Lõputöö

Tõhusate Andmestruktuuride ja Algoritmide Arendamine Veebirakendustele: Pythoni Jõudluse Optimeerimine WebAssembly (Wasm) abil

Sergei Ivanov

IT Süsteemide arendus, Tartu Ülikooli

Juhendaja: Andre Sääsk

October 31, 2025

Contents

1. Sissejuhatus

Andmestruktuuride ja algoritmide (DSA) optimeerimine on skaleeruvate infosüsteemide loomisel kriitilise tähtsusega. CPythoni standardimplementatsioonides puuduvad teatud tõhusad andmestruktuurid, näiteks isetasakaalustuvad puud (nt Red-Black Tree), mis tekitab vajaduse kasutada spetsiifilisi kolmanda osapoole lahendusi. Käesoleva uurimuse fookuses olevas rakenduses leiti, et DSA lahenduse esialgne implementeerimine näitas suure andmemahu (üle 100 miljoni kirje) töötlemisel ebapiisavat jõudlust, pikendades kriitiliste ülesannete täitmise aja kuni kahe minutini. See empiiriline tähelepanek viitab interpreteeritava keele (Pythoni) jõudluspiirangutele ja loob aluse optimeerimismeetodite uurimiseks.

1.1 Teema aktuaalsus

Hoolimata sellest, et arendajad saavad kasutada olemasolevaid teeke, on spetsiifiliste DSA moodulite arendamine otstarbekas, kuna see võimaldab kiiremat prototüüpimist ja uute ideede testimist. Python on endiselt üks enimkasutatavaid programmeerimiskeeli, eriti valdkondades, kus jõudlus ja skaleerimine on olulised:

1. Masinõpe ja andmeanalüüs
2. Veebirakendused (taustsüsteemid)
3. Automatiseeritud skriptid

RESTful API-de arendamine microteenuste abil on levinud praktika. Python pakub selleks efektiivseid raamistikke, nagu FastAPI, kuid kiirete serveripoolsete DSA lahenduste puudumine pärsib nende rakenduste maksimaalset potentsiaali. Käesolev töö loob lahenduse sellistele probleemidele, avades võimalused DSA sujuvaks integreerimiseks suure jõudlusega veebirakendustesse.

1.2 Uurimisprobleem

Pythoni kui interpreteeritava keele jõudluspiirangute tõttu on kriitilise tähtsusega uurida ja võrrelda erinevaid optimeerimistehnikaid, et tagada DSA-de maksimaalne läbilaskevõime. Peamised uuritavad lahendused on:

1. Cython ja C integreerimine (Native Extensions)
2. Zig ja Rust
3. WebAssembly (WASM)
4. JIT (Just-in-Time) kiirenduse kasutamine.

Käesoleva lõputöö raames uuritakse ja võrreldakse eelnimetatud Cythoni, Zig-i, Rusti ja WASM-i lahendusi.

1.3 Hüpoteesid

H1 (Peamine Hüpotees): Kriitiliste, arvutusmahukate andmestruktuuride (DSA) implementeerimine süsteemikeeles (nt Rust) ja nende WebAssembly (Wasm) kaudu Pythoni veebirakendusse integreerimine pakub standardse Pythoni implementatsiooniga võrreldes vähemalt X-kordse (Tähelepanu: X peab olema konkreetne number, näiteks 5 või 10) jõudlusvõidu sama ülesande täitmisel).

H2 (Integratsiooni Jõudlus): Wasm-põhise mooduli integreerimisega kaasnev andmevahetuse overhead (st side Pythoni ja Wasm-i vahel) on piisavalt väike, et ei nulli ära madala taseme keele arvutuslikku jõudlusvõitu.

H3 (Arenduse Efektiivsus): Wasm/Python hübriid-lähenemine on arenduslikult otstarbekas (mõistlikult keerukas seadistada ja hooldada) kriitiliste jõudlusosade jaoks, pakkudes paremat lahendust kui CPythoni Native Extension (C/Cython) lähenemine.

1.4 Töö eesmärk

Töö eesmärk on arendada korduvkasutatav DSA-moodul süsteemikeeles (nt Rust/Zig/Cython) ja integreerida see Pythoni veebirakendusse Wasm-i kaudu, et empiiriliselt testida püstitatud hüpoteese jõudluse ja arenduse efektiivsuse osas.

2. Kirjanduse Ülevaade

2.1 Andmestruktuuride ja Algoritmide Teoreetiline Alus

Tarkvarasüsteemide skaleeritavuse tagamiseks on kriitilise tähtsusega analüüsida andmestruktuuride ja algoritmide asümptootilist jõudlust (Cormen et al., 2009). Peamiseks hindamisvahendiks on Big O notatsioon ($O(\cdot)$), mis kirjeldab algoritmi aeg- ja mälukomplekssust sisendi suuruse (N) kasvades. Kuna Red-Black Tree (RBTree) on isetasakaalustuv binaarne otsingupuu (BST), on selle operatsioonide (sisestamine, otsing, kustutamine) teoreetiline keerukus tagatud logaritmiline, s.o $O(\log N)$. See tagab optimaalse jõudluse suurte andmehulkade puhul. Käesoleva uurimuse fookuses on DSA, mille teoreetiline kompleksus on madal, kuid praktiline jõudlus interpreteeritavas keeles on osutunud ebarahuldavaks.

2.2 Pythoni Jõudluspiirangud ja Optimeerimisvajadus

Pythoni kui interpreteeritava keele peamine jõudluspiirang on seotud Global Interpreter Lock'iga (GIL).

2.3 WebAssembly (Wasm) Rakendamine Serveri Taustsüsteemides

Wasm-i tutvustus ning selle rakendamine väljaspool brauserit, kasutades Wasm-runtime'e (nt Wasmtime) serveripoolses kontekstis.

2.4 Süsteemikeelte Võrdlus (Rust, Zig, Cython)

Võrdlus Rusti ja Zig-i pakutava mäluturvalisuse ja Cythoni traditsioonilise Native Extension lähenemisega Pythoni jõudluse optimeerimise kontekstis.

3. Metoodika

Käesolevas töös kasutatakse eksperimentaalset uurimismeetodit, et võrrelda kolme DSA-implementatsiooni jõudlust.

4. Viidatud Allikad

- Beazley, D. (2010). Understanding the Python GIL.
- Cormen, T. H., et al. (2009). *Introduction to Algorithms*.
- Haas, A., et al. (2017). Bringing the Web Up to Speed with WebAssembly.
- Matsakis, N., & Klock, F. (2014). The Rust Language Design.
- Wagner, L., et al. (2022). Server-Side WebAssembly.
- Zakai, A. (2021). Emscripten and the Evolution of WebAssembly Performance.