# Tehnična dokumentacija – projekt PIPR

Na področju prometa smo izvedli paralelizacijo pri pridobivanju podatkov o stanju na cestah. Raziskali smo veliko možnosti in našli potencialne dele, kjer bi bila paralelizacija lahko zelo učinkovita.

Podatke smo pridobili s pomočjo praskanja s strani :

[**https://promet.si/portal/sl/stevci-prometa.aspx**](https://promet.si/portal/sl/stevci-prometa.aspx)

Strežnik, ki smo ga implementirali tekom prejšnjega semestra, in za katerega smo že praskali nekatere podatke iz zgoraj omenjene strani že deluje in obdeluje podatke v nodejs okolju, toda teh podatkov je relativno malo in se nam paralelizacija tega procesa ni zdela smiselna. Odločili smo se praskati druge podatke, sicer iz iste strani, toda v mnogo večji količini.

Tukaj smo opazili, da je polnjenje glavne tabele strani, kjer se nahajajo naši podatki zelo počasno, na nekaterih računalnikih je namreč trajalo tudi po 5 sekund in več. Z orodji smo pregledali, kako se sama tabela polni, saj bi lahko v primeru, da najdemo vir, to optimizirali s pomočjo paralelizacije in z več procesi tabelo hitreje napolnili. Po raziskavi nam samega vira ni uspelo najti, strinjali smo se, da bomo namesto paralelizacije prenosa opravili paralelizacijo nad obdelovanjem prenesenih podatkov, kar je resda manjša pohitritev, toda vseeno zelo koristna. V namen pridobitve podatkov smo napisali nov scraper.

Poleg podlage **nodejs** za delovanje potrebujemo module **puppeteer, cheerio** ter **fs**.

Puppeteer je zaslužen za izvajanje skript strani in pravziranje prenosa, dokler se dinamična vsebina strani v celoti ne naloži. Cheerio uporabljamo, da iz surovega tekstovnega izhoda puppeteerja poberemo samo podatke, ki nas zanimajo, nato pa jih s fs modulom sproti zapisujemo v izhodno datoteko.

Sama parelelizacija je izvedena s uporabo MPI, dosti smo si lahko pomagali s primeri iz prejšnjih vaj. Program, napisan v c++, izvršujejo štirje procesi na način master/slave oz. manager/worker. Proces 0 (master) najprej iz vhodne datoteke prebere vse podatke, jih shrani v ločene podatkovne strukture, ki jih v nadaljevanju posreduje worker procesom 1, 2 in 3. Ti izvajajo računski del programa, po koncu seštevke vrednosti pošljejo nazaj procesu 0, ki pred izpisom izračuna še končni rezultat, ki nas pravzaprav zanima, torej povprečja vrednosti.

## Paralelizacija in porazdeljenost

Pri sami paralelizaciji smo uporabili VPN, katerega smo vzpostavili sami. Sam host našega VPN je bil DigitalOcean, VPN pa smo ustvarili s pomočjo **Algo.** Postopek je izgledal nekako takole:

Graphical user interface, text, application

Description automatically generated

Slika : Ustvarjanje ključev za člane skupine

Text

Description automatically generated

Slika : Zagon na strežniku

Potem smo vsak svoj ključ uporabili na orodju **WireGuard** in se s tem povezali v svoje omrežje.

Graphical user interface

Description automatically generated

Slika : VPN strežnik