|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p\_r** | **p\_w** | **deltaT** | **deltaX** | **time** |
| **4** | **4** | **1** | **1** | **2984149444** |
| **2** | **3023242510** |
| **4** | **3002577860** |
| **2** | **2** | **1** | **4004678122** |
| **2** | **3937576887** |
| **4** | **3990071326** |
| **4** | **2** | **1** | **3008426260** |
| **2** | **2989504928** |
| **4** | **2994601243** |
| **4** | **8** | **1** | **2999680560** |
| **2** | **2992522544** |
| **4** | **2987899281** |
| **4** | **4** | **2** | **10** | **3043914880** |
| **4** | **4** | **2** | **50** | **6191907994** |

Folosim TCP pentru comunicarea dintre clienti si server, fiecare find lansati intr-un thread separate din main. Alocam un thread pool pentru citirea datelor prin socket trimise de client serverului, iar la primirea unui rezultat adaugam in blocking queue. In acelasi timp, avem threaduri worker care prelucreaza datele din blocking queue.

        for (int i = 0;i < countries;++i)

            thread\_pool.detach\_task([&, i] {

            if ((sockets[i] = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen)) < 0) {

                perror("Accept failed");

                close(server\_fd);

                exit(EXIT\_FAILURE);

            }

...

Flowul actualizarii listei functioneaza la fel ca laboratorul trecut, diferenta fiind ca datele vor veni la intervalul deltaX. Ce difera este cererea clientilor pentru rankinguri, pe care serverul le calculeaza asincron si le trimite cand sunt gata. In acelasi timp, un ranking nu va fi recalculat daca acesta a fost inceput sa fie facut cu un timp <= deltaT, si se va astepta terminarea rankingului (daca acesta nu este deja gata). Gestionarea se face folosind apeluri asincrone din C++, utilizand std::future.

auto calculateRankings = [&](bool isFinal) {

            return std::async(std::launch::async, [&, isFinal] {

                chrono::time\_point<chrono::system\_clock> now = chrono::system\_clock::now();

                mutex.lock();

                auto diff = chrono::duration\_cast<chrono::milliseconds>(now - last);

                if (isFinal || std::chrono::milliseconds(diff) > std::chrono::milliseconds(deltaT)) {

                    last = now;

                    auto future = std::async(std::launch::async, [&] {

…

                        });

                    cerr << format("recalculating {} {} {}\n", isFinal, std::chrono::milliseconds(diff), std::chrono::milliseconds(deltaT));

                    future.wait();

                    currentFuture = std::move(future);

                    mutex.unlock();

                    currentFuture = nullopt;

                }

                else {

                    mutex.unlock();

                    cerr << "reusing\n";

                    if (currentFuture.has\_value())

                        currentFuture.value().wait();

                }

                });

            };

Din benchmark, observam ca overheadul este dat de comunicare si de producatorii de date, un p\_r mai mare influentand foarte mult timpul de executie. In acelasi timp, cresterea p\_w nu a avut aproape nicio diferenta. O valoare deltaT mai mare scade timpul de executie (desi foarte putin), asa cum era de asteptat. O valoare deltaX mai mare creste timpul de executie, desi asta este sesizabil doar de la un anumit prag.