|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления» (ИУ)

КАФЕДРА «Системы обработки информации и управления» (ИУ5)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

**«Сравнение psql14 и psql15 для метаграфового хранилища»**

Студент группы ИУ5-33М **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Е. А. Белоусов

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Ю. Е. Гапанюк

*2022 г.*

Оглавление

[Введение 3](#_Toc123075198)

[Основная часть 4](#_Toc123075199)

[Эксперимент 5](#_Toc123075200)

[Выводы 7](#_Toc123075201)

[Приложение 8](#_Toc123075202)

# **Введение**

Метаграфовая модель на основе СУБД psql14 оказалась довольно медленной. Одним из возможных вариантов решения этой проблемы является переход на psql15. Данная работа содержит сравнение СУБД psql14 и psql15 для использования в метаграфовом хранилище.

# **Основная часть**

Главным критерием, по которому будет происходить сравнение, является скорость работы алгоритмов, использующихся в метаграфовом хранилище. Так как на данном этапе реализованы только сохранение и выгрузка метаграфа, будем проверять работу СУБД на этих двух операциях.

В эксперименте мы будем устанавливать зависимость времени выполнения функций от количества элементов метаграфа. Для этого зафиксируем одинаковое количество всех элементов и поварьируем по одному из них. Так как метаграфы генерируются случайно, полученные выборочные временные зависимости будут отражать зависимости реальные. Тестовый скрипт приведен в приложении.

После получения временных значений нужно будет прогнать тестовый скрипт еще раз с подключением к psql15. Затем нужно будет построить графики зависимостей, сравнить их и сделать выводы.

# Эксперимент

Тестовый скрипт для эксперимента приведен во вложении.

Метод generateXML генерирует XML со случайным метаграфом с заданным количеством элементов, который можно загрузить в метаграфовое хранилище.

Метод load\_xml сохраняет XML с метаграфом в метаграфовое хранилище.

Метод save\_xml выгружает XML с метаграфом из метаграфового хранилища.

Функция delete\_all очищает таблицы базы данных.

Конфигурации экспериментов и их результаты приведены в таблице 1. Синим цветом отображена временная зависимость, полученная в результате эксперимента с psql14, красным – временная зависимость psql15.

Как видно, полученные случайные зависимости весьма похожи.

Таблица 1

| **Изменяемый параметр** | **Конфигурация эксперимента** | **Время загрузки** | **Время выгрузки** |
| --- | --- | --- | --- |
| Вершины |  |  |  |
| Метавершины |  |  |  |
| Метаребра |  |  |  |
| Ребра |  |  |  |
| Атрибуты |  |  |  |

# Выводы

По полученным зависимостям можно сделать следующие выводы:

1. Время загрузки метаграфа в метаграфовое хранилище на psql15 в среднем ниже времени загрузки такого же метаграфа в метаграфовое хранилище на psql14 на константное значение.
2. Характер зависимостей времени загрузки psql15 такой же как psql14.
3. Время выгрузки метаграфа psql15 в среднем такое же как и в psql14.

Следовательно, я считаю:

1. Нужно производить оптимизацию функций метаграфового хранилища
2. Нужно рассмотреть варианты отличные от psql.

# Приложение

import sys

sys.path.append('C:\\Users\\Evgeny\\PycharmProjects\\MetagraphDB\\metagraphdb\\modules\\Module\_input\_output\_xml')

sys.path.append('C:\\Users\\Evgeny\\PycharmProjects\\MetagraphDB\\metagraphdb\\modules\\Module\_for\_creating\_and\_modifying\_elements\_of\_a\_metagraph\_model')

import parcer

import generator

import psycopg2

gen = generator.Generator()

gen.generateXML('test.xml', 1000, 150, 400, 35)

test = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

test.load\_xml('test.xml', 'test2')

test.save\_xml(str(test.all\_nodeid\_id['Main\_metanode\_id'][0]), 'test2', 'result.xml')

