Statistiques du centre de données Résif-DC

Jonathan Schaeffer, Philippe Bollard, Jérôme Touvier

27 Mai 2021

https://www.resif.fr https://seismology.resif.fr datacentre@resif.fr

Contexte et besoins

Le centre de données sismologiques Résif-DC

- concentre toutes les données sismologiques des expériences françaises (plusieurs SNO, 6 OSU producteurs de données)
- 70 réseaux sismologiques (10 permanents depuis 1986, 60 temporaires)
- 80To de données sous forme de séries temporelles
- distribution des données par webservices normalisés et protocole temps réel
- intégration dans EPOS
- conforme aux principes FAIR

Démarche qualité et indicateurs

Besoins :

- En interne : bien connaître l'utilisation pour dimensionner le centre de données
- Le pilotage : disposer d'outils d'évaluation
- *Principal Investigators* : statistiques pour les rapports annuels

Démarche Top-Down du pilotage européen (EIDA Management Board)

- Liste d'indicateurs (volume de données hébergée, volume distribué, décompté des requêtes, ...)
- Matrice des besoins
  - qui en a besoin ? (opérateurs, pilotage du datacentre, PI, pilotage européen)
  - à quel point ?

Difficultés à surmonter

Physionomie d'une donnée sismologique

Identification de la données : `FR.ILLK.00.HNZ`

Network le réseau sismologique, sur 2 caractères (ex. FR)

Station la station sismologique, sur 5 caractères max (ex. ILLK)

Location la localisation des instruments dans la station, sur 2 chiffres (ex. 00)

Channel le canal de prise de données (direction, bande passante, unité de mesure, ...) sur 3 caractères (ex. HNZ)

Toutes les statistiques doivent permettre de faire des selections sur ces identifiants. Il y en a 20000 au centre de données.

Volumes de requêtes et difficultés techniques

- 500000 requêtes par jour
- grande finesse des critères de sélection dans la requêtes (jokers, selection géographique, temporelle, etc.)
- requêtes HTTP GET ou POST => difficile de connaître la requête à partir de frontend http
- plusieurs formats de sortie possibles
- réponses streamées à l'utilisateur
- ne pas pénaliser les temps de réponse

Choix d'implémentation

Schéma d'architecture

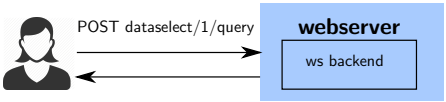
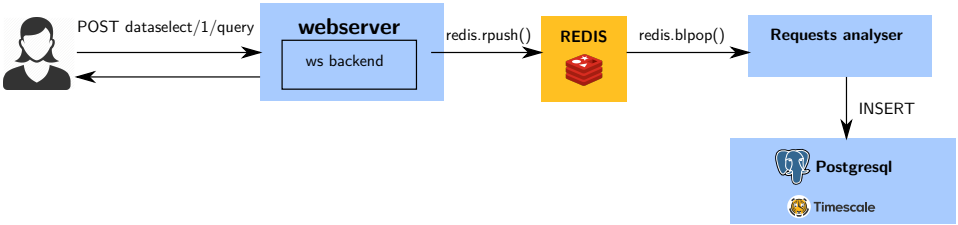


Schéma d'architecture



Aggrégation

Motivation

- Trafic important (2 enregistrements/s)
- Répondre aux requêtes rapidement
- Garder une base de données réduite

- il faut agréger sur une période de temps plus longue
- conserver les événements sur un temps court

Aggrégation

Motivation

- Trafic important (2 enregistrements/s)
- Répondre aux requêtes rapidement
- Garder une base de données réduite

- il faut agréger sur une période de temps plus longue
- conserver les événements sur un temps court

Difficulté : count-distinct problem

Si on connait le nombre de client unique sur la période d'aggrégation (pour chaque mois par exemple), le nombre de client unique sur l'année ne peut en être déduit ! (Lien Wikipedia)
⇒ Obstacle majeur à l'aggrégation

Aggrégation et cardinalité : HyperLogLog

HyperLogLog est un algorithme d'estimation de la cardinalité d'un ensemble.

Implémentation en Java, Python et dans PostgreSQL : On n'a plus rien à faire ! 👌

```
CREATE EXTENSION hll;
CREATE TABLE sent_data_summary_weekly(
...
clients hll(11,5),
...)
```

On n'a plus qu'à remplacer les `count(distinct())` par des `hll_cardinality()`

Aggrégation et cardinalité : HyperLogLog

```
INSERT INTO sent_data_summary_weekly(date, network, station, location,
↪  channel, country, clients, requests_set, bytes, protocol)
  SELECT time_bucket('1 week', date) as date, network, station,
   ↪  location, channel, country,
  hll_add_agg(hll_hash_bigint(userid)),
   ↪  hll_add_agg(hll_hash_text(requestid)), sum(bytes) as bytes,
   ↪  'dataselect' as protocol
  FROM dataselectvol WHERE date BETWEEN date_trunc('week', now() -
   ↪  interval '1 week') AND date_trunc('week', now())  GROUP BY 1, 2,
   ↪  3, 4, 5, 6;
```

Inconvénient : Les fonction postgresql de hyperloglog ne sont pas parallélisables (attente de la prochaine release). TimescaleDB ne peut pas les utiliser dans son système d'aggrégation continue, donc il faut aggréger à la main. Peut-être déjà résolu dans PostgreSQL13 ...

