Karol Kasperek Kamil Sulkowski

Laboratorium nr 4
SKPS

1. Przetestowanie działania programów na gospodarzu.

Przygotowaliśmy i podłączyliśmy w prawidłowy sposób płytkę RPi4. Postawiliśmy system OpenWRT w ten sam sposób jak na poprzednim laboratorium. Pracę rozpoczęliśmy od pobrania i rozpakowania pliku https://moodle.usos.pw.edu.pl/pluginfile.php/241562/mod_folder/content/0/skps_lab4_student.tar.xz?forcedownload=1. Po rozpakowaniu wykonaliśmy polecenie make w folderze z plikami źródłowymi i dodaliśmy do PATH ścieżkę pliku cw4b i uruchomiliśmy komendą ./cw4a 3 100 1000 1. Program działał poprawnie.

2. Zbudowanie pakietu dla OpenWRT.

Dodaliśmy na koniec pliku feeds.conf.default linie z ścieżką do naszego pakietu: src-link skps /home/user/Pulpit/lab4/skps_lab4_student/cw4_owrt_pkg. Wykonaliśmy następnie polecenia scripts/feeds update -a oraz scripts /feeds install -p skps -a. Wykonaliśmy polecenie make menuconfig i dodaliśmy nasze pakiety do kompilacji i instalacji. Zaznaczyliśmy również opcję aby maksymalnie przyśpieszyć kompilację. Wykonaliśmy make na naszym pakiecie poleceniem: make package/feeds/sksps/cwicz4mak. Polecenie wykonało się pomyślnie. Następnie zainstalowaliśmy nasz pakiet poleceniem opkg install cwicz4mak_1_aarch64_cortex-a72.ipk. Uruchomiliśmy OpenWRT na RPi4 i przesłaliśmy na niego zbudowany pakiet. Zainstalowaliśmy go tam poleceniem opkg install cwicz4mak_1_aarch64_cortex-a72.ipk. Instalacja zakończyła się powodzeniem i możemy korzystać z pogramów cw4a i cw4b.

3. Ustalanie granicznej wartości czasu przetwarzania

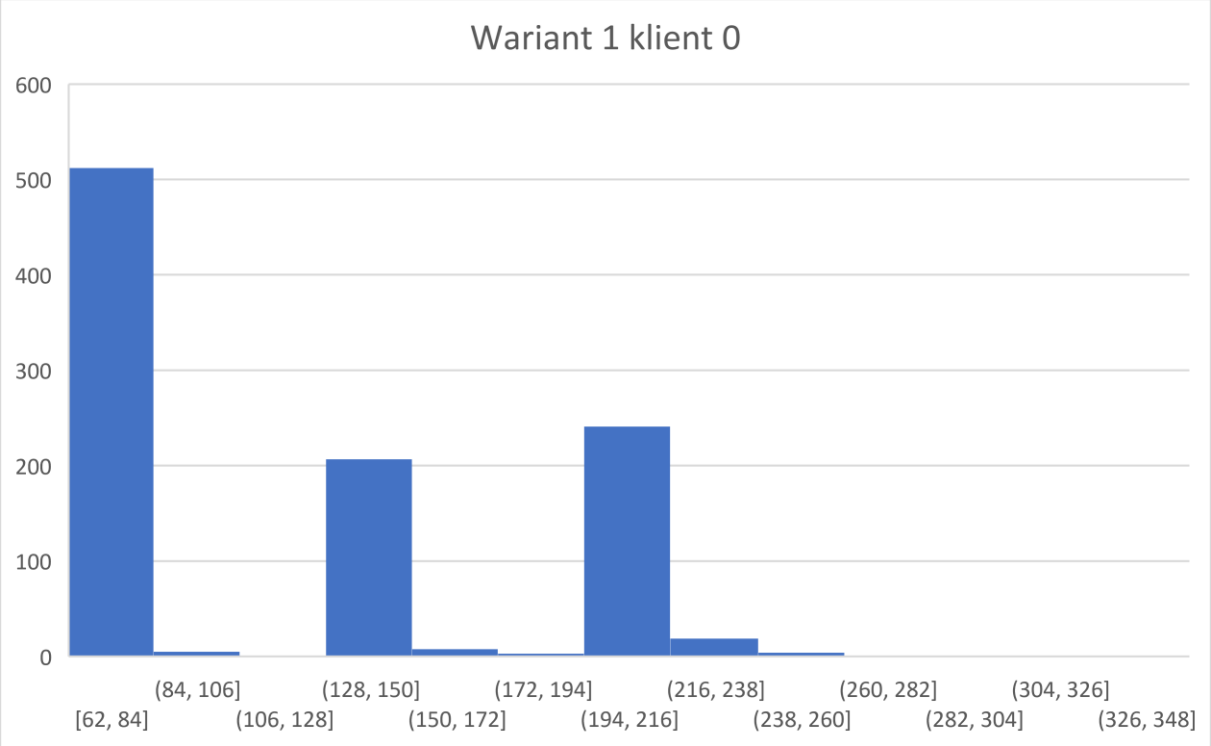
W pliku cmdline.txt dodaliśmy parametr maxcpus=N i będziemy go zmieniać w zależności od podpunktu.

