[GreenPlum] Performance TPC-DS

Description

Pivotal (main Greenplum vendor) recommends to use 2 testing tools gpcheckperf and TPC-DS (TPC-H). We decide to use TPC-DS for complete knowledge about Yandex. Cloud VMs performance compared with bare-metal servers.

Code of TPC-DS which we running in GP may be found in following repo: https://github.com/diarworld/TPC-DS

Complete specification of this test may be downloaded here.

Parameters

Provider

We tested 2 servers providers: onprem & yandex cloud.

Onprem

Bare-Metal segment servers have following configuration:

- HPE DL380 Gen10 5115 Xeon-G (Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz)
- 256GB Memory (HPE 32GB 2Rx4 PC4-2666V-R)
- 10TB RAID10 software array (HPE 1.8TB SAS 10K SFF SC 512e)
- 4x10GB bonded network interfaces

Yandex Cloud

VMs with differend configurations. When we used 5 machines we select 32 cores per VM&256GB memory. We tested 14cores&112GB memory also.

- Intel Cascade Lake (Intel Xeon Gold 6230) 100%
- 10GB network

In results page we calculated total segment CPU cores & total memory available for segments on Greenplum cluster.

Scale Factor

One of the main test parameter is Scale Factor - size of generated data:

> *The TPC-DS benchmark defines a set of discrete scaling points ("scale factors") based on the approximate size of the raw data produced by dsdgen. The actual byte count may vary depending on individual hardware and software platforms.*

Table 3-2 Database Row Counts

Table	Avr Row Size in bytes	Sample Row Counts.					
		1GB	1TB	3TB	10TB	30TB	100TB
call_center	305	6	42	48	54	60	60
catalog_page	139	11,718	30,000	36,000	40,000	46,000	50,000
catalog_returns	166	144,067	143,996,756	432,018,033	1,440,033,112	4,319,925,093	14,400,175,879
catalog_sales	226	1,441,548	1,439,980,416	4,320,078,880	14,399,964,710	43,200,404,822	143,999,334,399
customer	132	100,000	12,000,000	30,000,000	65,000,000	80,000,000	100,000,000
customer_address	110	50,000	6,000,000	15,000,000	32,500,000	40,000,000	50,000,000
customer_ demographics	42	1,920,800	1,920,800	1,920,800	1,920,800	1,920,800	1,920,800
date_dim	141	73,049	73,049	73,049	73,049	73,049	73,049
household_ demographics	21	7,200	7,200	7,200	7,200	7,200	7,200
income_band	16	20	20	20	20	20	20
inventory	16	11,745,000	783,000,000	1,033,560,000	1,311,525,000	1,627,857,000	1,965,337,830
item	281	18,000	300,000	360,000	402,000	462,000	502,000
promotions	124	300	1,500	1,800	2,000	2,300	2,500
reason	38	35	65	67	70	72	75
ship_mode	56	20	20	20	20	20	20
store	263	12	1,002	1,350	1,500	1,704	1,902
store_returns	134	287,514	287,999,764	863,989,652	2,879,970,104	8,639,952,111	28,800,018,820
store_sales	164	2,880,404	2,879,987,999	8,639,936,081	28,799,983,563	86,399,341,874	287,997,818,084
time_dim	59	86,400	86,400	86,400	86,400	86,400	86,400
warehouse	117	5	20	22	25	27	30
web_page	96	60	3,000	3,600	4,002	4,602	5,004
web_returns	162	71,763	71,997,522	216,003,761	720,020,485	2,160,007,345	7,199,904,459
web_sales	226	719,384	720,000,376	2,159,968,881	7,199,963,324	21,600,036,511	71,999,670,164
web_site	292	30	54	66	78	84	96

We have tested 4 SFs - 100GB, 300GB, 1TB, 3TB.

Concurrency

We used concurrency parameter = 5.

Score

Score calculates with following formula:

$$QphDS@SF = SF * 3600 \frac{198 * S}{(T_{QR1} + T_{DM} + T_{QR2} + 0.01 * S * T_{Load})}$$

T_{QR1}:elapsed time of Query Run 1
T_{QR2}:elapsed time of Query Run 2.
T_{DM}:elapsed time of the Data Maintenance run.
T_{Load}: elapsed time of database load test.
S:number of simulated concurrent users (streams)
SF:scale factor.

