Neo4j 2.2 Cypher - Poor query performance with multiple OPTIONAL MATCH statements

New issue

Open **bunkat** opened this issue Apr 9, 2015 · 9 comments

bunkat commented Apr 9, 2015

I'm running into an issue with quickly degrading performance on queries that use multiple OPTIONAL MATCH statements, especially with the COST planner.

If the query only has a single OPTIONAL MATCH statement, the query planner creates an appropriate `OptionalExpand(All)` query term. However, when multiple OPTIONAL MATCH statements are used the query planner starts creating `NodeByLabelScan` query terms with large parallel branches that then get merged. I would have expected that the multiple OPTIONAL MATCH statement query would have looked more similar to the single OPTIONAL MATCH query with just multiple `OptionalExpand(All)` terms.

I believe that this issue exists in previous versions of Neo4j as well, but I'm only noticing it now as my database grows in size (30k nodes and 100k relationships, not even that large). Queries that used to take 100ms are now taking 2+ minutes to complete.

The following query concerns 670 contacts that are associated with approximately 1800 phone, address, tag, and invite nodes. I'm trying to create a query that will return the contacts along with the associated nodes if any exist.

Example 1: Query with a single OPTIONAL MATCH STATEMENT

```
PROFILE
MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
WHERE HAS(ur.teamid)
WITH uc, ur

MATCH (c:contact {teamid: uc.teamid})
WITH DISTINCT c, {accessedby: uc.id, accessedrole: ur.name} AS r

OPTIONAL MATCH (c)-[:TAG]->(t:tag)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby,
  isinvited: r.isinvited,
  tags: EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name})
  } AS r

RETURN c, r LIMIT 10
```

Result: Cypher version: CYPHER 2.2, planner: COST. 8364 total db hits in 70 ms.
Full plan available at http://i.imgur.com/QAPvO39.png.

Example 2: Query with a multiple OPTIONAL MATCH STATEMENTs

Labels

None yet

Milestone

No milestone

Assignee

No one assigned

5 participants

```
PROFILE
MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
WHERE HAS(ur.teamid)
WITH uc, ur

MATCH (c:contact {teamid: uc.teamid})
WITH DISTINCT c, {accessedby: uc.id, accessedrole: ur.name} AS r

OPTIONAL MATCH (c)-[:INVITE]->(i:invite)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby,
  isinvited: NOT i IS NULL
  } AS r

OPTIONAL MATCH (c)-[:PHONE]->(p:phone)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby,
  isinvited: r.isinvited,
  phones: EXTRACT(x IN COLLECT(p) | { id: x.id, object: x.object })
  } AS r

OPTIONAL MATCH (c)-[:ADDRESS]->(a:address)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby,
  isinvited: r.isinvited, phones: r.phones,
  addresses: EXTRACT(x IN COLLECT(a) | { id: x.id, object: x.object, street: x.street, l
      region: x.region, postcode: x.postcode, country: x.country, description: x.descrip
    })
  } AS r

OPTIONAL MATCH (c)-[:TAG]->(t:tag)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby,
  isinvited: r.isinvited, phones: r.phones, addresses: r.addresses,
  tags: EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name})
  } AS r

RETURN c, r LIMIT 10
```

Result: Cypher version: CYPHER 2.2, planner: RULE. 99988 total db hits in 731 ms.
Full expanded query plan available at http://i.imgur.com/dvvEt48.png.

Result: Cypher version: CYPHER 2.2, planner: COST. 5620191 total db hits in 4821 ms.
Full expanded query plan available at http://i.imgur.com/L0nAJIB.png.

jexp commented Apr 10, 2015 Collaborator

Try to use more descriptive variable/identifier names

I think the biggest problem is that your are creating cross products.

Make sure your have an index on `:contact(teamid)` so that can be used instead of a label scan.

Also you check for WHERE HAS(ur.teamid) but then use uc.teamid for the lookup which seems wrong to me.

If you don't get the cardinalities down each optional match adds a new cartesian product.

```
PROFILE
 MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
 WHERE HAS(uc.teamid)
 WITH uc, collect(ur.name) as accessedroles
 // todo index on :contact(teamid)
 MATCH (c:contact {teamid: uc.teamid})
  // get cardinality down
 WITH c, collect({accessedby: uc.id, accessedroles: accessedroles}) AS access
 OPTIONAL MATCH (c)-[:TAG]->(t:tag)
 // get cardinalitiy down again
 RETURN c, access, COLLECT({id: t.id, object: t.object, name: t.name}) as tags
 LIMIT 10
```

jexp commented Apr 10, 2015 Collaborator

Btw. GitHub is uses for bug tracking. Please direct performance discussion to StackOverflow or the

Google Group. Thanks a lot

 **jexp** closed this Apr 10, 2015

bunkat commented Apr 11, 2015

Nobody in the Google Group or StackOverflow can help because the query planner is not working as intended for OPTIONAL MATCH statements. For example, adding in one additional OPTIONAL MATCH (that doesn't even match anything) can cause a query to require 100x the database accesses. Adding in a single OPTIONAL MATCH in an otherwise performant query can cause a 10x slowdown (see https://groups.google.com/forum/#!topic/neo4j/eR2wYaMLFL8). I've had discussions with a few more experienced Neo4j users and they agree that the results I'm seeing are unexpected.

Cardinality is not the problem. I've spent the last couple of days eliminating all use of OPTIONAL MATCH but keeping the same query structure (store results in a MAP to reduce cardinality). If my structure was causing a problem and OPTIONAL MATCH statements worked similar to MATCH statements then I should still see similar results. Instead I'm able to get 10x to 100x better performance.

Bugs:

- OPTIONAL MATCH statements generate NodeByLabelScan queries that touch the entire database. An OPTIONAL MATCH rooted at a node should never touch the entire database. I'm trying to express `return any connected nodes if they exist, if not, that's okay continue with the query` and not `please look for anything in my database that looks sort of like this`. I think this is where the primary confusion is.
- NodeByLabelScan queries as part of OPTIONAL MATCH statements produce nonsensical PROFILE results. A NodeByLabelScan query for a node label with 1000 total nodes will produce a PROFILE result of 500,000 database hits.
- PROFILE view displays NaN for OPTIONAL MATCH statements that did not have any matches, but yet still required hundreds of thousands of database hits to determine.
- PROFILE view displays estimated rows that make no sense with values in the quintillions for a database with 30k nodes.
- OPTIONAL MATCH statements often do not use indexes correctly even when they exist. USE INDEX does not work with OPTIONAL MATCH statements in order to provide hints to Neo4j. I do have an index on :contact(teamid) and it does nothing.

I understand I'm not on a support contract and am not asking for somebody here to fix my problems for me. I just thought you might want to know that the query planner is not working correctly in a bunch of different scenarios. You're obviously free to do whatever you want with these reports.

boggle commented Apr 11, 2015 Collaborator

Thank you for reporting these issues. This is very helpful; we will investigate them immediately.

In the mean time, you might want to consider prefixing queries that suffer from these issues with "PLANNER RULE" to use the old query planner. Alternatively, you could try to force treating the optional match separately by separating it with "UNWIND [1] AS ignored". This is of course only a temporary workaround.

Regarding estimated rows: Cardinality estimation is difficult; these numbers are allowed to be off by multiple orders of magnitude in various situations as that usually still leads to the right plans (even if not in this case).

jexp commented Apr 11, 2015 Collaborator

What you could also try is to use a pattern expression as a workaround:

Instead of:

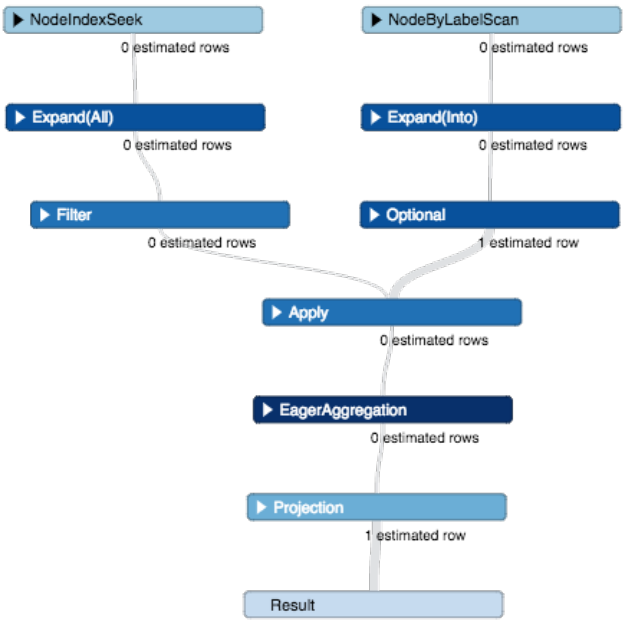
```
OPTIONAL MATCH (c)-[:TAG]->(t:tag)
```

use:

```
WITH c, [p in (c)-[:TAG]->(:tag) | last(nodes(p))] as tags
```

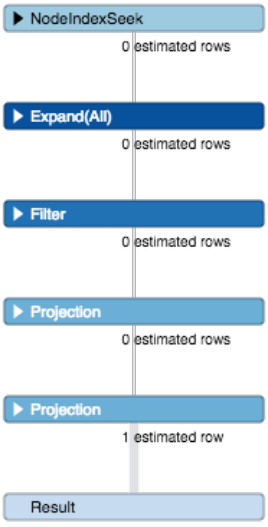
E.g.

```
explain
  MATCH (:user { id:"foobar" })-[:CONTACT]->(c:contact)
  OPTIONAL MATCH (c)-[:TAG]->(t:tag)
  return c, EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name}) as tags
```



becomes:

```
explain
  MATCH (:user { id:"foobar" })-[:CONTACT]->(c:contact)
  with c, [p in (c)-[:TAG]->(:tag) | last(nodes(p))] as tags
return c, [x in tags| {id: x.id, object: x.object, name: x.name}] as tag
```





jexp reopened this Apr 11, 2015

neemah commented Apr 13, 2015

We also use a lot of OPTIONAL MATCH in queries, and with upgrade to 2.2 such queries result in Java Heap Space (as of 2.1.7 there was no exceptions and was much faster).

This is +1 for initial message.

davidegrohmann commented Apr 17, 2015 Collaborator

Hi **@bunkat**,

thanks a lot for pointing out this bug, we are gonna fix it soon.

I can suggest the following workaround until the bug is fixed:

```
PROFILE
  MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
  WHERE HAS(ur.teamid)
  WITH uc, ur SKIP 0

  MATCH (c:contact {teamid: uc.teamid})
  WITH DISTINCT c, {accessedby: uc.id, accessedrole: ur.name} AS r

  OPTIONAL MATCH (c)-[:TAG]->(t:tag)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby,
    isinvited: r.isinvited,
    tags: EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name})
    } AS r

  RETURN c, r LIMIT 10
```

Please note the `SKIP 0` on the first `WITH` clause.

I hope this is helpful.

Cheers,
Davide

bunkat commented Apr 17, 2015

Thanks David. What does the SKIP 0 indicate to the planner? How does that help the issue with OPTIONAL MATCH statements?

boggle commented Apr 17, 2015 Collaborator

@bunkat this is a workaround that forces building an Apply-based plan which then does not suffer from the bug you found. You should only use this to circumvent this problem for the time being and remove it again from your queries as soon as there's a fix and you've upgraded your installation.