Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Distribuovaný výpočet parametrů modelu dynamiky glukózy

Plzeň, 2016 Martin Šlapa

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin ŠLAPA**
Osobní číslo:	**A13N0088P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Distribuované systémy a počítačové sítě**
Název tématu:	**Distribuovaný výpočet parametrů modelu dynamiky glukózy**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Rámcově se seznamte s homeostázou glukózy.
2. Seznamte se s již navrženým modelem dynamiky glukózy.
3. Navrhněte dynamický load-balancer, který rozdělí výpočetní zátěž existující metody výpočtu mezi několik uzlů.
4. Implementujte navržený load-balancer a dodržte stávající rozhraní navržené vedoucím práce.
5. Zhodnoťte dosažené výsledky.

Rozsah grafických prací:	**dle potřeby**
Rozsah kvalifikační práce:	**doporuč. 50 s. původního textu**
Forma zpracování diplomové práce:	**tištěná**

Seznam odborné literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Ing. Tomáš Koutný, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání diplomové práce:	**1. září 2015**
Termín odevzdání diplomové práce:	**12. května 2016**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Doc. Ing. Přemysl Brada, MSc. PhD.
vedoucí katedry

V Plzni dne 15. září 2015

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Tomáši Koutnému, Ph.D. za odborné vedení, podnětné námitky a rady k realizaci diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborných zdrojů a literatury, které jsou citovány a uvedeny v seznamu literatury.

V Plzni dne 23. 6. 2016 _____

Abstrakt

Česky

Tato práce rozšiřuje způsob výpočtů programu Glucose Predictor, který hledá vztah mezi naměřenými hodnotami z podkoží a skutečnou hladinou glukózy v krvi. Jelikož některé metody výpočtu jsou velmi náročné nebo je příliš velké množství dat ke zpracování, výpočet na jediném počítači může být příliš dlouhý. Proto je potřeba nalézt způsob, jak výpočet realizovat distribuovaně. Tato práce implementuje řešení distribuovaného výpočtu pro více počítačů. Seznamuje čtenáře se základními problémy diabetu prvního a druhého typu u postižených pacientů. Zabývá se obecnými mechanizmy pro distribuovaný výpočet.

English

This thesis enhances the computation capabilities of the Glucose Predictor software. Glucose Predictor is used to calculate the parameters of relation between interstitial glucose levels and blood glucose levels. Because some of the calculations are resource intesive or there is too much data to process the calculation on single machine can take too long. To solve this issue it is necessary to do the calculation in distributed manner. This work implements the distributed solution for multiple machines. This work introduces the basic problems of type 1 and type 2 diabetes. And covers the general approaches to distributed computation systems.

Obsah

1 Úvod

Cukrovka je onemocnění sužující lidstvo ve značné míře již od minulého století. Dříve byla cukrovka vzácností, ale v průběhu 20. století, hlavně v souvislosti se změnou životního stylu lidí, došlo k jejímu prudkému nárůstu jak u nás, tak i ve světě. V dnešní době by se dalo říci, že se jedná o epidemii.

Program Glucose Predictor, vyvíjen a dodán vedoucím diplomové práce, hledá vztah mezi naměřenými hodnotami z podkoží a skutečnou hladinou glukózy v krvi. Pro nalezení parametrů modelu dynamiky glukózy v krvi využívá několik algoritmů – metod výpočtu, které jsou velmi náročné.

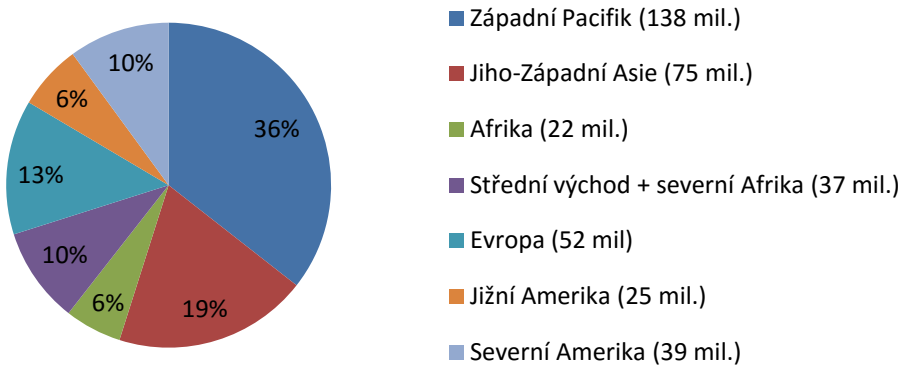
Účelem této práce je vytvořit distribuované řešení výpočtu pro program Glucose Predictor. Navrhnout a implementovat dynamický load-balancer, který rozdělí výpočetní zátěž existující metody výpočtu mezi několik uzlů a zároveň se bude řídit rozhraním navrženým vedoucím práce.

Dále pak u implementovaného řešení provést měření jednotlivých metod a porovnat časy výpočtů s původním nedistribuovaným řešením. Dále otestovat, jakým způsobem load-balancer přerozděluje části distribuovaného výpočtu podle různého zatížení řešitelských stanic.

2 Diabetes

Diabetes mellitus[1] je heterogenní skupina chronických onemocnění, které se vyznačují společným rysem – zvýšenou hladinou glukózy v krvi. Diabetes může mít různé příčiny, které doposud nejsou úplně objasněné. [1]

Podle IDF[2] mělo v roce 2014 diabetes přibližně 8,3% světové populace (tj. asi 38,8 milionu lidí). Do roku 2035 se odhaduje, že se toto číslo zvýší až na 243,8 milionu lidí. Jednotlivé zastoupení výskytu diabetu na jednotlivých kontinentech je vidět na obrázku *Obrázek 1*. [1]



Obrázek 1: Zastoupení diabetu na jednotlivých kontinentech, převzato z [1]

Přibližně polovina lidí, kteří trpí cukrovkou, o svém onemocnění neví. Nemoc se projevuje pozvolna a zjištění onemocnění lze odhalit až při preventivní prohlídce při rozboru krve. Ale ani nahodilý rozbor krve nemusí s určitostí odhalit problém.

Zejména u diabetiků typu-1 se po několik dní zaznamenává hladina glukózy pomocí přístroje kontinuálního měření koncentrace glukózy v podkoží - CGMS[3]. Malý senzor se zapíchne do podkoží a diabetik a lékaři získávají představu o tom, jak se mění koncentrace glukózy v těle pacienta.

U zdravého člověka se pohybuje hodnota hladiny cukru v krvi v rozmezí 3,5 – 6,5 mmol/l krve. Pokud je hodnota vyšší než 7 mmol/l, může to být příznakem diabetu.

[1] česky úplavice cukrová

[2] International Diabetes Federation

[3] Continual Glukose Monitoring System

2.1 Typy diabetu

Nejčastějšími typy Diabetes mellitus jsou:

- 1. typ – *absolutní* nedostatek inzulinu,
- 2. typ – *relativní* nedostatek inzulinu.

Dále existují *gestační diabetes* a *ostatní specifické typy diabetu* (tzv. sekundární diabetes), které mohou být způsobeny genetickým defektem ß buněk a insulinu, změnou glukotolerance, onemocněním slinivky apod.

2.1.1 Diabetes 1. typu

Typ 1 je způsoben nedostatečnou produkcí inzulínu, což je hormon slinivky břišní. Jedná se o autoimunitní onemocnění, kdy dochází k ničení buněk slinivky břišní vlastní imunitním systémem, tzv. fáze *autoimunity*. Konkrétně jde o buňky, jež jsou zodpovědné za tvorbu inzulínu. Inzulín je pro lidské tělo životně důležitou součástí, umožňuje utilizaci glukózy buňkami. Glukóza je energie pro všechny buňky, bez které zanikají. Diabetes prvního typu se objevuje v dětství či dospívání. V individuálních případech je možný i pozdní vznik onemocnění po třicátém roce věku označován jako LADA[4]. [1]

2.1.2 Diabetes 2. typu

Cukrovka 2. typu postihuje častěji starší osoby a osoby s nadváhou či obezitou. Málo pohybu, nepravidelný příjem potravy, nadměrný stres, ale také genetické dispozice výrazně přispívají ke vzniku tohoto typu cukrovky. Častěji než nedostatek mívají tito pacienti normální množství či dokonce nadbytek inzulínu.

92 % nemocných cukrovkou tvoří diabetici 2. typu. [2]

Cukrovka tohoto typu bývá charakterizována zejména nedostatečnou citlivostí tkání k účinkům inzulinu, tj. inzulinovou rezistencí. K dosažení normální hladiny cukru v krvi je nutné zvýšené množství inzulinu. Určitý stupeň poruchy vylučování inzulinu najdeme i v tomto případě. Vázne uvolňování již vytvořeného inzulinu ze slinivky po stimulaci potravou. Postižená bývá hlavně časná fáze uvolnění inzulínu. Inzulínu je uvolněno zpočátku nedostatečné množství. Proto v další fázi musí slinivka uvolnit zvýšené množství inzulínu, aby korigovala takto zvýšenou hladinu cukru v krvi.

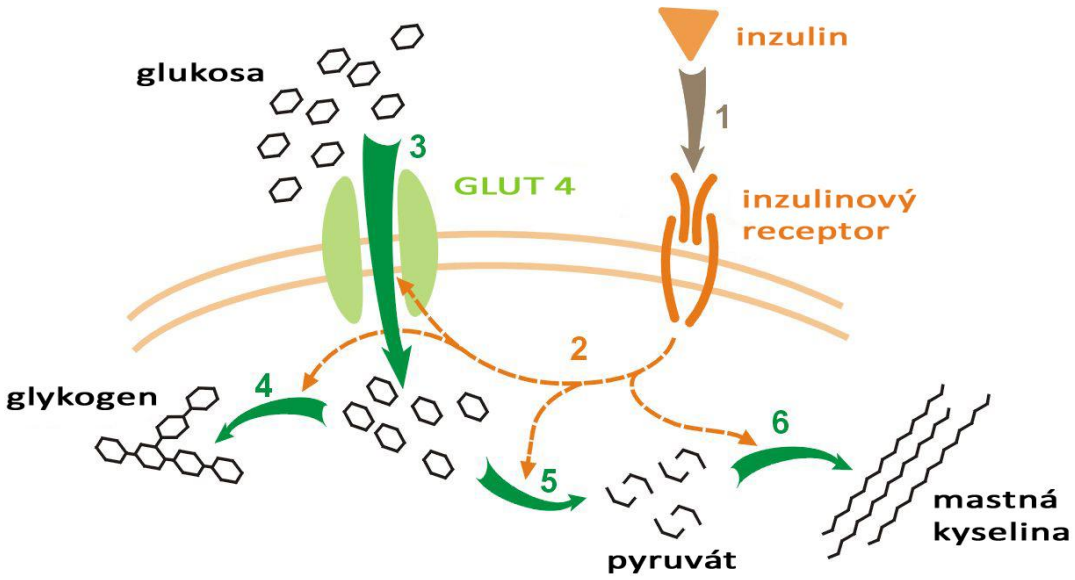
[4] latent autoimmune diabetes of adults

2.2 Glukóza

Glukóza[5], hovorově hroznový cukr nebo krevní cukr, je monosacharid. Pro všechny buňky je energetickým substrátem, který se podílí na intermediárním metabolismu[6]. Je přijímána potravou buď volně (ovoce nebo jiná jídla s obsahem cukru), nebo jako součást disacharidů a polysacharidů. Z trávicího traktu se do krve vstřebává pouze volná glukóza. Distribuce k buňkám probíhá v těle prostřednictvím krevního oběhu.

2.3 Inzulín

Inzulín je hormon, který produkují ß buňky Langerhansových ostrůvků slinivky břišní. Inzulín moderuje spotřebu glukózy v krvi. Je do krve vylučován postupně nehledě na příjem potravy. Množství inzulínu, které je vylučováno nezávisle na potravě, koresponduje s obsahem cukru v krvi. Jednoduše řečeno, čím více cukru je v krvi, tím by mělo být i více inzulínu. Následující obrázek zobrazuje reakci inzulínu s glukózou.



Obrázek 2: Systém inzulin – glukagon, převzato z [1]

[5] chemický vzorec $C_6H_{12}O_6$

[6] biochemické přeměny odehrávající se mezi katabolickou a anabolickou částí metabolismu

2.4 Příznaky

Nejčastější příznaky začínající cukrovky 1. typu:

- Žízeň, hlad, ale také hubnutí, únava, pocení a tzv. vlčí hlad (kdy musíme sníst, co vidíme), zkrátka musíme stále jíst a pít, následně hodně močíme a tento koloběh se pořád opakuje. Tyto příčiny se většinou projevují u diabetu 1. typu.
- Cukrovku 1. typu dostávají také lidi po operacích, v těhotenství, kvůli stresu.

Typické příznaky a rizika cukrovky 2. typu:

- Diabetes přichází s vyšším věkem, špatnou životosprávou, stresem a nedostatkem pohybu.
- Začátek diabetu 2. typu je na rozdíl od diabetu 1. typu často nenápadný a někdy se zjistí až při rozvoji diabetických komplikací.
- U tohoto typu diabetu je současně přítomná porucha vylučování a účinku inzulinu ve tkáních.
- Některé situace nebo události podporují vznik diabetu 2. typu (rizikové faktory). Patří mezi ně nadváha, nedostatek pohybu, nevhodná strava a stres. Diabetes 2. typu je v určité míře dědičný.
- Vzhledem k tomu, že u diabetu 2. typu je nedostatek inzulínu částečný a nikoliv úplný, stoupá koncentrace glukózy v krvi pomalu a tělo se vyšší hladině přizpůsobuje. V důsledku toho se mohou projevit stejné příznaky, jako u diabetu 1. typu, budou však méně intenzivní.

2.5 Komplikace

Onemocnění diabetes zvyšuje riziko vzniku nejrůznějších zdravotních komplikací. V případě, kdy dochází ke značnému zvýšení hladiny glukózy v krvi, zvyšuje se riziko poškození srdce, cév, zrakového ústrojí, ledvin, nervů nebo zubů. Naopak u nižší hladiny může vést ke snížení psychické výkonnosti, slabosti, bolesti hlavy, pocitu hladu apod., v extrémním případě až ke stavu bezvědomí.

Nejčastějším případem jsou kardiovaskulární choroby, selhání zrakového ústrojí nebo ledvin. Dále také tzv. diabetická noha, kdy vysoká hladina glukózy v krvi poškozuje cévy, což může mít za následek amputaci části nohy.

2.5.1 Onemocnění ledvin, nervů a zrakového ústrojí

Onemocnění ledvin se nazývá *diabetická nefropatie* a objevuje se u lidí, kteří mají dlouhodobě zvýšenou hladinu glukózy v krvi. Tím se snižuje účinnost ledvin a může to vést až k úplnému selhání funkce ledvin.

Onemocnění nervů se nazývá *diabetická neuropatie* a opět se objevuje u lidí, kteří mají dlouhodobě zvýšenou hladinu glukózy v krvi. Projevuje se poškozením funkce a struktury periferního nervstva. Onemocnění zasahuje nervy motorické, senzitivní a vegetativní[7].

Onemocnění zrakového ústrojí se nazývá *diabetická retinopatie* a projevuje se obvykle u pacientů v průběhu deseti let od propuknutí diabetu. U pacienta dochází ke snížení zrakové ostrosti, v nejhorším případě až slepotě. Cukrovka způsobuje slepotu zhruba ve 2 % případů a je její nejčastější příčinou ve vyspělých zemích. [1] [3]

2.5.2 Kardiovaskulární onemocnění

Tato onemocnění postihují srdce a cévy a jsou nejčastějším důvodem úmrtí diabetiků. Mohou zapříčinit nepříjemné onemocnění koronární artérie[8]. Příčinou těchto onemocnění je obvykle zvýšený krevní tlak nebo zvýšená hladina cholesterolu.

2.5.3 Komplikace v těhotenství

Pokud jsou ženy diabetičky, mají významně menší šanci otěhotnět oproti zdravým ženám. Naopak šance porodu zdravého dítěte se téměř neliší za předpokladu, že diabetes je správně léčen a stabilizován.

Dědičnost diabetu je závislá na rodičích. V případě, že ho má pouze matka, je pravděpodobnost přenosu okolo 2 %. Pokud pouze otec, je riziko 5-6%. Pokud ho mají oba rodiče, riziko přenosu na narozeného jedince dosahuje mezi 10-15 %.

