,
  <Interval [2020-05-09 17:50:00+02:00, 2020-05-09 17:55:00+02:00] 9 mai 2020 17:50>],
 [<Interval [2020-05-09 17:50:00+02:00, 2020-05-09 17:55:00+02:00] 9 mai 2020 17:50>,
  <Interval [2020-05-09 17:55:00+02:00, 2020-05-09 18:00:00+02:00] 9 mai 2020 17:55>],
 [<Interval [2020-05-09 17:55:00+02:00, 2020-05-09 18:00:00+02:00] 9 mai 2020 17:55>],
 [],
 [],
 ... (en fait 5279 lignes vides)
 []]
```

Demandes liées:

Lié à Chrono - Bug #42332: Erreur dans le filtrage de rendez-vous bookés (dan...	**Rejeté**	**30 avril 2020**

Révisions associées

Révision 3c5d056f - 19 mai 2020 14:54 - Benjamin Dauvergne

add test on metz data (#42169)

Révision ece063b2 - 19 mai 2020 18:19 - Benjamin Dauvergne

api: optimize get_all_slots() and around it (#42169)

Workflow in get_all_slots() is simplified :

- first we accumulate, for each desk, the set of time slots when a booking cannot occur or is already booked,
- then we generate the list of possible time slots and match them to the exclusion and already booked set.

Intervals is replaced by a simpler data-structure, IntervalSet, it does not need to be a map, a simple set is enough.

Also :

- moved TimePeriod.get_effective_timeperiods() to the agenda level , it deduplictes TimePeriod between desks and remove excluded TimePeriod for virtual agendas.

- added a named-tuple WeekTime to represent a TimePeriod base unit, so we can use them in IntervalSet easily (as they can be compared) to compute the effective time periods,

- the fact that base_duration is unique for a given virtual agenda is now accounted and stated everywhere,

- the fact that generated time slots must have time in the local timezone for the API to work is now stated everywhere,

- In get_all_slots(), also : * integrated the code of get_exceptions_by_desk() into get_all_slots() to further reduce the number of SQL queries. * used_min/max_datetime is reduced by the exclusion periods, and effective time periods are grouped based on the used_min/max_datetime of each agenda. * pre-filter slots for uniqueness when generating available datetimes (but for filling slot we still need exact availability information for each desk)

Historique

#1 - 28 avril 2020 23:21 - Benjamin Dauvergne

Je suis retourné voir le truc qu'on utilisait à l'origine (https://pypi.org/project/intervaltree/) ça semble beaucoup plus stable et le bug qu'on avait est corrigé depuis deux ans[1] (et surtout l'API est la même vu qu'elle vient de là). La dernière version a la bonne idée d'être disponible en buster et c'est du pur python.

J'ai mesuré un cycle de 5000 add() puis 5000 remove_overlap() (avec très peu de collision en fait ce qui me semble notre cas), c'est 24x plus rapide que l'implémentation dans chrono.

[1] https://dev.entrouvert.org/issues/20732https://github.com/chaimleib/intervaltree/issues/41

#2 - 29 avril 2020 08:01 - Frédéric Péters

J'ai fait

```
class Intervals(IntervalTree):
    def add_expanded(self, begin, end, data=None):
        self.add(Interval(begin, end, data))

    def iter_data(self):
        'Iterate data element attached to intervals'
        for interval in self:
            yield interval.data

    def search_data(self, begin, end):
        'Search data elements of overlapping intervals'
        for interval in self.overlap(begin, end):
            yield interval.data
```

pour compléter l'API avec ce qu'on utilise, + bien sûr remplacer le .add() par ce .add_expanded(), les tests passent nickel (j'ai juste regardé test_api.py).

J'ai utilisé ce code sur la situation metz, j'ai eu un timeout à 60 secondes. (là où on a un résultat à 20 secondes avec le code actuel).

J'ai regardé les perfs sur les tests, 396 secondes, vs avec le code actuel 365 secondes.

#3 - 29 avril 2020 11:00 - Benjamin Dauvergne

Ok, mon test était pas significatif (j'ai testé remove() et pas remove_overlap(), notre remove_overlap() est beaucoup plus rapide); je regarderai pour

optimiser le code actuel (sauf si tu préfères regarder) mais ce serait bien d'avoir un test de perf approchant du cas Metz dans les tests (je ne sais pas à quel point c'est simple à reproduire).

#4 - 29 avril 2020 11:19 - Benjamin Dauvergne

~~Sur un microbenchmark je vois que le plus coûteux c'est add(), idem un micro-benchmark je gagne pas mal en pré-allaount self.points et self.container (avec une politique de doublement quand c'est plein) et en utilisant des move l[i+1:i+1+usage] = l[i:usage].~~

Bêtise c'est l'allocation des listes le plus coûteux.

#5 - 29 avril 2020 11:39 - Frédéric Péters

- Fichier metz.json ajouté

export metz en pièce jointe, https://.../api/agenda/vdmvirtuel-retraitmasque/meetings/un-retrait/datetimes/ pour l'url testée.

#6 - 30 avril 2020 06:08 - Benjamin Dauvergne

Voilà

