Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Факультет информационных систем и технологий

Кафедра: «Вычислительная техника»

Дисциплина: «Параллельное программирование»

Отчет по практическим занятиям на тему:

«Исследование параллельных реализаций обработки элементов дополненной реальности»

Выполнил

студент группы ИВТАСмд-11

Кондратьев П. С.

Проверил:

Негода В. Н.

г. Ульяновск, 2022

Оглавление

[Введение 2](#_Toc99656110)

[Постановка задачи 4](#_Toc99656111)

[Произведем оптимизацию загрузки файлов в Postgres 4](#_Toc99656112)

[Произведем оптимизацию запросов в Postgres 9](#_Toc99656113)

[Список источников 16](#_Toc99656114)

# Введение

Современный инженер имеет дело со сложными техническими системами. В рамках одного методологического подхода [1], к решению таких систем, могут существовать ряд разных теорий, взглядов, положений, имеющих одну и ту же концептуальную основу исследования.

Значительное место в организации исследования на предприятиях занимает работа с информацией, ее анализ, идентификация, классификация и обработка данных. Банк данных предприятий включает большую группу документов, которые могут являться информационной основой исследования. Это устав, баланс и другие документы финансовой отчетности, структура, приказы, отчеты функциональных служб, планы, протоколы, нормативные документы, рекламации и т.д. Получение данных экспертной оценки [2] и социологические опросы существенно расширят проблемное поле исследования, позволят восполнить недостаток информации, полученной из других источников.

При изучении закономерностей функционирования сложных систем приходится прибегать к различным способам познания. Основным методом эмпирического познания стал эксперимент [3], т.е. совокупность операций и в случае необходимости воздействий на изучаемый объект, выполняемых для получения информации о нем на основе результатов опытов.

Для эксперимента, когда количество задач, которые приложение должно выполнять одновременно, увеличивается, есть несколько вариантов решения этой проблемы:

* Запустить больше машин;
* Приложение может использовать больше процессов на одной машине;
* Используйте больше процессов операционной системы;
* Запуск большего количества потоков операционной системы для каждого процесса.

# Постановка задачи

Для исследования возможностей параллельных реализаций экспериментальных исследований в проектировании автоматизированных систем, для функциональной реализации учебных планов преподавателей вуза, реализуем многопоточную обработку plx файлов (учебных планов), используя стандартные библиотеки, такие как threading, multiprocessing. В рамках данной работы также будет проводиться эксперимент над функцией выбора всех дисциплин, которые ведет кафедра. Эта функция имеет вид:

fSelectDisc: splans x sdepartment => sdisc\*, где

sdisc\* ∈ sdiscipline

Таким образом, не взирая на степень вовлечения содержания учебного плана в реализацию тех или иных функций организации учебного процесса, все они обслуживают так или иначе реализацию совокупности учебных планов.

**В ходе выполнения работы решим следующие задачи:**

1. Произведем оптимизацию загрузки файлов в Postgres с помощью Multipocessing
2. Произведем оптимизацию запросов в Postgres

# Произведем оптимизацию загрузки файлов в Postgres

Модуль threading обработки использует потоки, модуль multiprocessing обработки использует процессы. Разница в том, что потоки выполняются в одном и том же пространстве памяти, в то время как процессы имеют отдельную память. Это немного усложняет совместное использование объектов между процессами с многопроцессорной обработкой. Поскольку потоки используют одну и ту же память, необходимо принять меры предосторожности, иначе два потока будут записывать в одну и ту же память одновременно. Для этого и предназначена глобальная блокировка интерпретатора.

Эксперименты организуются как двухфакторные: первым фактором является число параллельно работающих потоков, вторым – размер рабочей нагрузки. Варьирование размера рабочей нагрузки обеспечивается через изменение количества XML-файлов.

Для создания достаточно большого числа файлов создан генератор, который формирует заданное число экземпляров учебных планов путем создания копий реальных учебных планов.

План эксперимента представляет собой сочетание всех значений, указанных выше факторов.

|  |
| --- |
| def processesed(procs, config, listFiles):  # procs - количество ядер    processes = []    # делим вычисления на количество ядер  for proc in range(procs):  p = multiprocessing.Process(target=main, args=(len(listFiles[proc]), proc, config, listFiles[proc]))  processes.append(p)  p.start()  # Ждем, пока все ядра завершат свою работу.  for p in processes:  p.join()  if \_\_name\_\_ == "\_\_main\_\_":  config = configparser.ConfigParser()  config.read("conf.ini")    filesDirectory = list(files("./genfiles/files"))  n\_proc = multiprocessing.cpu\_count()  listFilesChunks = chunks(filesDirectory, n\_proc)  start = time.time()    processesed(n\_proc, config, listFilesChunks)    end = time.time()  print(f"Всего {n\_proc} ядер в процессоре")  print(f"Итого: ", end - start) |

Для организации параллелизма на Python была использована библиотека multiprocessing, для замера времени – функция time(), которая возвращает время в секундах с начала эпохи как число с плавающей запятой.