tables\_for\_cleaning = ['timetable', 'nodes', 'linkpoints', 'attributes', 'temporal\_mutables', 'temporal\_immutables']

def delete\_all():

conn = psycopg2.connect(dbname='metagraph\_db',

user='postgres',

password='1',

host='localhost',

port='5433')

with conn.cursor() as cursor:

sql = "DELETE FROM metagraph\_db.{}"

for table in tables\_for\_cleaning:

cursor.execute(sql.format(table))

conn.commit()

conn.close()

delete\_all()

from tqdm.auto import tqdm

from IPython.display import clear\_output

from time import time

import pandas as pd

excel\_name = 'nodes.csv'

result\_xml = 'result.xml'

xml\_name = 'DAMDID.xml'

time\_stamp = 'DAMDID'

nodes = 1000

metanodes = 500

edges = 1000

metaedges = 100

load\_times = []

save\_times = []

num = []

MAX\_VALUE = 2000

MIN\_VALUE = 100

STEP = 100

for nodes in range(MIN\_VALUE, MAX\_VALUE+1, STEP):

clear\_output(wait=False)

pbar = tqdm(total=MAX\_VALUE\*4, initial=(nodes - MIN\_VALUE)\*4)

pbar.update(STEP)

gen = generator.Generator()

loader = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

print('GENERATION')

gen.generateXML(xml\_name, nodes, metanodes, edges, metaedges)

pbar.update(STEP)

print('LOADING')

start\_time = time()

loader.load\_xml(xml\_name, time\_stamp)

end\_time = time()

load\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('SAVING')

start\_time = time()

loader.save\_xml(str(loader.all\_nodeid\_id['Main\_metanode\_id'][0]), time\_stamp, result\_xml)

end\_time = time()

save\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('DELETEING')

delete\_all()

num.append(nodes)

df = pd.DataFrame({"nodes\_num": num,

"load\_times": load\_times,

"save\_times": save\_times})

df.to\_csv(excel\_name)

excel\_name = 'metanodes.csv'

result\_xml = 'result.xml'

xml\_name = 'DAMDID.xml'

time\_stamp = 'DAMDID'

nodes = 1000

metanodes = 500

edges = 1000

metaedges = 500

attributes = 5

load\_times = []

save\_times = []

num = []

MAX\_VALUE = 2000

MIN\_VALUE = 100

STEP = 100

for metanodes in range(MIN\_VALUE, MAX\_VALUE+1, STEP):

clear\_output(wait=False)

pbar = tqdm(total=MAX\_VALUE\*4, initial=(metanodes - MIN\_VALUE)\*4)

pbar.update(STEP)

gen = generator.Generator()

loader = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

print('GENERATION')

gen.generateXML(xml\_name, nodes, metanodes, edges, metaedges, max\_number\_of\_atributes=attributes)

pbar.update(STEP)

print('LOADING')

start\_time = time()

loader.load\_xml(xml\_name, time\_stamp)

end\_time = time()

load\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('SAVING')

start\_time = time()

loader.save\_xml(str(loader.all\_nodeid\_id['Main\_metanode\_id'][0]), time\_stamp, result\_xml)

end\_time = time()

save\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('DELETEING')

delete\_all()

num.append(nodes)

df = pd.DataFrame({"metanodes\_num": num,

"load\_times": load\_times,

"save\_times": save\_times})

df.to\_csv(excel\_name)

excel\_name = 'edges.csv'

result\_xml = 'result.xml'

xml\_name = 'DAMDID.xml'

time\_stamp = 'DAMDID'

nodes = 1000

metanodes = 500

edges = 1000

attributes = 5

metaedges = 500

load\_times = []

save\_times = []

num = []

