Sprawozdanie z Projektu CSP

System Producent-Konsument z Tablicami Wag i Dispatcherami

Marcel

10 grudnia 2025

Spis treści

1 Wprowadzenie **2**
1.1 Cel projektu 2

2 Architektura Systemu **2**
2.1 Schemat działania 2
2.2 Algorytm równoważenia obciążenia (skrót) 2

3 Konfiguracje testów **2**

4 Wyniki eksperymentów i wykresy **3**
4.1 Wykresy obciążeń 3

5 Analiza jakości równoważenia **4**

6 Fragmenty kluczowego kodu **4**

7 Wnioski i rekomendacje **6**

1 Wprowadzenie

Sprawozdanie przedstawia implementację i wyniki eksperymentów systemu producent-konsument w paradygmacie CSP (JCSP) z mechanizmem wag i dispatcherami. Dokument wykorzystuje dane wyjściowe z katalogu `lab7` projektu (pliki `output*.csv`, `buffers.csv`, `results.csv`) oraz wygenerowane wykresy.

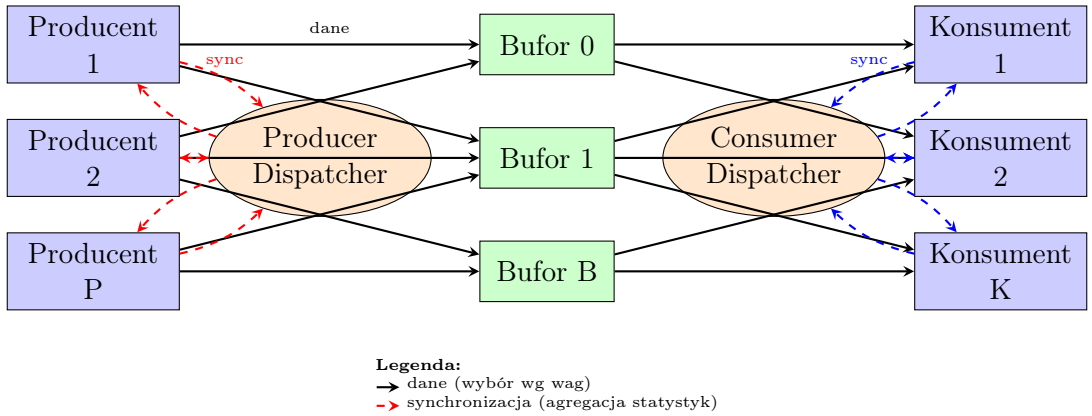
1.1 Cel projektu

Głównym celem projektu było:

- Implementacja systemu producent-konsument z wieloma buforami,
- Wprowadzenie adaptacyjnego mechanizmu wag i dispatcherów do równoważenia obciążenia,
- Przeprowadzenie serii testów i analiza rozkładu obciążeń między buforami,
- Prezentacja wyników i propozycje usprawnień.

2 Architektura Systemu

2.1 Schemat działania



Rysunek 1: Architektura systemu: producenci wybierają bufory wg wag; dispatchery agregują metryki i korygują wagi.

2.2 Algorytm równoważenia obciążenia (skrót)

Wagi aktualizowane są lokalnie i globalnie (EMA). W dokumentacji kodu i w katalogu `lab7` dostępny jest pełny opis algorytmu i implementacji.

3 Konfiguracje testów

W katalogu `lab7` są pliki wynikowe: `output555.csv`, `output51010.csv`, `output10105.csv`, `output101010.csv`, `output10510.csv`, `buffers.csv` i `results.csv`. Testy obejmują różne kombinacje liczby producentów (P), konsumentów (K) i buforów (B). Dokładne parametry każdego pliku widoczne są w nazwie pliku oraz w nagłówkach CSV.

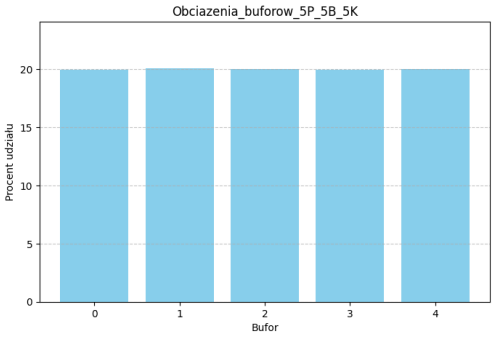
4 Wyniki eksperymentów i wykresy

`plot.py` służy do generacji wykresów.

