Отчет по лабораторной работе 3

Двухсервисная модель с эластичным трафиком

Генералов Даниил, 1032212280

0. теоретическая информация

Проаназируем соту сети емкостью C. Пусть пользователи генерируют запросы на передачу данных двух типов. Запросы на передачу данных представляют собой ПП с интенсивностью $\lambda_i, i = 1, 2$. Средняя длина передаваемого файла $\theta_i, i = 1, 2$. Минимальная емкость, необходимая для передачи данных равна $b_i, i = 1, 2$

- C -- пиковая пропускная способность соты;
- λ_1, λ_2 -- интенсивность поступления запросов 1, 2-го типа;
- θ_1, θ_2 -- длина передаваемого файла 1, 2-го типа;
- ρ_1, ρ_2 -- интенсивность предложенной нагрузки, создаваемой запросами 1, 2-го типа;
- a_1, a_2 -- доля нагрузки, создаваемой запросами 1, 2-го типа, которая приходится на единицу пропускной способности;
- b_1, b_2 -- минимальное требование к ресурсам сети, необходимое для передачи данных 1, 2-го типа;
- $X_i(t), i = 1, 2$ -- число запросов, обслуживаемых в системе в момент времени $t, t \geq 0$
- $X_i(t), i = 1, 2$ -- случайный процесс (СП), описывающий функционирование системы в момент времени t, $t \geq 0$;
- X -- пространство состояний системы;
- n_1, n_2 -- число передаваемых блоков данных 1, 2 типа;
- B_1, B_2 -- множество блокировок запросов 1, 2-го типа;
- S_1, S_2 -- множество приема запросов 1, 2-го типа.

Пространство состояний системы: $X = \{(n_1, n_2) : n_1 >= 0, n_2 >= 0\}$

Множество блокировок запросов на передачу данных: $B_i = \{\varnothing\}, i = 1, 2$

Множество приема запросов на передачу данных:
$S_i = \overline{B_i} = X \setminus B_i = \{0, 1, 2, \dots\}, i = 1, 2$

Система уравнений глобального баланса (СУГБ):

$$\left(\lambda_1+\lambda_2+\frac{C}{(n_1+n_2)\theta_i}+\frac{C}{(n_1+n_2)\theta_2}n_2\right)\times p(n1,n2)=$$

$$=\lambda_1 p(n_1-1,n_2)\times U(n_1)+\lambda_2 p(n_1,n_2-1)\times U(n_2)+$$

$$+\frac{C}{(n_1+1+n_2)\theta_1}(n_1+1)p(n_1+1,n_2)+$$

$$+\frac{C}{(n_1+n_2+1)\theta_2}(n_2+1)p(n_1,n_2+1),(n_1,n_2)\in X$$

Чтобы выписать систему уравнений частичного баланса (СУЧБ), проверим критерий Колмогорова. Рассмотрим произвольный замкнутый контур. Рассмотрим произведение интенсивностей переходов:

- по часовой стрелке $\frac{n_2}{n_1+n_2}\frac{C}{\theta_2}\frac{n_1}{n_1+n_2-1}\frac{C}{\theta_1}\lambda_1\lambda_2$;
- против часовой стрелке $\frac{n_2}{n_1+n_2}\frac{C}{\theta_1}\frac{n_2}{n_1+n_2-1}\frac{C}{\theta_2}\lambda_1\lambda_2$;

Произведения равны. Критерий выполнен, следовательно, СП $(X_1(t),X_2(t))$, описывающий поведение системы является обратимым марковским процессом, СУЧБ существует.

СУЧБ:

- $p(n_1,n_2)\frac{C}{(n_1+n_2)\theta_1}n_1=\lambda_1 p(n_1-1,n_2),n_1>0$,
- $p(n_1,n_2)\frac{C}{(n_1+n_2)\theta_2}n_2=\lambda_2 p(n_1,n_2-1),n_2>0$,

где $(n_1,n_2)\in X$.

Обозначим $\rho_i=\lambda_i\theta_i, a_i=\frac{\rho_i}{C},\rho_i<C,i=1,2$.