Tzv. *těhotenská cukrovka* může nastat, i přestože matka vůbec není diabetičkou. Stačí pouze, aby se projevila zvýšená hladina glukózy během těhotenského vyšetření. Stav je obvykle pouze dočasný a po porodu se vše vrací k normálu. Pokud je ženám těhotenská cukrovka diagnostikována, musí chodit na pravidelné prohlídky. Kdyby nebyla cukrovka během těhotenství objevena, mohlo by to vést k akutním komplikacím po

[7] tzn., že nepodléhají naší vůli – např. pohyb střev nebo pocení

[8] tepny přivádějící krev do srdeční svaloviny

porodu, popřípadě se velmi zvyšuje riziko předání cukrovky druhého typu na narozeného jedince. [1]

2.6 Akutní komplikace

Mezi hlavní akutní komplikace patří hypoglykémie (nízká hladina glukózy v krvi) a hyperglykémie (vysoká hladina glukózy v krvi). Jsou zde i další komplikace spojované se srdeční činností, zrakem nebo trávením. [1]

2.6.1 Hypoglykémie

Hypoglykémie je nízká hladina glukózy (krevního cukru) pod hodnotu 3,3 mmol/l – vzniká v důsledku nedostatečného krytí potřeby glukózy např. při:

- neobvykle intenzivní fyzické zátěži,
- vysokému stresu,
- alkoholu,
- opomenutí dávky jídla, atd.

Nerozpoznané a neléčené hypoglykémie mohou vést v extrémních případech k trvalému poškození mozku. [1]

2.6.2 Hyperglykémie

Hyperglykémie je vysoká hladina glukózy (krevního cukru) nad hodnotu 11 mmol/l a nastává v důsledku nedostatku inzulínu v krvi např. při:

- vynechané dávce inzulinu,
- nízké dávce bolusu[9] k jídlu,
- vytažení nebo zalomení kanyly u pacientů na inzulínové pumpě,
- přerušení dodávky inzulínu, atd.

Pokud není hyperglykémie zavčasu odhalena a kompenzována, obvykle dochází k diabetické katoacidóze, což je velmi závažný stav rozvratu vnitřního prostředí organizmu, který si může vyžádat hospitaci. [1]

[9] dávka inzulínu, která je dodána tělu před jídlem nebo během jídla, aby vyrovnala glukózu ze stravy nebo snížila zvýšenou hladinu cukru v krvi.

2.7 Možnosti podpůrné léčby

I přes dnešní nové metody léčby je vždy důležité zavčas objevit chorobu. Pokud se ji podaří objevit u pacienta v raném stádiu, je předán do péče imunologa, který se v první fázi snaží zastavit imunitní systém léky. Ty potlačují další poškozování buněk slinivky. Pokud ale léky nezabírají nebo imunitní systém zničil již všechny ß buňky, nemá pacient jinou možnost, než do konce života užívat inzulín.

Aby se choroba dále nezhoršovala, je nutné držet se správného stravování, sportovat a vyhýbat se stresu. V každém případě je vyžadováno přesné dávkování inzulinu. Za předpokladu, že pacient vše dodrží, má velkou pravděpodobnost, že nebude žít kratším životem než zdravý člověk. [1]

2.7.1 Transplantace

Transplantací slinivky břišní nebo Langerhansových ostrůvků lze dnes úplně vyléčit cukrovku 1. typu. Avšak po transplantaci je pacient donucen trvale užívat léky s látkami, které potlačují imunitní systém – tzv. *imunosupresiva*. Je to z důvodu, aby tělo neodmítalo transplantovanou tkáň. Kvůli značným vedlejším účinkům těchto léků se k transplantaci přistupuje pouze v krajních případech, kdy stádium cukrovky přímo ohrožuje život pacienta. [1]

2.7.2 Umělá slinivka

Umělá slinivka již dnes není záležitost pouze laboratoří či přísně kontrolovaného nemocničního prostředí. Testuje se na diatáborech[10], v každodenním životě i za kontrolovaných podmínek v domácím prostředí. Umělá slinivka se skládá z:

- **inzulínové pumpy**, která dodává potřebné množství inzulínu pacientovi do těla,
- **CGM[11] senzoru**, který měří hladinu glukózy v podkoží (bohužel zatím v dnešní době není natolik přesný, aby nahradil funkci glukometrů[12])
- **programového vybavení**, které zaznamenává údaje ze senzoru, predikuje hladinu glukózy v krvi a řídí dávkování skrze inzulínovou pumpu.

[10] tábor pro diabetiky zaměřený na zábavu, odpočinek, sportování, soutěže, hry, táboráky, výlety apod.

[11] continuous glucose monitoring – volně přeloženo jako kontinuální měření glukózy

[12] přístroj určený k domácímu měření hladiny glukózy v krvi

Výzkumy ukazují, že umělá slinivka je schopná dávkovat inzulín lépe než průměrný diabetolog nebo pacient. Roman Hovorka, český rodák působící na University of Cambridge, se zabývá výzkumem umělé slinivky. Jeho predikční software pracuje na bázi učení, kdy nejprve testuje, jak by organismus diabetika reagoval na podanou dávku inzulínu a současně měří hodnoty. Následně již dávkuje inzulín správně, dle potřeby pacienta. Algoritmus pro odhad hladiny glukózy je velmi náročný kvůli častým opakováním výpočtu nad spousty dat. Často je proto využíváno moderního „chytrého" mobilního telefonu, který má pacient u sebe. Komunikace s CGM snímačem a inzulínovou pumpou probíhá obvykle přes rozhraní Bluetooth. [1]

2.7.3 Prevence onemocnění

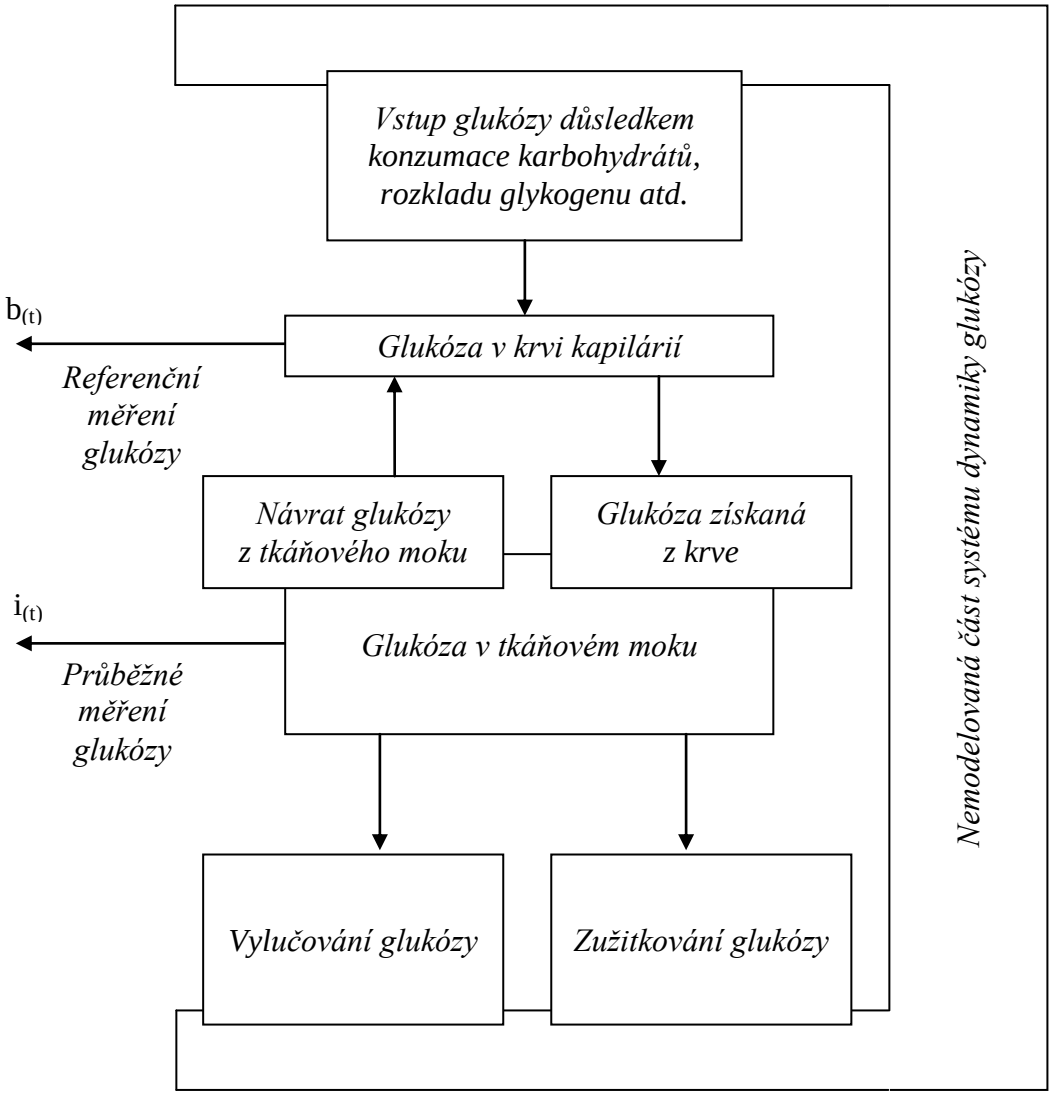
Předejít diabetu 1. typu není možné. Předchází mu mnoho faktorů, které nejsme schopni ovlivnit. Naopak diabetu 2. typu se můžeme vyvarovat a to tím, že odstraníme ze svého života všechny rizikové faktory pro toto onemocnění. To znamená dodržování přísných pravidel v každodenním životě. Mezi hlavní pravidla patří:

- kvalitní a vyvážená strava,
- pravidelné cvičení,
- snížení váhy (v případě obezity),
- omezit, v nejlepším případě zbavit se návykům na kouření nebo konzumaci alkoholu,
- snažit se vyhnout jakýmkoliv stresovým situacím.

3 Model dynamiky glukózy

Model je založen na systému dynamiky glukózy a spojuje aktuální hodnoty BG[13] a IG[14] s predikovanými hodnotami IG [4]. Popisuje vztah mezi BG a IG, který potřebujeme znát pro léčbu cukrovky.

Na obrázku Obrázek 3 je zobrazen tok glukózy, který se přímo vztahuje k modelu. Glukóza se může objevit v krvi např. důsledkem konzumace karbohydrátů, rozkladem jaterního glykogenu nebo infuzí. Z krve je glukóza přenášena skrze kapilární membrány do tkáňového moku. Rychlost tohoto přenosu je omezena velikostí povrchu kapilární membrány, propustností membrány a koncentračním gradientem mezi BG a IG. Dále



Obrázek 3: Blokové schéma toku glukózy ve vybrané části dynamiky glukózy [4]

[13] z ang. *blood glucose* - glukóza v krvi

[14] z ang. *interstitial glucose* - glukóza v podkožní tkáni

způsobuje zpoždění v přenosu glukózy z krve do moku. V tkáňové tekutině je glukóza buď spotřebována, nebo vyloučena. V závislosti na BG může být část IG přenesena zpět do krve s tím, že uvedené omezení přenosu platí i v opačném směru. Kromě toho se může část IG vyloučit jiným způsobem, např. lymfatickým systémem, eventuelně se objevit zpět v krvi. [4]

BG a IG se vzájemně ovlivňují a zároveň jsou obě ovlivňované množstvím hormonů, neurálních signálů a účinky substrátu. Modelování takto komplikovaného systému by znamenalo zanesení velkého množství parametrů, které by zvýšily náchylnost na přeučení. Pokud by k přeučení došlo, model by popisoval šum místo vztahů mezi jednotlivými částmi systému. Navíc v případě, že bychom měli příliš mnoho parametrů, nemuseli bychom poznat, že nemáme dostatek úrovní k zachycení dynamiky systému glukózy. [4]

Model byl inspirován *federativní kosimulací* (viz literatura [5]), kdy je celý systém rozdělen do menších částí a každá může být simulována vlastním simulátorem. Tyto simulátory spolu komunikují a ostatní simulátory se pro ně jeví jako *blackbox*. Pro zvýšení celkové přesnosti simulace může být simulátor ve federativní kosimulaci propojen s CGM zařízením. Model popisuje korelaci mezi IG a BG přes kapilární membránu. Krev a mok představují rozhraní, které spojuje tento model (simulátor) a další zařízení v systému. Ve výsledku se nemusí počítat s dalšími látkami jako např. inzulínem, který ovlivňuje spotřebu glukózy v buňkách. Namísto takového výpočtu biologický systém sám o sobě provádí potřebné kroky a aplikuje výsledky na IG a BG, které následně můžeme přečíst – díky tomu máme následky inzulínu přesně zpracované. Ve výsledku model nepotřebuje na vstupu data o dávkování inzulínu nebo příjmu cukru. Vyžaduje pouze souvislé měření IG a několik vzorků BG k odhadu parametrů modelu pro vyřešení předurčené soustavy rovnic. BG může být měřeno pomocí SMBG nebo získáváním vzorku krve z katetru. [4]

Model může běžet paralelně s přenosem glukózy skrz kapilární membránu. Předpovídá, že glukózový systém mění IG takovým způsobem, že IG měřené ve třech různých časech stačí k výpočtu BG v jednom z těchto časů. Na začátku se použije BG k určení parametrů modelu, tj. kvantifikaci účinků, které vyvíjejí vliv během transportu glukózy přes kapilární membránu. Následně se spojitě měří IG a zpětně vypočítá spojitá BG. Porovnáním naměřené a odhadnuté BG se následně zjistí chyba výpočtu. Naměřená BG

může být dále použita ke kalibraci CGMS nebo k novému nastavení parametru modelu. [4]

Následující rovnice (1) a (2) popisují model dynamiky glukózy. Parametr *p* je konstantou, která představuje přírůstek glukózy z krve skrz kapilární membránu. *cg* je konstanta, která popisuje vliv plochy a propustnosti membrány. *b(t)* - *i(t)* je rozdíl koncentrací přes membránu podle Flickova rozptylového zákona. *c* je úroveň glukózy, která představuje rozdíl mezi příjmem a vyloučením glukózy v podkoží. [4]

$$\varphi_{(t)} = \mathrm{t} + \Delta\mathrm{t} + \begin{cases} \mathrm{k} \times \frac{i_{(t)} - i_{(t-h)}}{\mathrm{h}} & \mathrm{h} \neq 0 \\ 0 & \mathrm{h} = 0 \end{cases} \qquad (1)$$

$$p \times b_{(t)} + cg \times b_{(t)} \times \left[b_{(t)} - i_{(t)}\right] + c = i_{(\varphi_{(t)})} \qquad (2)$$

$\varphi_{(t)}$ představuje posun změn IG v čase. Skládá se z pevné části Δt a proměnné části – parametry *h* a *k*. *t* je čas od nějakého bodu v historii, který je vyjádřen počtem dní od 1. ledna 1900 00:00UTC, desetinná část vyjadřuje čas v rámci jednoho dne. *h* je vyjadřován ve stejných jednotkách jako Δt a *t*. Vyjadřuje posun zpět od času *t*. Pro dlouhé intervaly *h*, parametr *k* vyjadřuje jak změna koncentračního gradientu ovlivňuje IG. Převádí změnu naměřené IG na proměnnou čas – vzdálenost, která je potřeba pro získání tří IG, které potom slouží k výpočtu BG. Pro v čase okrajové hodnoty IG rovnice může vrátit takový čas, pro který není naměřena hodnota IG. BG potom nemůže být vypočtena. Jelikož časový úsek (segment) je dlouhý několik dní, je toto omezení zanedbatelné. [4]

4 Distribuovaný výpočet

Distribuovaný výpočet je výpočet rozdělený na několik menších, na sobě nezávislých, částí výpočtu. Takový výpočet lze provést pouze u algoritmů, které můžeme paralelizovat, kdy vzájemně nezávislé části výpočtu běží souběžně. Důvodem rozdělení je obvykle zkrácení celkového času výpočtu.

4.1 Komunikace v distribuovaných výpočtech

Data pro distribuovaný výpočet jsou obvykle přenášená skrze počítačovou síť. V následujících podkapitolách jsou popsány některé z typů komunikací.

4.1.1 Source/Destination

Jedná se o přenos dat mezi dvěma počítači. Informace přenášené ve formě paketů mají jednoznačně určený zdroj (adresa odesílatele) a jedinečný cíl (adresa příjemce). V případě, že je potřeba poslat stejná data více příjemcům, musí se vytvořit pro každého z příjemců zvláštní zpráva, což může vést k nadměrnému zatížení sítě. Navíc vzniká odlišné zpoždění obdržení zprávy u jednotlivých příjemců a může tak znesnadnit synchronizaci mezi stanicemi. [6]

4.1.2 Master/Slave

Princip komunikace Master/Slave spočívá v tom, že vyslat zprávu svévolně může pouze Master. Slave může vyslat zprávu pouze tehdy, pokud je k tomu vyzván Masterem. Přímý přenos mezi jednotlivými zařízeními typu Slave není obvykle možný. Běžně se komunikace mezi zařízeními typu Slave zprostředkovává přes řídící zařízení typu Master. Některé sítě a protokoly umožňují existenci více zařízení typu Master na jedné síti.

4.1.3 Peer-to-Peer

Komunikace mezi zařízeními typu P2P[15] vychází z principu, že všechna zařízení v síti jsou rovnocenná. Takže vysílat mohou kdykoliv, kdy je potřeba. Na síti neexistuje žádné zařízení, které by si vyhrazovalo vyšší prioritu pro možnost komunikace. Aby se ale zabránilo chaosu na síti, jednotlivá zařízení si předávají tzv. *Token*, což je oprávnění

[15] Peer to Peer – přímá komunikace dvou klientů

k vysílání. Toto oprávnění se cyklicky předává mezi jednotlivé stanice[16] po určité době, aby nedocházelo k zahlcení sítě jedinou stanicí.