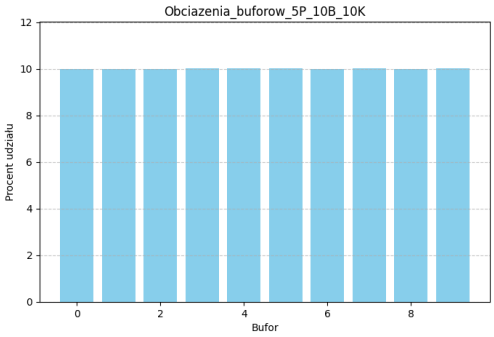
```
cd lab7
python3 plot.py
```

4.1 Wykresy obciążeń

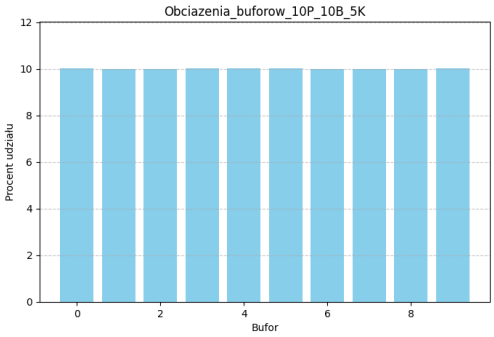
Poniżej załączam wykresy wygenerowane przez skrypty w `lab7`. Jeśli nie masz PNG, uruchom `plot.py`.



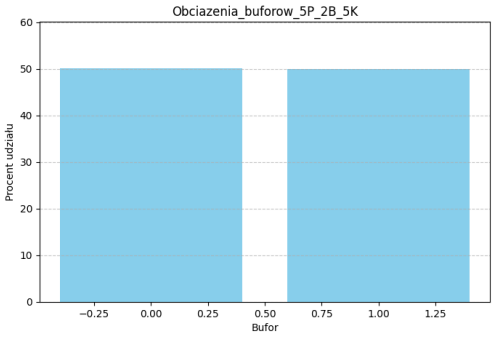
(a) obciazenie buforow (P=5,K=5,B=3)



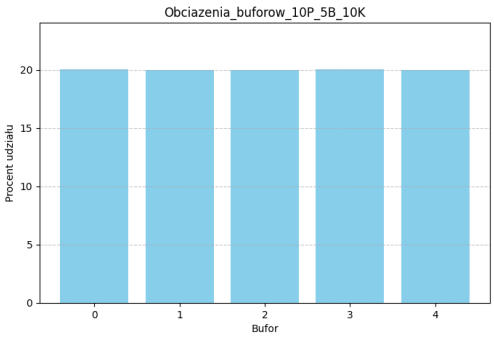
(b) obciazenie buforow (P=5,K=10,B=10)



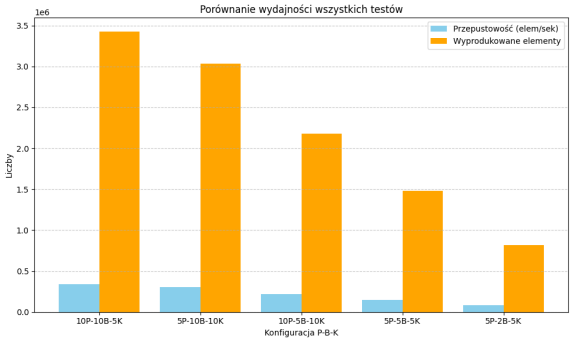
(c) obciazenie buforow (P=10,K=10,B=5)



(d) obciazenie buforow (P=5,K=2,B=5)



(e) obciazenie buforow (P=10,K=5,B=10)



(f) porownanie roznych testow

5 Analiza jakości równoważenia

Ogólne obserwacje:

- W testach z małą liczbą buforów fluktuacje krótkookresowe są większe, ale dispatcher stabilizuje rozkład.
- Zwiększenie liczby buforów zwykle podnosi przepustowość i obniża CV między buforami.
- Straty (różnica wyprodukowanych i skonsumowanych) są minimalne — szczegóły w `lab7/results.csv`.

6 Fragmenty kluczowego kodu

Poniżej najważniejsze fragmenty (zgodne z repo). Pełne pliki źródłowe znajdują się w `projektcsp/src`.

