Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Петрозаводский государственный университет»

Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

Отчет по выполнению проекта в рамках курса
“Оценивание производительности сетевых систем”

Оценивание производительности системы ClickHouse

Выполнила:
студентка группы 22603
Кобелева Алена

Петрозаводск — 2020

Содержание

1 Постановка задачи

ClickHouse — это колоночная аналитическая СУБД с открытым кодом, позволяющая выполнять аналитические запросы в режиме реального времени на структурированных больших данных, разрабатываемая компанией Яндекс. Целью работы является проведение анализа сетевого трафика системы ClickHouse

2 Описание системы

ClickHouse - столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP).

Разный порядок хранения данных лучше подходит для разных сценариев работы. Сценарий работы с данными - это то, какие производятся запросы, как часто и в каком соотношении; сколько читается данных на запросы каждого вида - строк, столбцов, байт; как соотносятся чтения и обновления данных; какой рабочий размер данных и насколько локально он используется; используются ли транзакции и с какой изолированностью; какие требования к дублированию данных и логической целостности; требования к задержкам на выполнение и пропускной способности запросов каждого вида и т. п.

Чем больше нагрузка на систему, тем более важной становится специализация под сценарий работы, и тем более конкретной становится эта специализация. Не существует системы, одинаково хорошо подходящей под существенно различные сценарии работы. Если система подходит под широкое множество сценариев работы, то при достаточно большой нагрузке, система будет справляться со всеми сценариями работы плохо, или справляться хорошо только с одним из сценариев работы.

Ключевые особенности OLAP сценария работы:

- подавляющее большинство запросов - на чтение;
- данные обновляются достаточно большими пачками ($>$ 1000 строк), а не по одной строке, или не обновляются вообще;
- данные добавляются в БД, но не изменяются;
- при чтении, вынимается достаточно большое количество строк из БД, но только небольшое подмножество столбцов;
- таблицы являются «широкими», то есть, содержат большое количество столбцов;

- запросы идут сравнительно редко (обычно не более сотни в секунду на сервер);
- при выполнении простых запросов, допустимы задержки в районе 50 мс;
- значения в столбцах достаточно мелкие - числа и небольшие строки (пример - 60 байт на URL);
- требуется высокая пропускная способность при обработке одного запроса (до миллиардов строк в секунду на один сервер);
- транзакции отсутствуют;
- низкие требования к консистентности данных;
- в запросе одна большая таблица, все таблицы кроме одной маленькие;
- результат выполнения запроса существенно меньше исходных данных - то есть, данные фильтруются или агрегируются;
- результат выполнения помещается в «оперативку» на одном сервере.

Легко видеть, что OLAP сценарий работы существенно отличается от других распространённых сценариев работы (например, OLTP или Key-Value сценариев работы). Таким образом, не имеет никакого смысла пытаться использовать OLTP или Key-Value БД для обработки аналитических запросов, если вы хотите получить приличную производительность («выше плинтуса»). Например, если вы попытаетесь использовать для аналитики MongoDB или Redis - вы получите анекдотически низкую производительность по сравнению с OLAP-СУБД.

По результатам внутреннего тестирования в Яндексе, ClickHouse обладает наиболее высокой производительностью (как наиболее высокой пропускной способностью на длинных запросах, так и наиболее низкой задержкой на коротких запросах), при соответствующем сценарии работы, среди доступных для тестирования систем подобного класса. [1]

[1] Документация ClickHouse: Введение

3 Метрика анализа

Будут рассчитаны следующие метрики для TCP протокола:

1. Среднее время установления соединения с сервером (Connection Time).
2. Среднее круговое время передачи по сети (Round Trip Time). Дельта между отправкой пакета со стороны клиента или сервера и получением подтверждения на этот пакет.

А также метрики для HTTP протокола:

1. Среднее время необходимое для передачи всех данных от клиента или сервера на его запрос (Data Transfer Time).
2. Среднее время отклика (Server Response Time). Время необходимое серверу на подготовку первого пакета с данными в ответ на запрос клиента.

Был составлен ряд требований:

- Время подключения к серверу не должно превышать 100мс;
- Круговое время передачи по сети должно быть не выше 100мс;
- Время отклика системы не должно превышать 1с;
- Время передачи всех данных не должно превышать 1с.

Эти требования актуальны для следующей конфигурации сервера:

- ОС Ubuntu 20.04.1 LTS;
- Процессор Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz;
- Оперативная память: 16Гб LPDDR3
- SSD 256Гб

4 План экспериментов

Для проведения экспериментов будет использоваться снифер Wireshark v.3.4.0 на системе Windows 10, ClickHouse сервер v.20.11.5.18.