На основе средств библиотеке multiprocessing процессы порождаются путём создания объекта Process и последующего вызова его метода start().

Последовательность операций, выполняемых в одной итерации программы проведения экспериментов, такова:

2. Разбиение множества файлов рабочей нагрузки на части, каждая из которых реализуется в одном потоке;

3. Порождение процессов и их запуск с фиксацией стартового времени;

4. В каждом потоке в объекты Python-программы вводятся данные из xml (plx) файлов соответствующей части рабочей нагрузки и затем выводятся через SQL-запросы в базу данных с последующей фиксацией времени завершения.

В экспериментах использовался компьютер на базе процессора I5 – 5700U (2,5 ГГц, 2 ядра, 4 потока), имеющий оперативную память 8 GB.

В серию экспериментов были вовлечены файлы с размером 3082КБ. Результаты измерения времени вычисления работы программы при различных сочетаниях значений факторов для этой серии экспериментов приведены в таблице 2.1. График зависимости коэффициента ускорения от числа потоков при различных значениях рабочей нагрузки приведен на рис. 2.2.

Таблица 2.1. Результаты замеров времени (ceк) вычисления

при размере одного файла 3082КБ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **10** | **25** | **50** | **100** | **150** |
| **1** | 27,39569044 | 70,04511309 | 145,63227749 | 260,33342431 | 400,47826409 |
| **2** | 16,99670386 | 41,44480205 | 79,63571000 | 178,74103045 | 262,01687837 |
| **3** | 14,01764774 | 34,79354596 | 69,55757856 | 147,78097653 | 230,82868576 |
| **4** | 10,22254515 | 28,32334232 | 57,35386205 | 120,69074678 | 170,14653563 |

Таблица 2.1.1. Результаты замеров времени (ceк) вычисления

при размере одного файла 3082КБ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **10** | **25** | **50** | **100** | **150** |
| **1** | 27,39569044 | 70,04511309 | 145,63227749 | 260,33342431 | 400,47826409 |
| **2** | 16,99670386 | 41,44480205 | 79,63571000 | 178,74103045 | 262,01687837 |
| **3** | 14,01764774 | 34,79354596 | 69,55757856 | 147,78097653 | 230,82868576 |
| **4** | 10,22254515 | 28,32334232 | 57,35386205 | 120,69074678 | 170,14653563 |

Рис. 2.2. Графики коэффициентов ускорения при варьировании числа потоков

Эффект от распараллеливания довольно заметен, несмотря на то, что значительную долю затрат занимает собственно ввод из файлов. Эффект от распараллеливания для четырех потоков достигает 2,7. Учитывая, что в используемой платформе всего 2 ядра и дополнительные два потока в режиме hyperthreading разделяют ресурсы этих двух ядер, ускорение можно считать существенным.

# Произведем оптимизацию запросов в Postgres

1. **Запросы к БД**
2. Теперь можно приступить непосредственно к тестированию. Команда для запуска такого теста будет выглядеть следующим образом:
3. *sudo -u postgres pgbench -f script.sql postgres*
4. В файле script.sql храниться наш запрос, а postgres – это название бд с которой мы работает в данный момент.
5. Будем запускать один и тот же скрипт несколько раз, так как при последующих запусках может отработать кэш, что заметно повлияет на результаты тестирования.
6. Перейдем к **перовому** экспериментальному тестированию.

*sudo -u postgres pgbench -c -j -f script.sql postgres , где*

После проведения первого эксперимента зафиксируем количество соединений (50) и потоков (4).