Figure 2: TPC-DS Primary Metric

Results

Log parser

```
import boto3
import re
import pandas as pd
def sizeof_fmt(num, suffix='B'):
    for unit in ['','Ki','Mi','Gi','Ti','Pi','Ei','Zi']:
        if abs(num) < 1024.0:
            return "%3.1f%s%s" % (num, unit, suffix)
        num /= 1024.0
    return "%.1f%s%s" % (num, 'Yi', suffix)
def sf_to_size(num):
    # http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.1.0.pdf - 3.1.3
    bytes = int(num) * 1024 * 1024 * 1024 * 1024 / 1000
    return (sizeof_fmt(bytes))
endpoint_url='https://storage.yandexcloud.net'
access_key = '***'
secret_key = '***'
session = boto3.session.Session(aws_access_key_id=access_key,aws_secret_access_key=secret_key)
s3 = session.client(service_name='s3',endpoint_url=endpoint_url)
s3_resource = session.resource(service_name='s3',endpoint_url=endpoint_url)

df = pd.DataFrame(columns=['Nodes', 'SegemntsPerNode', 'Cores', 'Memory', 'Scale', 'Provider', "CustomChange",
'Date', 'Load', 'Analyze', '1 User Queries', 'Concurrent Queries', 'Q', 'TPT', 'TTT', 'TLD', 'Score'])

for obj in s3_resource.Bucket('t-dp-tests').objects.filter(Prefix='perftests_results/'):
    if re.search('/tpcds.log', obj.key):
        skip=False
        print(obj.key + " Size:" + str(sizeof_fmt(obj.size)) + " Date: " + obj.last_modified.strftime("%d-%m-%
Y, %H:%M:%S"))
        test=obj.key.split('/')[1]
        test_case = test.split('_')
        if test.endswith('yandex') or test.endswith('onprem'):
            test_case.append('default')
        elif test.endswith('failed'):
            skip=True
        if not skip:
            c2=False
            c3=True
            result=test_case
            for line in obj.get()['Body']._raw_stream:
                if re.search('^2019.*', line.decode('utf-8')) and c3:
                    print(line.decode('utf-8').split(":")[0])
                    result.append(line.decode('utf-8').split(":")[0])
                    c3=False
                if re.search('Load        .*', line.decode('utf-8')) or c2:
                    if c3:
                        result.append('20191010')
                        c3=False
                    result.append(line.decode('utf-8').split()[-1])
                    c2=True
            result_df=pd.DataFrame([result], columns=['Nodes', 'SegemntsPerNode', 'Cores', 'Memory', 'Scale',
'Provider', "CustomChange", 'Date', 'Load', 'Analyze', '1 User Queries', 'Concurrent Queries', 'Q', 'TPT',
'TTT', 'TLD', 'Score'])
            df = df.append(result_df, ignore_index = True)
df['Size'] = df.apply(lambda row: sf_to_size(row.Scale), axis = 1)
df.set_index('Date')
df['Segments'] = df.apply(lambda row: int(row.Nodes) * int(row.SegemntsPerNode), axis = 1)
df['ClusterCores'] = df.apply(lambda row: int(row.Nodes) * int(row.Cores), axis = 1)
df['ClusterMemory'] = df.apply(lambda row: sizeof_fmt(int(row.Nodes) * int(row.Memory)* 1024 * 1024 * 1024),
axis = 1)
df['Queries Time'] = df.apply(lambda row: float(row.TPT) / int(row.Q), axis = 1)
df['NumScore'] = df.apply(lambda row: int(row.Scale) * int(row.Q), axis = 1)
df['DemScore'] = df.apply(lambda row: (float(row.TPT) + (float(2) * float(row['Concurrent Queries'])) + float
(row.TLD)), axis = 1)
df['CalcScore'] = df.apply(lambda row: (float(row.NumScore) / float(row.DemScore)), axis = 1)
basic_score = df.loc[(df['Provider'] == 'onprem') & (df['Nodes'] == '5') & (df['Scale'] == '1000') & (df
['CustomChange'] == 'default')].CalcScore.values[0]
df['PrcChange'] = df.apply(lambda row: (((float(row.CalcScore) / float(basic_score)) - 1) * 100), axis = 1)
df['PrcChange'] = df['PrcChange'].round(3).apply(str) + '%'
df[['Date', 'Provider', 'Nodes', 'Segments', 'ClusterCores', 'ClusterMemory', 'Size', "CustomChange",
'CalcScore', 'PrcChange']].sort_values(by=['CalcScore'], ascending=False)
```

