Verarbeitung von Logdateien mit PostgreSQL

am Beispiel von Apache Logdateien

Wer bin ich?

Danilo Endesfelder

- Technical Consultat/Projektsupport
- Plusserver GmbH

- Mail: daniloe@posteo.de

- Xing: …/Danilo_Endesfelder

Um was geht es heute?

- Warum
- Viele (native) Wege führen nach Rom
- Des Autors Lösung
- Json vs. JsonB
- Das hätte ich gern vorher gewusst
- Fragen

Warum

- Weil es geht
- Backup/Recovery der Funktionen im DB-Backup
- Einfache Erweiterung der Funktionalität innerhalb der DB
- Sicherheit durch Zugriffsschutz auf den Code
- Keine Zugänge im normalen Dateisystem

Viele (native) Wege führen nach Rom

Ausgangspunkt:

Logdatei: 1 Mio Zeilen, 308 MB

PostgreSQL 9.5 beta2

CPU: Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz

Vorbereitungen

- Datenbank und Tabelle anlegen:

```
# CREATE DATABASE ptest;
# \c ptest
# CREATE TABLE logdata (
    pk_data_id bigint NOT NULL,
    domain text,
    remip cidr,
    remtime timestamp without time zone,
    request text,
    status smallint,
    size integer,
    referer text,
    agent text
);
# CREATE TABLE temp (
    lines text
);
```

Import der Daten via Copy/Trigger

- Triggerfunktion: Splitte Datensatz mittels regex und gib Array zurück
- Import via einfachem \copy
- Ergebnis: Nach 30 Minuten abgebrochen, nur 57MB importiert
- Mission Failed...

Import der Daten via Funktion

```
CREATE OR REPLACE FUNCTION split_data()
RETURNS setof void
AS $$
    DECLARE m text[];
    DECLARE line text;
BEGIN
    FOR line IN
        SELECT * from temp
    LOOP
        m := regexp_matches(line, E'(\\S+) (\\S+) (.+) (.+) \\[(.+)\\] "(.+)" (\\d+) (.+) "(.*)" "(.*)"
"(.*)"');
        IF m[7] = '-' THEN
            m[7] := 0;
        END IF;
        IF m[8] = '-' THEN
            m[8] := 0;
        END IF;
        INSERT INTO logdata (domain,remip,remtime,request,status,size,referer,agent)
            VALUES (m[1],m[2]::cidr,m[5]::timestamp,m[6],m[7]::integer,m[8]::integer,m[9],m[10]);
    END LOOP;
    RETURN;
END;
$$
LANGUAGE plpgsql;
```

Import der Daten via Funktion

Ergebnis:

```
# \copy temp (lines) from
/var/lib/pgsql/9.5/data/appopy_import/appopy_import.log;
   COPY 1000000
   Time: 5803.049 ms


# select split_data();
   Time: 1707694.331 ms (~28,5 Min),
   Tabellengröße: 227MB
```

Import der Daten via jsonb-Funktion

```
CREATE OR REPLACE FUNCTION split_data_json()
RETURNS setof void
AS $$
    DECLARE m text[];
    DECLARE ar text[];
    DECLARE line text;
BEGIN
    FOR line IN
        SELECT * from temp
    LOOP
        m := regexp_matches(line, E'(\\S+) (\\S+) (.+) (.+) \\[(.+)\\] "(.+)" (\\d+) (.+) "(.*)"
"(.*)" "(.*)"');
        IF m[7] = '-' THEN
            m[7] := 0;
        END IF;
        IF m[8] = '-' THEN
            m[8] := 0;
        END IF;
        INSERT INTO logdata_jsonb (content) VALUES
( jsonb_object('{domain,remip,remtime,request,status,size,referer,agent}',
                ar || m[1] || m[2] || m[5] || m[6] ||m[7] || m[8] || m[9] || m[10]));
    END LOOP;
    RETURN;
END;
$$
LANGUAGE plpgsql;
```

Import der Daten via jsonb-Funktion

Ergebnis:

```
# \copy temp (lines) from
/var/lib/pgsql/9.5/data/appopy_import/appopy_import.log;
   COPY 1000000
   Time: 5803.049 ms


# select split_data_json();
   Time: 1660717.533 ms (~27,5 Min)
   Tabellengröße: 353MB
```

Import via Subselect

```
# INSERT INTO logdata
(domain,remip,remtime,request,status,size,referer,agent) SELECT
    m[1],m[2]::cidr,m[5]::timestamp,m[6],
    case when m[7] = '-' then 0
        else m[7]::integer end,
    case when m[8] = '-' then 0
        else m[8]::integer end,
    m[9],m[10]
from (
    select regexp_matches(lines, E'(\\S+) (\\S+) (.+) (.+) \\[(.
+)\\] "(.+)" (\\d+) (.+) "(.*)" "(.*)" "(.*)"') as m
    from temp) as foo;
```

Import via Subselect

Ergebnis:

```
# \copy temp (lines) from
/var/lib/pgsql/9.5/data/appopy_import/appopy_import.log;
```

COPY 1000000

Time: 5803.049 ms

```
# INSERT INTO...
```

Time: 1481560.557 ms (~25 Min),

Tabellengröße: 227MB

Warum dauert das so lange?

