Quassel IRC - Bug #680

slow sql queries with postgres backend and prepared statements

04/22/2009 01:53 PM - honk

Status:	Feedback	**Start date:**	04/22/2009
Priority:	Normal	**Due date:**	
Assignee:	EgS	**% Done:**	0%
Category:	Quassel Core	**Estimated time:**	0.00 hour
Target version:			
Version:	0.5.0	**OS:**	Linux

Description

When prepared statements with postgresql those queries will be optimized at creation time. This can lead to suboptimal execution patterns. In this particular case fetching the backlog takes ~500ms per bufferid which leads to a timeout when connecting with a client to the core.

Reproducible: unknown (always on my system)

Steps to Reproduce:
1. connect to database with pqsl
2.

    PREPARE quassel_3 AS SELECT messageid, time, type, flags, sender, message FROM backlog JOIN sender ON backlog.senderid = sender.senderid WHERE bufferid = $1 ORDER BY messageid DESC LIMIT $2;

3.

    EXPLAIN ANALYZE EXECUTE quassel_3(13,1000);

Actual Results:

```
Limit  (cost=0.00..2953.41 rows=1244 width=100) (actual time=52.738..481.666 rows=656 loops=1)
  ->  Nested Loop  (cost=0.00..29522.22 rows=12435 width=100) (actual time=52.731..480.869 rows=65
6 loops=1)
        ->  Index Scan Backward using backlog_pkey on backlog  (cost=0.00..24116.20 rows=12435 wid
th=59) (actual time=52.601..473.359 rows=656 loops=1)
              Filter: (bufferid = $1)
        ->  Index Scan using sender_pkey on sender  (cost=0.00..0.42 rows=1 width=49) (actual time
=0.006..0.008 rows=1 loops=656)
              Index Cond: (sender.senderid = backlog.senderid)
Total runtime: 497.364 ms
(7 rows)
```

Expected Results:

```
Limit  (cost=0.00..3659.73 rows=1000 width=100) (actual time=8.370..27.696 rows=656 loops=1)
  ->  Nested Loop  (cost=0.00..18020.51 rows=4924 width=100) (actual time=8.366..26.780 rows=656 l
oops=1)
        ->  Index Scan using backlog_bufferid_idx on backlog  (cost=0.00..14749.11 rows=4924 width
=59) (actual time=8.327..17.198 rows=656 loops=1)
              Index Cond: (bufferid = 13)
        ->  Index Scan using sender_pkey on sender  (cost=0.00..0.65 rows=1 width=49) (actual time
=0.008..0.010 rows=1 loops=656)
              Index Cond: (sender.senderid = backlog.senderid)
Total runtime: 28.243 ms
(7 rows)
```

Workaround:
1. Create a function that dynamically calls the SQL query in order to force query optimization at runtime:

```
CREATE TYPE select_messages_result AS (messageid integer, time timestamp without time zone, type i
nteger, flags integer, sender varchar(128), message text);
CREATE OR REPLACE FUNCTION select_messages(integer, integer)
   RETURNS SETOF select_messages_result
   STABLE
   AS $$
      DECLARE result select_messages_result;
      BEGIN
       FOR result IN EXECUTE 'SELECT messageid, time,  type, flags, sender, message
        FROM backlog
        JOIN sender ON backlog.senderid = sender.senderid
        WHERE bufferid = ' || quote_literal($1) || '
        ORDER BY messageid DESC
        LIMIT ' || quote_literal($2)
       LOOP
        RETURN NEXT result;
       END LOOP;
      END;
   $$ LANGUAGE plpgsql;
```

2. Call function from the prepared statement

    SELECT * from select_messages($1, $2)

Related files:

    src/core/SQL/PostgreSQL/14/select_messages.sql

History

#1 - 05/22/2009 01:12 AM - EgS

- Status changed from New to Rejected

This is rather a bug in Postgres than in Quassel. The error is, that Postgres doesn't pick up the proper index which is - for such a simple query - pretty surprising.

