Министерство образования и науки Российской Федерации

[Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования   
Московский государственный технический университет им. Н.Э. Баумана](http://elibrary.ru/publisher_about.asp?pubsid=787)

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



Курс  
«Имитационное моделирование дискретных процессов»

Отчёт по домашнему заданию

Леонтьев Алексей   
ИУ5-74

Москва - 2014 год

Оглавление

[Постановка задачи 3](#_Toc408093103)

[Описание модели 3](#_Toc408093104)

[Клиент 3](#_Toc408093105)

[Фронтенд (балансировщик) 3](#_Toc408093106)

[Бэкенд сервера 4](#_Toc408093107)

[Сервер СУБД 4](#_Toc408093108)

[Реализация 5](#_Toc408093109)

[Статистический анализ 7](#_Toc408093110)

[Описание эксперимента 7](#_Toc408093111)

[Задача 7](#_Toc408093112)

[Результаты 7](#_Toc408093113)

# Постановка задачи

Смоделировать взаимодействие множества клиентов с распределённой вычислительной системой, состоящей из одного балансирующего сервера (фронтенд), переменного количества бэкенд-серверов, а также сервера СУБД. Найти минимальное количество бэкенд-серверов при минимальном среднем времени ответа.

# Описание модели

Схема модели представлена на Рис 1.



**Рис 1.** Схема модели

Модель состоит из нескольких клиентов (Клиент 1 … Клиент 1000), отправляющих различные запросы через интернет на фронтенд-сервер, который либо отправляет ответ клиенту, либо отправляет запрос дальше – на 1 из веб-серверов (Бэкенд 1 … Бэкенд N), те в свою очередь делают несколько запросов к серверу СУБД, затем возвращают запрос на фронтенд для возможного кеширования.

Рассмотрим каждый элемент модели подробнее.

## Клиент

Клиент последовательно отправляет несколько разных запросов на фронтенд-сервер. Затем, после паузы всё повторяется. Здесь же происходит измерение времени прохождения запроса через всю систему.

## Фронтенд (балансировщик)

В задачи этого сервера входит:

1. определение типа запроса:
   1. динамический  
      нужно отправить на один из бэкендов или взять из кеша
   2. статический  
      нужно взять из файловой системы
2. балансировка нагрузки по одному из алгоритмов: RoundRobin или LeastConnections
3. кэширование ответов с бэкендов  
   для незарегистрированных пользователей
4. отдача статических файлов из ФС

## Бэкенд сервера

Данный сервер выполняет запросы к серверу СУБД в соответствии с данными в запросе и отправляет ответ обратно клиенту через кэширующий фронтенд.

## Сервер СУБД

Обрабатывает запросы от бэкендов, запросы типизированы и различаются временем обработки. Максимальное количество одновременно обрабатываемых запросов ограничено.

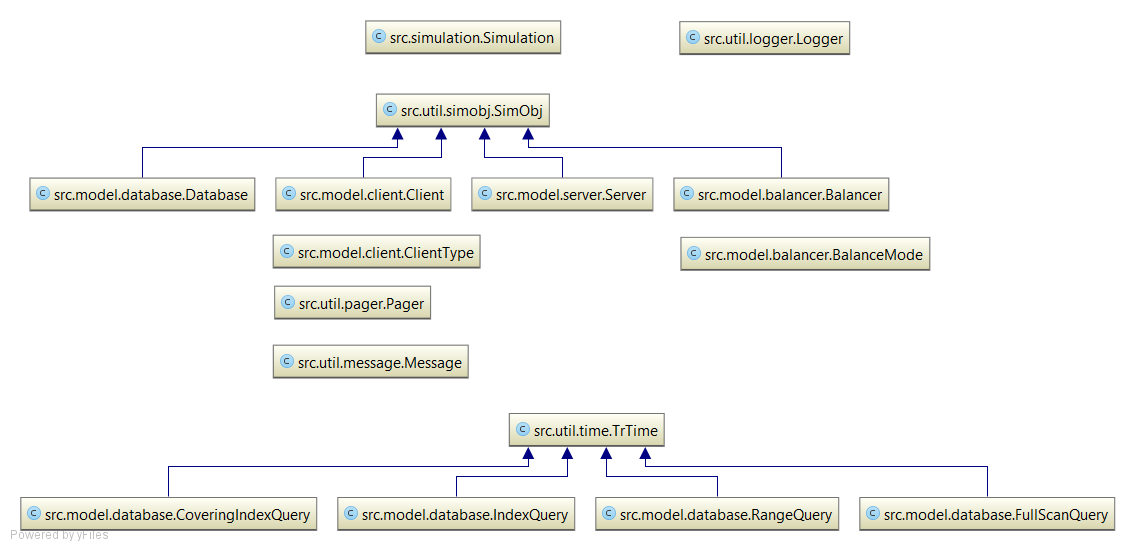
# Реализация

Для моделирования используется язык программирования Python 3.4 и библиотека simpy. Отчёт строится на JavaScript/HTML/CSS при помощи библиотек jQuery и flot и представляет собой интерактивную HTML-страничку.

Эксперименты проводятся параллельно в нескольких процессах (количество процессов равно количеству ядер процессора).

