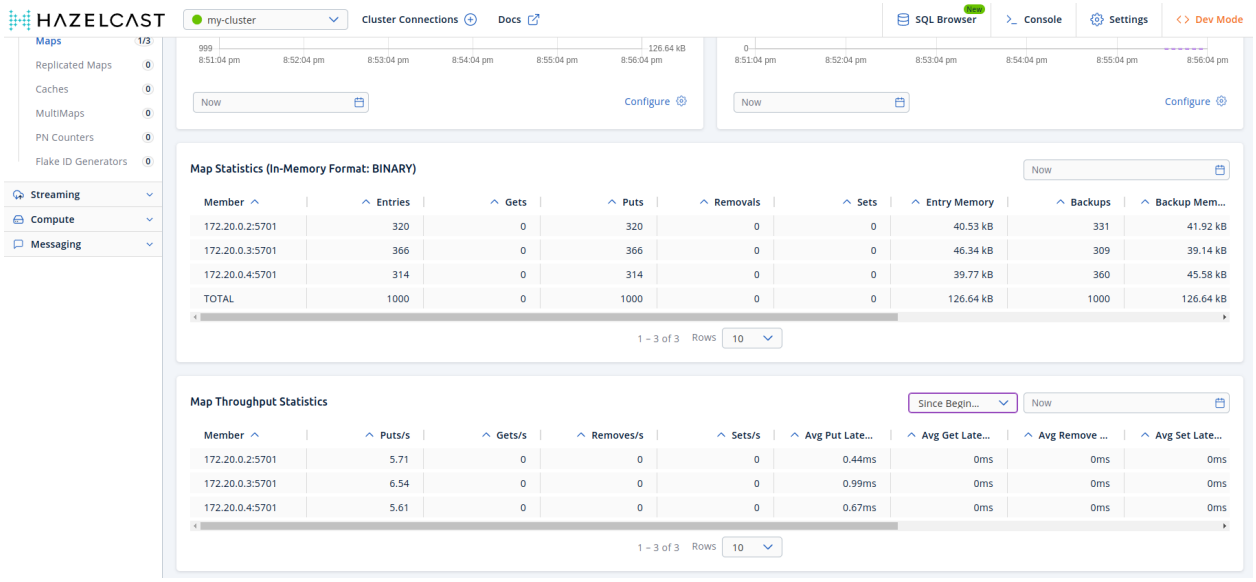
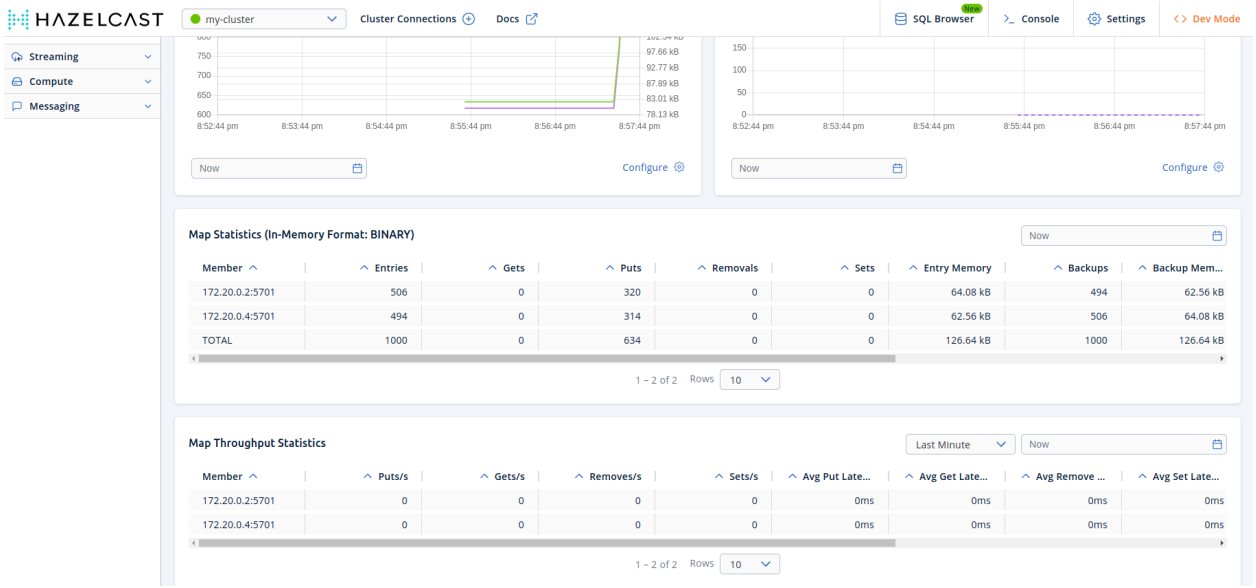
3) Результат try_distributed_map.py:

3 трьома нодами:



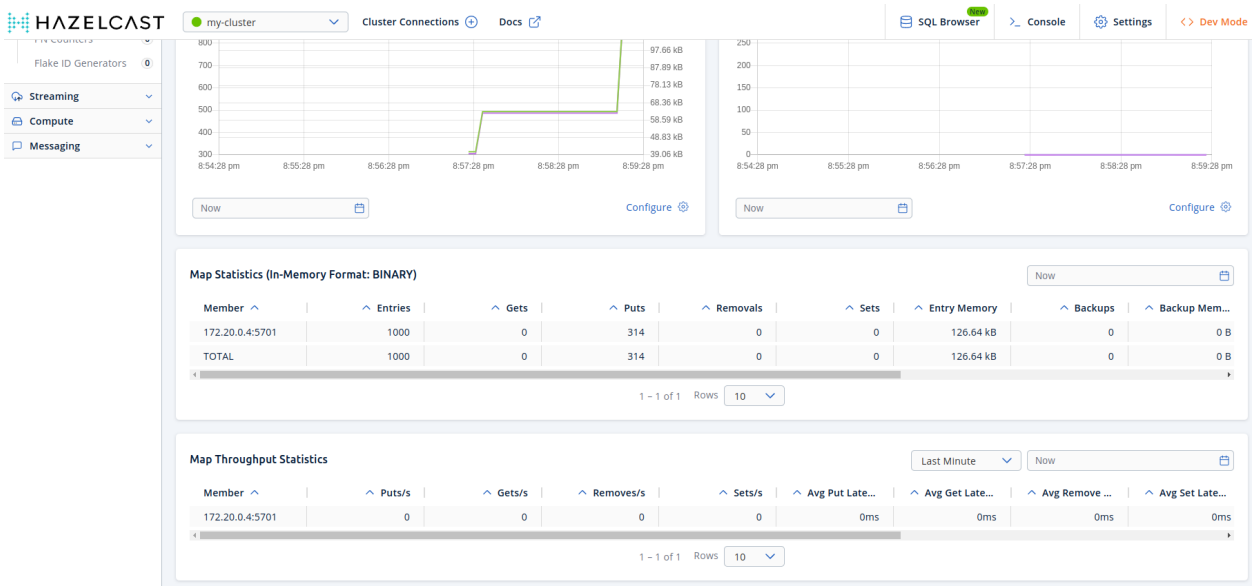
Ми бачимо що тут дані розподіляються +- одинаково так само як і backup меморі (Якщо якийсь мембер виходить з ладу, інші учасники кластера використовують свої резервні копії для повторного розподілу даних між іншими членами кластера)

Після відключення однієї ноди:



Тут ми можемо побачити що мембери які залишились до розподілили між собою ентріс які зберігались у третього ноду. Якщо ж подивитись на бекап дані то можна явно побачити що учасники зберігають дані одни одного.

Після відключення ще однієї ноди:

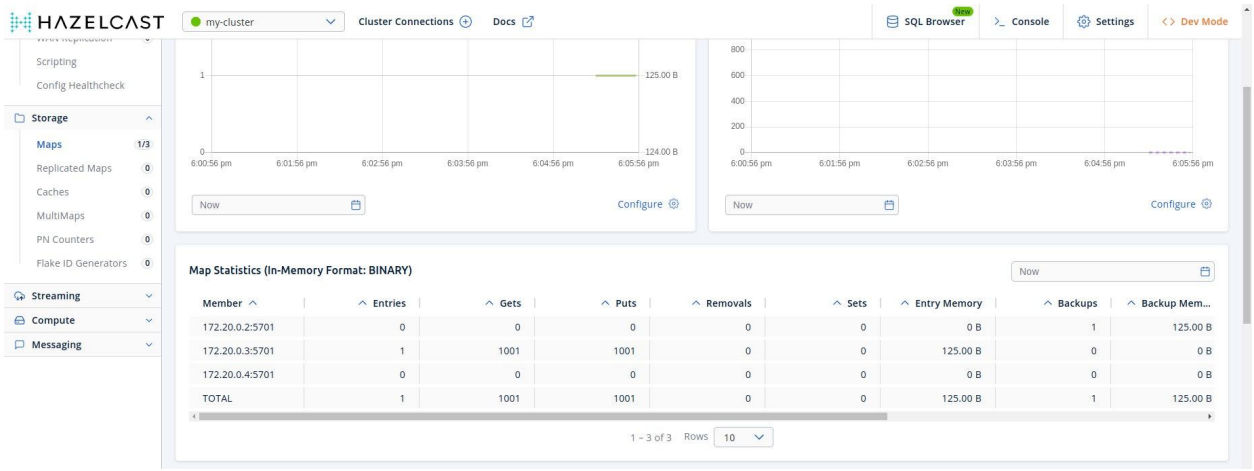


Як і попереднього разу, всі дані збереглись і тепер лише один учасник ними володіє. Тому втрати даних у нас немає.

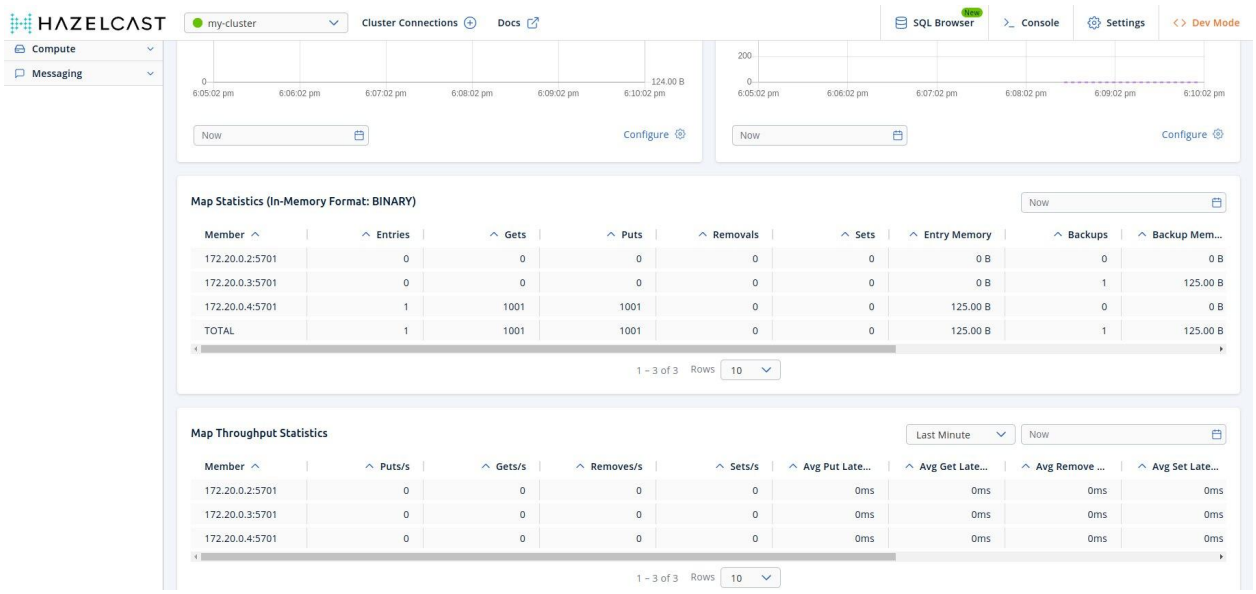
4) Результат no-blocking.py, pessim-blocking.py, optim-blocking.py:

Тут я реалізувала в циклі зчитування значення та фактично поновлений запис його (все з одним ключем), через це результат був однаковим у всіх варіантах:

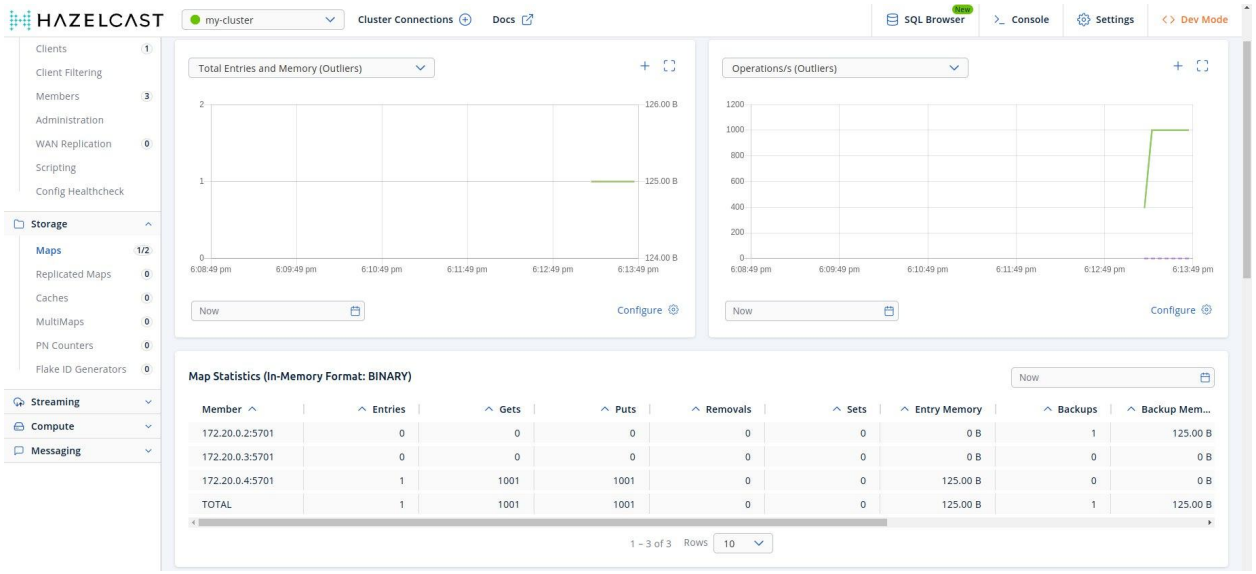
No blocking :



Pessimistic blocking:



Optimistic blocking:

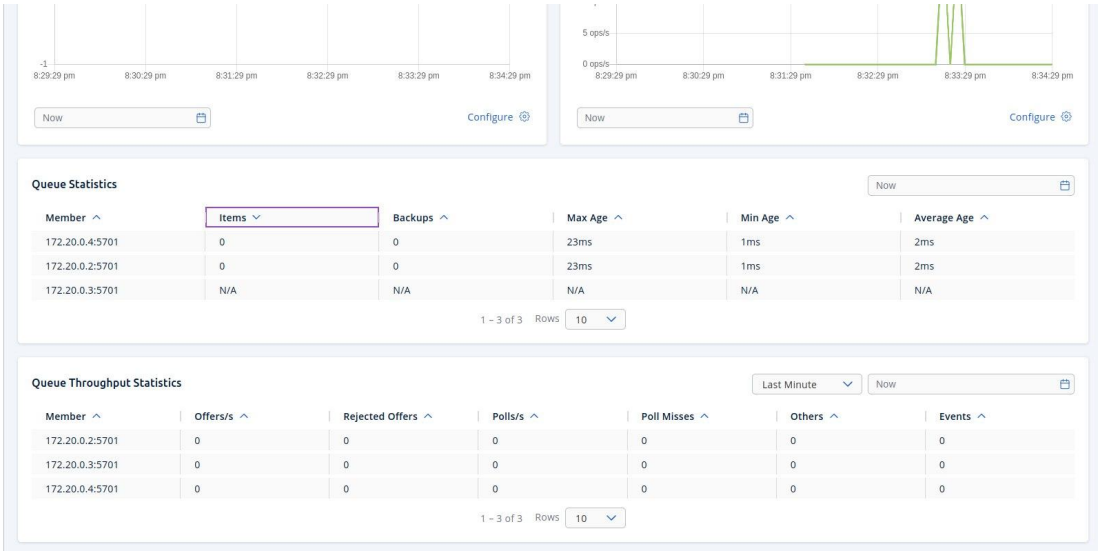


Як ми бачимо тільки один клієнт виконує зчитування та запис, через те що ми працюємо тільки з одним значенням і так як не відбувається запису чи зчитування багатьох ключів у нас немає потреби залучати двох інших клієнтів. В даному випадку найкраще виконувати песимістичне блокування - воно більш надійне коли ми часто апдейтимо один ключ. Проте якщо б ми лише зчитували краще використовувати оптимістичне - при зчитуванні нам не потрібен такий сильний захист як при записі і оптимістичне блокування є краще з точки зору перформансу. Щодо опції без локування - тут ми не побачили рейс кондинишу. Все через те, що Hazelcast сам виконує розподіл за допомогою тредів (точніше кожен розподіл керується тільки одним тредом) і для більшості систем це гарантує надійність. Однак оскільки рейс кондишин є вкрай небажаним краще використовувати локи.