Стационарное распределение вероятностей состояний системы:

$p(n_1,n_2)=\frac{a_1^{n_1}}{n_1!}\frac{a_2^{n_2}}{n_2!}(n_1+n_2)!p(0,0)$

где $p(0,0)=\left(\sum_{(n_1,n_2)\in X}(n_1+n_2)!\frac{a_1^{n_1}}{n_1!}\frac{a_2^{n_2}}{n_2!}\right)^{-1}$

Основные вероятностные характеристики (ВХ) модели:

- Вероятность блокировки по времени $E_i,i=1,2$ запроса на передачу данных первого/второго типа: $E_1=E_2=0$;
- Среднее число $\overline{N_i},i=1,2$ обслуживаемых в системе запросов на передачу данных первого/второго типа: $\overline{N_i}=\lambda_i\frac{\theta_i}{(\theta_1\lambda_1+\theta_2\lambda_2)},i=1,2$

- Среднее время $T_i, i = 1, 2$ обслуживания запроса на передачу данных первого/второго типа: $T_i = \frac{\overline{N_i}}{\lambda_i}$

1. подключение библиотек, определение функций

Для расчета больших факториалов нам потребуется длинная арифметика, а для рисования графиков -- библиотека для визуализации данных.

In [2]:
```
:dep num = { version = "^0.4.3" }
:dep plotters = { version = "^0.3.6", default-features = false, features = [

extern crate num;
use num::BigRational as R;
use num::BigInt as I;
use num::BigUint as U;
use num::Integer;
use num::traits::ConstZero;
use num::FromPrimitive;
use num::ToPrimitive;

extern crate plotters;
use plotters::prelude::*;
```

Для удобства конвертации стандартных чисел в числа длинной арифметики используются helper-функции.

In [3]:
```
fn u(i: usize) -> U {
    U::from_usize(i).unwrap()
}

fn rr(i: f64) -> R {
    R::from_float(i).unwrap()
}
```

Для вычисления факториала нет стандартной функции, и очевидные подходы не работают с длинной арифметикой, поэтому эта функция считает это за нас.

In [4]:
```
fn factorial(n: &U) -> R {
    let mut c = n.clone();
    let one = I::from_i8(1).unwrap();
    let mut out = R::new(one.clone(), one.clone());
    while c > U::ZERO {
        out *= R::new(I::from_biguint(num::bigint::Sign::Plus, c.clone()), o
        c -= 1u32;
    }
    out
}
```

Эта функция отображает график функции, принимая на вход список X-Y пар.

In [5]:

```
fn draw_chart(data: &Vec<(f32, f32)>, name: impl ToString) -> plotters::evcx
    let minx = data.iter().min_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(std
    let maxx = data.iter().max_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(std
    let miny = data.iter().min_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(std
    let maxy = data.iter().max_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(std
    let figure = evcxr_figure((640, 480), |root| {
        root.fill(&WHITE)?;
        let mut chart = ChartBuilder::on(&root)
            .caption(name.to_string(), ("Arial", 50).into_font())
            .margin(5)
            .x_label_area_size(30)
            .y_label_area_size(30)
            .build_cartesian_2d(minx..maxx, miny..maxy)?;

        chart.configure_mesh().draw()?;

        chart.draw_series(LineSeries::new(
            data.clone(),
            &RED,
        )).unwrap();

        // chart.configure_series_labels()
        //     .background_style(&WHITE.mix(0.8))
        //     .border_style(&BLACK)
        //     .draw()?;
        Ok(())
    });
    return figure;
}

fn draw_chart_2(data1: &Vec<(f32, f32)>, data2: &Vec<(f32, f32)>, name: impl
    let minx1 = data1.iter().min_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(s
    let maxx1 = data1.iter().max_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(s
    let miny1 = data1.iter().min_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(s
    let maxy1 = data1.iter().max_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(s
    let minx2 = data2.iter().min_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(s
    let maxx2 = data2.iter().max_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(s
    let miny2 = data2.iter().min_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(s
    let maxy2 = data2.iter().max_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(s

    let minx = minx1.min(minx2);
    let maxx = maxx1.max(maxx2);
    let miny = miny1.min(miny2);
    let maxy = maxy1.max(maxy2);

    let figure = evcxr_figure((640, 480), |root| {
        root.fill(&WHITE)?;
        let mut chart = ChartBuilder::on(&root)
            .caption(name.to_string(), ("Arial", 50).into_font())
            .margin(5)
            .x_label_area_size(30)
            .y_label_area_size(30)
            .build_cartesian_2d(minx..maxx, miny..maxy)?;

        chart.configure_mesh().draw()?;
```

```
        chart.draw_series(LineSeries::new(
            data1.clone(),
            &RED,
        )).unwrap();
        chart.draw_series(LineSeries::new(
            data2.clone(),
            &BLUE,
        )).unwrap();

        // chart.configure_series_labels()
        //     .background_style(&WHITE.mix(0.8))
        //     .border_style(&BLACK)
        //     .draw()?;
        Ok(())
    });
    return figure;
}
```

