xmpp.chapril.org - Demande #5997

Optimiser la table postgresql `ejabberd.archive`

22/10/2022 12:13 - Anonyme

Statut:	Nouveau	**Début:**	22/10/2022
Priorité:	Normale	**Echéance:**	
Assigné à:	pitchum .	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0.00 heure
Version cible:	Backlog		
Difficulté:	5 Difficile		

Description

Le problème

Cette table `ejabberd.archive` conserve l'historique de tous les messages (dans XMPP ça s'appelle MAM - Message Archive Management).
Elle est donc très volumineuse. Et elle est très souvent sollicitée, aussi bien en lecture qu'en écriture.
Des "SELECT" ou "SELECT COUNT" sur cette table durent souvent plus de 30 secondes, parfois plusieurs minutes. Ça doit probablement se ressentir côté client (mais je n'en suis pas certain, c'est peut-être habilement masqué dans les applis).

Une piste de solution

Une vraie solution nécessiterait de patcher le code ejabberd. Par exemple pour maintenant des compteurs de messages pour éviter d'avoir à faire des "SELECT COUNT" violents.
Mais comme je ne suis pas prêt à apprendre à code en erlang ni en elixir, je cherche ce que je peux faire côté postgresql.

Une amélioration possible serait de "saucissonner" la table (ça s'appelle Table Partitioning dans la doc postgresql).
Ce mécanisme se prête à priori très bien à cette table pour laquelle les écritures sont de type "append only" et qui dispose d'un champ d'horodatage.

L'idée serait de regrouper les messages par trimestres par exemple. Ça devrait rendre beaucoup plus rapides certaines requêtes SELECT basés sur l'horodatage.

Début d'analyse

Avec le module pg_stat_statements activé depuis un bon moment maintenant, on peut avoir un aperçu des requêtes les plus fréquentes et leurs durées moyennes d'exécution.
Je filtre les requêtes qui correspondent aux backups et à la métrologie. Leurs durées d'exécution nous importent peu.

```
postgres=# SELECT
  substring(query, 1, 120) as short_query,
  ROUND(total_exec_time :: numeric, 2) AS total_time,
  calls,
  ROUND(mean_exec_time :: numeric, 2) AS mean,
  ROUND (
    ( 100 * total_exec_time / SUM (total_exec_time :: numeric) OVER () ) :: numeric,
    2
    ) AS percentage_overall
FROM pg_stat_statements
WHERE calls > 10
  AND query NOT LIKE 'COPY %'
  AND query NOT LIKE '% where peer not like %'
  AND query NOT LIKE 'alter database %'
ORDER BY total_time DESC
LIMIT 20;
                                                  short_query
                        | total_time  |  calls  |  mean  | percentage_overall
-----------------------------------------------------------------------------------------------------------
-------------------------+-------------+---------+--------+--------------------
 SELECT COUNT(*) FROM archive WHERE  username=$1 and server_host=$2
                        | 30119788.74 |  357780 |  84.19 |              60.09
```

```
 SELECT timestamp, xml, peer, kind, nick FROM (SELECT  timestamp, xml, peer, kind, nick FROM archi
ve WHERE username=$1 an |  5670916.86 |   96454 |  58.79 |              11.31
 INSERT INTO archive(username, server_host, timestamp, peer, bare_peer, xml, txt, kind, nick) VALU
ES ($1, $2, $3, $4, $5, |  2507067.71 |  669008 |   3.75 |               5.00
 SELECT  timestamp, xml, peer, kind, nick FROM archive WHERE username=$1 and server_host=$2 AND ti
mestamp > $3 ORDER BY t |  2436240.96 |  346850 |   7.02 |               4.86
 select feature from caps_features where node=$1 and subnode=$2
                         |   712322.72 |  216250 |   3.29 |               1.42
 insert into caps_features(node, subnode, feature) values ($1, $2, $3)
                         |   616188.86 |   34179 |  18.03 |               1.23
 INSERT INTO spool(username, server_host, xml) VALUES ($1, $2, $3)
                         |   589124.91 |   89893 |   6.55 |               1.18
 select name, val from pubsub_node_option where nodeid=$1
                         |   572234.72 | 6463986 |   0.09 |               1.14
 SELECT COUNT(*) FROM archive WHERE  username=$1 and server_host=$2 and bare_peer=$3 and timestamp
 >= $4 and timestamp <= |   565019.23 |  179901 |   3.14 |               1.13
 select password, serverkey, salt, iterationcount from users where username=$1 and server_host=$2
                         |   520670.40 |  121422 |   4.29 |               1.04
 select itemid, publisher, creation, modification, payload from pubsub_item where nodeid=$1
                         |   435599.69 | 2176425 |   0.20 |               0.87
 WITH upsert AS (UPDATE last SET seconds=$1, state=$2 WHERE username=$3 AND server_host=$4 RETURNI
NG *) INSERT INTO last( |   400578.15 |    1146 | 349.54 |               0.80
 select node, parent, plugin, nodeid from pubsub_node where host=$1
                         |   266939.35 |  538148 |   0.50 |               0.53
 SELECT COUNT(*) FROM archive WHERE  username=$1 and server_host=$2 and timestamp <= $3
                         |   261728.44 |    1068 | 245.06 |               0.52
 delete from spool where username=$1 and server_host=$2
                         |   254015.13 |  163593 |   1.55 |               0.51
 select node, plugin, i.nodeid, affiliation from pubsub_state i, pubsub_node n where i.nodeid = n.
nodeid and jid=$1 and h |   248160.76 |  203433 |   1.22 |               0.50
 select username, jid, nick, subscription, ask, askmessage, server, subscribe, type from rosteruse
rs where username=$1 an |   239060.98 |  191271 |   1.25 |               0.48
 WITH upsert AS (UPDATE pubsub_item SET publisher=$1, modification=$2, payload=$3 WHERE nodeid=$4
AND itemid=$5 RETURNING |   233871.77 |    1397 | 167.41 |               0.47
 SELECT timestamp, xml, peer, kind, nick FROM (SELECT  timestamp, xml, peer, kind, nick FROM archi
ve WHERE username=$1 an |   228362.38 |    4484 |  50.93 |               0.46
 select itemid, publisher, creation, modification, payload from pubsub_item where nodeid=$1 order
by creation asc         |   221947.50 |  179670 |   1.24 |               0.44
(20 lignes)
```

Les cas d'usage typiques que je pense avoir identifiés :

- une appli se déconnecte/reconnecte, même brièvement, elle vérifie immédiatement si son historique de discussion est à jour => "SELECT COUNT FROM archive"
- pareil, mais en pire, si un utilisateur configure son compte XMPP déjà existant sur un nouvel appareil, celui-ci va potentiellement chercher à récupérer tout l'historique de ce compte
- un utilisateur rallume un appareil inactif depuis longtemps, l'appli va interroger MAM pour récupérer tous les messages qui ont été archivés depuis tout ce temps : "SELECT .... FROM archive WHERE timestamp >= ...."
- un utilisateur reçoit ou envoie un message => "INSERT INTO archive ..."
- un utilisateur rejoint un salon existant => "SELECT COUNT ... FROM archive WHERE "
- certaines applis ne récupère l'historique que au fil de l'eau, lorsque l'utilisateur scrolle vers le haut : "SELECT .... FROM archive WHERE timestamp >= .... AND timestamp <= ..."

Plan d'action

Yapuka ;)