Neo4j 2.2 - Cypher performance with large number of nodes / relationships

3 posts by 2 authors

bra...@wayblazer.com Apr 9

Hello,

I have reviewed many threads on this group related to neo4j performance with large data sets (and tried a few conf changes, see below). We're running into slow cypher queries andI have a few questions to see if we could speed them up. Here are some stats on our sample graph:

SETUP
- vm: aws ec2, ubuntu 64bit 14.04, 8 core, 61gb ram, 160 ssd
- neo4j: 2.2.0
- node total: 3,924,945
- edge total: 25,050,584
- db size: 21G

As for our neo4j conf, everything is stock except for the following:

neo4j-wrapper.conf:
- wrapper.java.additional=-XX:+UseNUMA
- wrapper.java.initmemory=32768
- wrapper.java.maxmemory=32768

neo4j.properties
- dbms.pagecache.memory=32g
- cypher_parser_version=2.2
- dbms.cypher.planner=COST
- cache_type=hpc (we are testing enterprise trial)

system
- increased ulimit to 40000 for neo4j per http://neo4j.com/docs/stable/linux-performance-guide.html

QUESTIONS
We have some performance issues on a few of our key queries and I was not 100% sure if it was a conf issue or not. Here are some examples of queries we would like to be faster:

1) PROFILE MATCH (attraction:ATTRACTION)-[:isLocatedIn*1..5]->(destination:DESTINATION)
WHERE (destination.nameLower IN ['austin'])
RETURN attraction.name

Cypher version: CYPHER 2.2, planner: COST. 1088900 total db hits in 3990 ms.

(notes)
- there is an index on destination.nameLower
- there are 231548 attractions
- there are 82729 destinations
- there are approx 200,000 attractions in "austin" in this example
- an attraction can have multiple destinations chains / taxonomy (EX: (attraction)-[]->(postalCode)-[]->(city)-[]->(province)-[]->(country)-[]->(continent))

2) PROFILE MATCH (attraction:ATTRACTION)-[:hasCategory*1..4]->(category:CATEGORY)<-[:hasCategory]-(lat:LAT)
WHERE (lat.name IN ['lodging'])

RETURN attraction.name

Cypher version: CYPHER 2.2, planner: COST. 640849 total db hits in 2186 ms.

(notes)
- there is an index on lat.name
- there are 231548 attractions
- there are 483 categories
- there are 9 lats
- there are approx 125,000 attractions for the lat "lodging" in this example
- an attraction can have multiple categories chains / taxonomy (EX: (attraction)-[]->(childCategory)-[]->(childCategory)-[]->(childCategory)-[]->(parentCategory)-[]->(lat))

INITIAL FINDINGS
- It seems cost is doing a better job than rule for cypher (in our use case), these queries were almost double the time with rule.
- Cypher queries were working fast until our sample data set started to become large

Let me know if I can provide any other information, our goal would be to try and get the queries .

Thanks,
Brandon

Michael Hunger Apr 10

Hi Brandon,

your setup seems to be sane.

> Am 09.04.2015 um 15:22 schrieb bra...@wayblazer.com:
>
> Hello,
>
> I have reviewed many threads on this group related to neo4j performance with large data sets (and tried a few conf changes, see below). We're running into slow cypher queries andI have a few questions to see if we could speed them up. Here are some stats on our sample graph:
>
> **SETUP**
> - vm: aws ec2, ubuntu 64bit 14.04, 8 core, 61gb ram, 160 ssd
> - neo4j: 2.2.0
> - node total: 3,924,945
> - edge total: 25,050,584
> - db size: 21G
>
> -> which files are those, nodestore, relstore, property store?
>
> As for our neo4j conf, everything is stock except for the following:
>
> neo4j-wrapper.conf:
> - wrapper.java.additional=-XX:+UseNUMA
> - wrapper.java.initmemory=32768
> - wrapper.java.maxmemory=32768

I'd reduce the heap size to 24G

neo4j.properties
- dbms.pagecache.memory=32g
- cypher_parser_version=2.2
- dbms.cypher.planner=COST
- cache_type=hpc (we are testing enterprise trial)

system
- increased ulimit to 40000 for neo4j perhttp://neo4j.com/docs/stable/linux-performance-guide.html

QUESTIONS
We have some performance issues on a few of our key queries and I was not 100% sure if it was a conf issue or not. Here are some examples of queries we would like to be faster:

1) PROFILE MATCH (attraction:ATTRACTION)-[:isLocatedIn*1..5]->(
destination:DESTINATION)
WHERE (destination.nameLower IN ['austin'])
RETURN attraction.name

Cypher version: CYPHER 2.2, planner: COST. 1088900 total db hits in 3990 ms.

-> is that the first query or a subsequent one?

What does the query plan look like?
How does isLocatedIn fan out?
How many rows are returned?

(notes)
- there is an index on destination.nameLower
- there are 231548 attractions
- there are 82729 destinations
- there are approx 200,000 attractions in "austin" in this example

-> so you return 200k rows?