Объём кода модели слишком велик, чтобы включать его целиком в отчёт.

Диаграмма классов приведена на Рис 2.



**Рис 2.** Диаграмма классов

Параметры модели, описанные в simulation.py:

*# create db*db = Database(env, self.logger, 0, dict(  
 max\_connections=100  
))  
  
*# create bd query pattern*query\_pattern = (  
 dict(type=FullScanQuery, count=**lambda**: maybe.maybe(0, 1, 90)),  
 dict(type=CoveringIndexQuery, count=**lambda**: random.randint(0, 10)),  
 dict(type=IndexQuery, count=**lambda**: random.randint(0, 2)),  
 dict(type=RangeQuery, count=**lambda**: random.randint(0, 2)),  
)  
  
*# create servers*servers = []  
**for** i **in** range(0, servers\_count):  
 server = Server(env, self.logger, i, dict(  
 cores=20,  
 db=db,  
 render\_time=TrTime(30, 100),  
 query\_pattern=query\_pattern,  
 db\_latency\_time=TrTime(20, 30),  
 balancer\_latency\_time=TrTime(10, 20)  
 ))  
  
 servers.append(server)  
 server.start()  
  
*# create balancer*balancer = Balancer(env, self.logger, 0, dict(  
 mode=BalanceMode.ROUND\_ROBIN,  
 servers=servers,  
 cache\_size=100,  
 cache\_time=1\*30\*1000, *# half of minute* render\_time=TrTime(2, 8),  
 balance\_time=TrTime(1, 2),  
 sender\_processes=4,  
 receiver\_processes=4,  
 max\_clients=100000))  
balancer.start()  
  
*# create pager*pager = Pager(env, dict(  
 dynamic\_pages\_count=10,  
 static\_files\_count=50,  
 static\_files\_per\_page=3  
))  
  
*# create clients*clients = []  
**for** i **in** range(0, 1000):  
 client = Client(env, self.logger, i, dict(  
 balancer\_pipe=balancer.get\_clients\_pipe(),  
 pager=pager,  
 type=ClientType.PC,  
 guest=maybe.maybe(**True**, **False**, 50),  
 page\_idle\_time=TrTime(1\*1000, 3\*1000),  
 request\_idle\_time=TrTime(0.1\*1000, 0.2\*1000),  
 uplink\_speed=TrTime(1, 3),  
 downlink\_speed=TrTime(10, 30)  
 ))  
 clients.append(client)  
 client.start()

# Статистический анализ

## Описание эксперимента

Эксперимент состоит в многократном запуске симуляции модели при разном количестве бэкенд-серверов.

## Задача

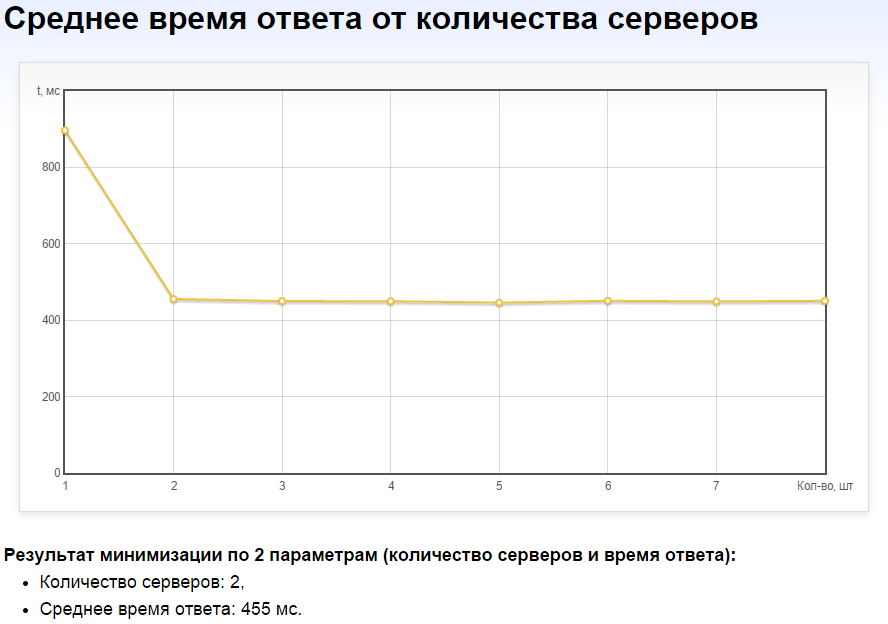
Задача состоит в нахождении минимального количества бэкенд-серверов при минимальном среднем времени ответа (клиент-система-клиент). То есть стоит задача минимизации по двум параметрам одновременно: по времени ответа и по количеству серверов.

Применимость задачи к реальным системам заключается в следующем: нужно потратить как можно меньше денег на закупку серверов, при этом минимизировав неудобства клиентам.

## Результаты

На Рис 3. приведён график зависимости среднего времени ответа от количества бэкенд-серверов.

Из него видно, что оптимальным количеством серверов при данных параметрах системы является 2 сервера, дальнейшее увеличение их количества не вносит сколько-нибудь значимого улучшения времени ответа для клиентов.



**Рис 3.** Зависимость среднего времени ответа от количества бэкенд-серверов