#	Date	Provider	Nodes	Segments	ClusterCores	ClusterMemory	Size	CustomChange	CalcScore	PrcChange
1	20191010	onprem	21	336	672	5.2TiB	3.0TiB	GP6	40.686125	255.447%
2	20191007	onprem	21	336	672	5.2TiB	1.0TiB	GP6.AdminPool90	34.656664	202.771%
3	20191006	onprem	21	336	672	5.2TiB	1.0TiB	default	31.895572	178.65%
4	20191007	onprem	21	336	672	5.2TiB	1.0TiB	GP6	30.014088	162.212%
5	20191007	onprem	21	336	672	5.2TiB	102.4GiB	default	13.701748	19.703%
6	20191001	yandex	18	90	252	2.0TiB	1.0TiB	default	12.931680	12.975%
7	20191002	yandex	18	90	252	2.0TiB	307.2GiB	default	11.957135	4.461%
8	20191003	onprem	5	90	160	1.2TiB	1.0TiB	AdminPool90	11.638090	1.674%
9	20191003	onprem	5	90	160	1.2TiB	1.0TiB	HugePagesTuning	11.601731	1.356%
10	20191001	onprem	5	90	160	1.2TiB	1.0TiB	default	11.446483	0.0%
11	20191007	onprem	21	336	672	5.2TiB	102.4GiB	GP6	11.232275	-1.871%
12	20191002	onprem	5	90	160	1.2TiB	307.2GiB	default	11.112115	-2.921%
13	20191005	onprem	5	90	160	1.2TiB	1.0TiB	GP6	10.893043	-4.835%
14	20191005	onprem	5	90	160	1.2TiB	1.0TiB	GP6#2	10.874319	-4.999%
15	20190930	yandex	18	90	180	1.4TiB	3.0TiB	default	10.178751	-11.075%
16	20191002	onprem	5	90	160	1.2TiB	102.4GiB	default	9.858725	-13.871%
17	20191002	onprem	5	90	160	1.2TiB	102.4GiB	GPBlockSize8k	9.782165	-14.54%
18	20191002	onprem	5	90	160	1.2TiB	102.4GiB	AdminPool90	9.679785	-15.434%
19	20191002	yandex	18	90	252	2.0TiB	102.4GiB	default	9.590719	-16.213%
20	20191001	yandex	18	90	180	1.4TiB	1.0TiB	default	9.197611	-19.647%
21	20191006	onprem	5	90	160	1.2TiB	102.4GiB	GP6	9.183538	-19.77%
22	20191006	onprem	5	90	160	1.2TiB	102.4GiB	GP6.AdminPool90	9.103244	-20.471%
23	20191001	yandex	18	90	180	1.4TiB	307.2GiB	default	8.642192	-24.499%
24	20191001	yandex	18	90	180	1.4TiB	102.4GiB	default	7.197199	-37.123%
25	20191002	yandex	18	90	252	2.0TiB	10.2GiB	default	3.341427	-70.808%
26	20190930	yandex	18	90	180	1.4TiB	10.2GiB	default	2.730759	-76.143%
27	20191005	onprem	5	90	160	1.2TiB	10.2GiB	GP6	2.627309	-77.047%
28	20191007	yandex	5	80	160	1.2TiB	102.4GiB	default	2.262148	-80.237%

Custom change

Custom Change - is set of changes which we apply to default installation of GreenPlum:

- **GP6** - Version of greenplum cluster is 6.0.0 instead of 5.21.
- **AdminPool90** - we ran 2 following commands for set memory&cpu limits to utilize all cluster resources:

```
ALTER RESOURCE GROUP admin_group SET MEMORY_LIMIT 90;
ALTER RESOURCE GROUP admin_group SETCPU_RATE_LIMIT 90;
```

- **HugePagesTuning** - https://gpdb.docs.pivotal.io/6-0/install_guide/prep_os.html disabling of Transparent Huge Pages (THP) and some sysctl tuning (not tuned by default installer) by running following playbook:
- **GPBlockSize8k** - change of default block size for generated tables - https://github.com/diarworld/TPC-DS/commit/e9a1a9ec4a17ad4a1a7f51ba745df00dda3066cb

Findings

- Decided to run gp_tunung playbook by default (disable THP, additional tuning sysctl, etc).
- Block size influence is not so much as expected. Decide to use default 32k block size.
- Bare-metal server performance is ~20% better than cloud installation. When GP is installed on virtual machines, need to order ~20% more CPU&memory resources to receive the same performance.
- GP on Ya.Cloud VMs need to have smaller overcommit parameters, cause on high CPU workloads affected virtual networking layer - it will lose performance (lose packets, slow responses, etc.). Because of this FTS probe will mark segments as failed and then all cluster will be down. We had no successfully 3TB SF tests on ya.cloud (tried 2 times, decided to not run, cause of high pricing).
- Decided to upgrade to **GP 6.0** version: https://greenplum.org/oltp-workload-performance-improvement-in-greenplum-6/ Cause now we have high OLTP workload - our users uses greenplum is not correctly, we need to explain it them (OLAP - OLTP differences), *but we need to provide best performance as we can.*
- Yandex Cloud shows better performance when we use 18 small nodes instead of 5 huge nodes, because high workload less affected on hosts machines. So 18 nodes (10 cores&64gb memory) * 5 segment per VM is better than 5 nodes (36 cores&256 memory) * 18 segments.