4.1.4 Producer/Consumer

Tento způsob komunikace se od ostatních liší v tom, že není vůbec podstatné, kdo zprávu odeslal, ale jakého typu zpráva je. Adresa odesílatele je nahrazena identifikátorem obsahu zprávy. V síti tuto zprávu získají všechny stanice a podle identifikátoru usoudí, zdali budou nějakým způsobem reagovat nebo ji ignorují. Obecně se tento způsob používá při distribuci nových údajů, kdy např. naměřená hodnota teploty zajímá pouze některé stanice v síti. Při odeslání zprávy nemá odesílatel zpravidla vůbec přehled o příjemcích v síti, zdali vůbec nějaký je. V síti jsou si všechna zařízení rovna a k vysílání zpráv dochází na základě některého z podnětu:

- obdržení žádosti o data,
- potřeba získat data (např. pro pokračování ve výpočtu),
- změna v datech (např. teploty),
- vypršení času pro periodické odeslání dat.

4.2 Load balancing

Load balancing (česky vyvažování zátěže) je obecná technika, která rozděluje zatížení mezi více entit (počítačů, síťových linek, procesorů, atd.), aby bylo dosaženo optimálního využití, propustnosti nebo času odezvy. Platí, že každá entita, mezi kterou se rozděluje zatížení, poskytuje identickou funkci. To znamená, že ať už požadavek zpracuje kdokoliv, výsledek musí být vždy stejný.

4.3 RPC mechanismus

Procedurální komunikace připomíná běžnou proceduru a jedna z jejích forem je proto standardně nazývána voláním vzdálených procedur (Remote Procedure Call, RPC mechanismus). Ve své klasické formě zahrnuje RPC mechanismus kromě vlastní výměny žádosti mezi klientem a serverem, provedení funkce serverem a vrácení odpovědi klientovi ještě mechanismus, který dává procedurální komunikaci formu běžného volání procedury. [7]

[16] tzv. token passing

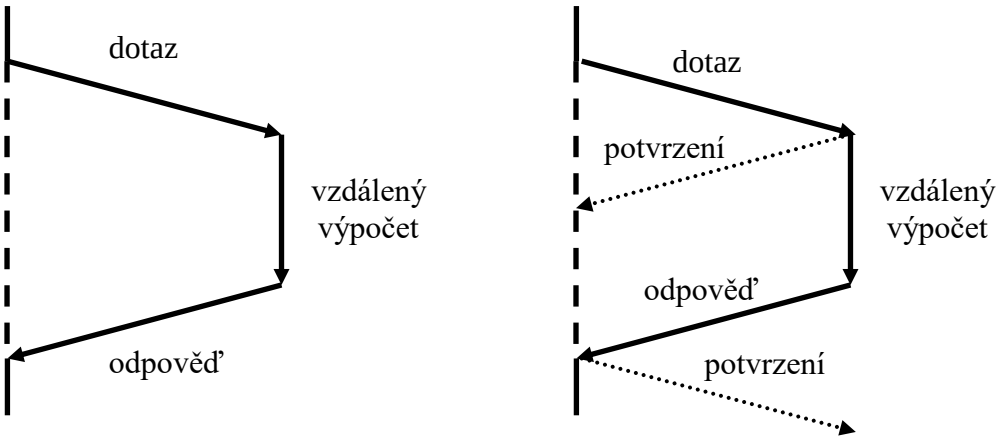
Mechanismus RPC je složen z: [7]

- části kódu na straně klienta, který transformuje volání RPC procedury na zprávu předávanou komunikačním systémem vzdálenému serveru a zpětně transformuje odpověď serveru,
- části kódu na straně serveru, který transformuje přijatou zprávu na volání RPC procedury a výsledek transformuje na odpověď odesílanou klientovi,
- vlastního kódu RPC procedury na serveru.

Mechanismus RPC je postaven nad komunikační systém, který vzájemně propojuje jednotlivé uzly a procesy distribuovaného systému. Na jeho vlastnostech závisí, jak dokonale podpoří procedurální komunikaci a nakolik ho budeme nuceni doplnit. Z našeho pohledu můžeme komunikační systémy rozdělit do tří skupin: [7]

- virtuální kanál mezi klientem a serverem,
- datagramová komunikace zachovávající pořadí (např. lokální síť),
- datagramová komunikace nezachovávající pořadí (např. internet).

Virtuální kanál, realizovaný například transportním komunikačním protokolem TCP, zabezpečí svými vnitřními mechanismy spolehlivé předání požadavku a odpovědi. Vlastní RPC mechanismus v takovém jednoduchém případě popisuje Obrázek 4 vlevo. Nemáme-li takový spolehlivý prostředek, nebo nám takový prostředek svou vysokou režií nevyhovuje, musíme doplnit méně spolehlivou datagramovou službu vlastním koncovým potvrzováním. Jednoduché potvrzování dovolí automaticky opakovat poškozený požadavek nebo odpověď a jeho funkci ilustruje Obrázek 4 vpravo. V uvedené formě ovšem znamená výměnu čtyř zpráv pro jedno RPC (proti dvěma u ideální komunikace). [7]



Obrázek 4: Výměny zpráv pro mechanismus RPC, převzato z [7]

Procedurální komunikace RPC se podobá běžnému volání procedur a mechanismus RPC rozdíly záměrně zakrývá. Mechanismus RPC a běžné volání procedury se však liší svou sémantikou. Rozdíl je způsoben tím, že každý z procesů, klient a server, může běžet na jiném počítači. Požadavek efektivity pak přirozeně omezuje typy parametrů, které RPC mechanismus používá. Nejsou povolené odkazy, seznamové struktury, procedury apod. [7]

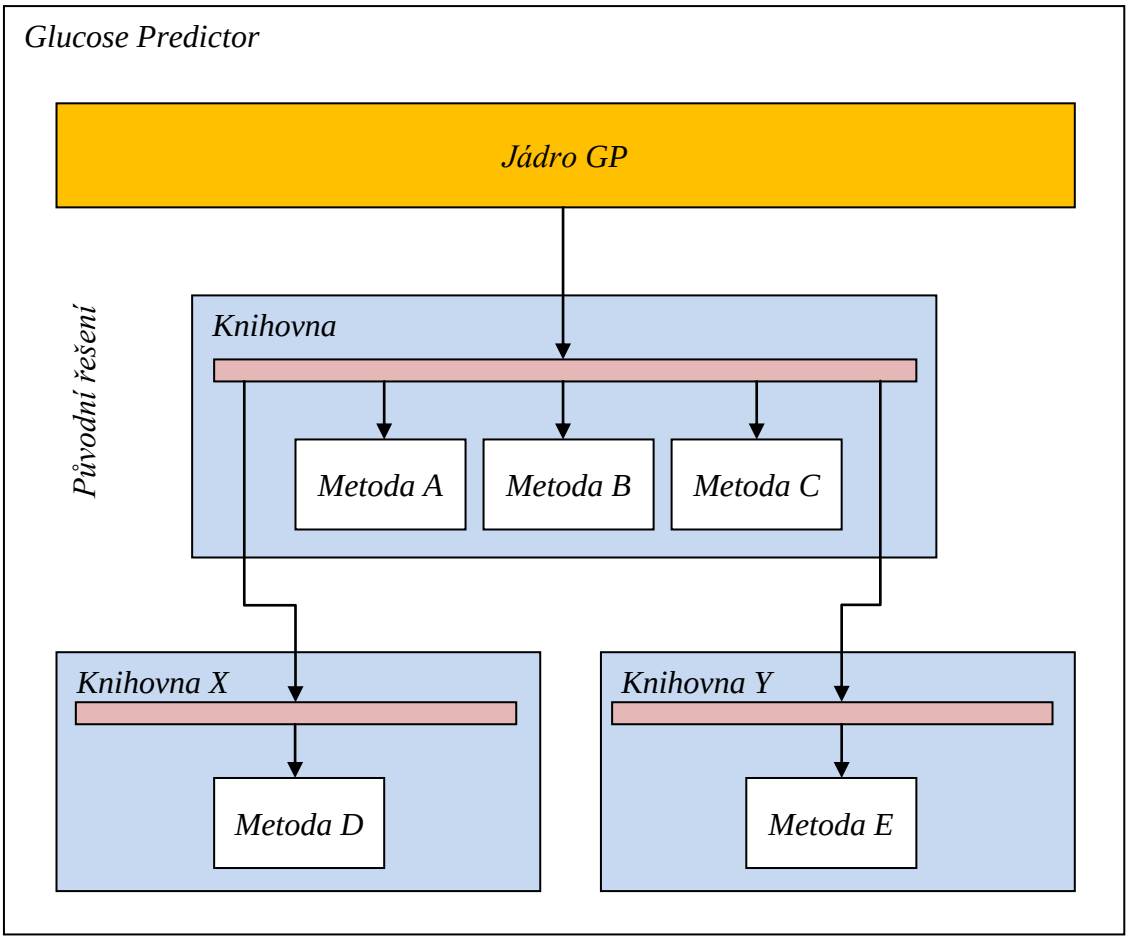
Problémem, se kterým se musí RPC mechanismus vypořádat, jsou důsledky ztrát požadavků a odpovědí a výpadky systémů, na nichž běží proces. Kromě ztráty požadavku a odpovědi a výpadku serveru působí potíže i výpadek klienta. Odpovědi na požadavky vyslané před výpadkem klienta je nutné rozlišit od odpovědí na požadavky vyslané po jeho restartu. [7]

Komunikace Remote Procedure Call (RPC) je podporovanou formou komunikace v řadě programovacích prostředků/jazyků. Předpokládá nesymetrické postavení aplikačních procesů a nevyžaduje explicitní navazování spojení. Model spolupráce aplikačních procesů, pro který se RPC používá, označujeme obvykle jako model Client–Server. Klient žádá konkrétní server o provedení určité akce zasláním požadavku. Server, který na takovou žádost od libovolného klienta čeká, požadovanou akci provede. Po jejím ukončení odešle server odpověď, na kterou klient mezitím čekal. [7]

5 Implementace řešení

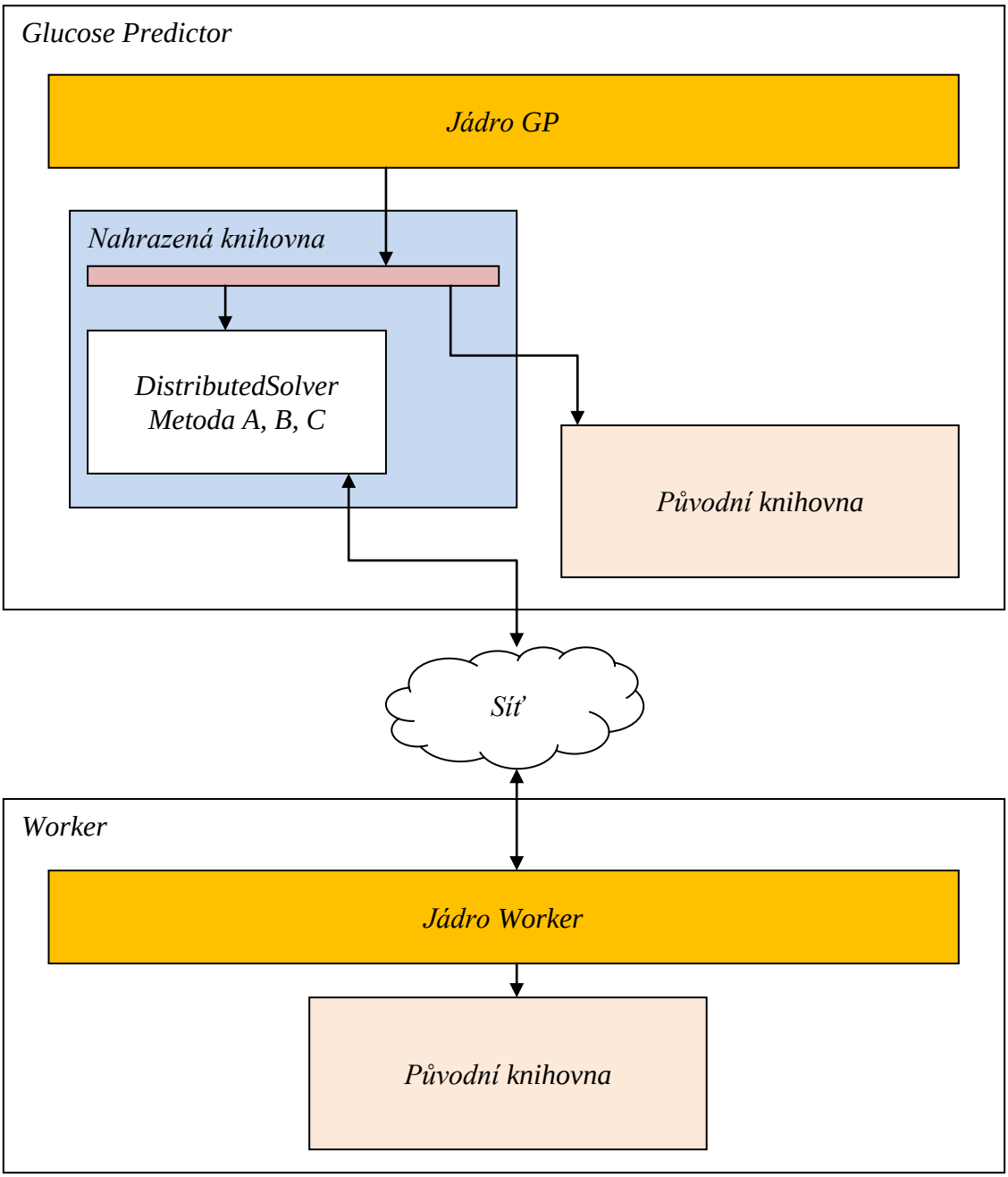
Program Glucose Predictor obsahuje rozhraní pro výpočet řešení modelu různými metodami. Podle typu metody se volí, která knihovna provede daný výpočet. Mým úkolem je vytvořit knihovnu, vlastní implementaci tohoto rozhraní, která bude výpočet zvolených metod počítat distribuovaně a nahradí tak původní nedistribuované řešení.

Původní způsob výpočtu (výběru knihovny a metody) je zjednodušeně nastíněn na obrázku *Obrázek* 5. Program Glucose Predictor (GP) zastřešuje a využívá jednotlivé knihovny podle metody, které poskytují.



Obrázek 5: Zjednodušené schéma programu Glucose Predictor - původní řešení

Na obrázku *Obrázek 6* je vidět jakým způsobem nyní program provádí výpočet distribuovaně. Původní knihovna zastřešující výpočty je nahrazena mojí knihovou (dále DS - *DistributedSolver*), ve které se provádí výpočet distribuovaně. Původní knihovna se nadále používá pro některé funkce nebo metody výpočtu, proto je vidět „průchod“ mojí implementací. V DS je obsažena celá funkcionalita, která zajišťuje distribuci částí výpočtu, oslovování řešitelských stanic apod.



Obrázek 6: Zjednodušené schéma programu Glucose Predictor - distribuované řešení

5.1 Distribuovaný výpočet

Distribuovaný výpočet jsem navrhnul na principu *Farmer-Workers*, kde se celý výpočet rozdělí na několik menších výpočtů, které jsou následně odesílány mezi jednotlivé řešitelské stanice – počítače s programem *Worker*.

5.1.1 Rozdělení výpočtu

U každé metody lze nadefinovat, jaký parametr se má pro výpočet rozdělovat. V první fázi se parametry výpočtu rozdělí do staticky definovaného počtu – ten je stejný jako maximální počet spojení s řešiteli. Následně se čeká na připojení řešitelů a po

zhodnocení se provede dodatečné rozšíření nebo naopak snížení počtu částí, které se budou distribuovaně počítat. Přidělení částí výpočtu jednotlivým řešitelům probíhá od začátku seznamu, naopak rozdělení a slučování od konce.

Např. je zbytečné, aby byl výpočet rozdělen na sto částí, pokud máme aktuálně k dispozici pouze jednoho řešitele. Algoritmus ale vždy zajišťuje, aby bylo minimálně třikrát více částí výpočtu, než je aktuální počet řešitelů. Limitní hranice pro toto dělení je maximální počet současně připojených řešitelů. Důvodem zvolení konstanty 3 pro minimální rozdělení je, že někteří řešitelé mohli být v době zahájení ještě nedostupní (prováděl se výpočet pro jinou řídicí stanici nebo řešitel nebyl zapnutý a připravený). Konstantu lze v konfiguraci změnit.

Pokud tedy máme ve výchozím stavu rozdělení na 30 částí a po vypršení prvotního čekání na připojení řešitelů se připojí pouze tři, tak se před zahájením tyto části spojí do 9 částí. Následně je každému přidělena jedna část a zahájí se distribuovaný výpočet. Takže tři části se počítají a dalších šest čeká. Pokud se nikdo dále nepřipojí, jednotliví řešitelé si postupně po vyřešení jedné části řeknou o další. V případě ale, že se připojí další řešitelé, provede se další analýza, zdali se mají zbývající části ještě dále rozdělit.