The benefit of prepared queries is exactly to avoid query planing for the same query over and over again. The proposed solution is to discard the benefits of prepared queries by introducing overhead via a new pgsql function...

If you want to force query planing for each executed query, the solution would be to simply waive prepared queries and go for regular - non prepared - queries.

#2 - 06/27/2009 05:58 PM - EgS

- Status changed from Rejected to Resolved

Seems to be fixed in git:

http://bugs.quassel-irc.org/repositories/revision/1/4bb04fc5758f566e58d560d6f8c510c5cb038486

#3 - 07/14/2009 02:58 PM - m4yer

Postgresql-8.4 seems to bring huge advantages (in connection with the latest patches).

Maybe a improvment in query planing (or so^^)

#4 - 07/29/2009 03:12 PM - sph

PostgreSQL 8.4 fixes this bug indeed.

#5 - 09/25/2009 12:04 PM - teo

- Status changed from Resolved to Feedback

This doesn't seem to be fixed yet, in Postgres 8.4.1, as confirmed by al and myself:

```
psql (8.4.1)

Type "help" for help.

quassel=> PREPARE quassel_3 AS SELECT messageid, time, type, flags, sender, message FROM backlog JOIN sender O
N backlog.senderid = sender.senderid WHERE bufferid = $1 ORDER BY messageid DESC LIMIT $2;
PREPARE

quassel=> EXPLAIN ANALYZE EXECUTE quassel_3(13,1000);

                                                                             QUERY PLAN

--------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Limit  (cost=0.00..73077.69 rows=7300 width=100) (actual time=10354.128..10354.128 rows=0 loops=1)

   ->  Nested Loop  (cost=0.00..730826.97 rows=73005 width=100) (actual time=10354.123..10354.123 rows=0 loops
=1)
         ->  Index Scan Backward using backlog_pkey on backlog  (cost=0.00..244924.08 rows=73005 width=60) (ac
tual time=10354.118..10354.118 rows=0 loops=1)
               Filter: (bufferid = $1)

         ->  Index Scan using sender_pkey on sender  (cost=0.00..6.64 rows=1 width=48) (never executed)

               Index Cond: (sender.senderid = backlog.senderid)

 Total runtime: 10354.341 ms

(7 rows)
```

#6 - 01/11/2010 03:31 PM - APTX

I would suggest lowering random_page_cost described:
http://www.postgresql.org/docs/8.4/static/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS
Some more info about how and why changing this constant can help: http://archives.postgresql.org/pgsql-performance/2004-02/msg00274.php

I also see no reason why you shouldn't prefer to do index scans on a database like quassels as quassel only has very large or very small tables, so the slowdown in using an index on a small table would be negligible compared to the speedup I got in that one query (from 4.5 seconds to under 1 ms)

#7 - 05/08/2010 10:18 AM - amiconn

This problem is still reproducable with PostgreSQL 8.4.3, both on Debian x86 and x86_64. Lowering random_page_cost to 2.0 as suggested above does help (the default is 4.0). Just set this as a user variable for the database user 'quassel' (or whatever you called it):

ALTER ROLE quassel SET random_page_cost='2';

#8 - 06/18/2010 04:43 PM - sdancer

With 8.4.4, I get:

```
quassel=# EXPLAIN ANALYZE EXECUTE quassel_3(1,10);
                                                                               QUERY PLAN

--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
 Limit  (cost=83250.25..83285.42 rows=14068 width=103) (actual time=23129.867..23129.893 rows=10 loops=1)
   ->  Sort  (cost=83250.25..83601.96 rows=140683 width=103) (actual time=23129.863..23129.872 rows=10 loops=1
)
         Sort Key: backlog.messageid
         Sort Method:  top-N heapsort  Memory: 19kB
         ->  Hash Left Join  (cost=6385.04..55827.38 rows=140683 width=103) (actual time=1736.794..18053.591 r
ows=2551568 loops=1)
               Hash Cond: (backlog.senderid = sender.senderid)
               ->  Bitmap Heap Scan on backlog  (cost=2646.93..45593.47 rows=140683 width=61) (actual time=148
9.377..7584.482 rows=2551568 loops=1)
                     Recheck Cond: (bufferid = $1)
                     ->  Bitmap Index Scan on backlog_bufferid_idx  (cost=0.00..2611.76 rows=140683 width=0) (
actual time=1470.950..1470.950 rows=2551568 loops=1)
                           Index Cond: (bufferid = $1)
               ->  Hash  (cost=1781.27..1781.27 rows=89827 width=50) (actual time=247.141..247.141 rows=88872 
loops=1)
                     ->  Seq Scan on sender  (cost=0.00..1781.27 rows=89827 width=50) (actual time=0.016..85.0
11 rows=88872 loops=1)
 Total runtime: 23131.169 ms
(13 rows)
```

