Performance degradation when using parameter for "in" operator (v 2.2.0)

Open **rsk700** opened this issue Apr 2, 2015 · 12 comments

rsk700 commented Apr 2, 2015

With parameter this query will run much slower in neo4j 2.2.0:

```
$.ajax({
  type: "POST",
  url: "http://localhost:7474/db/data/cypher",
  async: false,
  data: JSON.stringify({
    query: "match (p:page) where p.id in {ids} return p limit 1",
    params: {ids: [1]}
  }),
  contentType: "application/json"
  });
```

678 ms

Same query without parameter:

```
$.ajax({
  type: "POST",
  url: "http://localhost:7474/db/data/cypher",
  async: false,
  data: JSON.stringify({
    query: "match (p:page) where p.id in [1] return p limit 1"
  }),
  contentType: "application/json"
  });
```

4 ms

With big DB this degradation can be significant up to several minutes even if query indexed properties.

Labels
None yet

Milestone
No milestone

Assignee
No one assigned

4 participants

jexp commented Apr 2, 2015 Collaborator

Is that the first or a subsequent run?
Then there is something wrong (btw. you should use the transaction endpoint instead)
Which query is taking several minutes? and how large is the db (nodes, rels of different kinds).
What kind of OS are you running on what disk?

Please share your `:schema` output, you `graph.db/messages.log`
and the visualized, expanded (little arrow bottom right) query plan of both queries?

rsk700 commented Apr 2, 2015

> Is that the first or a subsequent run?

For any run, as you see from my example difference is 170 times

> Then there is something wrong (btw. you should use the transaction endpoint instead)

No difference if use legacy or transaction endpoint, same 600 ms:

```
$.ajax({
  type: "POST",
  url: "http://localhost:7474/db/data/transaction/commit",
  async: false,
  data: JSON.stringify({
    statements: [{
      statement: "match (p:page) where p.id in {ids} return p limit 1",
      parameters: {ids: [1]}
   }]
  }),
  contentType: "application/json"
  });
```

> Which query is taking several minutes? and how large is the db (nodes, rels of different kinds).

I posted example for small db its 600 ms, for 70 Gb DB it becomes 7 min. Was tested on Ubuntu, on 3 different PC. Issue exists only if use "in" operator with parameter passed separately.

> Please share your :schema output, you graph.db/messages.log

ON :page(id) ONLINE

jexp commented Apr 2, 2015 Collaborator

You forgot the messages.log and the visualized, expanded (little arrow bottom right) query plan of both queries?

rsk700 commented Apr 2, 2015

messages.log

```
2015-04-02 04:14:13.237+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 424)
2015-04-02 04:25:37.982+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 427)
2015-04-02 04:25:38.045+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 427)
2015-04-02 07:20:13.624+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 429)
2015-04-02 07:20:13.689+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 429)
2015-04-02 07:46:11.333+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 435)
2015-04-02 07:46:11.397+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 435)
2015-04-02 08:45:54.221+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 438)
2015-04-02 08:45:54.284+0000 INFO  [o.n.k.EmbeddedGraphDatabase]: NodeCache purge (nr 438)
```

> and the visualized, expanded (little arrow bottom right) query plan of both queries?

how it related to rest api? its not 2 queries, it exactly same query runned defferently

jexp commented Apr 2, 2015 Collaborator

Full messages.log please there is a lot of diagnostic information that's important you can also email it to michael at neo4j.org.

Trust me, it is two queries, the literal values are treated a bit differently.
How large do your id-lists get?
You can also get the query plan-JSON from the HTTP endpoint by prefixing your query with `PROFILE`

```
$.ajax({
  type: "POST",
  url: "http://localhost:7474/db/data/transaction/commit",
  async: false,
  data: JSON.stringify({
    statements: [{
      statement: "PROFILE match (p:page) where p.id in {ids} return p limit 1",
      parameters: {ids: [1]}
   }]
  }),
  contentType: "application/json"
  });
```

jexp commented Apr 2, 2015 Collaborator

I just tested it, it seems the COST planner creates the wrong query-plan for the parametrized query.

Can you try to use

`PLANNER RULE match (p:page) where p.id in {ids} return p limit 1`

and see how it performs?

ikwattro commented Apr 2, 2015

👍 For the report !

rsk700 commented Apr 2, 2015

With `PLANNER RULE` it run fast even with parameter.

Marthym commented May 26, 2015

Hi, I have the exactly same problem in Neo4j 2.2.1

jexp commented May 26, 2015 Collaborator

can you try 2.2.2. ?