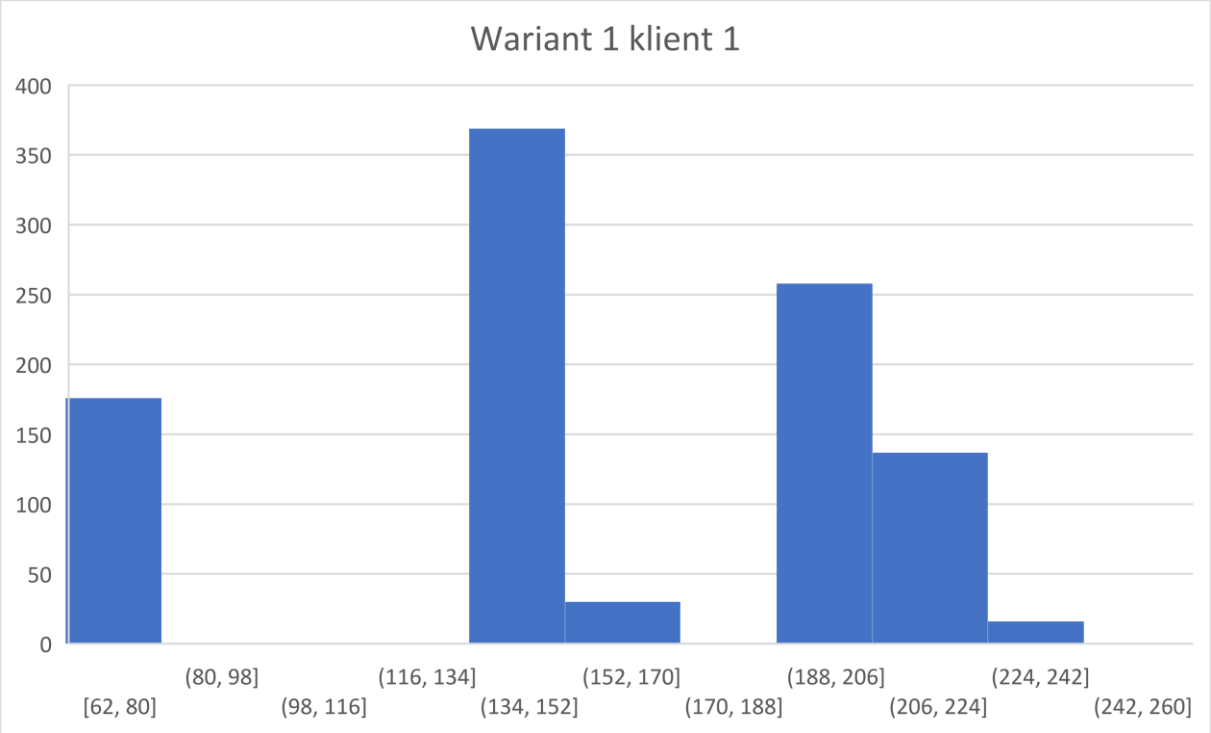
1) Pierwszy wariant: 3 klientów, 1 rdzeń, pełne obciążenie
   Maxcpus = 1
   Stress-ng –matrix 0 -t 5m&
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 X
   Gdzie X jest czasem przetwarzania, którym manipulowaliśmy. Testy przeprowadzaliśmy dla dużej ilości zmiennych: 1, 10, 100, 1000, 10000, 25000, 50000, 100000, 250000, 500000, 750000, 1000000. Zauważaliśmy, że w przedziale pomiędzy 250000 a 350000 powinna znajdować się wartość graniczna. Znaleźliśmy zadowalającą wartość graniczną, dla której występuje ciągły (z bardzo nielicznymi spadkami) wzrost opóźnienia, w okolicach 280000.
2) Drugi wariant: 3 klientów, 2 rdzenie, pełne obciążenie
   Macxpus = 2
   Stress-ng -matrix 0 -t 5m&
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 X
   Gdzie X jest czasem przetwarzania, którym manipulowaliśmy. Testy przeprowadzaliśmy na dużej ilości zmiennych. Znaleźliśmy zadowalającą wartość graniczą w okolicach 460000.
3) Trzeci wariant: 3 klientów, 2 rdzenie, bez obciążenia
   Macxpus = 2
   Bez obciążenia.
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 X
   Gdzie X jest czasem przetwarzania, którym manipulowaliśmy. Testy przeprowadziliśmy na dużej ilości zmiennych. Znaleźliśmy zadowalającą wartość graniczą w okolicach 540000.
4) Czwarty wariant: 1 klient, 4 rdzenie, bez obciążenia
   Macxpus = 4
   Bez obciążenia.
   Program cw4a uruchomiliśmy w podany sposób: cw4a 1 1000 10000 X