Все метрики будут рассматриваться для различного количества одновременно выполняющихся запросов в целях выявления понижения производительности системы при увеличении нагрузки. Запросы будут выполняться через http-интерфейс.

Для захвата трафика в Wireshark будет применен следующий фильтр: host 192.168.100.10 and port 8123 and tcp. В настройках Wireshark будут установлены разрешенные протоколы: tcp и http, а также добавлены колонки Source Port, Destination Port, Sequence Number, Acknowledgment Number и Flags.

В ходе экспериментов будут выполняться следующие шаги:

1. Запустить Wireshark.
2. Включить фильтрацию и настроить разрешенные протоколы и колонки.
3. Включить захват трафика.
4. Запустить скрипт, который выполняет параллельные запросы к системе ClickHouse.
5. Экспортировать данные из Wireshark в формате CSV.
6. Обработать данные.
7. Вычислить соответствующую метрику.

Анализ будет проводиться для 1, 5 и 10 параллельно выполняющихся запросов для каждого сценария (см. раздел 4.1). Для каждого количества параллельно выполняющихся запросов и для каждого сценария будет сделано 100 подходов для получения более достоверных результатов.

4.1 SQL запросы

Для проведения экспериментов был взят массив данных из раздела Tutorials документации ClickHouse. Эти данные представляют собой анонимные данные Яндекс.Метрики и включают две таблицы:

- hits — таблица с каждым действием, выполненным всеми пользователями на всех веб-сайтах, на которых подключен сервис.

- visits — таблица, содержащая сессии вместо отдельных действий.

Подробное описание этих таблиц можно прочитать в документации. Для проведения экспериментов были составлены следующие запросы:

- Сценарий 1A

```
SELECT
    StartURL AS URL,
    AVG(Duration) AS AvgDuration
FROM tutorial.visits_v1
GROUP BY URL
LIMIT 10000;
```

- Сценарий 1B

```
SELECT
    StartURL AS URL,
    AVG(Duration) AS AvgDuration
FROM tutorial.visits_v1
GROUP BY URL
ORDER BY AvgDuration DESC
LIMIT 10000;
```

- Сценарий 2

```
SELECT
    URL,
    groupUniqArray(HTTPError) AS HTTPErrors
FROM tutorial.hits_v1
GROUP BY URL
LIMIT 10000;
```

4.2 Скрипт, выполняющий запросы

```
import time

import click
import requests
from concurrent.futures import as_completed, ThreadPoolExecutor


```

```
@click.command()
@click.option("--ip", type=str, required=True, help="ip of ClickHouse server")
@click.option(
    "-p",
    "--port",
    type=int,
    default=8123,
    show_default=True,
    help="port of ClickHouse server",
)
@click.option(
    "-t",
    "--threads-number",
    type=int,
    required=True,
    help="number of concurrent sending queries",
)
@click.option("-q", "--query-filepath", type=str, help="file path to SQL query")
@click.option(
    "-n",
    "--number-of-iterations",
    type=int,
    default=100,
    show_default=True,
    help="number of iterations",
)
def main(
    ip: str,
    port: int,
    threads_number: int,
    query_filepath: str,
    number_of_iterations: int,
) -> None:
    with open(query_filepath) as f:
        query = f.read()
    for i in range(number_of_iterations):
        print(f"Start iteration {i}")
        with ThreadPoolExecutor(max_workers=threads_number) as
```

```
    executor:
46              futures = [
47                  executor.submit(send, ip=ip, port=port, query=query)
48                  for _ in range(threads_number)
49              ]
50              for future in as_completed(futures):
51                  future.result()
52          time.sleep(10)
53
54
55  def send(ip: str, port: str, query: str) -> None:
56      response = requests.get(
57          f"http://{ip}:{port}/",
58          headers={
59              "Connection": "keep-alive",
60              "User-Agent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome/53.0.2785.116 Safari/537.36 OPR
    /40.0.2308.81",
61          },
62          params={"query": query},
63      )
64      response.raise_for_status()
65
66
67  if __name__ == "__main__":
68      main()
```