```
tests/test_metz.py .
                                                                [100%]

============================================================================ warnings summary ================
================================================================
tests/test_metz.py::test_get_all_slots
  /tmp/tox-bdauvergne/chrono/py3-django111-pg/lib/python3.7/site-packages/django/db/models/fields/__init__.py:
436: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is d
eprecated, and in 3.8 it will stop working
    if name == "choices" and isinstance(value, collections.Iterable):

-- Docs: https://docs.pytest.org/en/latest/warnings.html
========================================================================== slowest 5 test durations ===========
================================================================
3.55s call     tests/test_metz.py::test_get_all_slots
3.46s setup    tests/test_metz.py::test_get_all_slots
0.15s teardown tests/test_metz.py::test_get_all_slots
```

Ce fut laborieux, je ne pense pas pouvoir aller plus loin.

#7 - 30 avril 2020 08:43 - Frédéric Péters

Je pense assez utile de parsemer ce code de commentaires/explications, peut-être déjà en docstring de EffectiveTimePeriod

~~

Aussi, tu affiches juste des perfs "après", c'était quoi, "avant" ?

~~

```
-    assert localtime(hours[2].begin).time() == datetime.time(14, 0)
-    assert localtime(hours[2].end).time() == datetime.time(17, 0)
+    assert localtime(hours[2][0]).time() == datetime.time(14, 0)
+    assert localtime(hours[2][1]).time() == datetime.time(17, 0)
```

Sans avoir tout lu, encore moins testé ou réfléchi, comme tu utilises par ailleurs namedtuple, ça peut pas être aussi utilisé ici, pour rester sur .begin/.end ?

~~

D'un côté "import itertools", de l'autre "from itertools import groupby", ça m'irait mieux d'avoir "import itertools", ce ne sont pas des fonctions qu'on utilise couramment du coup avoir au moment de l'appel le rappel de leur origine, je trouve ça utile.

~~

(j'arrête ici et passe la relecture à quelqu'un d'autre).

#8 - 30 avril 2020 11:28 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Je pense assez utile de parsemer ce code de commentaires/explications, peut-être déjà en docstring de EffectiveTimePeriod

Ok.

> Aussi, tu affiches juste des perfs "après", c'était quoi, "avant" ?

Sur ma machine c'était 20 secondes, sur base de tout le code dans 42142, d'ailleurs il faudrait le rebaser et le pousser pour que je puisse refaire un patch propre, ton patch sur les requêtes SQL s'est retrouvé mergé dans le mien.

> ~~
>
> [...]
>
> Sans avoir tout lu, encore moins testé ou réfléchi, comme tu utilises par ailleurs namedtuple, ça peut pas être aussi utilisé ici, pour rester sur .begin/.end ?

Oui, il y a beaucoup de code des fois j'ai fait au plus vite, mais pas de souci.

> D'un côté "import itertools", de l'autre "from itertools import groupby", ça m'irait mieux d'avoir "import itertools", ce ne sont pas des fonctions qu'on utilise couramment du coup avoir au moment de l'appel le rappel de leur origine, je trouve ça utile.

Ok.

#9 - 30 avril 2020 12:11 - Benjamin Dauvergne

J'ai poussé dans wip/42142-perfs-rebases ta branché 42142 rebasé avec les tests metz ça donne le temps de base dont je suis parti :

