Problem pagingu w cache

Karol Janic

Badane algorytmy

Zaimplementowano następujące metody działania cache'u:

- first in first out
- flush when full
- least recently used
- least frequently used
- random
- random markup

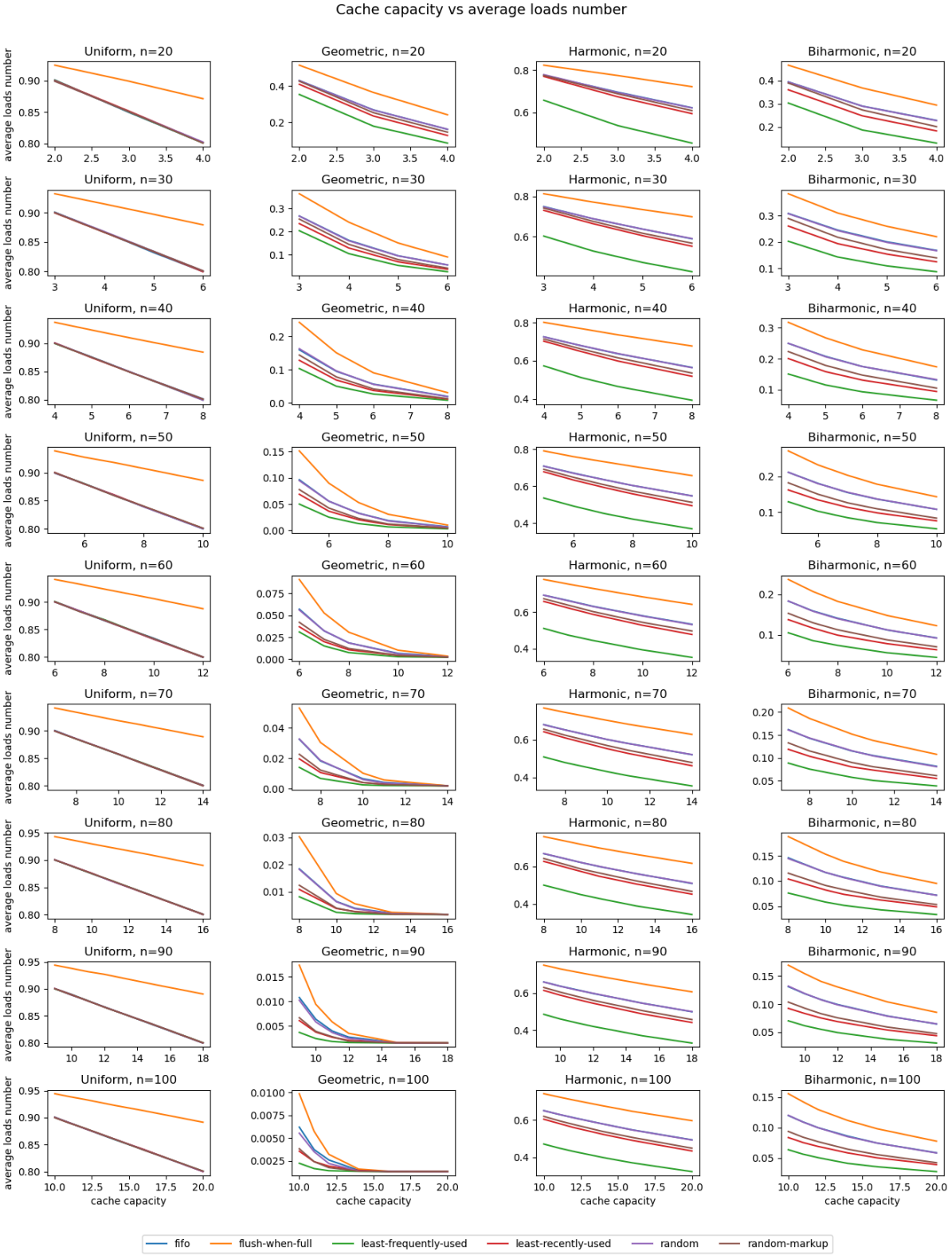
Opis eksperymentu

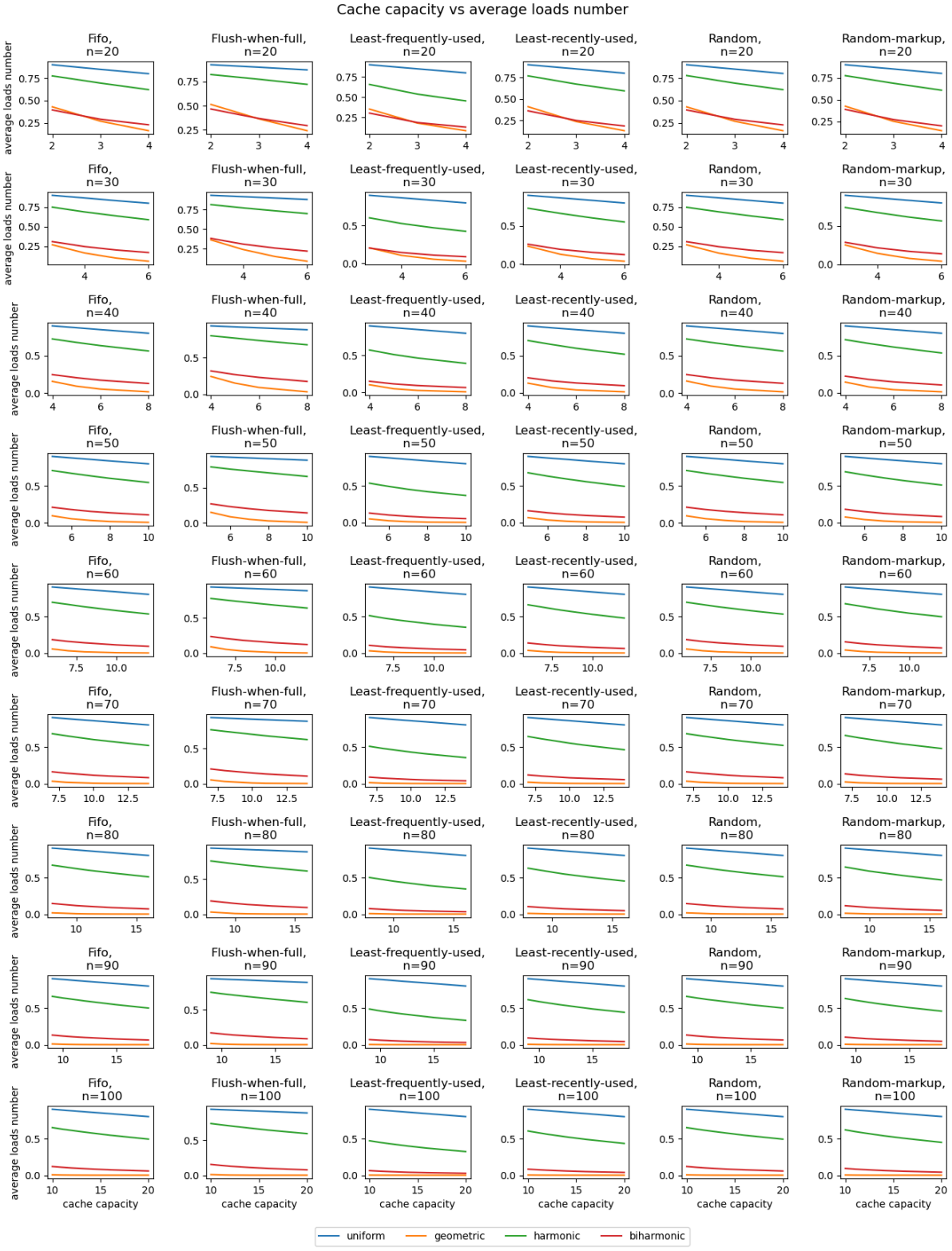
Eksperyment polegał na przeprowadzeniu symulacji cachu o pojemności k działającej na n stronach pamięci, gdzie $k = \{\frac{n}{10}, ..., \frac{n}{5}\}$. Wyszukiwane strony były wybierane losowo z kilku rozkładów prawdopodobieństwa:

- rozkład jednostajny ($\mathbb{P}[X = i] = \frac{1}{n}$)
- rozkład geometryczny ($\mathbb{P}[X = i] = \frac{1}{2^n}$)
- rozkład harmoniczny ($\mathbb{P}[X = i] = \frac{1}{iH_n^{(1)}}$)
- rozkład dwuharmoniczny ($\mathbb{P}[X = i] = \frac{1}{i^2H_n^{(2)}}$)

Dla każdej metody, rozkładu przeprowadzono i liczby stron przeprowadzono po 100 eksperymentów. W każdym z nich zadano $1000n$ zapytań o strony. Wynikiem eksperymentu był koszt działania cache'u, czyli liczba ładowań stron do pamięci szybkiej.

Wyniki eksperymentu




Cache capacity vs average loads number
Fifo, n=20
Flush-when-full, n=20
Least-frequently-used, n=20
Least-recently-used, n=20
Random, n=20
Random-markup, n=20
Fifo, n=30
Flush-when-full, n=30
Least-frequently-used, n=30
Least-recently-used, n=30
Random, n=30
Random-markup, n=30
Fifo, n=40
Flush-when-full, n=40
Least-frequently-used, n=40