4.3 Скрипт, собирающий метрики

```
1  import abc
2  import csv
3  import matplotlib.pyplot as plt
4  import numpy
5  import os
6  from collections import defaultdict
7  from queue import PriorityQueue
8
9  import attr
10 import pylatex
11 from typing import Any, Dict, List, Mapping, Tuple
12
```

```
13
14 def is_syn(mask: str) -> bool:
15     return bool((1 << 1) & int(mask, 16))
16
17
18 def is_ack(mask: str) -> bool:
19     return bool((1 << 4) & int(mask, 16))
20
21
22 def is_push(mask: str) -> bool:
23     return bool((1 << 3) & int(mask, 16))
24
25
26 class Metric(abc.ABC):
27     @abc.abstractmethod
28     def process(self, row: Mapping) -> None:
29         pass
30
31     @abc.abstractmethod
32     def flush(self) -> List[float]:
33         pass
34
35     @property
36     @abc.abstractmethod
37     def name(self) -> str:
38         pass
39
40
41 @attr.s(slots=True)
42 class ConnectionTime(Metric):
43     name = "connection_time"
44
45     _syn_times: Dict[int, float] = attr.ib(factory=dict, init=False)
46     _connection_times: Dict[int, float] = attr.ib(factory=dict, init=
       False)
47
48     def process(self, row: Mapping) -> None:
49         if row["Destination Port"] != 8123:
50             return
51         key = row["Source Port"]
```

```
52          if is_syn(row["Flags"]):
53              self._syn_times[key] = row["Time"]
54          elif is_ack(row["Flags"]):
55              self._connection_times.setdefault(key, row["Time"] - self.
    _syn_times[key])
56
57      def flush(self) -> List[float]:
58          result = list(self._connection_times.values())
59          self._syn_times.clear()
60          self._connection_times.clear()
61          return result
62
63
64  @attr.s(slots=True)
65  class DataTransferTime(Metric):
66      name = "data_transfer_time"
67
68      _first_segment_times: Dict[int, float] = attr.ib(factory=dict,
    init=False)
69      _data_transfer_times: Dict[int, float] = attr.ib(factory=dict,
    init=False)
70
71      def process(self, row: Mapping) -> None:
72          if row["Source Port"] != 8123:
73              return
74          key = row["Destination Port"]
75          if row["Protocol"] == "HTTP":
76              self._first_segment_times.setdefault(key, row["Time"])
77              self._data_transfer_times[key] = (
78                  row["Time"] - self._first_segment_times[key]
79              )
80
81      def flush(self) -> List[float]:
82          result = list(self._data_transfer_times.values())
83          self._first_segment_times.clear()
84          self._data_transfer_times.clear()
85          return result
86
87
88  @attr.s(slots=True)
```

```
class ServerResponseTime(Metric):
    name = "server_response_time"

    _request_times: Dict[int, float] = attr.ib(factory=dict, init=False)
    _server_response_times: Dict[int, float] = attr.ib(factory=dict, init=False)

    def process(self, row: Mapping) -> None:
        src_port = row["Source Port"]
        dst_port = row["Destination Port"]
        if row["Protocol"] == "HTTP":
            if dst_port == 8123:
                self._request_times.setdefault(src_port, row["Time"])
            else:
                self._server_response_times.setdefault(
                    dst_port, row["Time"] - self._request_times[dst_port]
                )

    def flush(self) -> List[float]:
        result = list(self._server_response_times.values())
        self._request_times.clear()
        self._server_response_times.clear()
        return result


@attr.s(slots=True)
class RoundTripTime(Metric):
    name = "round_trip_time"

    _send_times: Dict[Tuple[int, int], PriorityQueue] = attr.ib(
        factory=lambda: defaultdict(PriorityQueue)
    )
    _round_trip_times: List[float] = attr.ib(factory=list)

    def process(self, row: Mapping) -> None:
        src_port = row["Source Port"]
        dst_port = row["Destination Port"]
        self._send_times[(src_port, dst_port)].put(
```

```
            (row["Sequence Number"], row["Time"])
        )
        while not self._send_times[(dst_port, src_port)].empty():
            (seq, send_time) = self._send_times[(dst_port, src_port)].
    queue[0]