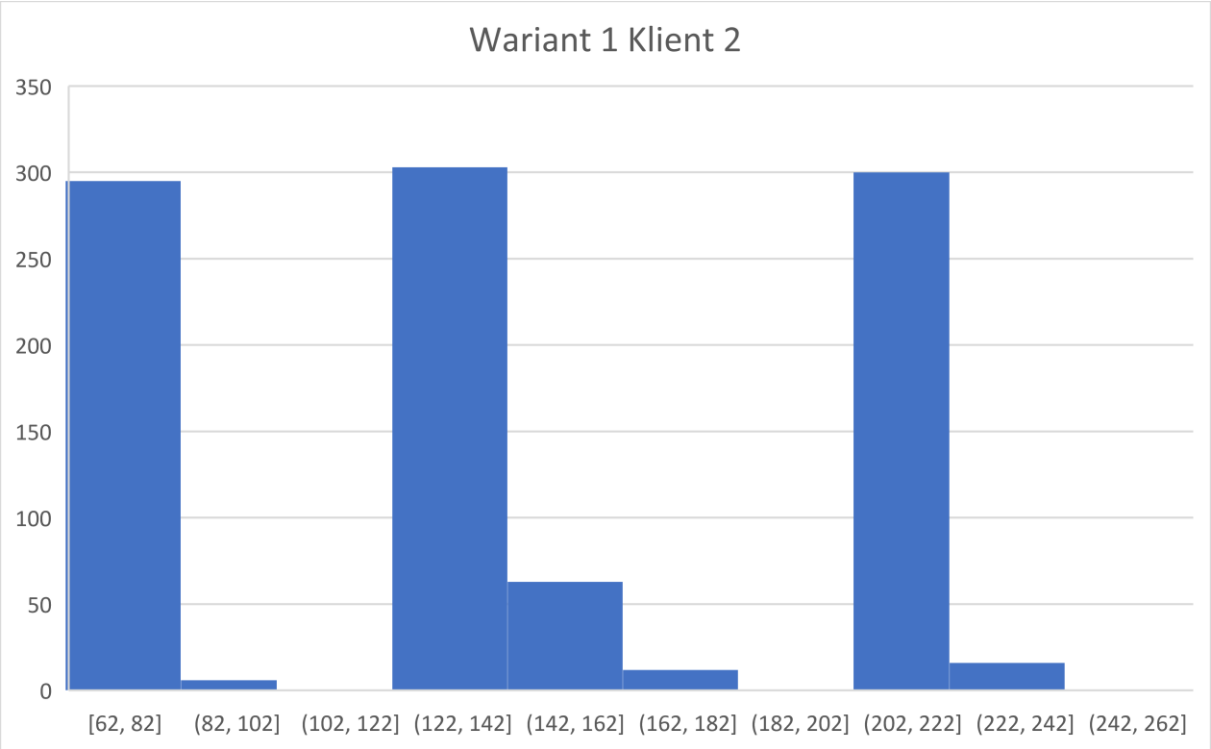
Gdzie X jest czasem przetwarzania, którym manipulowaliśmy. Testy przeprowadziliśmy na dużej ilości zmiennych. Znaleźliśmy zadowalającą wartość graniczną w okolicach 650000.