5.1.2 Selhání řešitele výpočtu

Během distribuovaného výpočtu může nastat situace, že některý z řešitelů z nějakého důvodu selže a odpojí se. Na to reaguje řídicí stanice tím, že přidělenou část výpočtu opět označí jako volnou pro výpočet. Vzhledem k tomu, že rozdělování práce se vždy provádí od konce a přidělování od začátku intervalu, může vzniknout při výpadku řešitele část, kterou nebude již možné v průběhu rozdělovat nebo slučovat. Pouze se přidělí dalšímu volnému řešiteli, který si bude říkat o další data.

Další problémy, které mohou během výpočtu nastat, jsou takové, že by se řešitel teoreticky mohl dostat do stavu deadlock nebo livelock. Přemýšlel jsem, zdali je možné nějakým způsobem tyto stavy detekovat a následně provést řízený restart. U deadlocku by se nabízelo řešení detekce tím, že aktuální výpočet neprodukuje dlouhou dobu žádné dotazy na řídicí jednotku a současně využití procesoru řešitelem je pod nějakou minimální vhodně zvolenou hodnotu. U livelocku je ale toto řešení nedostačující, protože procesor je i nadále využíván. Proto jsem od detekce zmíněných stavů upustil a pouze implementoval zrušení celého distribuovaného výpočtu, když je zrušen uživatelem v programu Glucose Predictor. Dalším důvodem proč neřešit tyto stavy je,

že obecně se může jednat o dlouhé a náročné výpočty, které by měly být dostatečně odladěné před nasazením do distribuovaného prostředí.

5.1.3 Load balancing

Abych efektivněji využíval jednotlivé stanice pro výpočet, vytvořil jsem hodnocení řešitelských stanic na základě několika parametrů. Mezi tyto parametry patří využití procesoru, teplota procesoru a velikost volné RAM. Každý z těchto parametrů je periodicky (každé 2 sekundy) odesílán do řídicí stanice (pokud je řešitelská stanice připojená), která provádí normalizaci do jednotného čísla, které určuje hodnocení řešitelské stanice. Čím je číslo vyšší, tím je pro výpočet dané stanice lepší.

$$m_c = \frac{m_{all} - m_{use} - m_{need}}{m_{all}} \cdot r_m \qquad (3)$$

$$cpu_{cl} = (1 - cpu_l) \cdot r_l \qquad (4)$$

$$cpu_{ct} = \frac{cpu_{max} - cpu_t}{cpu_t} \cdot r_t \qquad (5)$$

$$s = cpu_{ct} + cpu_{cl} + m_c \qquad (6)$$

Rovnice (6) vypočítává celkové skóre řešitelské stanice. cpu_{ct}, cpu_{cl} a m_c jsou mezivýsledky rovnic (3-5). Parametry r_m, r_l, r_t, cpu_{max} a m_{need} jsou nastavitelné konstanty v konfiguračním souboru. První tři vyjadřují poměr z celkového skóre a měly by tedy být volené od 0 do 1 s celkovým součtem 1. Parametr m_{need} je potřebná paměť v MB, se kterou řešitel počítá, aby se nemusel použít swap, cpu_{max} udává referenční teplotu, kterou nechceme překračovat na řešitelských stanicích. m_{all}, m_{use} cpu_l a cpu_t jsou hodnoty, které jsou získané z jednotlivých stanic.

Podle ohodnocení se seřadí sestupně všechny aktuálně připojené řešitelské stanice. A ty, které budou na konci seznamu a zároveň na nich neprobíhá výpočet, budou odpojeny. Tím se naskytne možnost pro další stanice v síti se připojit. Stručně řečeno, algoritmus se snaží využívat vždy ty nejlépe hodnocené řešitelské stanice.

S tímto algoritmem dále nastává příhodná situace, která nám zajistí, že v případě použití více souběžných programů Glucose Predictor nebudou dlouho čekat alespoň na jednoho volného řešitele.

Zvolené parametry pro vypočítání skóre stanice jsem zvolil na základě domluvy s vedoucím DP. Parametry se dají jednoduše rozšířit o další, které by mohly přinést určitým způsobem přesnější systém hodnocení.

5.1.4 Vyhodnocení výsledků

Po vypočtení všech částí distribuovaného výpočtu se provede vyhodnocení a vybrání nejlepšího výsledku, který se zvolí jako výsledný, a ten je následně předán programu Glucose Predictor. Výběr nejlepšího řešení je proveden porovnáním vypočítaných hodnot z metriky (tj. ohodnocení řešení), kterou zvolil uživatel pro výpočet v GP.

5.2 Komunikace

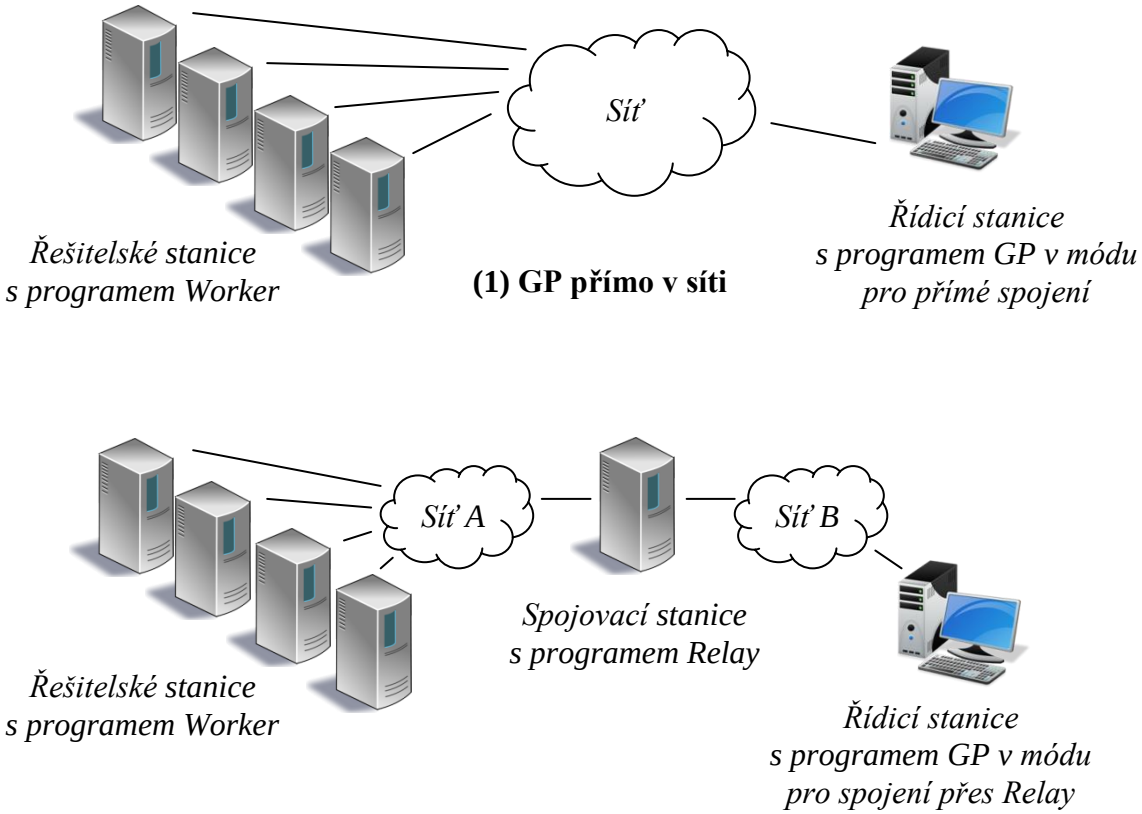
Komunikace mezi programem Glucose Predictor a jednotlivými řešiteli využívá několik technik. Pro vyhledání řešitelů v síti je použita multicastová komunikace. Pro přímou komunikaci potom TCP spojení, ve kterém je nejčastěji využito vzdálené volání metod pro získání dat pro distribuovaný výpočet. V případě použití *Relay* serveru se využívá k jeho oslovení UDP spojení.

5.2.1 Vyhledání řešitelů

Obecně nevíme, kolik je v síti k dispozici aktivních řešitelů. Proto jsem zvolil princip oslovování řešitelů přes multicastovou zprávu, ve které je obsažena informace s portem a adresou odesílatele. Oproti broadcastové zprávě se zbytečně nezahlcují stanice v síti, které nemají o tento typ zprávy zájem.

Každý z řešitelů se po spuštění připojí do skupiny, ve které čeká na oslovení. V případě oslovení se připojí k řídicí stanici, která mu následně zadává úkoly pro potřebný výpočet. Během trvajícího připojení již dále nereaguje na mutlicastové zprávy. Až teprve po ukončení spojení se „přepne“ do vyčkávacího režimu.

Řídicí stanice, na které běží program Glucose Predictor, může být přímo ve stejné síti s řešiteli a vysílá multicastové zprávy přímo do sítě ve stejné skupině. Může ale být i mimo síť, kam by se multicastové zprávy nemusely vypropagovat. Proto jsem vytvořil tzv. *Relay* server, který funguje jako vstupní brána do sítě řešitelů. V nastavení Glucose Predictor pak pouze stačí nastavit adresu *Relay* serveru, který normální zprávu převede na multicastovou a zajistí oslovení řešitelů.



Obrázek 7: Možné způsoby komunikace programu Glucose Predictor s řešiteli (zdroj obrázků: openclipart.org)

5.2.2 Vzdálené volání metod

V každé řešitelské stanici je řešena část distribuovaného výpočtu. Každá z metod pro svůj výpočet potřebuje různá data, která jsou poskytována programem GP. Jelikož ale tato data nejsou k dispozici v adresním prostoru řešitelských stanic, implementoval jsem vlastní řešení vzdáleného volání metod. Detailnější princip je popsán v programátorské dokumentaci v kapitole 6.4.2 - Vzdálené volání metod.

Hlavním důvodem, proč jsem se rozhodl pro vlastní řešení, bylo, že mám úplnou kontrolu nad přenášenými daty a nejsem limitován platformou OS.

6 Programátorská dokumentace

6.1 Struktura projektu a jeho zprovoznění

Diplomová práce je napsaná pro vývojové prostředí Microsoft Visual Studio 2015 s rozšířením QT5Package. Celé řešení je pod jedním tzv. *Solution*, ve kterém jsou jednotlivé projekty.

6.2 Instalace a přeložení

Návod jsem popsal pro nově nainstalovaný Windows 7 x64 Professional. Nejprve je potřeba nainstalovat vývojové prostředí **Microsoft Visual Studio 2015** (instalační soubor dostupný na přiloženém DVD **extra\vs_community_ENU.exe**). Při instalaci je potřeba zaškrtnout i modul **Visual C++**, aby se nainstalovalo studio spolu s překladačem.

Dále je potřeba nainstalovat program **Visual Leak Detector** (instalační soubor dostupný na přiloženém DVD **extra\vld-2.5-setup.exe**). Neměnit cestu, protože projekty mají absolutní cestu nastavenou na výchozí instalaci programu. Pro překlad je potřeba i nainstalovat **SDK Windows 8.1** (instalační soubor dostupný na přiloženém DVD **extra\sdksetup.exe**).

Nyní doporučuji překopírovat z DVD následující adresáře:

- **bin** – binární soubory aplikací
- **lib** – potřebné knihovny pro překlad projektů
- **src** – všechny zdrojové kódy

do adresáře **C:\projekt**. Cesta může být i jiná, jelikož návazné cesty jsou relativní.

Následně můžeme spustit vývojové studio. Před otevřením projektů je nutné ještě stáhnout a nastavit rozšíření **QT5Package.** To provedeme v menu **Tools/Excensions and Updates…**, vyhledáme a nainstalujeme. Po restartu v menu **QT5/Qt Options** přidáme verzi Qt knihovny. Do *Name* napíšeme třeba **x64** a do *Path* napíšeme **C:\projekt\lib\Qt\Qt5.4.1\5.4\msvc2013_64**.

Nyní je vše potřebné připraveno a je možné otevřít soubor **C:\projekt\src\dp\DP.sln**. Po otevření *Solution* je možné vybrat si profil překladu **Release** nebo **Debug.**

V *Properties* u *Solution* je vhodné nastavit, aby se při startu spouštělo více projektů zároveň (pro testování je dobré zvolit projekt DistributedSolver a Worker, případně i Relay). Upozorňuji, že ve výchozím stavu se Relay server nevyužívá. Pokud by to bylo potřeba, je nutné změnit nastavení GP, viz kapitola 6.6.3. Výsledné přeložené soubory se kopírují do příslušných adresářů v **C:\projekt\bin***.

6.3 Softwarové vybavení

V následujících podkapitolách jsou popsána softwarová vybavení, která jsem použil při vývoji.

6.3.1 Knihovna Qt

Knihovna Qt je již součástí programu Glucose Predictor ve verzi 5.4.1. Proto jsem pro další vývoj použil právě tuhle verzi. Hlavním využitím knihovny je podpora práce s objekty typu JSON. Formát JSON jsem použil pro jeho jednoduchost a hlavně protože je snadno čitelný (např. při zkoumání přenášených paketů) a byl mi doporučen vedoucím DP. Dále je např. oproti formátu XML úspornější v uchování informace. Využil jsem ho pro síťovou komunikaci mezi řídicí a řešitelskou stanicí. Primitivní datové typy jsou ukládány přímo, složitější datové struktury se konvertují do formátu base64, který je uložen pro přenos jako text.

6.3.2 Komponenta QtService

Komponenta QtService je rozšíření základní knihovny Qt. Umožňuje program jednoduše zaregistrovat do systému jako službu. Implementaci lze použít nejen pro systémy windows, ale i pro systémy založené na jádru Linux. V projektu je komponenta využita u programů *Worker* a *Relay*.

6.3.3 Program Core Temp

Jedná se o jednoduchý program, který dokáže získat ze systému aktuální hodnotu teploty procesoru a jeho aktuální zatížení. Naměřené údaje poskytuje ostatním aplikacím skrze dostupné SDK[17], včetně programu Worker. Tyto údaje se využívají pro celkové hodnocení daného počítače.

[17] Software Development Kit

Tento způsob získání dat jsem zvolil jako nouzové řešení, protože se mi nepodařilo zprovoznit rozhraní WMI (Windows Management Instrumentation), které mi vracelo nesmyslné nebo vůbec žádné údaje.

6.3.4 Knihovna Easylogging++

Knihovna Easylogging je skvělým řešením pro logování v programech, skládá se pouze z jednoho hlavičkového souboru a její použití a nastavení je velmi jednoduché. Hlavní důvod, proč jsem zvolil právě tuto knihovnu, byl, že již používaná knihovna Qt neobsahuje v dané verzi příliš použitelné rozhraní pro logování. Např. vůbec nemá zalogování typu INFO. To bylo přidáno až v pozdějších verzích, které zase ale nejsou kompatibilní s programem Glucose Predictor.

6.4 Shared

V tomto projektu je všechen sdílený kód, který se využívá v ostatních projektech. Obsahuje například třídy implementující jednotlivé RPC volání (vždy žádost a odpověď). Všechny implementace zpráv, které se využívají v mezimodulové komunikaci.

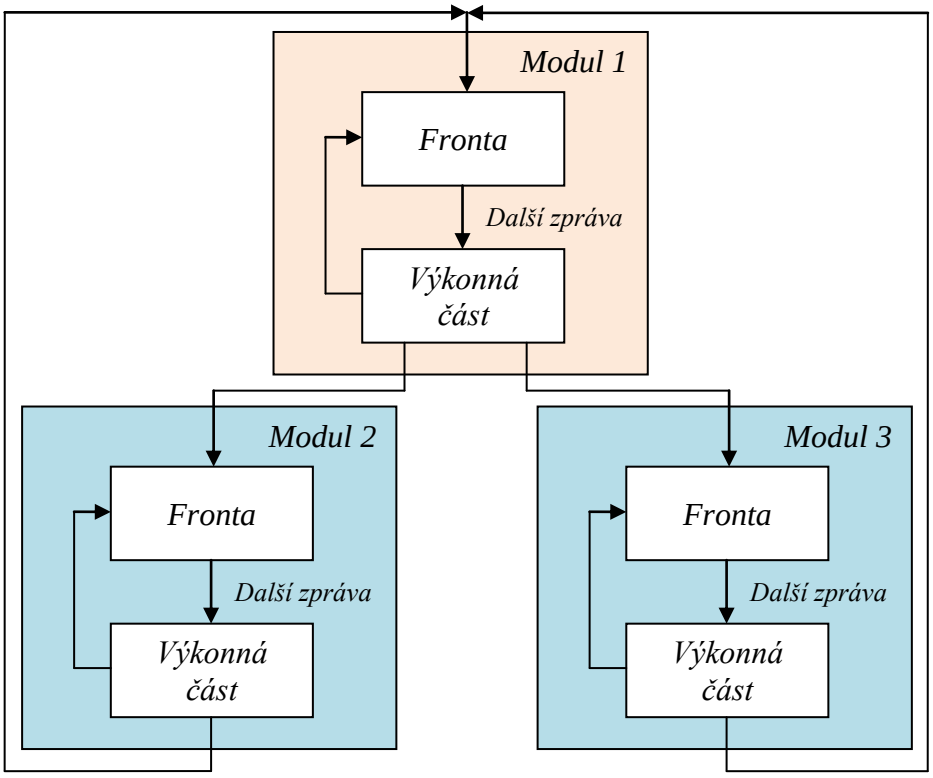
6.4.1 Mezimodulová komunikace

Každou dílčí aplikaci jsem rozdělil na moduly, které si mezi sebou vyměňují zprávy a jsou na sobě nezávislé. Modul běží ve vlastním vlákně a skrze frontu zpráv komunikují mezi sebou. Celý koncept komunikace modulů používám vždy ve stromové struktuře. Tzn., že vždy v aplikaci existuje hlavní modul, označen jako *Controller*, který řídí běh celé aplikace a předává získaná data mezi podřízenými moduly.