- Regex in Postgres anscheinend etwas träge
- Python regex-Funktion:

```
CREATE OR REPLACE FUNCTION pyreg(pattern text,string text)
RETURNS varchar[]
AS $$
    import re
    regray=re.match(pattern,string)
    if regray:
        reglist=regray.groups()
    else:
        reglist=''
    return reglist
$$ LANGUAGE plpythonu;
```

- Insert via Subselect: ~2 Min

Des Autors Lösung

- PL/Python
- schneller,flexibler,mehr Funktionalität (in weniger Code)
  - Inheritance (inkl. Erstellung der Tabellen)
  - GeoIP-Auswertung
  - Import gepackter Logfiles

- Voraussetzungen:
  - Python > 2.7
  - GeoIP inkl. python-geoip
  - Extension plpythonu in Datenbank aktiviert

Des Autors Lösung

- eine große Funktion
  - Aggregierung der Daten im json-Format zur Funktionslaufzeit
  - Anpassung des Verhaltens via Configtabelle
  - Prüfung ob gepackte Datei und entsprechendes Verhalten

Des Autors Lösung

- Tabellen und Definitionen

```
# CREATE TABLE logdata (
    pk_data_id      BIGSERIAL PRIMARY KEY,
    domain             TEXT ,
    remip              cidr,
    remtime            TIMESTAMP,
    request            TEXT,
    status             SMALLINT,
    size               INT,
    referer            TEXT,
    agent              TEXT
    );

# CREATE TABLE stat_daily (
    daily_data      JSONB
    );

# CREATE TABLE config (
    configname     TEXT,
    configvalue        TEXT
    );
```

Des Autors Lösung

- Konfigurationstabelle
  - Pfad zu den Logdateien
  - Pfad zur GeoIP-Datenbank
  - Könnte beliebig erweitert werden, z.B. um den bzw. verschiedene Regex

Des Autors Lösung

- Statistik/Aggregierung via Python-Klasse inkl. json-return-funktion

```
# class for daily domainstatistic

    class daily_domain:

        def __init__(self):

            self.hostname = ""

            self.date_time = ""

            self.count = 0

            self.traffic = 0

            self.land = {}


    def get_json(self):

            dd_dict={}

            dd_dict['host'] = self.hostname

            dd_dict['date'] = self.date_time

            dd_dict['count'] = self.count

            dd_dict['traffic'] = self.traffic

            for ll in self.land.keys():

                dd_dict[ll] = self.land[ll]

            return dumps(dd_dict)
```

Des Autors Lösung

- Parsen der Logzeilen via regex

```
parts = [
    r'(?P<host>\S+)',           # host %V
    r'(?P<rip>\S+)',            # Remote IP %a
    r'\S+',                     # indent %l (unused)
    r'(?P<user>.+)',            # user %u (unused)
    r'\[(?P<time>.+)\]',        # time %t
    r'"(?P<request>.+)"',       # request "%r"
    r'(?P<status>([0-9]+|-))',  # status %>s
    r'(?P<size>\S+)',           # size %O (careful, can be '-')
    r'"(?P<referer>.*)"',       # referer "%{Referer}i"
    r'"(?P<agent>.*)"',         # user agent "%{User-agent}i"
    r'".*"'                     # Cookie
]
```

Des Autors Lösung

- weitere Funktionalitäten
  - Überspringen der Zeile wenn diese nicht auf den regex passt
  - Kindtabelle mit Logdaten für jede Domain wird erstellt
  - Dictionary mit Einträgen der vorhandenen Kindtabellen („Cache“)
  - Pflege der täglichen Statistik in einem Dictionary aus Statistikklassen
  - Statistiken über Traffic, Zugreifende Länder nach Domain und Tag

```
{"DE": 1, "date": "2014-11-05", "host": "postgres.rocks", "count": 1,
"traffic": 607680}
```

  - Eintragen der Statistikdaten im jsonb-Format nach Ende der Logdatei
  - Löschen der abgeschlossenen Logfiles

Des Autors Lösung

Ergebnis:

```
# select appopy_import();
```

Time: 328808.642 ms (~6 Minuten),

DB Groesse: 235MB

```
# \d+
                                List of relations
 Schema |          Name           |   Type   |  Owner   |    Size    | Description
--------+-------------------------+----------+----------+------------+-------------
 public | config                  | table    | appopy | 16 kB      |
 public | d_42_info               | table    | appopy | 44 MB      |
 public | d_beispiel_com          | table    | appopy | 46 MB      |
 public | d_example_com           | table    | appopy | 46 MB      |
 public | d_foobar_biz            | table    | appopy | 46 MB      |
 public | d_postgres_rocks        | table    | appopy | 46 MB      |
 public | logdata                 | table    | appopy | 8192 bytes |
 public | logdata_pk_data_id_seq| sequence | appopy | 8192 bytes |
 public | stat_daily              | table    | appopy | 16 kB      |
```

json vs. jsonb

- json
  - 1zu1 Kopie des Eingangstextes
  - muss bei jeder Operation geparsed werden
  - behält Reihenfolge bei
  - doppelte Schlüssel möglich
- jsonb
  - bereits „zerlegte“ Daten, kein Parsing notwendig
  - insert etwas langsamer, dafür selects schneller
  - keine doppelten Schlüssel (last is kept)
  - keine feste Reihenfolge
  - unterstützt Indizierung

- Empfehlung Doku: Jsonb

Was ich gern vorher gewusst hätte

- Escape-Sequences und \copy sind böse, da direkt interpretiert, keine einfache Möglichkeit zum „entschärfen“
- Inheritance kann jede Menge Platz/Zeit benötigen
- Regex in nativem Postgres recht träge

Fragen?

Vielen Dank für die Aufmerksamkeit

E-Mail: daniloe@posteo.de

Github: https://github.com/DaniloE