4. Rozkład czasu dostarczana danych

1) Pierwszy wariant: 3 klientów, 1 rdzeń, pełne obciążenie
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 140000.
   Macxpus = 1
   Stress-ng -matrix 0 -t 5m&

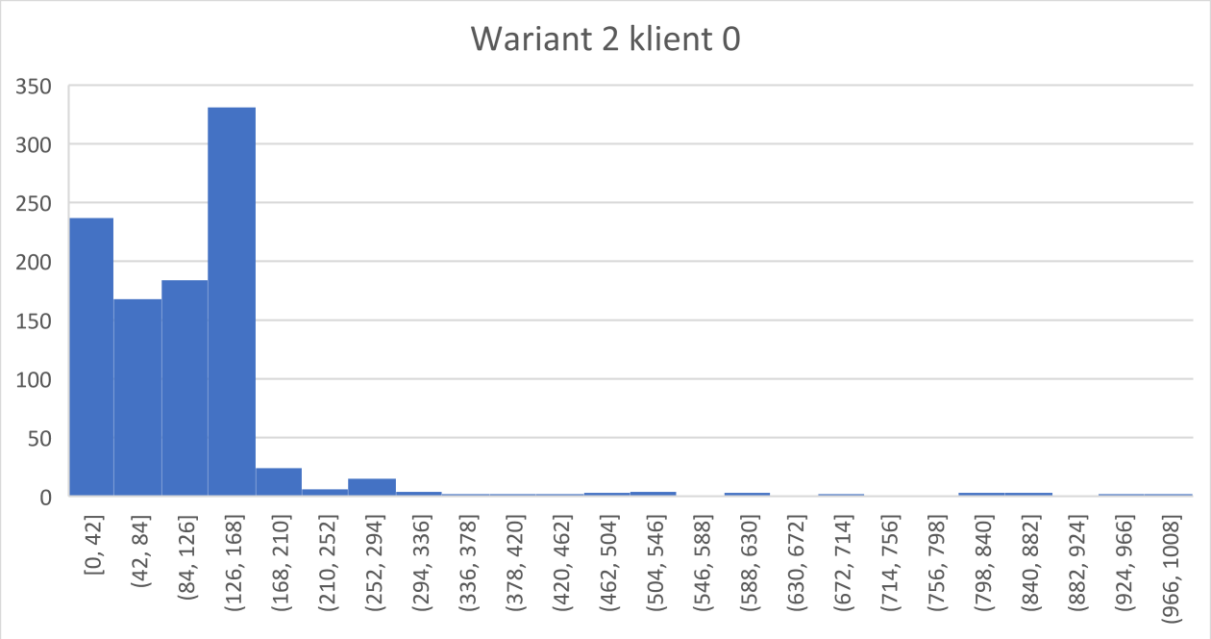