Least-recently-used, n=40
Random, n=40
Random-markup, n=40
Fifo, n=50
Flush-when-full, n=50
Least-frequently-used, n=50
Least-recently-used, n=50
Random, n=50
Random-markup, n=50
Fifo, n=60
Flush-when-full, n=60
Least-frequently-used, n=60
Least-recently-used, n=60
Random, n=60
Random-markup, n=60
Fifo, n=70
Flush-when-full, n=70
Least-frequently-used, n=70
Least-recently-used, n=70
Random, n=70
Random-markup, n=70
Fifo, n=80
Flush-when-full, n=80
Least-frequently-used, n=80
Least-recently-used, n=80
Random, n=80
Random-markup, n=80
Fifo, n=90
Flush-when-full, n=90
Least-frequently-used, n=90
Least-recently-used, n=90
Random, n=90
Random-markup, n=90
Fifo, n=100
Flush-when-full, n=100
Least-frequently-used, n=100
Least-recently-used, n=100
Random, n=100
Random-markup, n=100
average loads number
cache capacity
uniform
geometric
harmonic
biharmonic

Wnioski

- W przypadku rozkładu jednostajnego najgorszy okazał się algorytm flush when full. Reszta algorytmów uzyskiwała podobne wyniki.
- W przypadku reszty rozkładów najlepszy okazał się algorytm last-frequently used. Następne były algorytmy last-recently used, random markup, random, first in first out. Najgorszy okazał się algorytm flush when full. Jest to spowodowane tym, że w skoncentrowanych rozkładach liczba różnych wybieranych stron jest mała.
- Czym rozkład jest bardziej skoncentrowany tym algorytmy działają lepiej. Najlepiej działają dla rozkładu geometrycznego.