```
21.51s call     tests/test_metz.py::test_get_all_slots
```

Le gros du temps gagné c'est le changement d'ordre dans le temps : plutôt que d'accumuler tous les slots puis de les retirer, d'abord accumuler les moments impossibles soit pour exclusion soit parce que booké (indexé par desk), puis ensuite générer les slots et les générer à la volée, ça m'a amené vers des temps autours de 6 secondes.

Ensuite pour éviter de générer trop de fois le même slots dans period.get_time_slots() il y a le fait d'agréger les les timperiods par (desk, min_used_time, max_used_time).

#10 - 30 avril 2020 14:08 - Benjamin Dauvergne

- Lié à Bug #42332: Erreur dans le filtrage de rendez-vous bookés (dans get_all_slots()) ajouté

#11 - 30 avril 2020 16:01 - Benjamin Dauvergne

- Statut changé de Nouveau à Solution proposée

- Assigné à mis à Benjamin Dauvergne

Voilà doc et modifications apportées, à relire.

#12 - 04 mai 2020 10:40 - Benjamin Dauvergne

- Statut changé de Solution proposée à En cours

Je vais pousser la dernière version à 2,15s.

#13 - 04 mai 2020 22:14 - Benjamin Dauvergne

- Statut changé de En cours à Solution validée

#14 - 04 mai 2020 22:35 - Frédéric Péters

- Statut changé de Solution validée à Solution proposée

#15 - 05 mai 2020 15:27 - Emmanuel Cazenave

Quelques remarques sur la lisibilité et un peu plus.

1. Sur @IntervalSet.

_begin_or_interval peut renvoyer une valeur simplre ou un tuple mais c'est toujours appelé sous la forme :

```
begin, end = self._begin_or_interval(begin, end)
```

Sous cette forme ça ne fait rien d'autre que renvoyer begin, end du coup ça semble pouvoir dégager. Par domino ça pourrait faire dégager aussi simple et from_ordered, pour en rester au bon vieux *init* bien compréhensible (d'ailleurs je ne comprends pas vraiment ce qui y est attendu dans iterable)

Dans le même optique d'épure de l'interface, init_from_ordered appelé uniquement dans *init*, pas besoin de l'exposer à part.

Le cast, pour pouvoir passer des listes à certaines méthodes plutôt que des IntervalSet si je comprends bien, ça complique la lecture, ça semble être utilisé uniquement pour les tests, si c'est bien le cas et que ça peut dégager ce serait cool pour les neurones.

Dans les tests :

```
IS = IntervalSet
```

avec IS utilisé une seule fois, partout ailleurs IntervalSet.

1. Sur models.py.

Je préférerais un get_effective_time_periods plain and simple plutôt que la bidouille sur les get_effective_time_periods_xxx qui ne sont appelés nulle part ailleurs.

Un superflu :

```
min_datetime = min_datetime
max_datetime = max_datetime
```

Pas encore lu le reste.

#16 - 05 mai 2020 17:45 - Benjamin Dauvergne

Emmanuel Cazenave a écrit :

> IntervalSet...

Je vais ignorer ça, ça n'a pas vraiment de rapport avec cette relecture qui concerne les algos, le code comme il est me convient.

> Dans les tests :
> [...]
>
> avec IS utilisé une seule fois, partout ailleurs IntervalSet.

Ok.

> 1. Sur models.py.
>
> Je préférerais un get_effective_time_periods plain and simple plutôt que la bidouille sur les get_effective_time_periods_xxx qui ne sont appelés nulle part ailleurs.

Ok.

> Un superflu :
>
> [...]

Ok.

Branche à jour.

#17 - 05 mai 2020 17:51 - Frédéric Péters

> > IntervalSet...
>
> Je vais ignorer ça, ça n'a pas vraiment de rapport avec cette relecture qui concerne les algos, le code comme il est me convient.

On va imaginer de la relecture d'Emmanuel que côté algo c'est ok, et que la relecture de ces aspects du code sont alors opportuns ?

#18 - 05 mai 2020 18:50 - Benjamin Dauvergne

J'ai intégré init_from_ordered() dans *init*() le reste m'est utile.

#19 - 05 mai 2020 20:48 - Frédéric Péters

> le reste m'est utile.

Je trouve utile que le code de nos modules soit "cool pour les neurones" comme écrivait Manu, mais d'accord pour plein de trucs, expliqués. (parce que là je me disais que ça allait finir par être validé ainsi et que j'allais juste faire un ticket derrière avec le commentaire de Manu, pour voir ses propositions).

#20 - 05 mai 2020 23:44 - Benjamin Dauvergne

Frédéric Péters a écrit :

> > le reste m'est utile.
>
> Je trouve utile que le code de nos modules soit "cool pour les neurones" comme écrivait Manu, mais d'accord pour plein de trucs, expliqués. (parce que là je me disais que ça allait finir par être validé ainsi et que j'allais juste faire un ticket derrière avec le commentaire de Manu, pour voir ses propositions).

Je trouve plus simple de faire IntervalSet() (-,+,&,|) [(start_time, end_time)] et de ne pas différencier .addi(start_time , end_time) de .add(Interval(start_time, end_time)) comme dans l'API précédente héritée de python-intervaltree, ça me rend la lecture du code beaucoup plus simple à moi de me dire que tout itérable contenant des 2-tuples fonctionnera. Mais si on me pointe un bout de code obscure je n'ai aucun souci pour le documenter, par exemple *sub* qui est beaucoup moins documentée que .add() ou .overlaps().

#21 - 19 mai 2020 11:55 - Lauréline Guérin

Quelques remarques de relecture:

Dans la méthode get_effective_time_periods_meetings, est-ce qu'il serait pertinent d'ajouter un select ou prefetch related sur desk comme dans get_effective_time_periods_virtual ?

Dans get_all_slots, pourrais-tu renommer desks_exclusion en desks_exceptions ? On manipule ici des TimePeriodException, alors que exclusion pourrait renvoyer aux excluded_timeperiods d'un agenda virtuel; je trouve ça confusant :)

Aussi, ça manque peut-être un peu de commentaires côté models.py

J'ai à peu près tout compris, mais ça m'a demandé pas mal de neurones quand même :)

#22 - 19 mai 2020 15:14 - Benjamin Dauvergne

> Dans la méthode get_effective_time_periods_meetings, est-ce qu'il serait pertinent d'ajouter un select ou prefetch related sur desk comme dans get_effective_time_periods_virtual ?

Ouaip je fais ça, j'ai surtout regardé le nombre de query sur les appels à un agenda virtuel mais on va peut-être encore en gagner.

> Dans get_all_slots, pourrais-tu renommer desks_exclusion en desks_exceptions ? On manipule ici des TimePeriodException, alors que exclusion pourrait renvoyer aux excluded_timeperiods d'un agenda virtuel; je trouve ça confusant :)

Ok.

Branche rebasée sur master.

> Aussi, ça manque peut-être un peu de commentaires côté models.py

J'ai quand même l'impression qu'il y en a plus qu'avant, je vais documenter les get_effective_time_periods je ne vois que ça dans mon code qui ne soit pas particulièrement documenté.

#23 - 19 mai 2020 16:04 - Lauréline Guérin

- Statut changé de Solution proposée à Solution validée

Ok pour moi; sauf mention contraire d'un autre relecteur je valide le ticket dans 1H :)

#24 - 19 mai 2020 16:09 - Lauréline Guérin

- Statut changé de Solution validée à Solution proposée

#25 - 19 mai 2020 17:06 - Lauréline Guérin

- Statut changé de Solution proposée à Solution validée

#26 - 19 mai 2020 18:19 - Benjamin Dauvergne

- Statut changé de Solution validée à Résolu (à déployer)

```
commit ece063b2b357394533da93b0502ae99869ad057f
Author: Benjamin Dauvergne <bdauvergne@entrouvert.com>
Date:   Thu Apr 30 12:16:38 2020 +0200

    api: optimize get_all_slots() and around it (#42169)

    Workflow in get_all_slots() is simplified :
    * first we accumulate, for each desk, the set of time slots when a booking cannot
    occur or is already booked,
    * then we generate the list of possible time slots and match them to the
    exclusion and already booked set.

    Intervals is replaced by a simpler data-structure, IntervalSet, it does
    not need to be a map, a simple set is enough.

    Also :
    * moved TimePeriod.get_effective_timeperiods() to the agenda level , it
    deduplictes TimePeriod between desks and remove excluded TimePeriod for
    virtual agendas.

    * added a named-tuple WeekTime to represent a TimePeriod base unit, so
    we can use them in IntervalSet easily (as they can be compared) to
    compute the effective time periods,

    * the fact that base_duration is unique for a given virtual agenda is
    now accounted and stated everywhere,

    * the fact that generated time slots must have time in the local
    timezone for the API to work is now stated everywhere,

    * In get_all_slots(), also :
      * integrated the code of get_exceptions_by_desk() into get_all_slots()
      to further reduce the number of SQL queries.
      * used_min/max_datetime is reduced by the exclusion periods, and
        effective time periods are grouped based on the used_min/max_datetime of
        each agenda.
      * pre-filter slots for uniqueness when generating available datetimes
        (but for filling slot we still need exact availability information
        for each desk)

commit 3c5d056fa36e583d8042f3e059e7cc0142ca4b1b
Author: Benjamin Dauvergne <bdauvergne@entrouvert.com>
Date:   Thu Apr 30 00:08:02 2020 +0200

    add test on metz data (#42169)
```

#27 - 19 mai 2020 23:17 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

Fichiers

metz.json	709 ko	29 avril 2020	Frédéric Péters