- an attraction can have multiple destinations chains / taxonomy (EX: (attraction)-[]->(postalCode)-[]->(city)-[]->(province)-[]->(country)-[]->(continent))

2) PROFILE MATCH (attraction:ATTRACTION)-[:hasCategory*1..4]->(category:CATEGORY)<-[:hasCategory]-(lat:LAT)
WHERE (lat.name IN ['lodging'])
RETURN attraction.name

What does the query plan look like?

how many distinct categories per lat? you might want to make sure you touch the minimal set, e.g. like this:

PROFILE MATCH (category:CATEGORY)<-[:hasCategory]-(lat:LAT)
WHERE (lat.name IN ['lodging'])

WITH distinct category

MATCH (attraction:ATTRACTION)-[:hasCategory*1..4]->(category)

RETURN attraction.name

Cypher version: CYPHER 2.2, planner: COST. 640849 total db hits in 2186 ms.

(notes)
- there is an index on lat.name
- there are 231548 attractions
- there are 483 categories
- there are 9 lats
- there are approx 125,000 attractions for the lat "lodging" in this example
- an attraction can have multiple categories chains / taxonomy (EX: (attraction)-[]->(childCategory)-[]->(childCategory)-[]->(childCategory)-[]->(parentCategory)-[]->(lat))

INITIAL FINDINGS
- It seems cost is doing a better job than rule for cypher (in our use case), these queries were almost double the time with rule.
- Cypher queries were working fast until our sample data set started to become large

-> would you be able to enable gc-logging and share it?

If possible I'd love to have a look at your database and jvm-profile the execution.

-> Did you try to disable hpc to (cache_type=none). HPC consumes a lot of heap which competes with intermediate cypher results.

Cheers, Michael

Let me know if I can provide any other information, our goal would be to try and get the queries
.

Thanks,
Brandon

bra...@wayblazer.com Apr 20

Michael,

Thank you so much for your response and apologies that this response came so late. Here are updates and answers to your questions.

Which files are those, nodestore, relstore, property store?
For the 21G? Here is a list of our graph.db folder: http://pastebin.com/tCep7m9w

I'd reduce the heap size to 24G
Done! Reduced the following:
- wrapper.java.initmemory=24576
- wrapper.java.maxmemory=24576

1) PROFILE MATCH (attraction:ATTRACTION)-[:isLocatedIn*1..5]->(destination:DESTINATION)
WHERE (destination.nameLower IN ['austin'])
RETURN attraction.name

Is that the first query or a subsequent one?
It is part of a much larger query, but since the larger one takes approximately 20 seconds to run, we started breaking down each part to test latency. The large query is very fast with smaller datasets, but as the data has grown, the query got much slower.

What does the query plan look like?
Here is the query plan: http://i.imgur.com/KL4R74j.png

How does isLocatedIn fan out?
An :ATTRACTION can have many :isLocatedIn relationships, which are to :DESTINATION nodes. It depends on the number of data sources for the attraction, but it is not uncomment for an :ATTRACTION node to have one :isLocatedIn chain per data source (:POSTALCODE, :CITY, :PROVINCE, :COUNTRY, all of which have the :DESTINATION label as well). So approximately 3-4 :DESTINATION chain taxonomy per :ATTRACTION. :ATTRACTIONS can also have many :REGION nodes (also :DESTINATION), which are a smaller section of a city, province, or country (EX: South Austin, East Africa, basically tags). Hopefully that makes sense.

How many rows are returned?
For this query, we are returning everything (approx. 200k) because it is only part of our much large query we are trying to speed up. It is the same with the :LAT query in item 2 below. The end result with the larger query is 36 attractions, but we need to refine that data by destination (and other items) before we can get to the final 36.

2) PROFILE MATCH (attraction:ATTRACTION)-[:hasCategory*1..4]->(category:CATEGORY)<-[:hasCategory]-(lat:LAT)
WHERE (lat.name IN ['lodging'])
RETURN attraction.name

What does the query plan look like?
Here is the query plan: http://i.imgur.com/9Eq5Qoc.jpg

How many distinct categories per lat? you might want to make sure you touch the minimal set, e.g. like this:
A :LAT can have lots of :CATEGORY chains related to it, which ultimate relate to :ATTRACTIONS. I tried the distinct example you gave, and it didn't seem to improve much (still took about 2-3 seconds), we have application logic in place to try and restrict categories or lats from being duplicated in the graph on insert.
Here is it's query plan: http://i.imgur.com/dqhVoyp.png

Would you be able to enable gc-logging and share it?
I don't think that would be a problem. I added wrapper.java.additional=-Xloggc:logs/neo4j/neo4j-gc.log to the wrapper conf, but do not see the log file yet. Does it take a while to start to see GC logs?

If possible I'd love to have a look at your database and jvm-profile the execution.
What would be the next steps here?

Did you try to disable hpc to (cache_type=none). HPC consumes a lot of heap which competes with intermediate cypher results.
I disabled and re-ran queries 1 and 2 and they both took aboutu 3-4 seconds each, so seems about the same as with hpc enabled.

Let me know if there is anything else. Thanks again for your time.

Brandon