Для вычисления стационарного распределения можно заметить, что в формуле есть общая часть $\frac{\alpha_1^{n_1}\alpha_2^{n_2}}{n_1!n_2!}(n_1+n_2)!$. Ее мы вынесем в отдельную функцию.

In [6]:
```
fn common_part(alpha1: &R, alpha2: &R, n1: usize, n2: usize) -> R {
    (alpha1.pow(n1 as i32) / factorial(&u(n1))) *
    (alpha2.pow(n2 as i32) / factorial(&u(n2))) *
    factorial(&u(n1+n2))
}

fn stationary_distribution_at_zero(n1_max: usize, n2_max: usize, alpha1: R,
    let mut sum = R::from_float(0.0).unwrap();
    for n1 in 0..=n1_max {
        for n2 in 0..=n2_max {
            // println!("{n1} {n2}");
            sum += common_part(&alpha1, &alpha2, n1, n2);
        }
    }

    R::from_float(1.0).unwrap()/sum
}

fn stationary_distribution_at(zero: R, n1: usize, n2: usize, alpha1: R, alph
    zero * common_part(&alpha1, &alpha2, n1, n2)
}
```

2. входные параметры

Здесь задаются параметры, которые определяют модель. Чтобы попробовать запустить вычисления с другими значениями, вы можете поменять эту ячейку и перезапустить ее и все ячейки ниже.

При разработке я использовал следующие значения для теста:

- $\lambda 1 = 8$
- $\lambda 2 = 6$
- $\theta_1 = 2$
- $\theta_2 = 3$
- $C = 10$
- $n_1 = 20$
- $n_2 = 30$

Также мы здесь считаем распределение вероятности в ячейке (0,0), потому что относительно нее нормализуются все остальные ячейки.

In [7]:
```
let lambda1 = 8.;
let lambda2 = 6.;
let theta1 = 2.;
let theta2 = 3.;
let c = 10;
let n1_max = 20;
let n2_max = 30;

let rho1 = lambda1 * theta1;
let rho2 = lambda2 * theta2;
let a1 = rho1 / c as f64;
let a2 = rho2 / c as f64;

let zero = stationary_distribution_at_zero(n1_max, n2_max, rr(a1), rr(a2));
zero.to_f64()
```

Out[7]:
```
Some(1.9345027205245324e-26)
```

Затем мы собираем данные о распределении вероятностей в остальных ячейках от (0,0) до (n_1, n_2) в один двухмерный массив. Поскольку это распределение вероятностей, то сумма всех значений должна быть равна 1.

In [8]:
```
let mut v = vec![];
for n1 in 0..=n1_max {
    let mut r = vec![];
    for n2 in 0..=n2_max {
        r.push(stationary_distribution_at(zero.clone(), n1, n2, rr(a1), rr(a
    }
    v.push(r)
}
let sum: R = v.iter().map(|v| v.iter().sum::<R>()).sum();
println!("sum should be 1: {sum}");
```

```
sum should be 1: 1
```

In [13]:
```
for row in v.iter() {
    println!("{:?}", row.iter().map(|v| v.to_f64().unwrap()).map(|v| format!
}
```