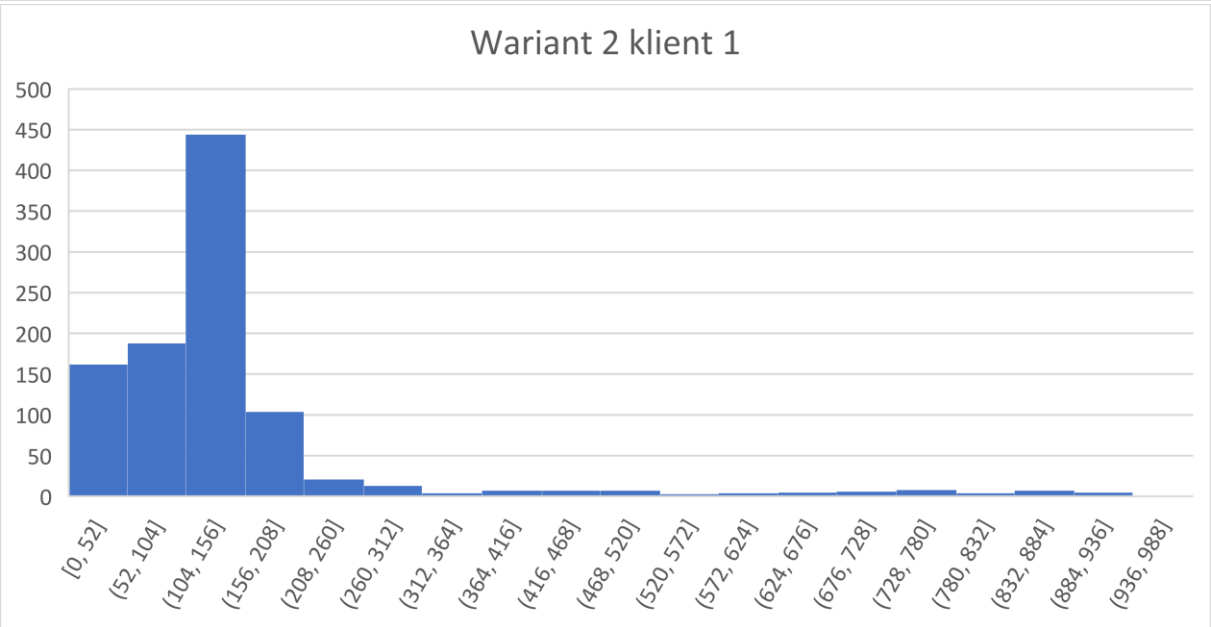


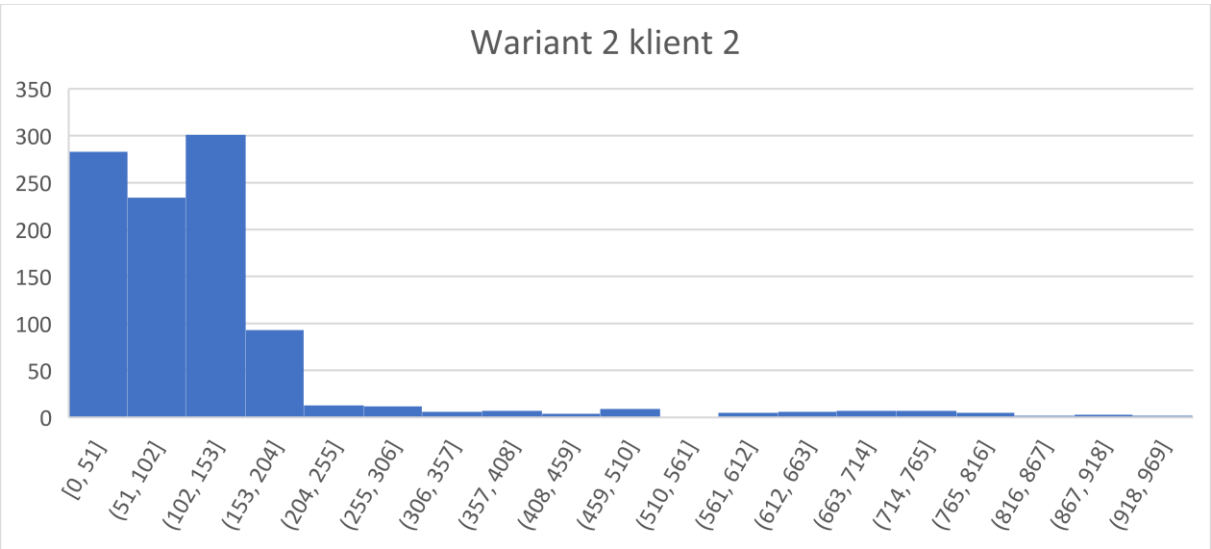


2) Drugi wariant: 3 klientów, 2 rdzenie, pełne obciążenie
   Macxpus = 2
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 230000.
   Stress-ng -matrix 0 -t 5m&