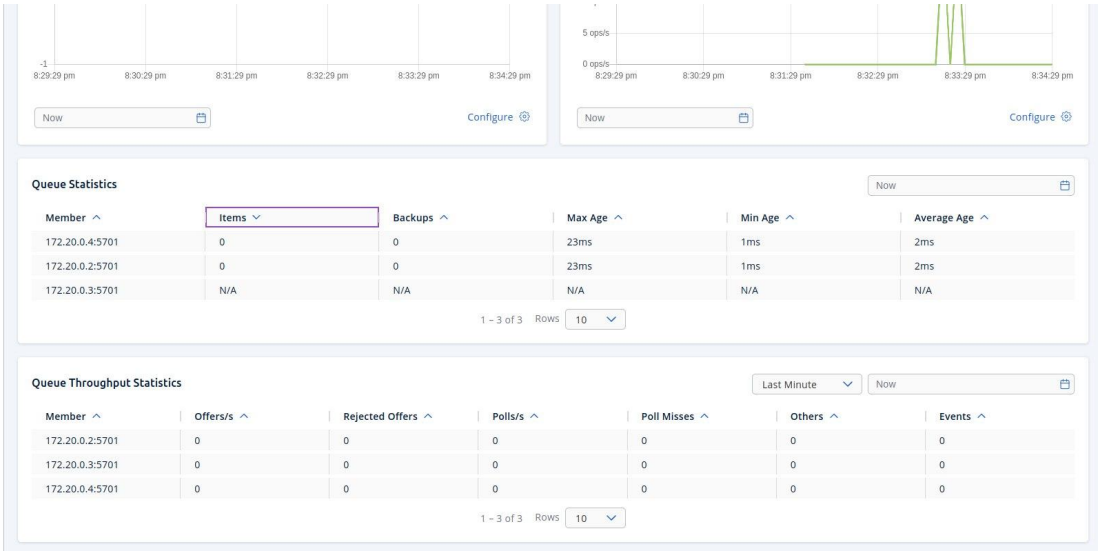
5) Результат bounded-queue.py:

Так як пайтон hazelcast не має підтримки bounded queue я створила її аналог сама

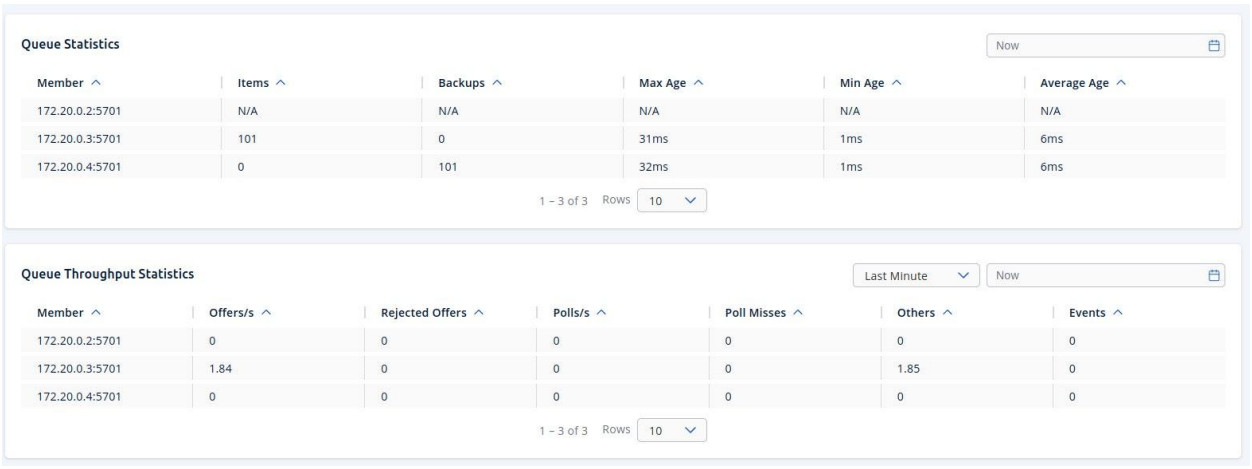
Запуск для двох тредів які забирають значення та одного який вписує:



Для одного який зчитує та одного який вписує:



По цій статистиці ми не можемо багато побачити адже на даний момент черга пуста. Спробуємо записати значення без зчитування:



Queue Statistics

Now

Member	Items	Backups	Max Age	Min Age	Average Age
172.20.0.2:5701	N/A	N/A	N/A	N/A	N/A
172.20.0.3:5701	101	0	31ms	1ms	6ms
172.20.0.4:5701	0	101	32ms	1ms	6ms

1 – 3 of 3 Rows 10

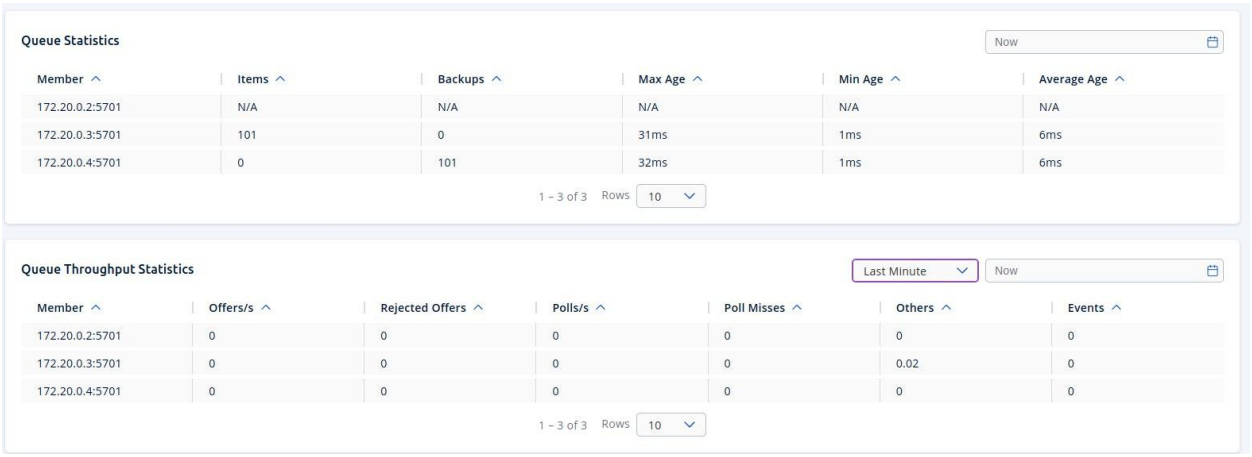
Queue Throughput Statistics

Last Minute Now

Member	Offers/s	Rejected Offers	Polls/s	Poll Misses	Others	Events
172.20.0.2:5701	0	0	0	0	0	0
172.20.0.3:5701	1.84	0	0	0	1.85	0
172.20.0.4:5701	0	0	0	0	0	0

1 – 3 of 3 Rows 10

Після цього ще раз викликаємо запис з заповненою чергою:



Queue Statistics

Now

Member	Items	Backups	Max Age	Min Age	Average Age
172.20.0.2:5701	N/A	N/A	N/A	N/A	N/A
172.20.0.3:5701	101	0	31ms	1ms	6ms
172.20.0.4:5701	0	101	32ms	1ms	6ms

1 – 3 of 3 Rows 10

Queue Throughput Statistics

Last Minute Now

Member	Offers/s	Rejected Offers	Polls/s	Poll Misses	Others	Events
172.20.0.2:5701	0	0	0	0	0	0
172.20.0.3:5701	0	0	0	0	0.02	0
172.20.0.4:5701	0	0	0	0	0	0

1 – 3 of 3 Rows 10

Як ми бачимо, весь запис відбувається тільки з одного ноду, а один нод не задіяний взагалі. Для розподілення роботи я використовувала python threading. Опісля того як ми запускаємо запис другий раз у заповненій черзі, виконання зразу переривається так як перед тим іде перевірка чи черга досягла максимального розміру.