["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000

```
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000000"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00
00000", "0.0000001"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000001", "0.00
00002", "0.0000005"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000000", "0.0000001", "0.0000002", "0.0000004", "0.00
00011", "0.0000027"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0000", "0.0000000", "0.0000001", "0.0000003", "0.0000008", "0.0000022", "0.00
00056", "0.0000145"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000
0001", "0.0000002", "0.0000005", "0.0000014", "0.0000038", "0.0000104", "0.00
00276", "0.0000729"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000001", "0.000
0003", "0.0000008", "0.0000022", "0.0000061", "0.0000172", "0.0000475", "0.00
01297", "0.0003501"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000000", "0.0000001", "0.0000003", "0.000
0010", "0.0000030", "0.0000089", "0.0000258", "0.0000739", "0.0002089", "0.00
05835", "0.0016105"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000000", "0.0000001", "0.0000004", "0.0000012", "0.000
0038", "0.0000116", "0.0000350", "0.0001043", "0.0003059", "0.0008848", "0.00
25262", "0.0071240"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000000", "0.0000001", "0.0000004", "0.0000013", "0.0000043", "0.000
```

```
0137", "0.0000432", "0.0001339", "0.0004078", "0.0012234", "0.0036178", "0.01
05541", "0.0303957"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
000", "0.0000001", "0.0000004", "0.0000013", "0.0000044", "0.0000148", "0.000
0486", "0.0001566", "0.0004960", "0.0015454", "0.0047391", "0.0143190", "0.04
26606", "0.1254222"]
["0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.000000
0", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.00000
00", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000000", "0.0000
001", "0.0000003", "0.0000011", "0.0000041", "0.0000144", "0.0000496", "0.000
1670", "0.0005512", "0.0017858", "0.0056870", "0.0178191", "0.0549848", "0.16
72296", "0.5016887"]
```

Out[13]: ()

3. рассчеты

Среднее число обслуживаемых запросов можно расчитать относительно исходных параметров модели.

In [14]:
```
let N1 = lambda1 * (theta1) / (theta1 * lambda1 + theta2 * lambda2);
let N2 = lambda2 * (theta2) / (theta1 * lambda1 + theta2 * lambda2);
println!("average number of requests in flight: {N1}, {N2}");
```

```
average number of requests in flight: 0.47058823529411764, 0.5294117647058824
```

Среднее время обслуживания запроса является обратной величиной к количеству запросов, и замвисит от интенсивности поступления этих запросов.

In [15]:
```
let T1 = N1 / lambda1;
let T2 = N2 / lambda2;
println!("Avg service time: {T1}, {T2}");
```

```
Avg service time: 0.058823529411764705, 0.08823529411764706
```

4. графики

Для того, чтобы построить графики среднего количества запросов от интенсивности поступления запросов, нужно зафиксировать одну из λ и изменять другую, вычисляя показатели при каждом значении. Сначала мы будем изменять λ_1, а затем λ_2.