Listing 1: Wybór bufora (fragment)

```
private static int selectByWeight(double[] weights, Random rand) {
    double total = 0;
    for (double w : weights) total += w;
    double r = rand.nextDouble() * total;
    double cumulative = 0;
    for (int i = 0; i < weights.length; i++) {
        cumulative += weights[i];
        if (r <= cumulative)
            return i;
    }
    return weights.length - 1;
}
```

Listing 2: Aktualizacja wag (fragment)

```
private static void updateWeight(double[] weights, int index, long
   waitTimeNanos) {
    double newWeight = 1.0 / (1.0 + waitTimeNanos / 1_000_000.0);
    weights[index] = 0.7 * weights[index] + 0.3 * newWeight;
    if (weights[index] < 0.01) weights[index] = 0.01;
}
```

Listing 3: Producent - uruchomienie i wagi

```
        for (int p = 0; p < numProducers; p++) {
            final int pid = p;
            final One2OneChannel myDispatcherResponse =
               producerDispatcherOut[p];
            final ArrayBlockingQueue<Integer>[] queues =
               bufferQueues;
            final long endTime = stopTimeMillis;

            processes[idx++] = () -> {
```

```
8                 Random rand = new Random(pid * 1000 + System.
                      nanoTime());
9                 CSTimer timer = new CSTimer();
10
11                double[] weights = new double[numBuffers + 1];
12                long[] waitTimes = new long[numBuffers + 1];
13
14                for (int i = 0; i <= numBuffers; i++) {
15                    weights[i] = 1.0;
16                    waitTimes[i] = 0;
17                }
```

Listing 4: Konsument - deklaracja i wagi

```
1         for (int c = 0; c < numConsumers; c++) {
2             final int cid = c;
3             final One2OneChannel myDispatcherResponse =
                  consumerDispatcherOut[c];
4             final ArrayBlockingQueue<Integer>[] queues =
                  bufferQueues;
5             final long endTime = stopTimeMillis;
6
7             processes[idx++] = () -> {
8                 Random rand = new Random(cid * 2000 + System.
                      nanoTime());
9                 CSTimer timer = new CSTimer();
10
11                double[] weights = new double[numBuffers + 1];
12                long[] waitTimes = new long[numBuffers + 1];
13                boolean[] bufferAlive = new boolean[numBuffers];
14
15                for (int i = 0; i <= numBuffers; i++) {
16                    weights[i] = 1.0;
17                    waitTimes[i] = 0;
18                }
19                for (int i = 0; i < numBuffers; i++) {
20                    bufferAlive[i] = true;
21                }
```

Listing 5: Deklaracja buforow, procesow, kanalow i zmiennych globalnych

```
1                 ArrayBlockingQueue<Integer>[] bufferQueues = new
                      ArrayBlockingQueue[numBuffers];
2         for (int i = 0; i < numBuffers; i++) {
3             bufferQueues[i] = new ArrayBlockingQueue<>(1000);
4         }
5
6         Any2OneChannel producerDispatcherIn = Channel.any2one();
7         One2OneChannel[] producerDispatcherOut = new One2OneChannel
              [numProducers];
8         for (int i = 0; i < numProducers; i++) {
9             producerDispatcherOut[i] = Channel.one2one();
```

```
10          }
11
12          Any2OneChannel consumerDispatcherIn = Channel.any2one();
13          One2OneChannel[] consumerDispatcherOut = new One2OneChannel
                [numConsumers];
14          for (int i = 0; i < numConsumers; i++) {
15              consumerDispatcherOut[i] = Channel.one2one();
16          }
17
18          AtomicInteger itemCounter = new AtomicInteger(0);
19          AtomicInteger activeProducers = new AtomicInteger(
                numProducers);
20          AtomicInteger activeConsumers = new AtomicInteger(
                numConsumers);
21
22          CSProcess[] processes = new CSProcess[numProducers +
                numConsumers + 2];
23          int idx = 0;
```

7 Wnioski i rekomendacje

Zalety:

- Skuteczne wyrównywanie obciążenia między buforami (niskie roznice w większości testów).
- Dobra skalowalność z liczbą procesów i buforów.
- Małe straty danych i stabilne czasy wykonania.

Wady / możliwości poprawy:

- Dispatcher może stać się wąskim gardłem przy ekstremalnej skali — rozważyć sharding dispatcherów.
- Koszt komunikacji przy częstych aktualizacjach wag — rozważyć rzadsze/batched aktualizacje lub lokalne polityki.
- Dodatkowe metryki (percentyle latencji, wykresy czasów oczekiwania) ułatwią dalszą diagnostykę.