            if seq >= row["Acknowledgment Number"]:
                break
            self._send_times[(dst_port, src_port)].get()
            self._round_trip_times.append(row["Time"] - send_time)

    def flush(self) -> List[float]:
        result = self._round_trip_times.copy()
        self._send_times.clear()
        self._round_trip_times.clear()
        return result


def process(reader: csv.DictReader) -> Mapping[str, float]:
    result_metrics: Dict[str, List[float]] = defaultdict(list)
    current_metrics = [
        ConnectionTime(),
        DataTransferTime(),
        ServerResponseTime(),
        RoundTripTime(),
    ]
    last_time = 0
    for row in reader:
        entry: Dict[str, Any] = row.copy()
        entry["Time"] = float(row["Time"]) * 1000
        entry["Sequence Number"] = int(row["Sequence Number"])
        entry["Acknowledgment Number"] = int(row["Acknowledgment
    Number"])
        entry["Source Port"] = int(row["Source Port"])
        entry["Destination Port"] = int(row["Destination Port"])
        for metric in current_metrics:
            if entry["Time"] - last_time > 9000:
                result_metrics[metric.name].extend(metric.flush())
            metric.process(entry)
        last_time = entry["Time"]
    return {name: numpy.mean(values) for name, values in
```

```
        result_metrics.items()}
164
165
166 def generate_tables(result: Mapping[str, Mapping[str, Mapping[str,
        float]]]):
167     for metric in result:
168         tabular = pylatex.Tabular("|c|c|c|c|")
169         tabular.add_hline()
170         tabular.add_row("", "1", "5", "10")
171         tabular.add_hline()
172         for query, value in result[metric].items():
173             query_name = query[5:].upper()
174             tabular.add_row(
175                 query_name,
176                 f"{value['1']:.2f}",
177                 f"{value['5']:.2f}",
178                 f"{value['10']:.2f}",
179             )
180             tabular.add_hline()
181         caption = " ".join(item.capitalize() for item in metric.split(
        "_"))
182         center = pylatex.Center()
183         center.append(tabular)
184         center.append(pylatex.NoEscape(r"\caption{" + caption + r"}"))
185         with open(f"tables/{metric}.tex", "w") as f:
186             f.write(center.dumps())
187
188
189 def generate_diagrams(result: Mapping[str, Mapping[str, Mapping[str,
        float]]]):
190     for metric in result:
191         data_for_plots: Dict[str, Dict[str, float]] = defaultdict(dict
        )
192
193         for query, value in result[metric].items():
194             query_name = query[5:].upper()
195             data_for_plots["1"][query_name] = value["1"]
196             data_for_plots["5"][query_name] = value["5"]
197             data_for_plots["10"][query_name] = value["10"]
198
```

```
199         fig, ax = plt.subplots()
200         x = numpy.arange(3)
201         width = 0.2
202         shift = -width
203 
204         for num_workers, values in data_for_plots.items():
205             ax.bar(
206                 x + shift,
207                 [value for _, value in sorted(values.items())],
208                 width=width,
209                 label=num_workers,
210             )
211             shift += width
212 
213         ax.set_xticks(x)
214         ax.set_xticklabels(["1A", "1B", "2"])
215         ax.set_xlabel("Query")
216         ax.set_ylabel(f"{metric} (ms)")
217         ax.legend()
218         plt.savefig(f"diagrams/{metric}.png")
219 
220 
221 def main():
222     result = defaultdict(lambda: defaultdict(dict))
223     directory = "data"
224     for query in os.listdir(directory):
225         for datafile in os.listdir(os.path.join(directory, query)):
226             with open(os.path.join(directory, query, datafile)) as f:
227                 reader = csv.DictReader(f)
228                 metrics = process(reader)
229                 for metric, value in metrics.items():
230                     result[metric][query][datafile.partition(".")[0]]
    = value
231     generate_tables(result)
232     generate_diagrams(result)
233 
234 
235 if __name__ == "__main__":
236     main()
```