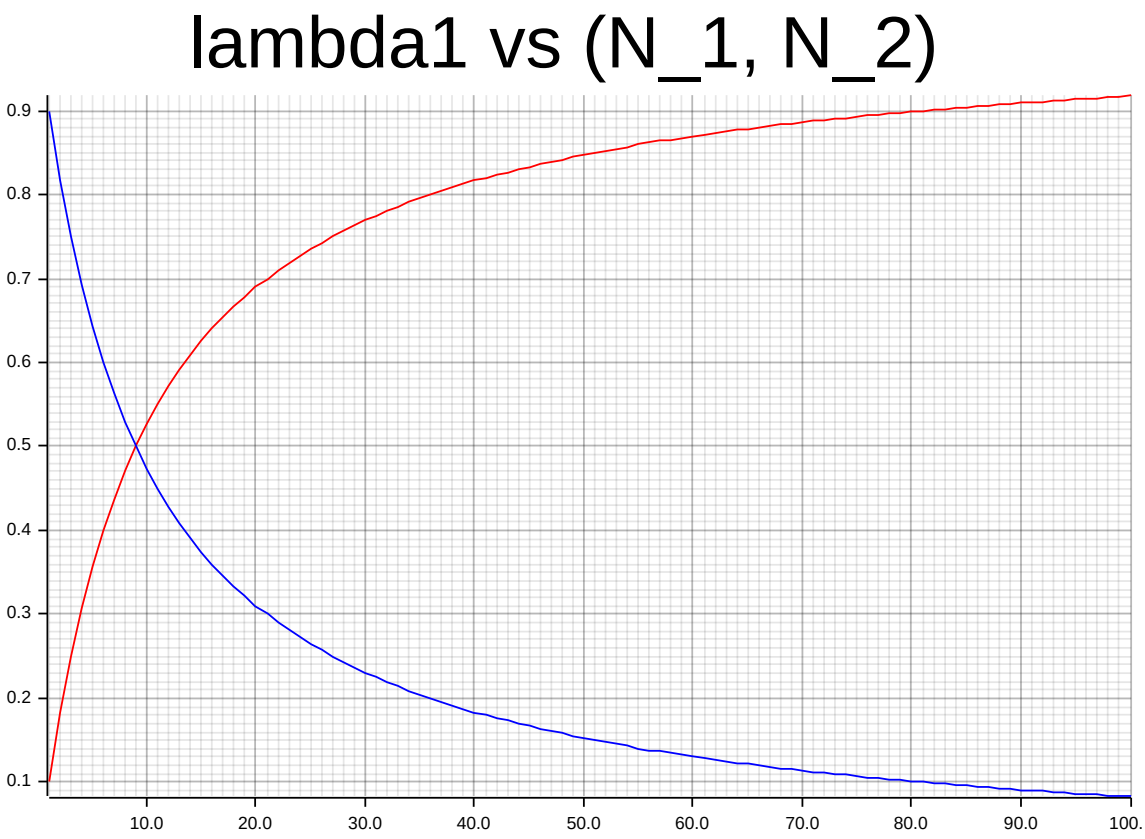
In [16]:
```
let mut avg_service_time1 = vec![];
let mut avg_service_requests1 = vec![];
let mut avg_service_time2 = vec![];
let mut avg_service_requests2 = vec![];

for lambda1 in 1..=100 {
```

```
    let lambda1 = lambda1 as f64;
    let n1 = lambda1 * (theta1) / (theta1 * lambda1 + theta2 * lambda2);
    let n2 = lambda2 * (theta2) / (theta1 * lambda1 + theta2 * lambda2);
    avg_service_requests1.push((lambda1 as f32, n1 as f32));
    avg_service_time1.push((lambda1 as f32, (n1/lambda1) as f32));
    avg_service_requests2.push((lambda1 as f32, n2 as f32));
    avg_service_time2.push((lambda1 as f32, (n2/lambda2) as f32));
}
```

Out[16]: ()

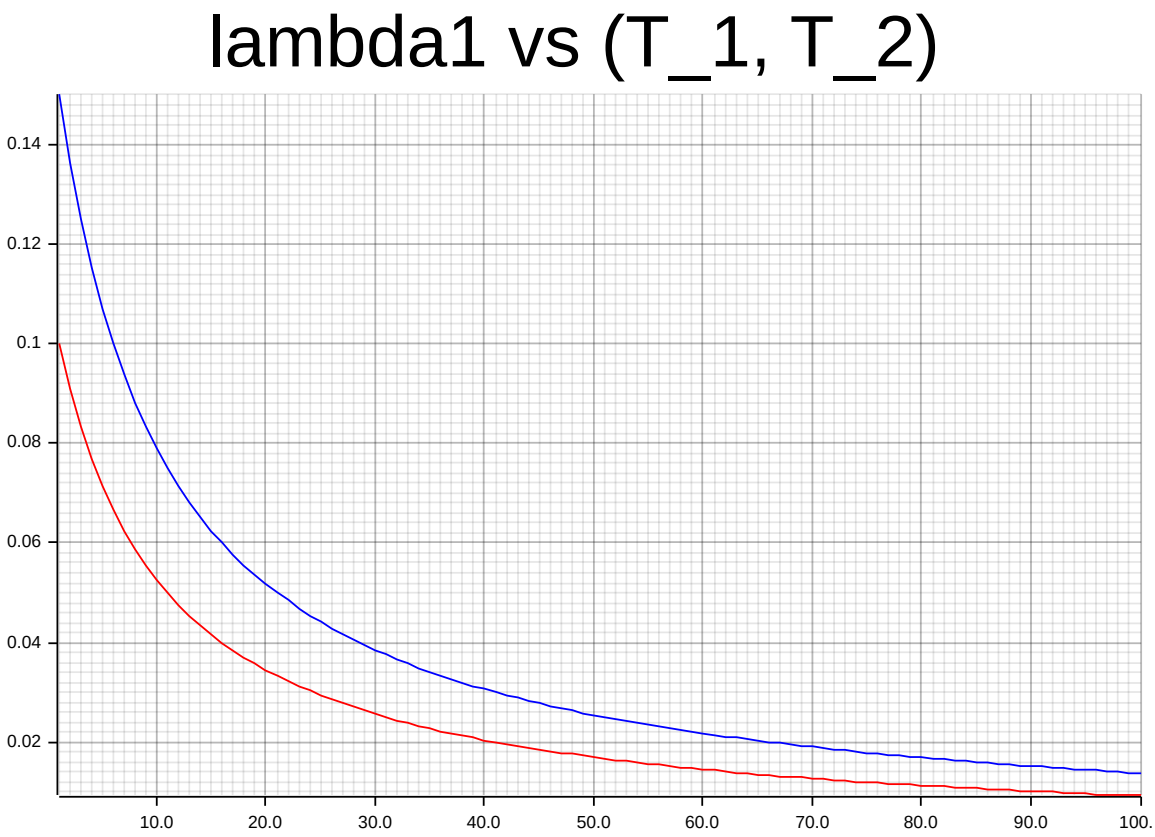
In [17]:
```
draw_chart_2(&avg_service_requests1, &avg_service_requests2, "lambda1 vs (N_
```