Marthym commented May 27, 2015

Hi, it seems to be good with 2.2.2

Thanks

Marthym commented May 27, 2015

After test with 2.2.2 the problem occur but in different condition :

I run this query without parameters :

```
{
   "statements": [
       {
           "statement" : "PROFILE MATCH (ne:NetworkElement {_type:'interface'})-[:Connect*
           "parameters" : {}

       }
       ]
}
```

The query execute in ~100ms without cache, the plan is :

```
{
   "root":{
      "operatorType":"EagerAggregation",
      "DbHits":1468,
      "Rows":1,
      "version":"CYPHER 2.2",
      "KeyNames":"s.tag",
      "EstimatedRows":0,
      "planner":"COST",
      "identifiers":[
         "collect(ne.tag)",
         "s.tag"
      ],
      "children":[
         {
            "operatorType":"Projection",
            "LegacyExpression":"ne",
            "Rows":734,
            "DbHits":1468,
            "EstimatedRows":0,
            "identifiers":[
               "  UNNAMED46",
               "ne",
               "s",
               "s.tag"
            ],
            "children":[
               {
                  "operatorType":"Filter",
                  "LegacyExpression":"(ne:NetworkElement AND ne._type == {  AUTOSTRING0})"
                  "Rows":734,
                  "DbHits":3402,
                  "EstimatedRows":0,
                  "identifiers":[
                     "  UNNAMED46",
                     "ne",
                     "s"
                  ],
                  "children":[
                     {
                        "operatorType":"VarLengthExpand(All)",
                        "ExpandExpression":"(s)-[  UNNAMED46:Connect*]->(ne)",
                        "Rows":1134,
                        "DbHits":2269,
                        "EstimatedRows":0,
                        "identifiers":[
                           "  UNNAMED46",
                           "ne",
                           "s"
                        ],
                        "children":[
                           {
                              "operatorType":"NodeUniqueIndexSeek",
                              "Index":":NetworkElement(tag)",
                              "Rows":1,
                              "DbHits":1,
                              "EstimatedRows":0.9999999999971109,
                              "identifiers":[
                                 "s"
                              ],
                              "children":[

                              ]
                           }
                        ]
                     }
                  ]
               }
            ]
         }
      ]
   }
}
```

Now I run the same query with parameters :

```
{
   "statements": [
       {
           "statement" : "PROFILE MATCH (ne:NetworkElement)-[:Connect*1..]->(s:NetworkElem
           "parameters" : {
               "endNeType" : "interface",
               "startTags" : ["mytag"]
           }

       }
       ]
}
```

The query take 980ms to execute and the plan is not the same :

```
{
   "root":{
      "operatorType":"EagerAggregation",
      "DbHits":1468,
      "Rows":1,
      "version":"CYPHER 2.2",
      "KeyNames":"s.tag",
      "EstimatedRows":0,
      "planner":"COST",
      "identifiers":[
         "collect(ne.tag)",
         "s.tag"
      ],
      "children":[
         {
            "operatorType":"Projection",
            "LegacyExpression":"ne",
            "Rows":734,
            "DbHits":1468,
            "EstimatedRows":0,
            "identifiers":[
               "  UNNAMED26",
               "ne",
               "s",
               "s.tag"
            ],
            "children":[
               {
                  "operatorType":"Filter",
                  "LegacyExpression":"(any(-_-INNER-_- in {startTags} where s.tag == -_-IN
                  "Rows":734,
                  "DbHits":104427,
                  "EstimatedRows":0,
                  "identifiers":[
                     "  UNNAMED26",
                     "ne",
                     "s"
                  ],
                  "children":[
                     {
                        "operatorType":"VarLengthExpand(All)",
                        "ExpandExpression":"(ne)-[  UNNAMED26:Connect*]->(s)",
                        "Rows":34809,
                        "DbHits":105113,
                        "EstimatedRows":0,
                        "identifiers":[
                           "  UNNAMED26",
                           "ne",
                           "s"
                        ],
                        "children":[
                           {
                              "operatorType":"NodeIndexSeek",
                              "Index":":NetworkElement(_type)",
                              "Rows":35495,
                              "DbHits":35496,
                              "EstimatedRows":0.9999999999971109,
                              "identifiers":[
                                 "ne"
                              ],
                              "children":[

                              ]
                           }
                        ]
                     }
                  ]
               }
            ]
         }
      ]
   }
}
```

I have an unique constraint on NetworkElement.tag and index on NetworkElement._type.