5 Результаты экспериментов

В результате экспериментов было получено значение каждой из метрик (в миллисекундах) для каждого сценария и каждого количества параллельно выполняющихся запросов. Эти результаты представлены в таблице 1 и на рисунке 1. В таблицах столбцы соответствуют количеству параллельно выполняющихся запросов, а строки — сценариям. На диаграммах метрики сгруппированы по сценариям, для каждой группы представлены значения для каждого количества параллельно выполняющихся запросов в виде столбцов.

	1	5	10
1A	48.09	39.95	43.45
1B	53.33	53.23	49.56
2	61.44	35.11	41.62

(a) Connection Time

	1	5	10
1A	26.42	81.89	191.56
1B	27.39	101.53	206.48
2	68.30	102.42	208.34

(b) Round Trip Time

	1	5	10
1A	136.70	528.46	1000.84
1B	206.14	766.99	1565.52
2	503.53	2358.72	4903.93

(c) Server Response Time

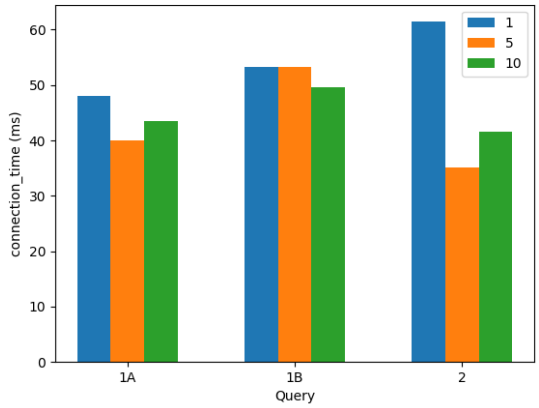
	1	5	10
1A	236.42	674.18	1387.67
1B	218.73	960.13	1878.53
2	381.35	954.18	1850.01

(d) Data Transfer Time

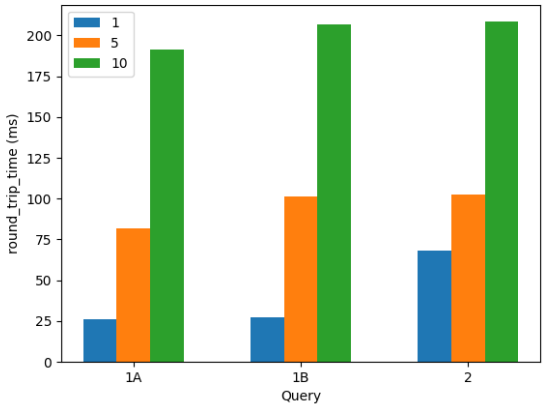
Таблица 1: Результаты измерений для каждой из метрик (мс)

Результаты экспериментов показали, что все требования сохраняются для неконкурирующих запросов. Однако, при увеличении одновременного количества запросов ситуация меняется для всех метрик, кроме Connection Time:

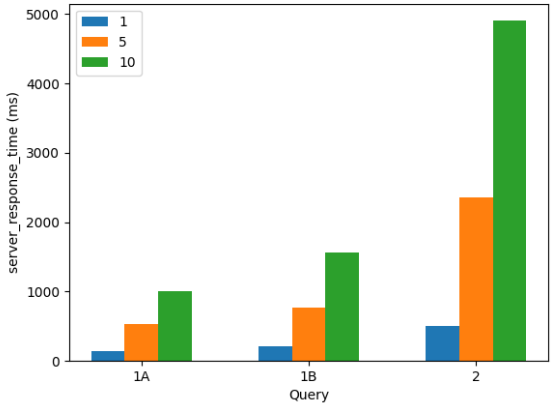
- RTT находится около пороговых пределов при 5 параллельных запросах, а при 10 уже превышает его в два раза.
- SRT для сценариев 1A и 1B находится в норме для 5 параллельных запросов, а для 10 уже ее превышает. В случае сценария 2 SRT сильно превышает норму уже при 5 запросах.
- DTT при 5 параллельных запросах находится ниже порогового значения, но для 10 параллельных запросов уже значительно превышает его.



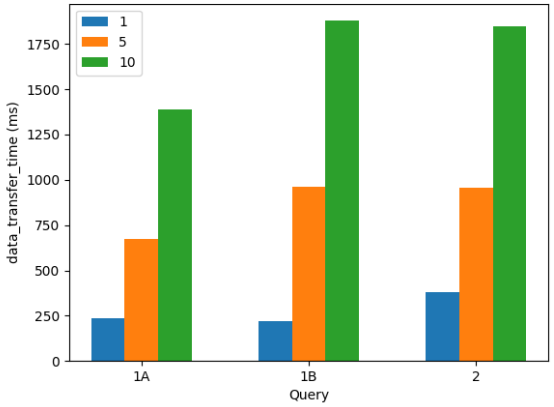
(a) Connection Time



(b) Round Trip Time



(c) Server Response Time



(d) Data Transfer Time

Рис. 1: Диаграммы для каждой из метрик

6 Выводы

Система удовлетворяет всем требованиям при неконкурирующих запросах, но в реальных условиях, как правило, к системе будет выполняться много запросов одновременно. С имеющимися у нас ресурсами данная система не удовлетворяет требованиям, поставленным в начале экспериментов. Однако, стоит отметить, что мы обращались только к обычным таблицам ClickHouse и не пользовались материализованными представлениями, которые могут помочь оптимизировать время ответа сервера. Несмотря на это, данный механизм не поможет исправить проблемы с временем передачи данных по сети (DTT) и круговым временем передачи по сети (RTT).