Out[17]:



In [18]:
```
draw_chart_2(&avg_service_time1, &avg_service_time2, "lambda1 vs (T_1, T_2)"
```

Out[18]:



In [19]:

```rust
let mut avg_service_time1 = vec![];
let mut avg_service_requests1 = vec![];
let mut avg_service_time2 = vec![];
let mut avg_service_requests2 = vec![];

for lambda2 in 1..=100 {
    let lambda2 = lambda2 as f64;
    let n1 = lambda2 * (theta1) / (theta1 * lambda1 + theta2 * lambda2);
    let n2 = lambda2 * (theta2) / (theta1 * lambda1 + theta2 * lambda2);
    avg_service_requests1.push((lambda2 as f32, n1 as f32));
    avg_service_time1.push((lambda2 as f32, (n1/lambda1) as f32));
    avg_service_requests2.push((lambda2 as f32, n2 as f32));
    avg_service_time2.push((lambda2 as f32, (n2/lambda2) as f32));
}
```

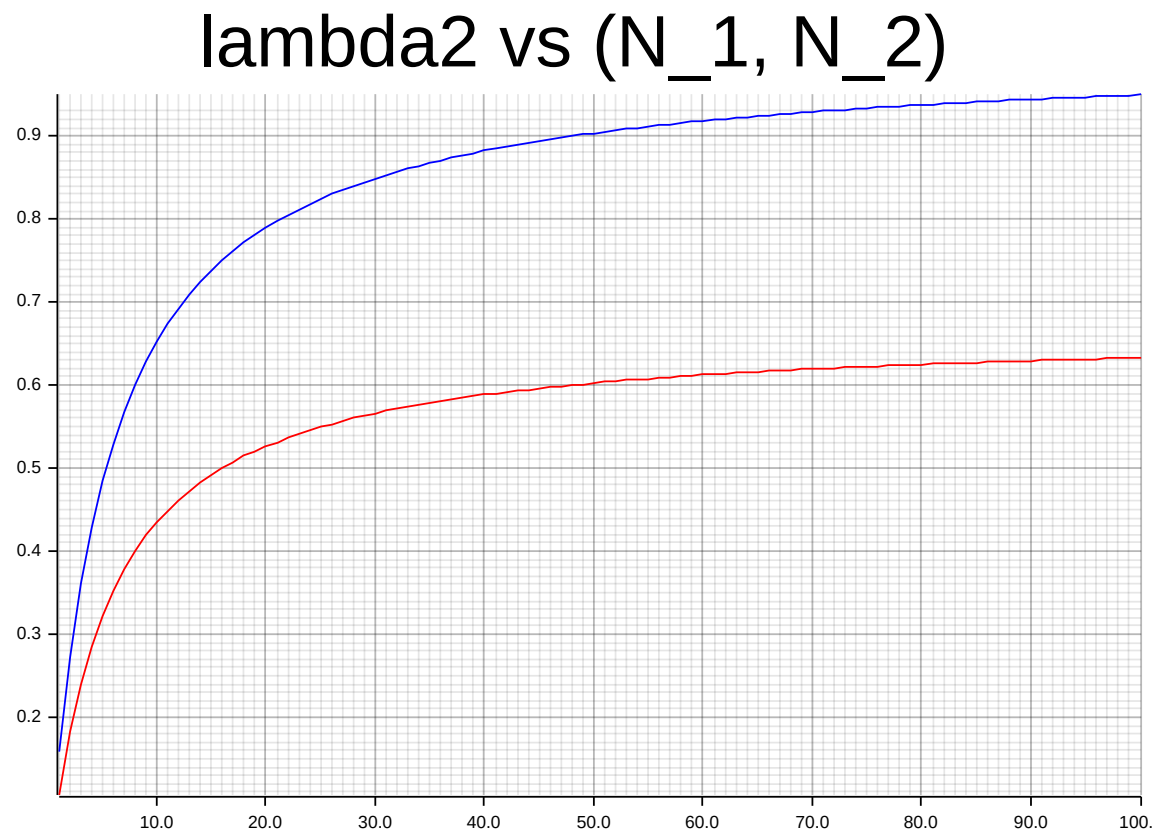
Out[19]:

```
()
```

In [20]:

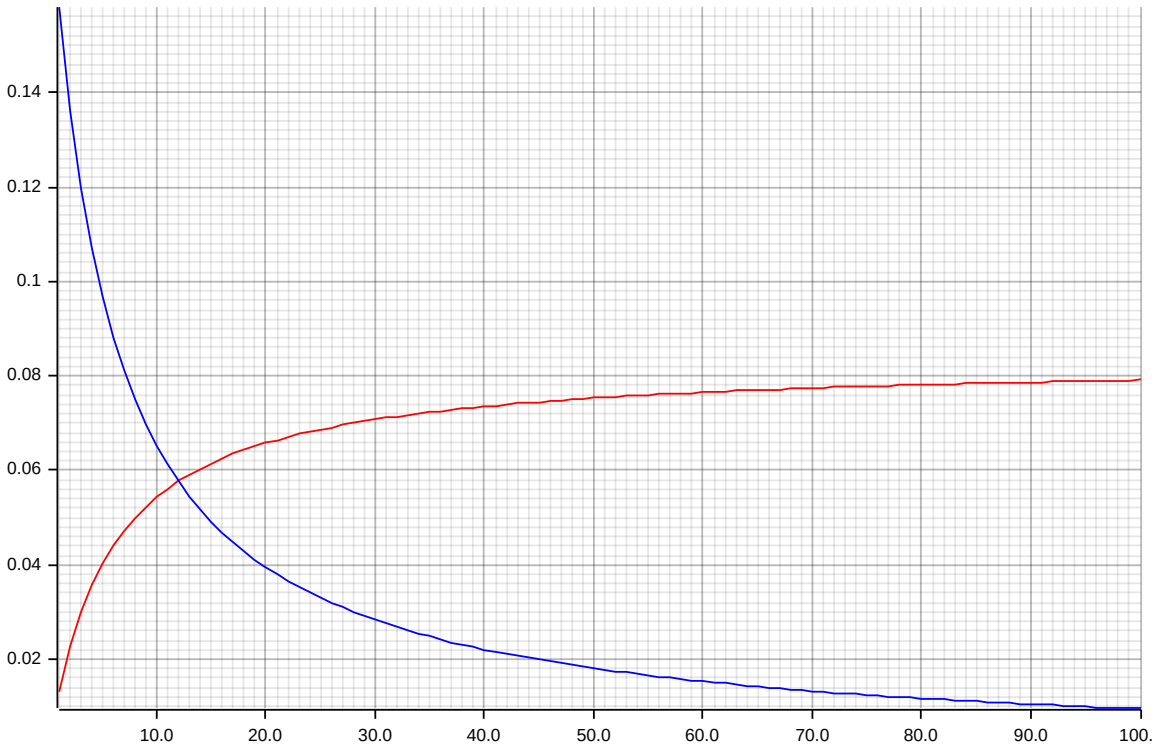
```rust
draw_chart_2(&avg_service_requests1, &avg_service_requests2, "lambda2 vs (N_
```

Out[20]:



In [21]:
```
draw_chart_2(&avg_service_time1, &avg_service_time2, "lambda2 vs (T_1, T_2)"
```

Out[21]:



In [ ]: