

# PostgreSQL et TimescaleDB pour la collecte de métriques à haute fréquence

Simon Delamare

LIP / CNRS

5 décembre 2019



## Données orientées *time series*

- Time series ?
  - ▶ données horodatées
  - ▶ temps + valeur + méta-données

time	value	device_id	metric_id
2019-11-28 08:16:54+00	77.186	nova-1	wattmetre_power_watt
2019-11-28 08:22:40+00	79.882	nova-1	wattmetre_power_watt
2019-11-28 08:32:49+00	74.966	nova-1	wattmetre_power_watt
2019-11-28 08:44:11+00	74.69	nova-2	wattmetre_power_watt
2019-11-28 08:49:52+00	77.146	nova-2	wattmetre_power_watt

## Données orientées *time series*

- Time series ?
  - ▶ données horodatées
  - ▶ temps + valeur + méta-données

time	value	device_id	metric_id
2019-11-28 08:16:54+00	77.186	nova-1	wattmetre_power_watt
2019-11-28 08:22:40+00	79.882	nova-1	wattmetre_power_watt
2019-11-28 08:32:49+00	74.966	nova-1	wattmetre_power_watt
2019-11-28 08:44:11+00	74.69	nova-2	wattmetre_power_watt
2019-11-28 08:49:52+00	77.146	nova-2	wattmetre_power_watt

- (nombreux) exemples : monitoring des systèmes informatiques, IoT, enregistrements d'évènements, ...
- Bases de données orientées *time series* :
  - ▶ InfluxDB, OpenTSDB, KairosDB (NoSQL)
  - ▶ **TimescaleDB (extension PostgreSQL)**

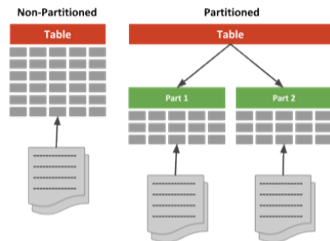
# TimescaleDB

## Pourquoi PostgreSQL ?

- SQL : syntaxe, cohabitation avec des données classiques
- Écosystème : clients, outils, savoir-faire, communauté, robustesse

## Mais... pas optimisé pour les *time series*

- Beaucoup de données  $\implies$  Grosses tables  
 $\implies$  Pas en RAM (les indexes)  $\implies$  Faibles perfs
- Solution : partitionnement (= on découpe les tables en sous-tables).  
Mais doit être réalisé par l'admin de la DB



(source: oracle-base.com)

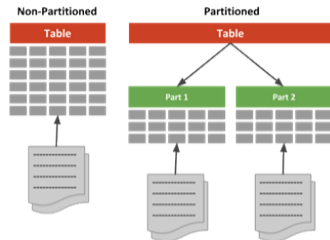
# TimescaleDB

## Pourquoi PostgreSQL ?

- SQL : syntaxe, cohabitation avec des données classiques
- Écosystème : clients, outils, savoir-faire, communauté, robustesse

## Mais... pas optimisé pour les *time series*

- Beaucoup de données  $\implies$  Grosses tables  
 $\implies$  Pas en RAM (les indexes)  $\implies$  Faibles perfs
- Solution : partitionnement (= on découpe les tables en sous-tables).  
Mais doit être réalisé par l'admin de la DB



(source: oracle-base.com)

## TimescaleDB

- Auto partitionnement sur des périodes de temps
- Les données insérées / accédées le sont généralement sur la partition la plus «récente»
- Cette partition est suffisamment petite pour tenir en RAM  $\implies$  Performances++
- Également : compression, agrégation de données, nouvelles opérations temporelles, ...

# Monitoring à haute fréquence avec Kwapi (NG)

*Prototype, déployé à l'essai dans Grid'5000*



Collecte et mise à disposition de métriques  
(données de capteur, mesures de perf.) :

- Wattmètre haute fréquence : 50 mesures par seconde
- SNMP, IPMI : Capteurs sur les interfaces de management des machines, équipements réseaux, PDUs...
- Focus sur les données «externes» aux machines

# Monitoring à haute fréquence avec Kwapi (NG)

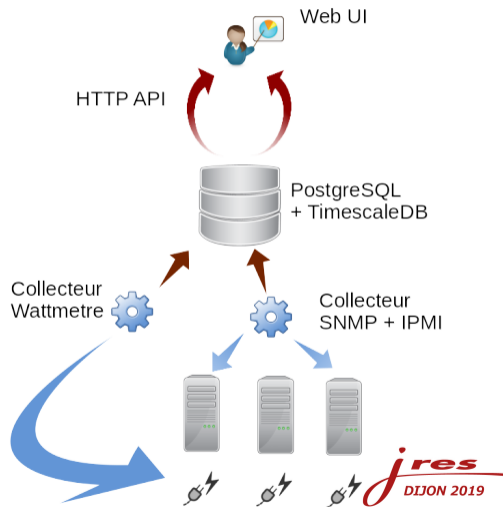
Prototype, déployé à l'essai dans Grid'5000



Collecte et mise à disposition de métriques  
(données de capteur, mesures de perf.) :

- Wattmetre haute fréquence : 50 mesures par seconde
- SNMP, IPMI : Capteurs sur les interfaces de management des machines, équipements réseaux, PDUs...
- Focus sur les données «externes» aux machines

**Architecture :**



## Premier retour d'expérience

- Déployé sur une VM «modeste» : 4 vCPUs @2.3Ghz, 16Go RAM
- Données collectées :  $> 3.10^6$  entrées/h
  - ▶ 30 valeurs/sec via SNMP (=10 requêtes SNMP/sec)
  - ▶ 1000 valeurs/sec via wattmetre (16 nœuds à haute fréquence, 120 nœuds à 1Hz)
  - ▶ 1Go de données par heure, 20Mo compressé
- Requêtes :
  - ▶  $\approx 1$  sec. pour  $10^5$  lignes (ex: toutes les métriques d'un cluster de machine pour les 4 dernières heures)
  - ▶  $\approx 7$  sec. pour  $10^6$  lignes (ex: toutes les métriques des wattmetres à haute fréquence pour la dernière heure)
  - ▶  $\approx 7$  sec. pour  $10^5$  lignes qui ne sont pas toutes en RAM (ex: toutes les métriques d'une machine depuis 1 jour)
- Quelques bugs. . .



# Premier retour d'expérience

- Déployé sur un serveur
- Données collectées
  - ▶ 30 valeurs
  - ▶ 1000 valeurs
  - ▶ 1Go de données
- Requêtes :
  - ▶ ≈ 1 sec. par heure
  - ▶ ≈ 7 sec. par dernière heure
  - ▶ ≈ 7 sec. par machine de données
- Quelques bugs

The screenshot shows a GitHub issue page for the repository 'timescale / timescaledb'. The issue title is 'Planner does not enforce join condition on hypertable with compression #1558'. It was opened by SimonDelamare 5 days ago. The issue description includes system information (OS: Debian 10, PostgreSQL version: 11.5, TimescaleDB version: 1.5.1) and a reproduction of the bug. A table of the 'metrics' table is shown with columns: timestamp, source, metric, and value. The table has a primary key on 'timestamp' and indexes on 'source' and 'timestamp'. The bug description states that a JOIN with several conditions between an hypertable with compression and another table selects rows that do not meet some of the conditions.

Relevant system information:

- OS: Debian 10
- PostgreSQL version : postgres (PostgreSQL) 11.5 (Debian 11.5-1+deb10u1)
- TimescaleDB version : 1.5.1
- Installation method: apt

Describe the bug

A JOIN with several conditions between an hypertable with compression and another table selects rows that do not meet some of the conditions.

It seems that the planner is wrong on conditions to apply on compressed chunks.

How I reproduce on my system

I have a "metrics" table:

```
\d+ metrics
+-----+-----+-----+-----+-----+
| Column | Type                | Modifiers          | Storage | St |
+-----+-----+-----+-----+-----+
| timestamp | timestamp with time zone | not null default now() | plain   | <n |
| source    | text                 | not null           | extended | <n |
| metric    | text                 | not null           | extended | <n |
| value     | double precision     | default 'NaN'::double precision | plain   | <n |
+-----+-----+-----+-----+-----+
Indexes:
    "metrics_pkey" PRIMARY KEY, btree ("timestamp", source, metric)
    "metrics_source_idx" btree (source, "timestamp" DESC)
    "metrics_timestamp_idx" btree ("timestamp" DESC)
Triggers:
    ts_insert_blocker BEFORE INSERT ON metrics FOR EACH ROW EXECUTE PROCEDURE _timescaledb
```

s à 1Hz)

e pour les 4 dernières

de fréquence pour la

métriques d'une



# Conclusion