Každý takový modul dědí od tříd *MessageQueue* a *IRunnable*. *MessageQueue* je standardní fronta rozšířená o synchronizační prvky, které bylo nutné přidat kvůli přístupu k frontě z více vláken. Obecně se jedná o návrhový vzor producent/konzument, kde v roli producenta je buď nadřízený, nebo podřízený modul. Někdy i modul samotný. Konzumentem je vždy modul samotný, který reaguje na příchozí zprávy. Vkládané zprávy do fronty musí být vždy děděné od rozhraní *IMessage*, ve kterém je vždy nutné definovat typ zprávy z výčtu ze souboru *Type.h*. *IRunnable* je rozhraní, které zajišťuje modulu samotný běh ve vlákně, poskytuje podporu pro spuštění a ukončení běhu.

Hlavní výhodou tohoto návrhu je obecnost modulů a jejich logická nezávislost na sobě. Další přínosem je i znovu použitelnost jednotlivých modulů napříč více projekty.

Na následujícím obrázku *Obrázek 8* je blokově naznačen princip komunikace mezi moduly. Kde *Modul 1* je zde v roli kontroléru, který vše řídí a nemá žádný nadřízený modul. Předává zprávy do front modulů *Modul 2* a *Modul 3*. Dále každý z podřízených modulů mají možnost zapisovat do fronty nadřízeného modulu, aby mohly poskytovat žádosti/výsledky formou zprávy. Každý z modulů má výkonnou část v podobně vlákna, které cyklicky čte zprávy z fronty a následně reaguje vygenerováním nějaké zprávy, nebo vykonáním akce.



Obrázek 8: Mezimodulová komunikace – předávání zpráv

6.4.2 Vzdálené volání metod

Pro vzdálené volání metod jsem implementoval vlastní řešení. Třída *IMethod* je hlavní třídou, od které všechny třídy pro vzdálené volání dědí a poskytuje základní práci pro ukládání a čtení parametrů do JSON objektu. Uchovává si jednoznačný identifikátor o typu třídy, aby bylo možné jednoznačně identifikovat, co za třídu je uloženo v objektu. Všechny typy jsou uložené jako výčet v souboru *MethodType.h*.

Přiklad vytvořeného JSON objektu:

```
{
      "type": 17,             // typ IGlucoseLevels_GetLevels_RESPONSE
      "id": 1,                // jednoznačný identifikátor pro vazbu se
                              // vzdáleným rozhraním

      "data": {               // předávaná data

            "levels": "dG9obGUgbmlrZG … VidWRlIHDFmQ==", // text
                              // ve formátu base64, který reprezentuje
                              // binární data přenášených struktur
                              // typu TGlucoseLevel

            "count": 10,      // počet přenášených levelů
            "result": 0       // návratová hodnota volané metody (S_OK)

      }
}
```

Dále se třídy vzdáleného volání dělí na *dotaz* a *odpověď*, které jsou implementovány třídami *IMethodRequest* a *IMethodResponse*. Od těchto tříd dále vždy dědí jednotlivé metody, které je potřeba volat na vzdáleném počítači (v projektu *Farmer*).

Implementace jednotlivých volání metod jsem volil na základě potřeby v jednotlivých třídách, které implementují rozhraní solveru zadané vedoucím DP. Rozhraní, pro které bylo potřeba implementovat vzdálené volání metod, jsou:

- IApproximatedGlucoseLevels
- ICalculation
- IGlucoseLevels
- IMetricCalculator
- IMetricFactory
- ITimeSegment

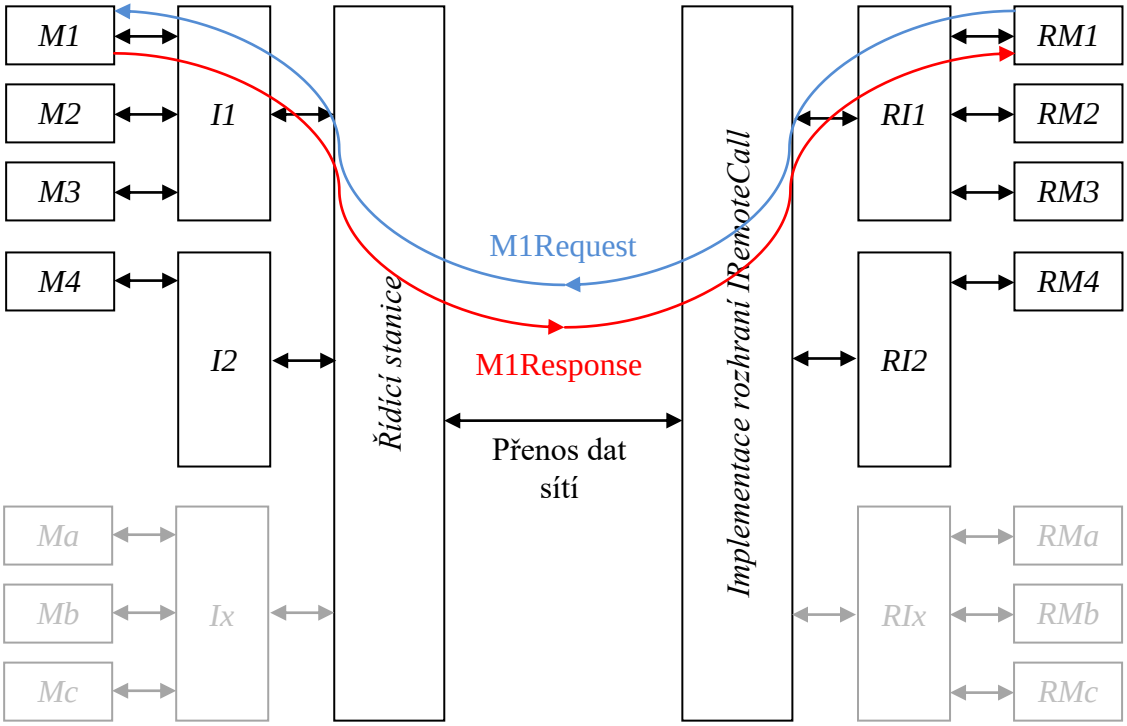
Pojmenování u jednotlivých implementací tříd jsem volil tak, že jsem přidal na začátek názvu každé třídy písmeno R, ve smyslu *Remote*.

Kvůli duplicitě názvů některých metod jsem z jednotlivých rozhraní musel použít *namespace* pro každé implementované rozhraní. Označení se opět odvíjí od názvu původního rozhraní přidáním prefixu NS.

Každá z implementací rozhraní má v konstruktoru jednoznačný identifikátor. Tento identifikátor slouží pro rozpoznání správné instance na straně, která vykonává vzdálenou metodu. Dalším parametrem v konstruktoru je rozhraní *IRemoteCall*, které slouží jako služba pro vzdálené volání metody. Jedinou implementací rozhraní

IRemoteCall je třída *WorkerSolve* z projektu *Worker*. Detailnější informace o implementaci jsou v kapitole 6.7.1.

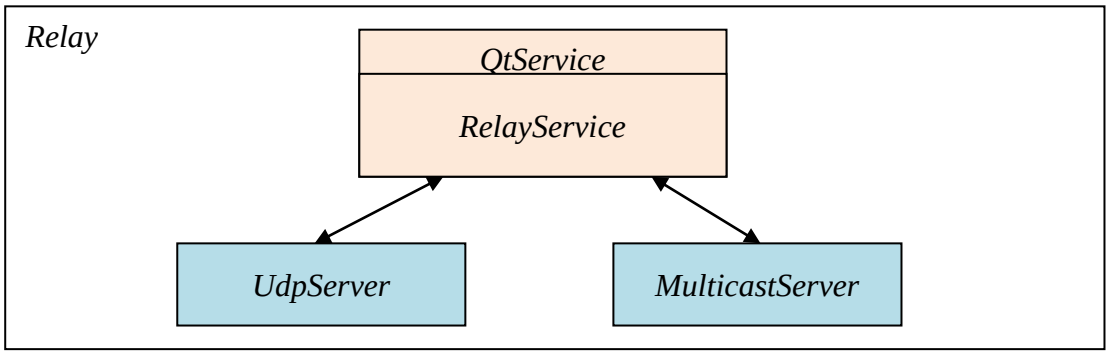
Na obrázku Obrázek *9* je vidět princip vzdáleného volání metod. Pokud je například zavolána metoda *RM1* z rozhraní *RI1*, vytvoří se instance třídy *M1Request*, které se předají vstupní parametry volané metody. Instanci je dále nastaven jednoznačný identifikátor rozhraní *RI1* a proběhne předání do implementace rozhraní *IRemoteCall*. V té se třída serializuje jako JSON text a zabalí do zprávy, která je odeslána po síti do cílové destinace (do řídicí stanice – projekt *Farmer*). Po přijetí zprávy řídicí stanicí se podle typu metody a identifikátoru rozhraní zjistí, že se jedná o metodu *RM1* z rozhraní *RI1*. Zavolá se tedy metoda *M1* z rozhraní *I1* s parametry předané *RM1*. Výsledek volání *M1* je předán vytvořené instanci třídy *M1Response*, která je následně navrácena po síti zpět, kde se výsledky předají původní volané metodě.



Obrázek 9: Blokové schéma principu volání vzdálených metod

6.5 Relay

Projekt *Relay* tvoří jednoduchý program, který má za úkol příchozí požadavek na oslovení volných pracovníků (počítače s programem *Worker*) přeposlat jako multicastovou zprávu do vnitřní sítě. Každý z pracovníků (pokud může) reaguje na přijatou zprávu přímým spojením k původnímu odesílateli.



Obrázek 10: Blokové schéma modulů v projektu Relay

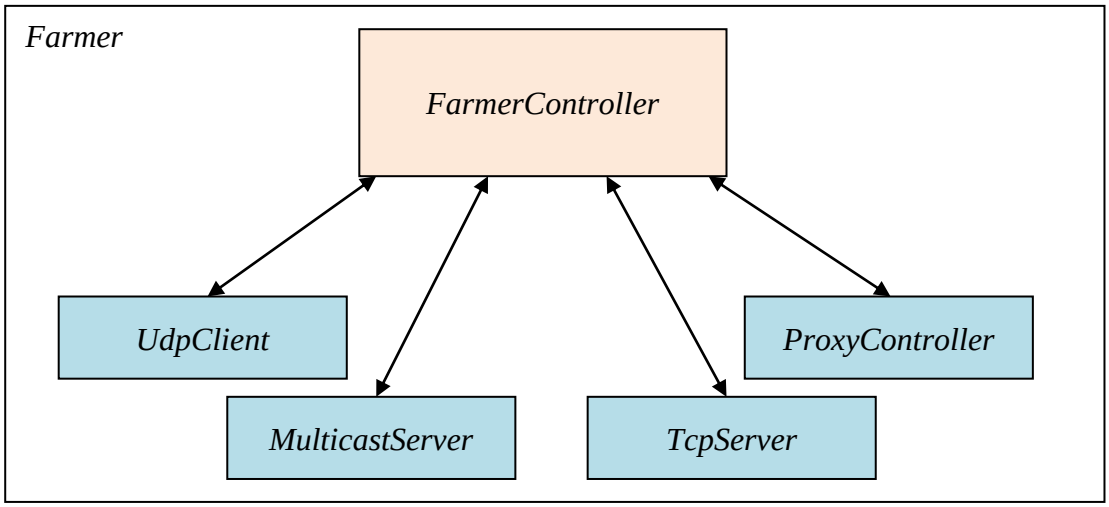
Tento projekt je navrhnut tak, aby mohl být spouštěn jako služba. S využitím komponenty *QtService* jsem vytvořil hlavní modul *RelayService*, který řídí podřízené moduly *UdpServer* a *MulticastServer*. *UdpServer* má za úkol číst přijaté zprávy, které následně předává *RelayService*, ten zprávu pouze předá podřízenému modulu *MulticastServer*, který zprávu pošle jako multicast paket do vnitřní sítě. Využití aplikace *Relay* je vidět na *Obrázek 7: Možné způsoby komunikace programu Glucose Predictor s řešiteli.*

Program lze konfigurovat souborem *relay.ini* (ten musí být na stejné adresářové úrovni), kde je možné nastavit následující parametry (v hranatých závorkách výchozí hodnota):

- **network/relay-port** [25001] – port pro zaslané zprávy GP
- **network/mc-address** [224.0.0.255] – adresa multicastu
- **network/mc-port** [25000] – port multicastu

6.6 Farmer

Projekt *Farmer* je jádrem celého řízení distribuovaného výpočtu. Má za úkol rozdělovat intervaly/parametry výpočtu na menší části, poskytuje potřebná data k výpočtu



Obrázek 11: Blokové schéma modulů v projektu Farmer

jednotlivým pracovníkům (programům *Worker*) a získané dílčí výsledky sloučit do jednoho celku, který je předán jako finální výsledek výpočtu.

FarmerController je tu opět ve funkci hlavního modulu, který řídí všechny podřízené. *UdpClient* a *MulticastServer* mají za úkol vyhledávat řešitele. Vždy je ale aktivní pouze jeden z nich. V případě, že je program GP mimo síť řešitelů, je využíván *Relay* server a zprávy o oslovení řešitelů se odesílají přes modul *UdpClient*. Naopak v případě, že je GP přímo ve stejné síti s řešiteli, využívá se pro oslovení řešitelů modul *MulticastServer*.

TcpServer slouží pro komunikaci s jednotlivými řešitelskými stanicemi. Maximální počet připojení lze nastavit v konfiguračním souboru, viz kapitola 6.6.3.

6.6.1 ProxyController

ProxyController je modul, ve kterém se provádí hlavní logika distribuovaného výpočtu. Vytváří instance třídy *ProxySession*, které se přiřazují 1:1 k připojeným řešitelům. Má za úkol dynamicky reagovat na změnu počtu řešitelů – přerozdělení nepřidělených částí pro distribuovaný výpočet. Je v něm implementovaná metoda pro výběr nejlepšího řešení, který je následně předán jako finální výsledek výpočtu. Uchovává výsledky z již vypočítaných částí výpočtů a provádí loadbalancing podle vypočítaného skóre z poskytnutých dat od řešitelských stanic.

Pro uchovávání informací části distribuovaného výpočtu slouží třída *ProxySolution*, ve které jsou uloženy parametry pro výpočet ve struktuře *TSolvingParams* a informace o stavu výpočtu. Třída pak má k dispozici metody pro zjištění, zdali je část již *vypočítaná*, *počítá se*, nebo *čeká na výpočet*. Ve stavu *počítá se* je uchováván i identifikátor spojení s pracovní stanicí, pro případ výpadku. Když by se řešitelská stanice z nějakého důvodu odpojila, je její přidělená část výpočtu opět přesunuta do stavu *čeká na výpočet*.

6.6.2 ProxySession

Každá instance *ProxySession* si uchovává stav svázaný s řešitelem. Pokud tedy řešitel volá některou vzdálenou metodu, je vykonávaná právě skrze tuto instanci. V třídě jsou implementovány všechny provázanosti mezi vzdálenými metodami na metody lokální. Generují se zde unikání identifikátory svázané s nově vytvořenými instancemi tříd rozhraní. Provázání je nutné kvůli správnému volání jednotlivých metod instancí. Při

volání vzdáleného volání je se zprávou poslán i jednoznačný identifikátor (viz 6.4.2), podle kterého se zjistí, která instanci daného rozhraní se má použít pro zavolání metody.