*sudo -u postgres pgbench -P 10 -T 30 -c 50 -j 4 -f script.sql postgres, где*

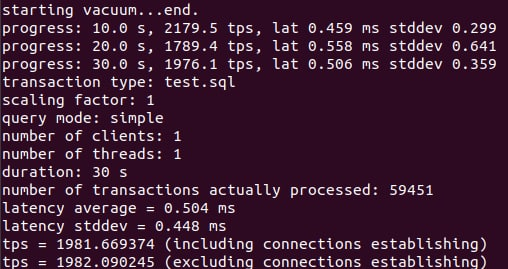


Рис. 1. Первый эксперимент, зависимость TPS от рабочей нагрузки

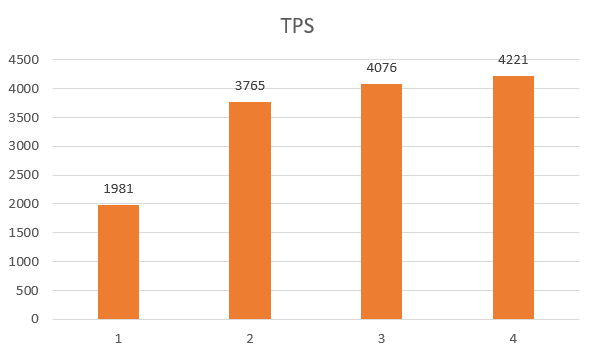


Рис. 2. Первый эксперимент, зависимость TPS от рабочей нагрузки

Из графика мы видим, что первые два столбца имеют линейную зависимость от рабочей нагрузки. При 3-х и более соединений, мы видим, что возрастания TPS не происходит, это говорит нам о том, что мы достигли порогового значения.

1. Перейдем ко **второму** экспериментальному тестированию.

*sudo -u postgres pgbench -P 10 -T 30 -c 50 -j 4 -f test.sql postgres*

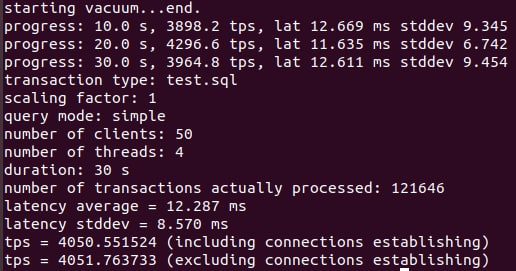


Рис. 3. Второй эксперимент

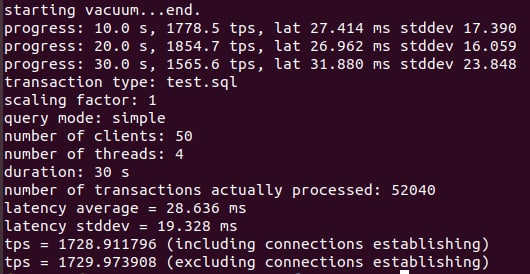


Рис. 4. Второй эксперимент, удвоили кол-во данных в таблицах

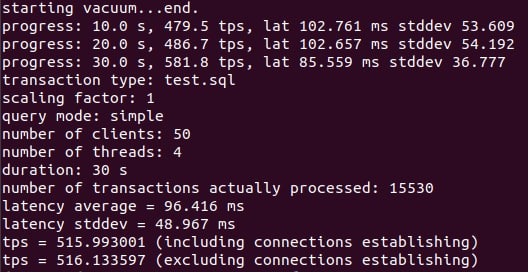


Рис.5. Второй эксперимент, учетверили кол-во данных

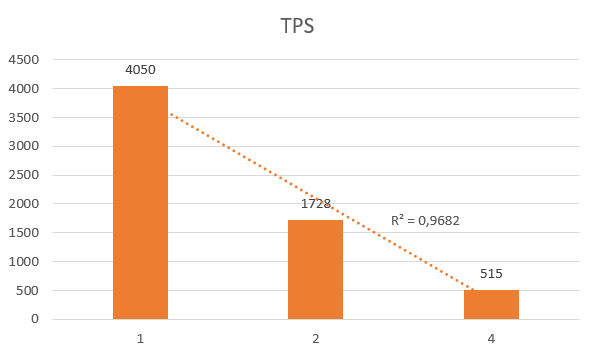


Рис. 6. Второй эксперимент – исследование зависимости TPS от количества строк в базе данных

Из гистограммы видно, что с увеличением объема данных линейно падает TPS. Построив линию тренда, мы убедились то, что стремиться к 1, а это значит, что зависимость действительно линейна.

1. Перейдем ко **третьему** экспериментальному тестированию.
2. *sudo -u postgres pgbench -P 10 -T 30 -c 50 -j 4 -f test.sql postgres*

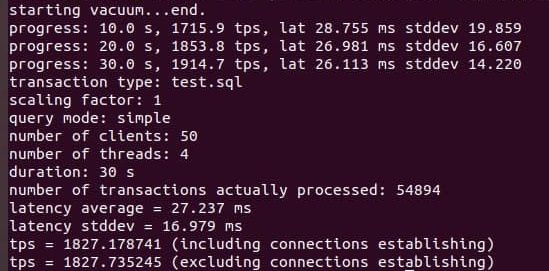


Рис. 7. Третий эксперимент, добавили индексы

При добавлении индексов на поля, которые используется в тестовом запросе, мы видим значительное увеличение значения TPS. Проверить то, что используются индексы, можно при использовании explain, который показывает план запроса.