MAX\_VALUE = 2000

MIN\_VALUE = 100

STEP = 100

for edges in range(MIN\_VALUE, MAX\_VALUE+1, STEP):

clear\_output(wait=False)

pbar = tqdm(total=MAX\_VALUE\*4, initial=(edges - MIN\_VALUE)\*4)

pbar.update(STEP)

gen = generator.Generator()

loader = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

print('GENERATION')

gen.generateXML(xml\_name, nodes, metanodes, edges, metaedges, max\_number\_of\_atributes=attributes)

pbar.update(STEP)

print('LOADING')

start\_time = time()

loader.load\_xml(xml\_name, time\_stamp)

end\_time = time()

load\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('SAVING')

start\_time = time()

loader.save\_xml(str(loader.all\_nodeid\_id['Main\_metanode\_id'][0]), time\_stamp, result\_xml)

end\_time = time()

save\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('DELETEING')

delete\_all()

num.append(nodes)

df = pd.DataFrame({"edges\_num": num,

"load\_times": load\_times,

"save\_times": save\_times})

df.to\_csv(excel\_name)

excel\_name = 'metaedges.csv'

result\_xml = 'result.xml'

xml\_name = 'DAMDID.xml'

time\_stamp = 'DAMDID'

nodes = 1000

metanodes = 500

attributes = 5

edges = 1000

metaedges = 500

load\_times = []

save\_times = []

num = []

MAX\_VALUE = 2000

MIN\_VALUE = 100

STEP = 100

for metaedges in range(MIN\_VALUE, MAX\_VALUE+1, STEP):

clear\_output(wait=False)

pbar = tqdm(total=MAX\_VALUE\*4, initial=(metaedges - MIN\_VALUE)\*4)

pbar.update(STEP)

gen = generator.Generator()

loader = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

print('GENERATION')

gen.generateXML(xml\_name, nodes, metanodes, edges, metaedges, max\_number\_of\_atributes=attributes)

pbar.update(STEP)

print('LOADING')

start\_time = time()

loader.load\_xml(xml\_name, time\_stamp)

end\_time = time()

load\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('SAVING')

start\_time = time()

loader.save\_xml(str(loader.all\_nodeid\_id['Main\_metanode\_id'][0]), time\_stamp, result\_xml)

end\_time = time()

save\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('DELETEING')

delete\_all()

num.append(nodes)

df = pd.DataFrame({"metaedges\_num": num,

"load\_times": load\_times,

"save\_times": save\_times})

df.to\_csv(excel\_name)

excel\_name = 'attributes.csv'

result\_xml = 'result.xml'

xml\_name = 'DAMDID.xml'

time\_stamp = 'DAMDID'

nodes = 1000

metanodes = 500

attributes = 5

edges = 1000

metaedges = 500

load\_times = []

save\_times = []

num = []

MAX\_VALUE = 10

MIN\_VALUE = 1

STEP = 1

for attributes in range(MIN\_VALUE, MAX\_VALUE+1, STEP):

clear\_output(wait=False)

pbar = tqdm(total=MAX\_VALUE\*4, initial=(attributes - MIN\_VALUE)\*4)

pbar.update(STEP)

gen = generator.Generator()

loader = parcer.IOParcer(dbname='metagraph\_db', user='postgres', password='1', host='localhost', port='5433')

print('GENERATION')

gen.generateXML(xml\_name, nodes, metanodes, edges, metaedges, max\_number\_of\_atributes=attributes)

pbar.update(STEP)

print('LOADING')

start\_time = time()

loader.load\_xml(xml\_name, time\_stamp)

end\_time = time()

load\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('SAVING')

start\_time = time()

loader.save\_xml(str(loader.all\_nodeid\_id['Main\_metanode\_id'][0]), time\_stamp, result\_xml)

end\_time = time()

save\_times.append(end\_time - start\_time)

pbar.update(STEP)

print('DELETEING')

delete\_all()

num.append(attributes)

df = pd.DataFrame({"attributes\_num": num,

"load\_times": load\_times,

"save\_times": save\_times})

df.to\_csv(excel\_name)