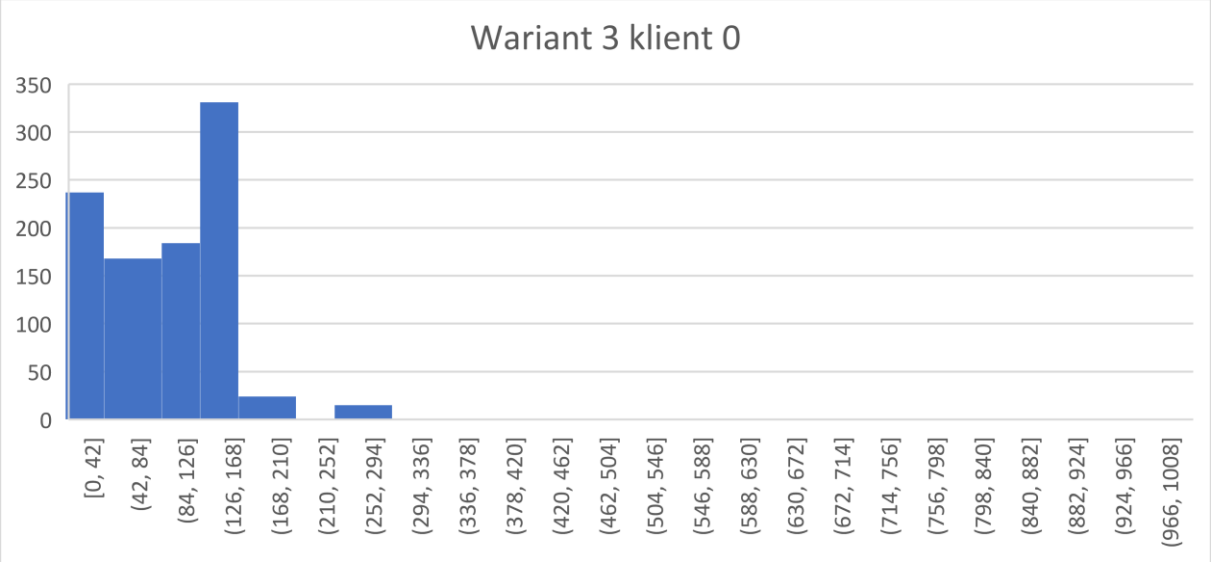
Karol Kasperek Kamil Sulkowski

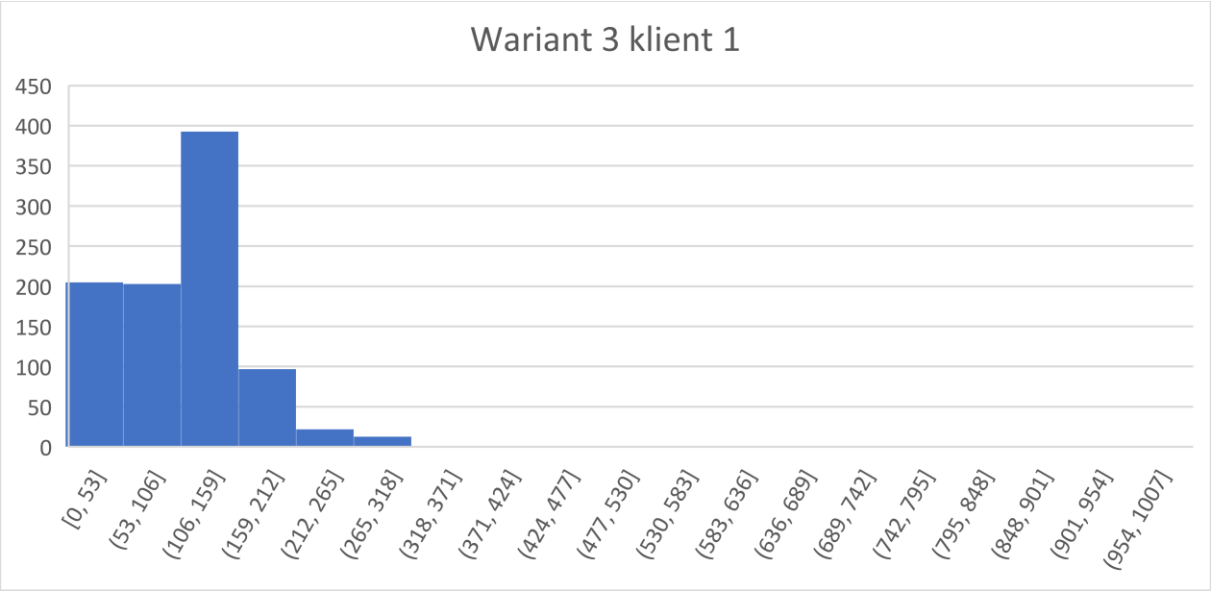


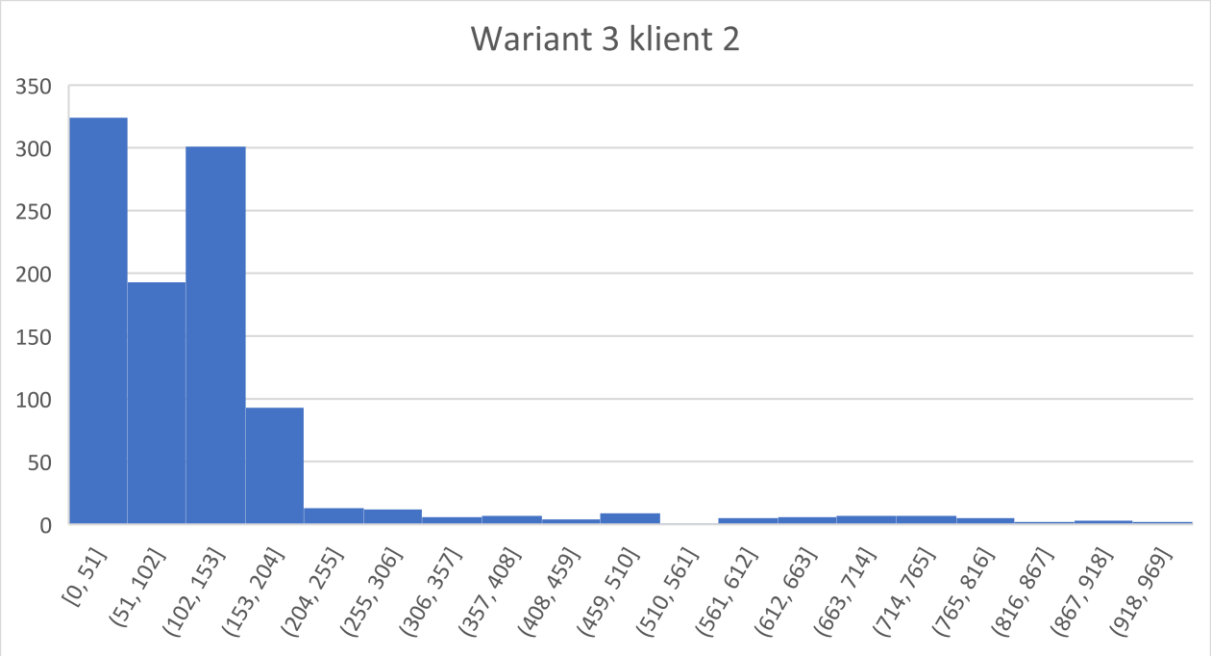




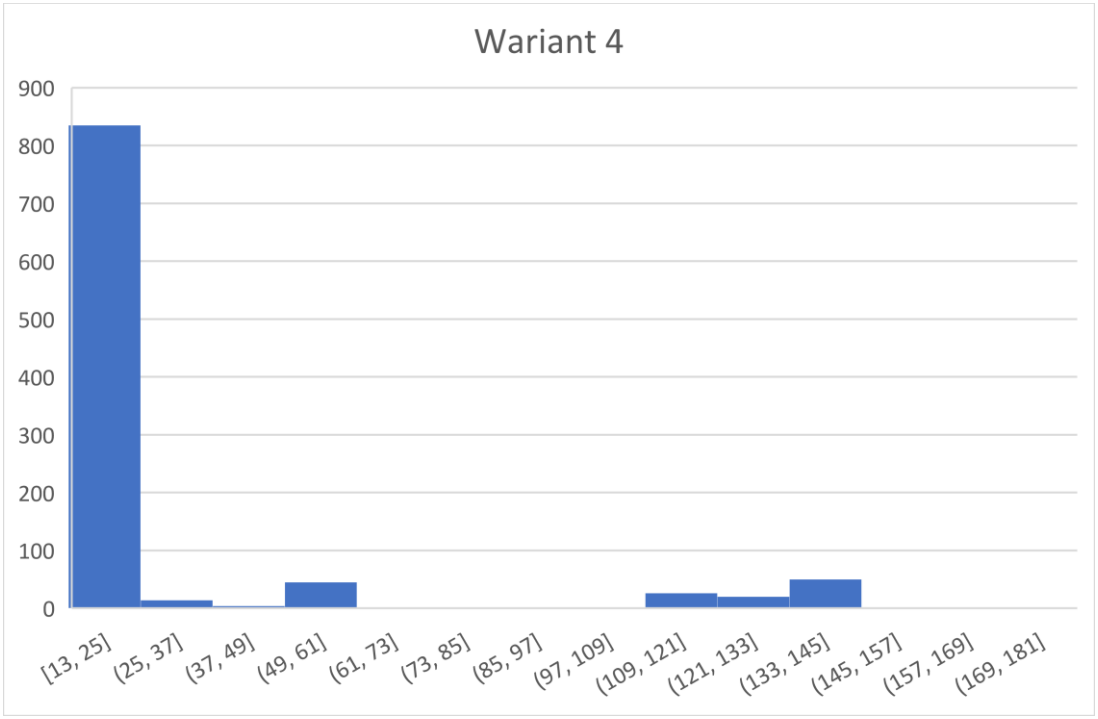
3) Trzeci wariant: 3 klientów, 2 rdzenie, brak obciążenia
   Macxpus = 2
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 270000.







4) Czwarty wariant: 1 klient, 4 rdzenie, brak obciążenia
   Maxpus = 4
   Program cw4a uruchomiliśmy w podany sposób: cw4a 3 1000 10000 325000.



W niektórych przypadkach wartości graniczne bardzo psuły czytelność histogramu. Z tego powodu postanowiliśmy nie prezentować ich na histogramie. Wszystkie wartości powyżej 1000 zostały nieuwzględnione.

5. Aktywne oczekiwanie

```
while(smpnum<nsmp) {
    int j;
    char line[200];