Aggrégation et cardinalité : HyperLogLog

Démonstration pour 1.7 millions d'entrées :

```
SELECT hll_cardinality(hll_union_agg(clients)) FROM
 ↪  sent_data_summary_weekly WHERE protocol='dataselect';
  hll_cardinality
-------------------
 7849.312471746741
(1 ligne)

Durée : 1255,696 ms (00:01,256)
```

Mise en base des événements

- Table SQL simple pour la mise en base.
- Beacoup d'insertions (2/s)
- Beaucoup de suppression (période de rétention de 4 mois)

date	timestamp without time zone
requestid	character varying(8)
useragent	text
clientip	character varying(15)
geohash	character varying(12)
network	character varying(8)
station	character varying(6)
location	character varying(2)
channel	character varying(3)
country	character varying(2)
bytes	bigint
nbfiles	bigint
userid	bigint

Mise en base des événements

- Table SQL simple pour la mise en base.
- Beacoup d'insertions (2/s)
- Beaucoup de suppression (période de rétention de 4 mois)

TimescaleDB

- extension PostgreSQL
- optimisation de performance pour séries temporelles
- aggrégation dynamique
- gestion des périodes de rétentions

date	timestamp without time zone
requestid	character varying(8)
useragent	text
clientip	character varying(15)
geohash	character varying(12)
network	character varying(8)
station	character varying(6)
location	character varying(2)
channel	character varying(3)
country	character varying(2)
bytes	bigint
nbfiles	bigint
userid	bigint

Aggrégation

Table Initiale :

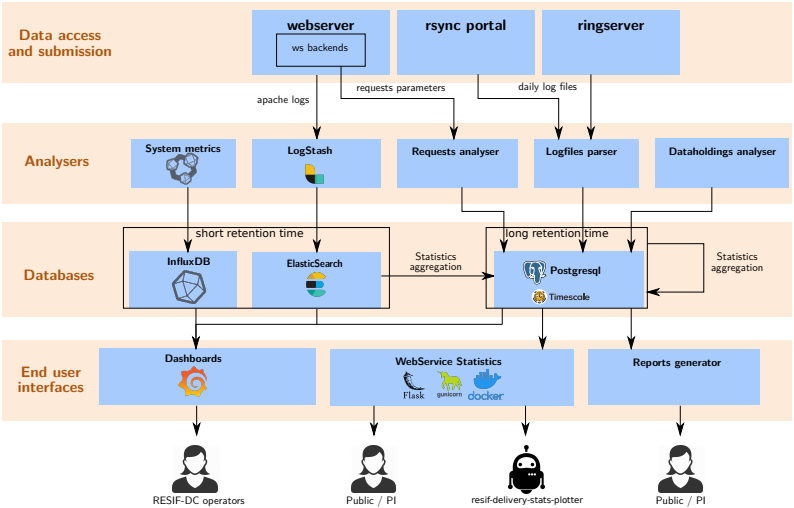
date	Grouper par semaine
useragent	Poubelle
clientip	Poubelle
geohash	Poubelle
nbfiles	Poubelle
network	Grouper
station	Grouper
location	Grouper
channel	Grouper
country	Grouper
requestid	Sommer
bytes	Sommer
userid	Hyperloglog
requestid	Hyperloglog

Table aggrégée :

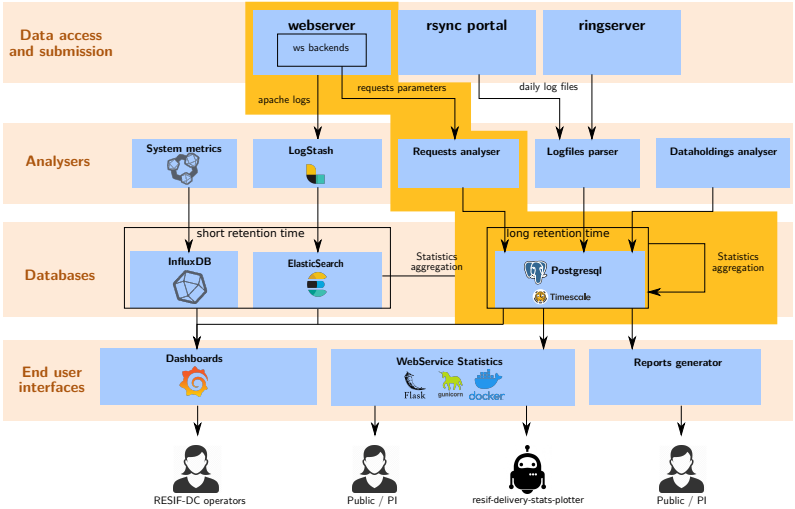
date	date par semaine
network	character varying(8)
station	character varying(6)
location	character varying(2)
channel	character varying(3)
country	character varying(2)
requests	integer
bytes	bigint
clients	hll
$\text{requests}_{\text{set}}$	hll