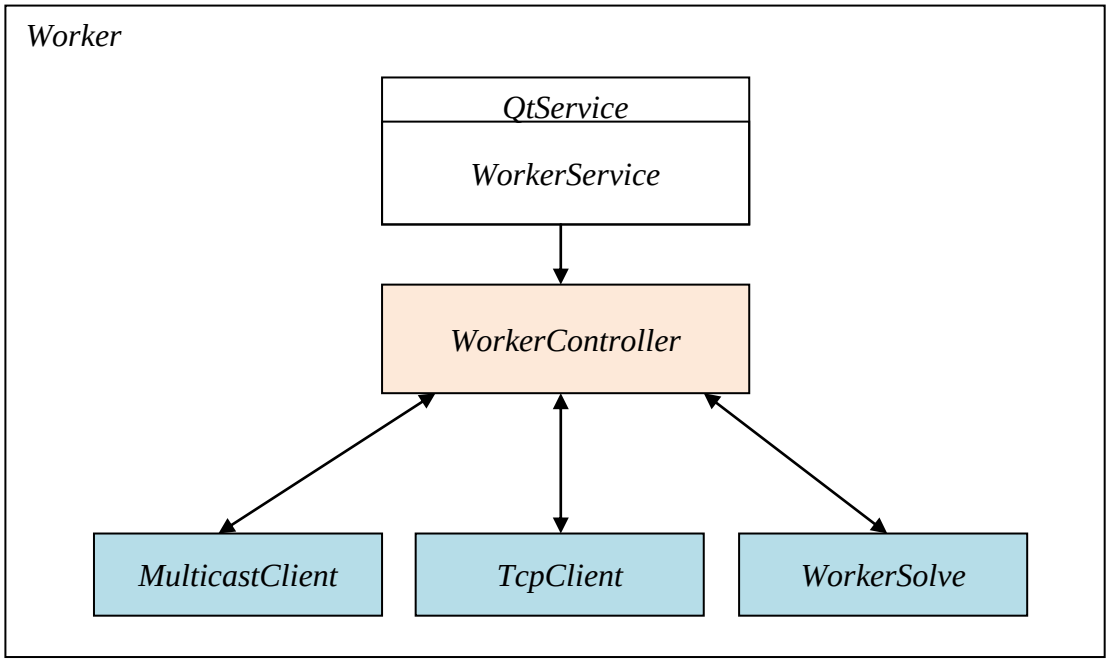
6.6.3 Konfigurační soubor

Konfigurační soubor *solver.ini* umožňuje změnit výchozí nastavení, které jsem zvolil při vývoji. S ohledem na formát souboru INI uvádím následující názvy parametrů vždy se skupinou, ve které se nachází a v hranatých závorkách výchozí hodnotu:

- **debug/console** [true] – zobrazení příkazového řádku pro výpis prováděných akcí
- **network/max-workers** [10] – maximální počet pracovních stanicí pro výpočet
- **network/port** [1975] – port pro spojení s řešiteli
- **network/mc-address** [224.0.0.255] – adresa multicastu
- **network/mc-port** [25000] – port multicastu
- **network/relay-use** [false] – zdali se má použít pro oslovení řešitelů relay server
- **network/relay-address** [localhost] – adresa relay serveru
- **network/my-address** [localhost] – adresa počítače s programem GP
- **network/relay-port** [25001] – port relay serveru
- **score/memory-need** [500] – předpokládaná velikost požadované paměti pro solver na řešitelské stanici v MB
- **score/cpu-max-temperature** [70] – maximální teplota procesoru pro výpočet skóre ve °C
- **score/ratio-memory** [0.4] – váhová konstanta pro paměť
- **score/ratio-cpu-load** [0.3] – váhová konstanta pro vytížení procesoru
- **score/ratio-cpu-temperature** [0.3] – váhová konstanta pro teplotu procesoru
- **solutions/k** [3] – konstanta pro rozdělování částí výpočtu, zmíněná v 5.1.1

6.7 Worker

Worker představuje program, který vykonává samotnou část distribuovaného výpočtu. Komunikuje s programem GP přes síť převážně způsobem vzdáleného volání metod pro získání dat potřebných pro výpočet.



Obrázek 12: Blokové schéma modulů v projektu Worker

Obrázek 12 zobrazuje blokové schéma modulů, ze kterých je tvořen projekt Worker. V projektu je, stejně jako v projektu *Relay,* použita komponenta *QtService,* díky které program může běžet jako služba. Implementace je v podobě třídy *WorkerService,* ve které je řešeno řízení modulu *WorkerController*. Ten je hlavním modulem, který řídí podřízené moduly *MulticastClient, TcpClient a WorkerSolve.*

MulticastClient je modul, ve kterém se čtou přijaté zprávy s žádostmi o výpočet. Tyto zprávy jsou zasílány do modulu *WorkerController*, který podle aktuální situace rozhoduje, zdali se má k žadateli (řídicí stanici) o výpočet připojit, nebo žádost ignorovat – to nastává obvykle v případě probíhajícího výpočtu pro jinou řídící stanici. *TcpClient* je modul, který je využíván pro komunikaci s řešitelskou stanicí během výpočtu.

6.7.1 WorkerSolve

Nejdůležitějším modulem v projektu je *WorkerSolve*, který poskytuje metodě výpočtu potřebná vzdálená data z řídicí stanice poskytnuté programem GP. Modul není vytvářen

hned při startu aplikace (jak se obecně u modulů děje), ale vždy až po připojení k řešitelské stanici, která dodá potřebné parametry pro vytvoření modulu.

Při vytváření instance se předávají parametry o typu metody, která se bude pro výpočet využívat, seznam unikátních identifikátorů pro vzdálené volání instancí třídy *ITimeSegment*, unikátní identifikátor pro vzdálené volání instance třídy *IMetricFactory* a parametry pro vytvoření lokální instance třídy *IMetricFactory* – nutnost využití lokální i vzdálené instance vysvětluji v následující kapitole 6.7.2.

Po startu vytvořeného modulu se vytvoří instance řešící metody (tzv. solver) zavoláním funkce *CreateSolver(…)*. Tato funkce je volána z původní knihovny (dynamická knihovna *solver.dll*). Původní data parametru (s názvem *segments*) se seznamem instancí implementace třídy *ITimeSegment* jsou nahrazeny nově vytvořenými instancemi třídy *RITimeSegment*, ve kterých jsou již zmiňované unikátní identifikátory pro zjištění instance při vzdáleném volání metod.

Po vytvoření solveru je zaslána zpráva řídicí stanici o připravenosti řešitelské stanice pro výpočet. Ta vzápětí odpovídá zprávou, ve které jsou již konkrétní parametry pro výpočet. Po obdržení zprávy se zavolá nad instancí solveru metoda *Solve(…)*, která spustí výpočet. Při výpočtu se postupně dále volají metody ze třídy *RITimeSegment*, které, mimo jiné, vytvářejí další návazné instance lokálních tříd *RIGlucoseLevels*, *RICalculation, a RIApproximatedGlucoseLevels* provázané přes unikátní identifikátory na vzdálené instance *IGlucoseLevels*, *ICalculation* a *IApproximatedGlucoseLevels*. O provázanosti a jednotném přidělování unikátních identifikátorů se stará konkrétní instance třídy *ProxySession*, kterou jsem popsal v kapitole 6.6.2.

6.7.2 Lokální a vzdálená instance třídy IMetricFactory

V původním řešení jsem měl pouze lokální instanci třídy *RIMetricFactory*, která vždy vytvářela přes vzdálené volání instance třídy *RIMetricCalculator*, ve které byly metody rovněž volané vzdáleně. Toto řešení se ale vůbec nevyplácelo, protože výpočet se vysoce zpomalil kvůli síťové režii a odezvě. Navíc se kalkulace prováděly na straně řídicí stanice, což také nedávalo smysl.

Z tohoto důvodu jsem si přidal vstupní parametry pro vytvoření vlastní instance *IMetricFactory*, které posílám společně s parametry na vytvoření instance modulu *WorkerSolve*. Instanci vytvářím funkcí *CreateMetricFactory(…)*, která je vykonávaná z původní knihovny (dynamická knihovna *solver.dll*). Všechny instance třídy

IMetricCalculator jsou nyní vytvářeny lokálně na řešitelské stanici, takže i potřebné kalkulace se provádí lokálně, bez zbytečného zbrzdění.

Původní řešení jsem ale nemohl zrušit, protože v třídě *ITimeSegment* existuje metoda *SuggestParams(...)*, ve které je jedním z parametrů ukazatel právě na původní instanci třídy *RIMetricFactory*. A vzhledem k tomu, že původní rozhraní nemůžu nijak změnit, při volání této vzdálené metody se využívá instance třídy *IMetricFactory* přímo na řídicí stanici.

6.7.3 Program Core Temp a informace o řešitelské stanici

Každý program *Worker* si udržuje po spuštění aktuální informace o využití a teplotě procesoru, dále údaje o velikost a využití paměti. Nasbíraná data se po připojení k řídicí stanici periodicky odesílají, aby měla řídicí stanice aktuální informace o využití jednotlivých řešitelských stanicích. Tyto informace využívá pro load balancing, popsaný v kapitole 4.2.

Důvody, proč jsem použil program Core Temp pro získání zatížení a teploty procesoru, jsem uvedl v kapitole 6.3.3.

6.7.4 Konfigurační soubor

Program lze konfigurovat souborem *worker.ini* (ten musí být na stejné adresářové úrovni), kde je možné nastavit následující parametry:

- **network/mc-address** [224.0.0.255] – adresa multicastu
- **network/mc-port** [25000] – port multicastu

6.7.5 Lokální metrika

Během vývoje jsem narazil na výkonnostní problém, který způsobovala vysoká režie na síti. Tento problém jsem vyřešil tím, že parametry pro vytvoření původní továrny (*IMetricFactory*) posílám společně s parametry při inicializaci části výpočtu. Takže instance továrny je vytvořena podle původních parametrů z programu GP (identifikátor továrny a hodnot z *TMetricParameters*) na každé lokální řešitelské stanici. Takže následné vytváření instancí *IMetricCalculator* nyní provádí každý *Worker* lokálně a nemusí být tedy výpočtem zatěžována řešitelská stanice.

6.8 DistributedSolver

Tento projekt je pouze pro zastřešení projektu *Farmer*. Výsledkem sestavení projektu je soubor *solver.dll*, který nahrazuje – překrývá původní nedistribuované řešení v programu Glucose Predictor. Původní soubor *solver.dll* jsem přejmenoval na *solver.orig.dll* (pouze u programu GP, u programu *Worker* je původní), jelikož je i nadále využíván pro některé metody, které nejsou počítané distribuovaně.

Rozcestník pro metody, které se mají počítat distribuovaně a které lokálně, je implementován v souboru *factory.cpp*, ve funkci *CreateSolver(...)*. Po domluvě s vedoucím DP se metody *Steil-Rebrin* a *AR Kalman* nepočítají distribuovaně. Toto lze kdykoliv změnit podle potřeby.

Ve třídě *DistributedSolver* je možné si u každého typu metody nadefinovat, podle kterého parametru se má provádět rozdělování částí výpočtů. Definování je formou labda výrazů. Lze zde naprogramovat i složitější algoritmus, který se bude rozhodovat nejen podle typu metody, ale i podle konkrétních vstupních parametrů pro výpočet.

7 Testování implementace

Data k testování mi byla poskytnuta vedoucím DP pouze pro testovací účely. V datech jsou obsaženy naměřené hodnoty celkem od 9 pacientů s diabetem typu 1.

7.1 Referenční stroje

Při měření v laboratoři jsem měl k dispozici celkem 8 referenčních počítačů. Jeden jsem použil jako řídicí stanici a sedm jako řešitelské stanice. Všechny stroje mají totožnou konfiguraci, viz Tabulka 1.

Tabulka 1: Důležité parametry referenčního stroje

CPU	Xeon E3-1246 v3 3.5GHz
RAM	16 GB DDR3
OS	Windows 7 x64 Professional

7.2 Funkčnost a rychlost

Pro prvotní ověření funkčnosti a rychlosti implementovaného řešení jsem provedl měření s vybranými metodami. Pro každou z vybraných jsem měřil dobu výpočtu, nejprve mnou implementované distribuované řešení, následně původní lokální řešení. V rámci metod jsem také měnil krok – což je parametr v programu Glucose Predictor. Dále jsem měnil i počet segmentů (jeden segment představuje naměřená data subjektu).

Každou kombinaci metody a kroku jsem vždy měřil třikrát (s výjimkou metod EPS Differential evolution a Artificial bee, které byly velmi časově náročné). V tabulkách 2 až 15 uvádím průměrný čas. Při měření původního řešení bylo těžké změřit dobu běhu, protože program neposkytuje žádné logování. Časy jsou tedy spíše orientační – ve většině případů menší než jedna sekunda. U distribuované verze jsem využíval čas z logování.

Tabulka 2: Difuse, segmentů 1, řešitelů 7

Krok [s]	10	1	0,1	0,01
Doba výpočtu - distribuovaně [s]	2,56	3,56	9,99	25,48
Doba výpočtu - lokálně [s]	<1	<1	<1	<1

Tabulka 3: Difuse, segmentů 5, řešitelů 7

Krok [s]	10	1	0,5
Doba výpočtu - distribuovaně [s]	15,21	24,14	30,28
Doba výpočtu - lokálně [s]	<1	<1	<1

Tabulka 4: Difuse v2 - Analytic, segmentů 1, řešitelů 7

Krok [s]	10	1	0,3	0,1
Doba výpočtu - distribuovaně [s]	2,47	**5,4**	**24,9**	**176**
Doba výpočtu - lokálně [s]	<1	11,8	126	1140

Tabulka 5: Difuse v2 - Analytic, segmentů 5, řešitelů 7

Krok [s]	10	1	0,3
Doba výpočtu - distribuovaně [s]	14,66	**39,8**	**262**
Doba výpočtu - lokálně [s]	3	135	1522

Tabulka 6: Difuse v2 - Diffusion Amoeba, segmentů 1, řešitelů 7

Krok [s]	10	1	0,1	0,01
Doba výpočtu - distribuovaně [s]	3,06	3,59	10,34	26,31
Doba výpočtu - lokálně [s]	<1	<1	<1	<1

Tabulka 7: Difuse v2 - Diffusion Amoeba, segmentů 5 řešitelů 7

Krok [s]	10	1	0,3
Doba výpočtu - distribuovaně [s]	18,98	22,28	33,29
Doba výpočtu - lokálně [s]	<1	<1	<1

Tabulka 8: Difuse v2 - EPS Differential evolution, segmentů 1, řešitelů 7

Krok [s]	10
Doba výpočtu - distribuovaně [s]	584
Doba výpočtu - lokálně [s]	192

Tabulka 9: Difuse v2 - EPS Differential evolution, segmentů 5, řešitelů 7

Krok [s]	10
Doba výpočtu - distribuovaně [s]	5490
Doba výpočtu - lokálně [s]	1857

Tabulka 10: Metoda Difuse v2 - Artificial bee, počet segmentů 1, počet řešitelů 7

Krok [s]	10
Doba výpočtu - distribuovaně [s]	873
Doba výpočtu - lokálně [s]	267

Tabulka 11: Metoda Difuse v2 - Artificial bee, počet segmentů 5, počet řešitelů 7

Krok [s]	10
Doba výpočtu - distribuovaně [s]	5490
Doba výpočtu - lokálně [s]	1857

Tabulka 12: Difuse v3 - Analytic, segmentů 1, řešitelů 7

Krok [s]	10	1	0,1	0,01
Doba výpočtu - distribuovaně [s]	2,24	2,94	10,1	26,03
Doba výpočtu - lokálně [s]	<1	<1	<1	<1

Tabulka 13: Difuse v3 - Analytic, segmentů 5, řešitelů 7

Krok [s]	10	1	0,3
Doba výpočtu - distribuovaně [s]	11,09	14,83	29,24
Doba výpočtu - lokálně [s]	<1	<1	<1

Tabulka 14: Difuse v3 - Diffusion v1 - SA v1, segmentů 1, řešitelů 7

Krok [s]	10	1	0,1
Doba výpočtu - distribuovaně [s]	1,9	3,76	15,74
Doba výpočtu - lokálně [s]	<1	<1	<1

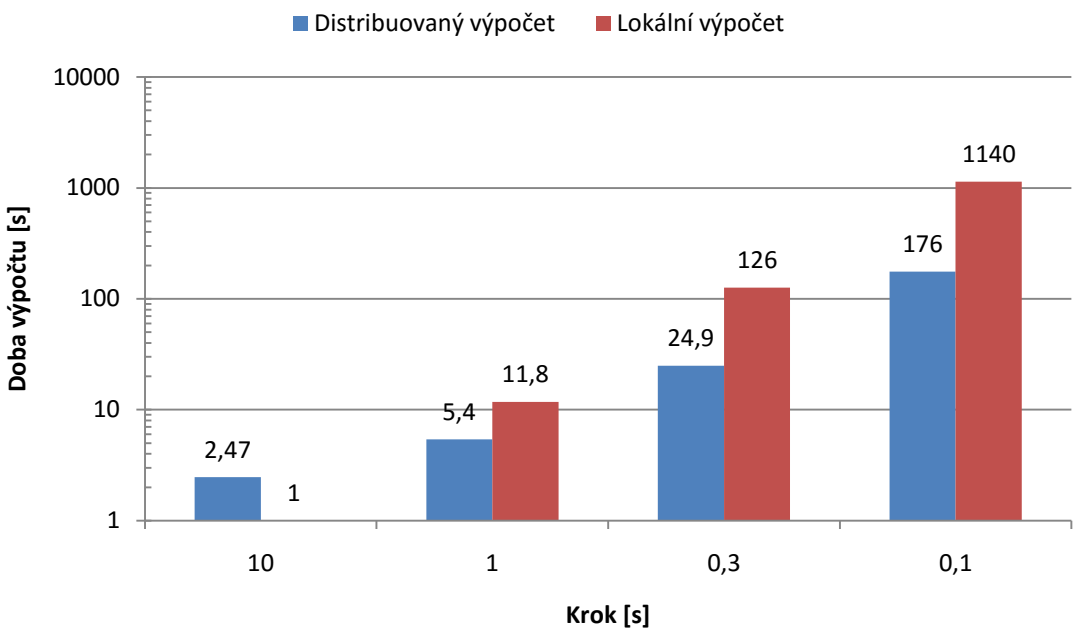
Tabulka 15: Difuse v3 - Diffusion v1 - SA v1, segmentů 5, řešitelů 7