    //pthread_rwlock_rdlock(&rbuf->buflock);
    //check if there is any the new data
    pthread_spin_lock(&rbuf->cvar_lock);
    if(rbuf->head != rbuf->tail[ncli]) {
        struct timeval tv1,tv2;
        gettimeofday(&tv1,NULL);
        //memcpy(&tv2,&rbuf->buf[rbuf->tail].tstamp);
        tv2 = rbuf->buf[rbuf->tail[ncli]].tstamp;
        rbuf->tail[ncli]++;
        if (rbuf->tail[ncli] == BUF_LEN) rbuf->tail[ncli] = 0;
        pthread_spin_unlock(&rbuf->cvar_lock);
        smptime = 1000000*tv2.tv_sec + tv2.tv_usec;
        deliverytime = 1000000*tv1.tv_sec + tv1.tv_usec;
        int tdel = deliverytime - smptime;
        printf("Sample %d, client %d, delivery time: %d\n",smpnum,ncli,tdel);
        sprintf(line,"%d, %lu, %lu, %d\n",smpnum,smptime,deliverytime,tdel);
        //The next instruction is an example of incorrect implementation!
        //Now we only detect possible error, but we should also check the number of written bytes
        //and repeat writing if only part of the line was written?
        assert(write(fout,line,strlen(line))>0);
        sync();
        smpnum++;
        //Here we simulate delay for data processing
        for(j=0;j<ndel;j++)
           dummy++;
    } else {
        // pthread_cond_wait(&rbuf->cvar,&rbuf->cvar_lock);
        pthread_spin_unlock(&rbuf->cvar_lock);
    }
```

Udało nam się wprowadzić poprawkę uwzględniającą aktywne oczekiwanie (wariant 1), jednak nie udało nam się jej przetestować. Zbudowaliśmy od nowa .ipk i wprowadziliśmy na RPi4 jednak nie zdążyliśmy przeprowadzić obserwacji.