That's on a very limited virtual host.

#9 - 08/20/2010 08:49 AM - amiconn

The bug suddenly came back even at random_page_cost=2 (PostgreSQL 8.4.4 on Debian x86_64, quasselcore 0.7-beta1+11 git-f16f4c5). Lowering random_page_cost further (random_page_cost=1) fixed it for now.

This really needs some in-depth analysis, given that other users see even different slow query plans.

#10 - 09/26/2010 01:03 PM - amiconn

The bug still exists with default settings and PostgreSQL 9.0.0 (tested on Debian 32 bit, quasselcore 0.8-pre+19 git-fdec4a8). Lowering random_page_cost helps as usual.

#11 - 02/18/2013 10:37 AM - Tobu

I'm seeing similar symptoms to either this or #754 (can't fetch initial backlog, wrong query plan) with Quassel 0.8.0 and PostgreSQL 9.1. Setting random_page_cost to 2 helps with the first comment's query plan (deallocate/prepare uses the correct index), but setting random_page_cost to 2 or 1 for the Quassel user with

```
ALTER ROLE quassel SET random_page_cost='1';
\drds
sudo service quasselcore restart
```

doesn't make the initial backlog fetch work (the client disconnects after a while).

#12 - 04/22/2013 11:36 PM - Tobu

Apparently prepared-statements use generic query plans, and they are known to cause poor performance.
http://postgresql.1045698.n5.nabble.com/Prepared-statements-performance-tp5699514p5699982.html

Some of the language APIs apparently have ways to make prepared statements work, for example the JDBC driver will only define a prepared statement after a query (with bind params) has been called five times, which AIUI increases chances that a decent plan is cached. PostgreSQL 9.2 will recompute the plans of prepared statements after a while.

Anyway, it doesn't seem prepared statements are useful here. They are only recommended for very frequent small queries. Securing against injection is a separate problem which is solved by bind params in the db api.

#13 - 04/22/2013 11:38 PM - Tobu

By the way, setting random_page_cost had no measurable effect for me on 9.0 and 9.1. Even with random_page_cost='0.5', connecting a client still killed the core.

#14 - 06/24/2013 01:46 PM - al

- Version changed from 0.5-pre to 0.3.0.x

#15 - 06/24/2013 01:48 PM - al

- Version changed from 0.3.0.x to 0.5.0

Maybe the correlation stats for backlog.messageid or .bufferid are misleading the planner.
On my database the values are:

```
quassel=> SELECT attname, inherited, n_distinct, correlation FROM pg_stats
WHERE (tablename = 'backlog' and attname in ('messageid', 'senderid', 'bufferid'))
   OR (tablename = 'sender' and attname in ('senderid'));
```

```
  attname  | inherited | n_distinct | correlation
-----------+-----------+------------+-------------
 messageid | f         |         -1 |    0.444176
 bufferid  | f         |        157 |    0.258886
 senderid  | f         |      37605 |    0.384319
 senderid  | f         |         -1 |           1
```

#16 - 06/24/2013 01:51 PM - Tobu

What PostgreSQL version? But I doubt there's anything wrong with those stats.

Quassel should just print a big fat warning when installing on PostgreSQL < 9.2. Restoring decent performance on earlier versions would require ditching prepared statements.