Krok [s]	10	1	0,3
Doba výpočtu - distribuovaně [s]	8,74	24,36	55,01
Doba výpočtu - lokálně [s]	<1	3,35	7,33

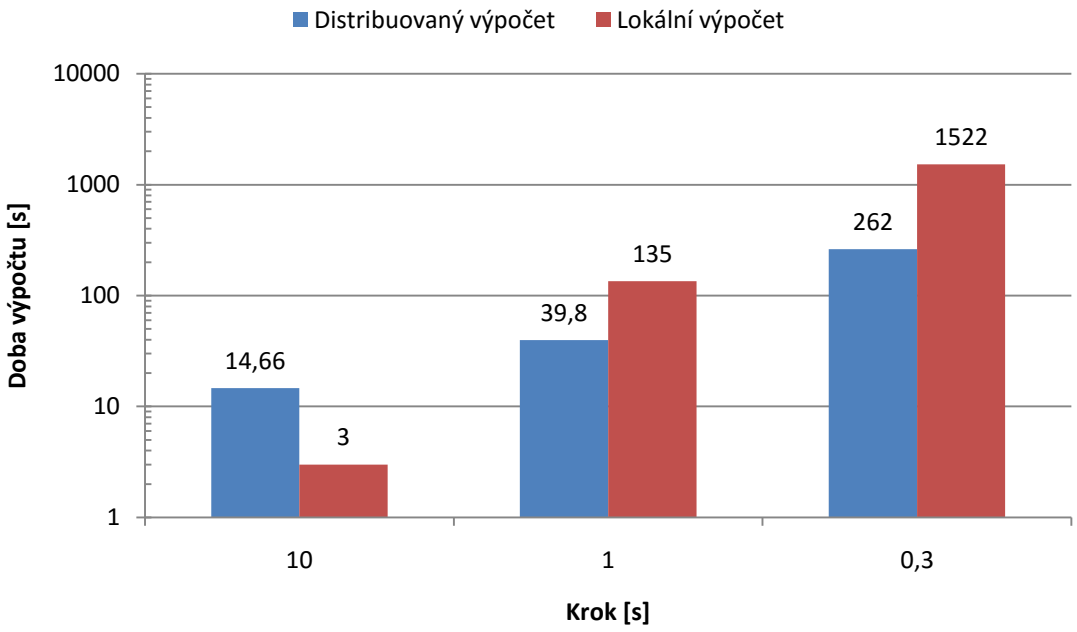
Z naměřených hodnot (Tabulka 4 a Tabulka 5) je vidět, že distribuovaný výpočet byl významně rychlejší pouze u metody *Difuse v2 – Analytic*. Z toho důvodu se budu v následujících měření soustředit pouze na tuto metodu.

Porovnání dob výpočtů mezi distribuovaným a lokálním výpočtem je zobrazen na Obrázek 13 a Obrázek 14.

Při měření některých metod jsem narazil na problém limitace knihovny Qt. Ten mi během vývoje nikdy nenastal. Na řídicí stanici se při výpočtu objevil text „*qjson document too large to store in data structure*“. Kvůli snižování velikost kroku narůstá velikost přenášených dat mezi řídicí a řešitelskou stanicí, to zapříčiní neúplnost přenesených dat a zkreslení výpočtů. Vyřešení tohoto problému by obnášelo detailnější analýzu a pravděpodobně povýšení verze knihovny Qt. Dalším řešením by bylo použití jiné knihovny pro práci s JSON, případně implementace úplně jiného způsobu přenosu dat.



Obrázek 13: Porovnání času výpočtů metodou Difuse v2 - Analytic, 1 segment, 7 řešitelů



Obrázek 14: Porovnání času výpočtů metodou Difuse v2 - Analytic, 5 segmentů, 7 řešitelů

7.3 Ověření vypočítaných parametrů

Při prvotním měření jsem rovněž porovnával vypočítané parametry. Všiml jsem si drobných odchylek – v řádu 10^{-4} u některých porovnávaných parametrů. Proto jsem se dále zaměřil na tuto nesrovnalost specificky zaměřeným měřením.

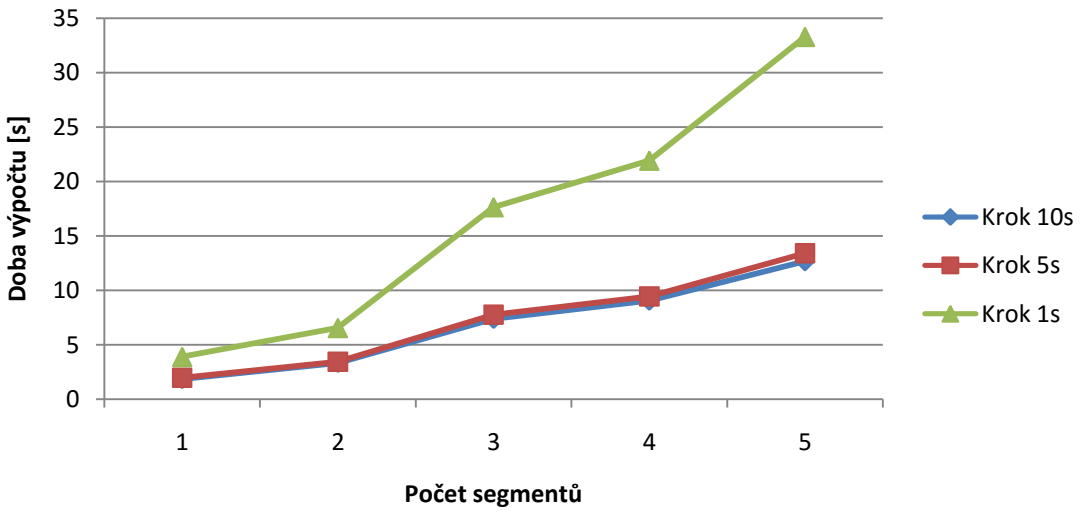
Abych vyloučil implementační chybu, provedl jsem test s jedinou řešitelskou stanicí a nastavil parametr **solutions/k** (konstanta pro rozdělování částí výpočtu) na hodnotu 1. Tím jsem docílil toho, že se nebude provádět žádné rozdělování práce a výpočet se bude provádět pouze vzdáleně.

Referenční hodnota metriky při lokálním výpočtu metodou *Difuse v2 – Analytic* je **0,364521**. Ta je shodná i pro vzdálený výpočet.

Se zvyšujícím počtem částí distribuovaného výpočtu se výsledná hodnota metriky mění v porovnání s referenční hodnotou. Při měření se zvyšujícím počtem částí (viz Tabulka 16) byla maximální absolutní chyba **0,00018**. Směrodatná odchylka naměřených hodnot je **0,0001163**. Odchylka hodnoty z metriky je pravděpodobně způsobená rozdělením výpočtu, přesný důvod jsem nebyl schopen zjistit.

7.4 Měření Difuse v2 - Analytic

Doba výpočtu je závislá na počtu řešitelských stanic, které distribuovaný výpočet provádějí, dále čas ovlivňuje počet segmentů a velikost kroku. Naměřené časy pro různé počty segmentů a velikosti kroku jsou v Tabulka 17. Počet řešitelských stanic byl 7, parametr **solutions/k** byl 1.



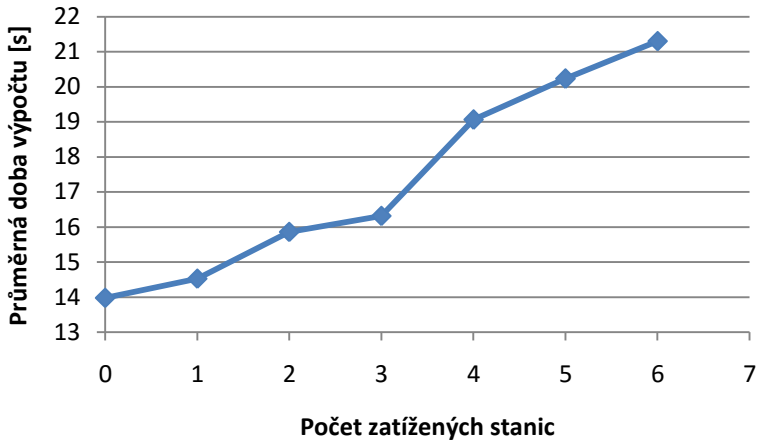
Obrázek 15: Čas distribuovaného výpočtu v závislosti na počtu segmentů a velikosti kroku

Obrázek 15 zobrazuje, jak se doba výpočtu prodlužuje se zvyšujícím se počtem počítaných segmentů. Z naměřených hodnot je vidět, že síťová režie výrazně převyšuje samotnou dobu výpočtu na řešitelské stanici, což je příčinou podobnosti časů u kroku 10s a 5s. V případě kroku 1s je doba výpočtu na řešitelské stanici výrazně vyšší, než je samotná síťová režie.

7.5 Load balancing

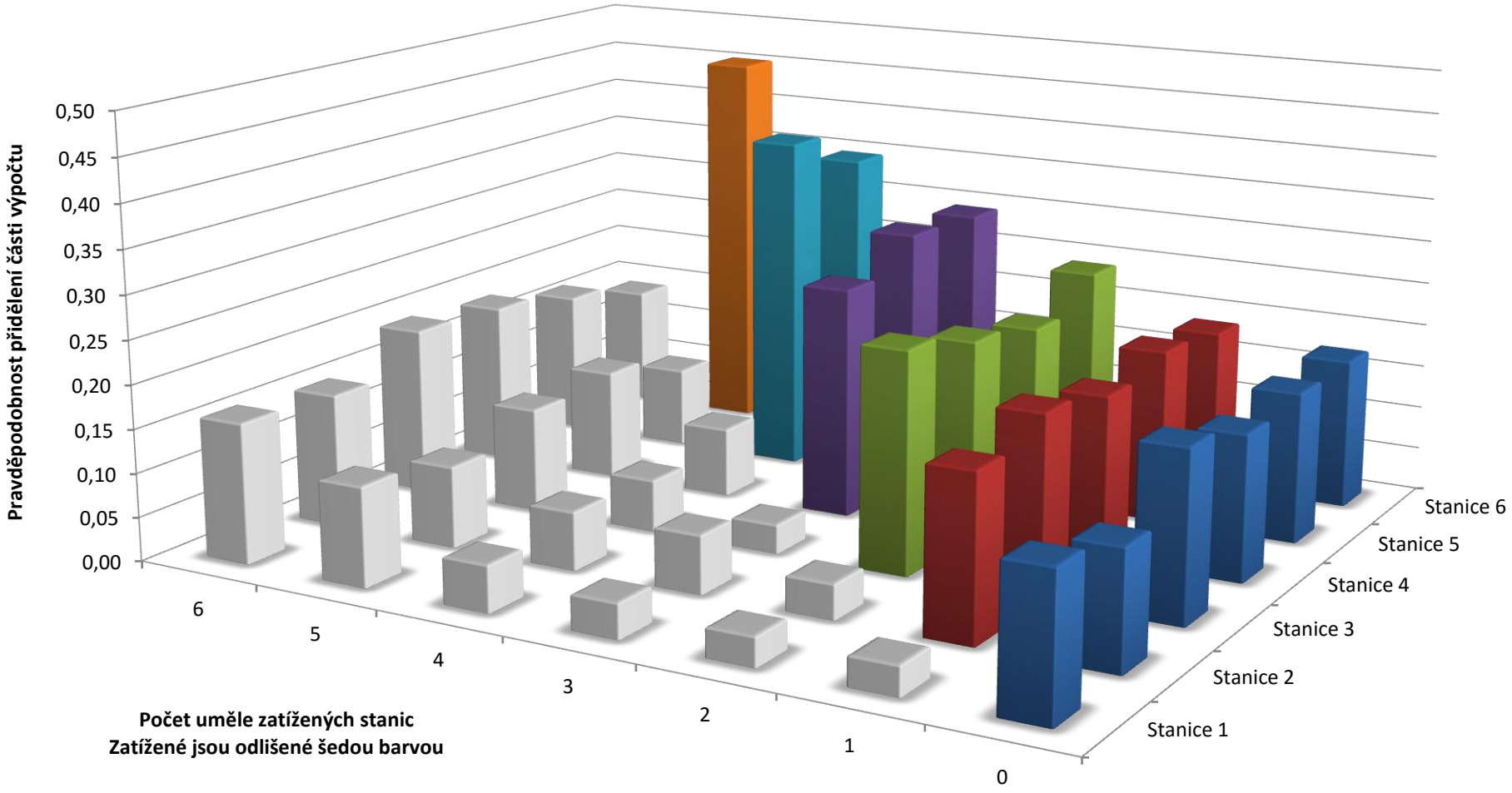
V následujícím měření jsem testoval, jakým způsobem ovlivňuje implementovaný load balancer celkový čas výpočtu. Povolený počet řešících stanic byl 3, z celkového počtu dostupných stanic 6. Během měření jsem postupně uměle zatěžoval procesor jednotlivých řešitelských stanic, abych mohl ověřit funkčnost implementovaného load balanceru. Procesor byl vytěžován programem Sleeper[18]. Paměť jsem během měření nevytěžoval, protože zatížení procesoru dostatečně ovlivňuje hodnocení řešitelské stanice (5.1.3).

Obrázek 16 zobrazuje průměrný čas výpočtu při měnícím se počtu zatížených řešitelských stanic. Dokud je počet nezatížených stanic větší než povolený počet řešících stanic, load balancer tyto stanice upřednostňuje. Při překročení této meze dochází k výraznému zpomalení celého výpočtu, protože některé části se musí počítat i na stanicích s horším hodnocením.



Obrázek 16: Vliv počtu zatížených stanic na celkový čas výpočtu

[18] dostupný z http://www.slunecnice.cz/sw/sleeper/



Obrázek 17: Pravděpodobnost přidělení části výpočtu konkrétní stanici podle počtu zatížených stanic

Obrázek 17 zobrazuje naměřené pravděpodobnosti přiřazení části výpočtu jednotlivým stanicím. Dokazuje, že load balancer upřednostňuje nezatížené stanice (protože mají lepší hodnocení) a snaží se vždy rozdělovat části rovnoměrně. Při žádném zatížení je vidět, že je rozložení částí mezi všechny stanice rovnoměrné. Dokud jsou k dispozici nezatížené stanice, jsou upřednostňovány pro přidělení částí výpočtu. Avšak se zvyšujícím se počtem vytížených stanic, se zvyšuje i pravděpodobnost, že vytížená stanice dostane část výpočtu. To je způsobené tím, jak se řešitelské stanice náhodně připojují a odpojují – na začátku můžou být všechny tyto stanice (vytížené se špatným hodnocením) připojené jako první a potom tedy nemá load balancer k dispozici lepší řešitelské stanice. Postupně se však tyto stanice vyloučí a nahradí je stanice s lepším hodnocením. Pokud jsou vytížené všechny, rozložení mezi stanice je opět rovnoměrné, celková doba výpočtu je však výrazně vyšší, než v případě kdy není zatížená žádná – viz Obrázek 16.

Pravděpodobnost přidělení části výpočtu byla vypočtena na základě četnosti přidělených částí výpočtu k jednotlivým řešitelským stanicím. Pro každé nastavení bylo provedeno 5 měření.

8 Závěr

Během vývoje řešení distribuovaného výpočtu jsem se seznámil s fungováním programu Glucose Predictor, abych zjistil a pochopil, jakým způsobem probíhá komunikace s jednotlivými implementacemi metod, které provádějí samotný výpočet. Poté jsem byl schopen vytvořit vlastní distribuované řešení. To se skládá ze dvou hlavních částí. První částí byla implementace knihovny *solver.dll*, která nahrazuje původní nedistribuované řešení. Druhou částí bylo vytvoření programu, který provádí část distribuovaného výpočtu. Dále jsem vytvořil, na základě přání vedoucího DP, tzv. Relay server, který umožňuje spouštět distribuovaný výpočet i mimo vnitřní síť.

V diplomové práci jsem popsal základní informace o onemocnění cukrovkou. Dále jsem popsal model dynamiky glukózy, využívaný ve výpočtech programu GP a vysvětlil základy distribuovaného výpočtu. Snažil jsem se podrobně popsat způsoby, jakým mé řešení distribuovaného výpočtu funguje a co všechno bylo potřeba vyřešit.