Schéma complet

Schéma


Data access and submission
webserver
ws backends
rsync portal
ringserver
apache logs
requests parameters
daily log files
Analysers
System metrics
LogStash
Requests analyser
Logfiles parser
Dataholdings analyser
short retention time
long retention time
Databases
InfluxDB
ElasticSearch
Statistics aggregation
Postgresql
Timescale
Statistics aggregation
End user interfaces
Dashboards
WebService Statistics
Flask
gunicorn
docker
Reports generator
RESIF-DC operators
Public / PI
resif-delivery-stats-plotter
Public / PI

Schéma


Data access and submission
webserver
ws backends
rsync portal
ringserver
apache logs
requests parameters
daily log files
Analysers
System metrics
LogStash
Requests analyser
Logfiles parser
Dataholdings analyser
short retention time
long retention time
Databases
InfluxDB
ElasticSearch
Statistics aggregation
Postgresql
Timescale
Statistics aggregation
End user interfaces
Dashboards
WebService Statistics
Flask
gunicorn
docker
Reports generator
RESIF-DC operators
Public / PI
resif-delivery-stats-plotter
Public / PI

Utilisations et perspectives

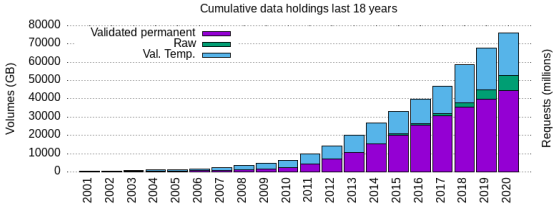
Webservice statistiques

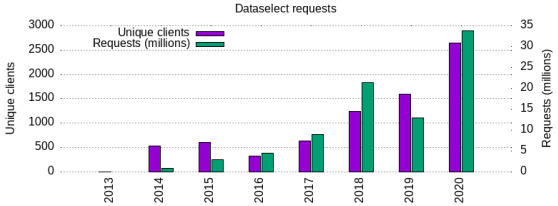
Un webservice exposant publiquement toutes les statistiques dont on dispose en base de données :
https://ws.resif.fr/resifws/statistics/1/

Sources : https://gricad-gitlab.univ-grenoble-alpes.fr/OSUG/RESIF/ws-statistics

Rapport annuel automatisé

- orgmode + babel
  - requêtes SQL
  - gnuplot
  - quelques autres outils (curl)
- gitlab pages pour publication automatique
  Consulter le rapport

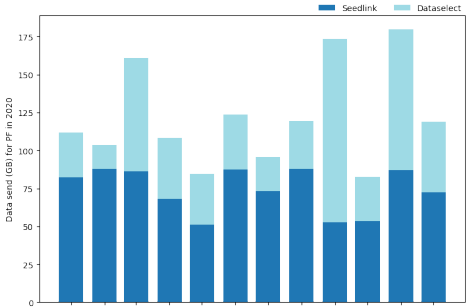




Rapport d'utilisation pour les responsables de réseaux

- programme CLI en python
- génération de rapport annuel pour un réseau
- utilisation d'informations publiques uniquement

Generated by Résif on 2021-05-20 at 15h00 UTC



Generated by Résif on 2021-05-20 at 15h00 UTC

Aggrégation de statistiques européennes

Problématiques identiques :

- aggregation
- count-distinct problem
- différentes implémentation du webservice dataselect

Réalisation :

- d'une base de données de stats des centres de données européens,
- d'un outil de création d'aggrégation pour tous les centres de données
- d'un webservice d'ingestion des aggregats centralisé
- d'une base de donnée centralisée
  https://github.com/EIDA/eida-statistics

Conclusion

Conclusion

- Mettre en œuvre des metriques est essentiel.
- Demande un important travail de conception.
- Projet important car visible par le pilotage.
- Un message à retenir : PostgreSQL rox !

Annexes

Liens vers les projets

- web service statistics (sources)
- fdsnwsstats
- déploiement kubernetes de fdsnws-station et fdsnws-dataseelect
- resif-delivery-stats-plotter
- rapport annuel automatisé 2020 (sources)