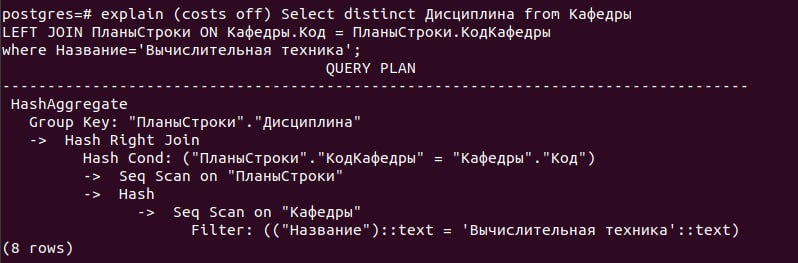


Рис.8. План запроса без индексов

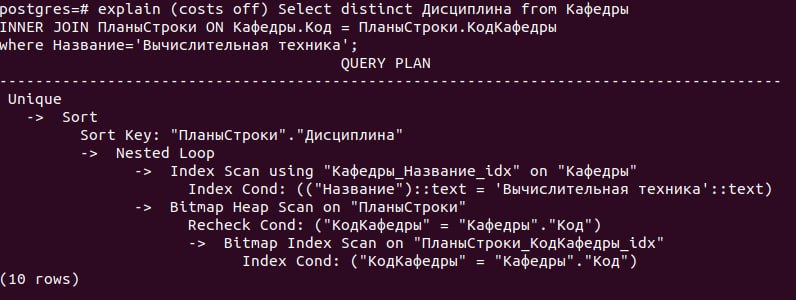


Рис.8. План запроса с индексами

После проведения экспериментов с запросами, можно было бы попробовать делать параллельные запросы к бд, но на простых запросах мы не получим выигрыш по времени.

PostgreSQL может вырабатывать такие планы запросов, которые будут задействовать несколько CPU, чтобы получить ответ на запросы быстрее. Эта возможность называется распараллеливанием запросов. Для многих запросов параллельное выполнение не даёт никакого выигрыша, либо из-за ограничений текущей реализации, либо из-за принципиальной невозможности построить параллельный план, который был бы быстрее последовательного. Однако для запросов, в которых это может быть полезно, распараллеливание часто даёт очень значительное ускорение. Многие такие запросы могут выполняться в параллельном режиме как минимум двое быстрее, а некоторые — быстрее в четыре и даже более раз. Обычно наибольший выигрыш можно получить с запросами, обрабатывающими большой объём данных, но возвращающими пользователю всего несколько строк.

**Запросы к Plx**

Есть несколько вариантов, первый вариант является частью оптимизации загрузки файлов в Postgres, в котором происходит парсинг файла с последующим сбором данных. Второй вариант XQuery – язык программирования, позволяющий делать выборки из XML данных.

Ниже приведен пример запроса: Выбора всех дисциплин которая ведет кафедра.

|  |
| --- |
| def main(listFiles):  with open("log", "a", encoding="utf-8") as file:  for fileName in listFiles:  tree = ET.parse("files/" + fileName)  root = tree.getroot()    tagList = []  for elem in tree.iter():  tagList.append(realName(elem.tag))  tagList = list(set(tagList))  codeDep = ''  for dep in root.iter('{http://tempuri.org/dsMMISDB.xsd}Кафедры'):  if(dep.attrib['Название'] == 'Вычислительная техника'):  codeDep = dep.attrib['Код']  for dis in root.iter('{http://tempuri.org/dsMMISDB.xsd}ПланыСтроки'):  try:  if (dis.attrib['КодКафедры'] == codeDep):  file.write(f"{dis.attrib['Дисциплина']}\n")  except (Exception) as error:  pass  break  >> Итого: 0.00886572456359863 sec |

**Вывод**: результаты загрузки объектов с использованием различных режимов позволяют говорить о том, что скорость композиции мультипроцессорного режима чуть больше скорости других вариантов загрузки объектов.

Подведя итоги экспериментов можно сделать следующие заключения. Многопроцессорность (multiprocessing) следует использовать при выполнение большого количества тяжелых вычислений, которые можно разделить на группы.

# Список источников

1. Основной методологический способ – это подход. URL: https://molotokrus.ru/osnovnoy-metodologicheskiy-sposob-eto-podhod/
2. Экспертное оценивание. URL: https://ru.wikipedia.org/wiki/Экспертное\_оценивание
3. Эксперимент как важнейший способ эмпирического познания. URL:https://studref.com/483556/filosofiya/eksperiment\_vazhneyshiy\_sposob\_empiricheskogo\_poznaniya
4. Эффективных примера асинхронных представлений Django без сна [Электронный ресурс] - Режим доступа: https://pythobyte.com/django-async-views-examples-18019/
5. Многопоточность в Python. [Электронный ресурс] - Режим доступа: https://docs-python.ru/tutorial/mnogopotochnost-python/