8.1 Dosažené výsledky

Z výsledů měření lze vyčíst, že implementované řešení distribuovaného výpočtu se vyplatí pouze u metody *Difuse v2 – Analytic*, ve které bylo urychlení výpočtu s krokem jedné sekundy více než trojnásobné a s krokem 0,3 sekundy dokonce 6 a půl násobné. Ostatní metody jsou buď natolik rychlé v nedistribuovaném řešení, nebo nejsou v současné implementaci použitelné pro distribuovaný výpočet. Například metoda DPS-DE[19], která využívá pro výpočet diferenciální evoluci, vždy provádí stejný počet iterací bez ohledu na zadané parametry výpočtu. Takže celkový čas distribuovaného výpočtu byl dokonce trojnásobný, než nedistribuovaný. To je zapříčiněno způsobem rozdělení výpočtu, které jsem popsal v kapitole 5.1.1.

Obrázek 17 zobrazuje naměřené pravděpodobnosti přiřazení části výpočtu jednotlivým stanicím. Lze vidět, že při postupném umělém (simulaci reálného) zatížení se implementovaný load balancer snaží upřednostnit stanice, které nejsou natolik vytížené, tzn., že mají lepší hodnocení na základně výpočtu uvedeném v kapitole 5.1.3.

[19] Differential Evolution With Dynamic Parameters Selection

8.2 Možnosti dalšího rozšíření

Asi hlavním důvodem, proč je výpočet ve většině metod pomalý, je fakt, že je příliš mnoho komunikace během inicializace mezi řídicí a řešící stanicí. To způsobuje zdržení při distribuovaném výpočtu. Proto by se chtělo zaměřit na samotné rozhraní, které by se mělo upravit tak, aby se minimalizovala komunikace.

Je možné přidat i další údaje o řešitelské stanici, aby vypočítané hodnocení bylo ještě užitečnější při vyhodnocování load balancerem. Dále by bylo možné se důkladněji zaměřit na výpočet hodnocení řešitelské stanice, se kterým hlavně souvisí přidělování části výpočtu v load balanceru.

Určitě by se pro distribuovaný výpočet měl upravit hlavní program Glucose Predictor, pro který by tak mohl lépe spolupracovat s mým implementovaným řešením – lepší průběžné informace o distribuovaném výpočtu, který je nyní pouze přes výpis v příkazové řádce. Bylo by rovněž vhodné, aby se program mohl jednoduše přepnout mezi distribuovaným a lokálním řešením výpočtu. Pro distribuované řešení také vytvořit konfigurační formulář přímo v programu, který by umožňoval rychlejší změnu nastavení.

Dalším užitečným rozšířením by bylo vzdálené ovládání jednotlivých řešitelských stanic.

Použitá literatura a zdroje

1. **Strnádek, Jan.** Dimplomová práce. *Metoda odhadu chyby výpočtu koncentrace glukózy.* Plzeň : Západočeká univerzita, 2015.

2. Diabetes 2. typu. *Diabetická asociace ČR.* [Online] [Citace: 10. 6 2016.] http://www.diabetickaasociace.cz/co-je-diabetes/diabetes-2-typu/.

3. Complications of Diabetes. *International Diabetes Federation.* [Online] [Citace: 4. 6 2016.] http://www.idf.org/complications-diabetes.

4. **Koutný, Tomáš.** Using meta-differential evolution to enhance a calculation of a continuous blood glucose level. *Computer Methods and Programs in Biomedicine.* 2016.

5. **Chen, D., Wang, L. a Chen, J.** *Large-Scale Simulation: Models, Algorithms, and Applications.* Boca Raton, Florida : CRC Press, 2012. 9781439868867.

6. **Otčenášek, Petr.** Diplomová práce. *Distribuované řídící systémy a jejich využití v praxi.* Brno : Petr Otčenášek, 2008.

7. **Janeček, Jan.** Skripta. *Distribuované systémy.* Praha : Jan Janeček, 2000.

8. **Prata, Stephen.** *Mistrovství v C++ 4. aktualizované vydání.* Praha : COMPUTER PRESS, 2013. 978-80-251-3828-1.

9. **Guyton, A. C. a Hall, J. E.** *Medical Textbook of Physiology.* Philadelphia : Elsevier Inc., 2006. 978-0-7216-0240-0.

10. **Poretsky, L.** *Principles of Diabetes Mellistus.* New York : Springer, 2010. 978-0-387-09840-1.

11. **Longo, D., a další.** *Harrison's principles of internal medicine.* New York : Mc Graw-Hill, 2011.

Seznam ilustrací

Seznam tabulek

Příloha A - Uživatelská dokumentace

Na přiloženém DVD jsou k dispozici všechny přeložené programy. Před spuštěním je potřeba překopírovat adresáře **bin** a **data** někam na lokální uložiště.

Spuštění programu Worker

Program Worker se všemi potřebnými knihovnami nalezneme na DVD v adresáři **bin\worker\release**. Program lze spustit buď přímo z příkazové řádky, nebo je možné ho zaregistrovat jako službu. Pro zaregistrování služby je potřeba spustit příkazový řádek v režimu správce. Po zadání příkazu **Worker.exe –i** se program zaregistruje mezi ostatní služby a je možné ho spustit. Pokud je potřeba službu odebrat, stačí použít příkaz **Worker.exe -u**.

Pokud se rozhodneme spustit program přímo, bez registrace služby, slouží k tomu příkaz **Worker.exe –exec**. Po spuštění program čeká na oslovení a přidělení práce.

Pokud nechceme použít výchozí nastavení, je potřeba vytvořit soubor *worker.ini*, ve kterém je možné nastavit multicastovou adresu a port. Obsah soubor může vypadat následovně:

```
[network]
mc-address = 224.0.0.255
mc-port = 25000
```

Před spuštěním programu **Worker** je potřeba spustit ještě program **CoreTemp**, který nalezneme v adresáři **bin\coretemp**, program poskytuje údaje o vytíženosti a teplotě procesoru. Důvod použití externího programu je popsán v kapitole 6.3.3.

Všechny podstatné údaje se vypisují do příkazového řádku a současně i zapisují do souboru *worker.log* v adresáři **logs**.

Spuštění programu Relay

Program Relay se všemi potřebnými knihovnami nalezneme na DVD v adresáři **bin\relay\release**. Program lze spustit buď přímo z příkazové řádky, nebo je možné ho zaregistrovat jako službu. Pro zaregistrování služby je potřeba spustit příkazový řádek v režimu správce. Po zadání příkazu **Relay.exe –i** se program zaregistruje mezi ostatní služby a je možné ho spustit. Pokud je potřeba službu odebrat, použijeme příkaz **Relay.exe –u**.

Pokud se rozhodneme spustit program přímo, bez registrace služby, slouží k tomu příkaz **Relay.exe –exec**. Po spuštění program čeká na zprávy odeslané programem GP, které následně přeposílá jako multicastovou zprávu do vnitřní sítě.

Pokud nechceme použít výchozí nastavení, je potřeba vytvořit soubor *relay.ini,* ve kterém je možné nastavit multicastovou adresu a port, dále ještě port pro příchozí zprávy. Obsah soubor může vypadat následovně:

```
[network]
mc-address = 224.0.0.255
mc-port = 25000
relay-port = 25001
```

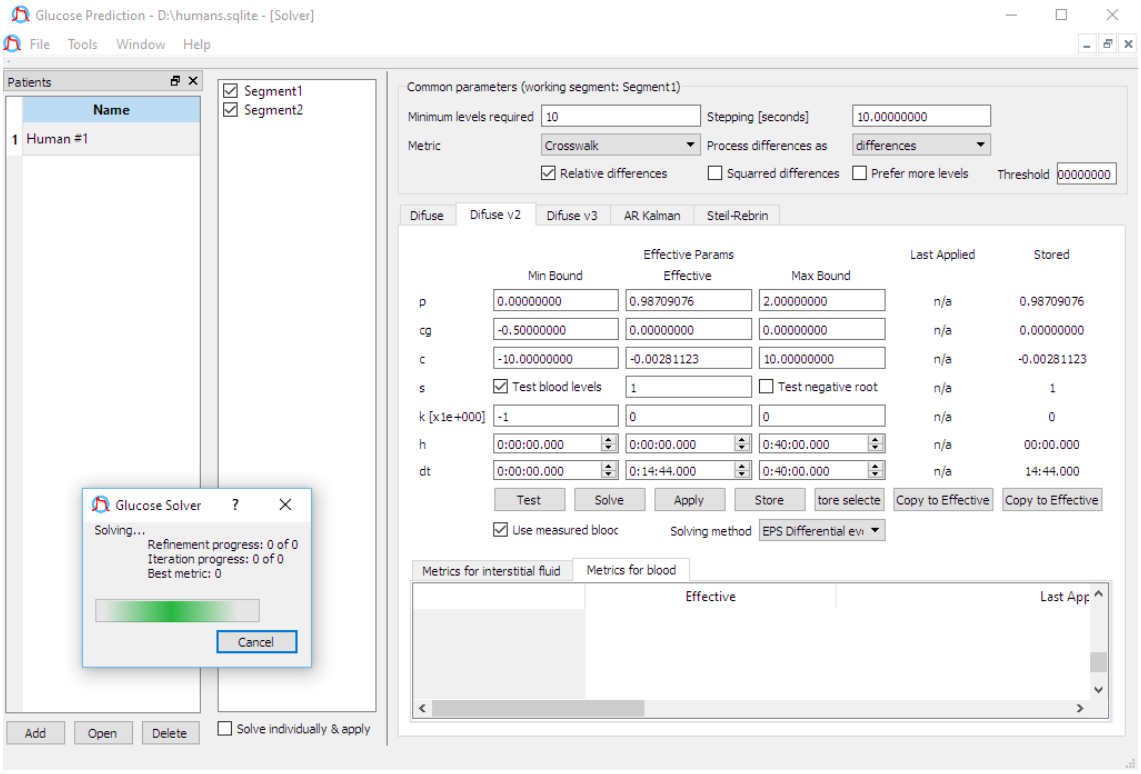
Všechny podstatné údaje se vypisují do příkazového řádku a současně i zapisují do souboru *relay.log* v adresáři **logs**.

Spuštění programu Glucose Predictor

Program GP je na DVD ve dvou verzích, v adresáři **bin\gpredict-original** je k dispozici v originálním stavu (nedistribuovaný výpočet) a potom v adresáři **bin\gpredict** upravený, který provádí výpočet distribuovaně. Obě verze se spouští stejně, programem **gpredict2.exe**. Nastavení programu je v souboru *config.ini,* kde lze nastavit cestu k souboru pro vstupní data.

Upravená verze může mít ještě druhý konfigurační soubor *solver.ini*, ve kterém lze upravit výchozí nastavení výpočtu a komunikace programu s řešitelskými stanicemi, na kterých je spuštěn program **Worker**. Obsah souboru a parametry, které je možné nastavit je popsáno v kapitole 6.6.3.

Po spuštění programu, viz Obrázek 18, je možné v levém panelu vybrat data pacienta dvojklikem na položku. Následně je potřeba vybrat ze seznamu jeden nebo více segmentů a stiskem tlačítka *Solve* se zobrazí obsáhlý formulář, ve kterém lze vybrat metodu výpočtu (*Solving method*), metriku (*Metric*) a krok (*Stepping*). Program disponuje mnoho dalšími parametry a funkcemi, ale to pro základní spuštění distribuovaného výpočtu není potřebné upravovat. Stiskem tlačítka *Solve* se začne provádět výpočet. V případě upravené verze se – podle nastavení – může zobrazit příkazový řádek, ve kterém lze vidět detailnější informace při výpočtu. Samozřejmě totožné údaje se zapisují i do souboru *solver.log* v adresáři *logs*.



Obrázek 18: Náhled spuštěného výpočtu parametrů v programu GP

Příloha B – Naměřené hodnoty

Tabulka 16: Závislost metriky na počtu částí - Difuse v2 - Analytic

Části	Metrika	Rozdíl metriky
1	0,364521	0
14	0,36468	0,000159
21	0,36468	0,000159
35	0,364594	0,000073
70	0,364701	0,00018
350	0,36463	0,000109
700	0,364374	-0,000147

Tabulka 17: Difuse v2 - Analytic - závislost času výpočtu na kroku a počtu segmentů

k = 1		Doba výpočtu [s] v n-tém měření															Statistické údaje						
Počet seg.	Krok [s]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Min.	Q1	$\tilde{x}$	Q3	Max.	Avg.	σ
1	10	1,60	1,66	1,96	1,89	1,95	2,06	1,81	1,82	1,84	1,97	1,82	1,82	1,97	1,86	1,81	1,60	1,81	1,84	1,95	2,06	1,86	0,12
1	5	2,05	2,06	2,04	1,87	2,02	1,92	2,10	2,03	1,90	1,90	1,90	1,87	1,92	1,89	2,02	1,87	1,90	1,92	2,04	2,10	1,97	0,08
1	1	3,85	4,09	3,84	3,91	3,97	3,86	3,84	3,79	3,94	3,69	4,15	3,87	3,82	3,89	3,88	3,69	3,84	3,87	3,92	4,15	3,89	0,11
2	10	3,29	3,33	3,31	3,29	3,25	3,35	3,45	3,45	3,32	3,34	3,38	3,42	3,26	3,24	3,33	3,24	3,29	3,33	3,36	3,45	3,33	0,07
2	5	3,45	3,42	3,42	3,43	3,44	3,34	3,41	3,44	3,45	3,38	3,48	3,48	3,41	3,46	3,38	3,34	3,41	3,43	3,45	3,48	3,43	0,04
2	1	6,58	6,56	6,62	6,53	6,57	6,53	6,51	6,50	6,56	6,61	6,55	6,46	6,47	6,57	6,59	6,46	6,52	6,56	6,58	6,62	6,55	0,05
3	10	7,71	7,32	7,35	7,41	7,32	7,32	7,34	7,43	7,38	7,37	7,39	7,33	7,30	7,32	7,39	7,30	7,32	7,35	7,39	7,71	7,38	0,10
3	5	7,79	7,78	7,76	7,78	7,78	7,77	7,72	7,76	7,71	7,73	7,72	7,75	7,72	7,88	7,78	7,71	7,73	7,76	7,78	7,88	7,76	0,04
3	1	17,66	17,50	17,71	17,64	17,56	17,68	17,59	17,74	17,69	17,77	17,58	17,60	17,53	17,68	17,60	17,50	17,59	17,64	17,69	17,77	17,64	0,08
4	10	9,78	8,92	9,03	9,02	8,94	8,87	8,98	9,08	9,01	8,99	9,13	8,97	9,01	9,04	8,92	8,87	8,96	9,01	9,03	9,78	9,05	0,21
4	5	9,50	9,48	9,46	9,67	9,35	9,28	9,37	9,35	9,48	9,38	9,43	9,34	9,60	9,40	9,56	9,28	9,36	9,43	9,49	9,67	9,44	0,11
4	1	21,90	21,95	21,87	21,90	21,75	21,87	21,92	21,83	21,96	22,05	21,88	21,91	22,10	22,01	21,77	21,75	21,87	21,90	21,96	22,10	21,91	0,09
5	10	12,69	12,59	12,71	12,69	12,69	12,61	12,63	12,70	12,69	12,67	12,72	12,56	12,68	12,62	12,65	12,56	12,63	12,68	12,69	12,72	12,66	0,05
5	5	13,45	13,32	13,34	13,33	13,37	13,36	13,36	13,64	13,35	13,37	13,62	13,34	13,34	13,52	13,46	13,32	13,34	13,36	13,45	13,64	13,41	0,11
5	1	34,08	33,59	33,61	34,07	33,13	33,73	33,45	33,76	33,50	33,45	32,49	32,54	32,53	32,51	32,60	32,49	32,57	33,45	33,67	34,08	33,27	0,59

Tabulka 18: Difuse v2 - Analytic - Závislost doby výpočtu na počtu částí

Počet stanic = 7	Doba výpočtu [s] v n-tém měření										Statistické údaje						
k	1	2	3	4	5	6	7	8	9	10	Min.	Q1	$\tilde{x}$	Q3	Max.	Avg.	σ
1	2,00	1,97	1,88	2,06	1,85	1,84	1,85	2,03	1,82	1,98	1,82	1,85	1,93	2,00	2,06	1,93	0,09
2	3,49	3,52	3,51	3,48	3,54	3,53	3,52	3,48	3,53	3,49	3,48	3,49	3,51	3,53	3,54	3,51	0,02
3	5,10	5,01	5,10	5,00	5,11	5,05	5,13	5,09	5,12	4,95	4,95	5,02	5,09	5,11	5,13	5,07	0,06
5	7,73	8,05	8,36	8,07	8,05	8,25	8,06	8,06	8,07	8,03	7,73	8,05	8,06	8,07	8,36	8,07	0,16
10	15,60	15,90	15,60	15,31	15,32	15,60	15,93	15,59	15,86	16,02	15,31	15,59	15,60	15,89	16,02	15,67	0,25
50	75,32	75,00	75,31	75,65	75,22	75,02	74,96	74,25	74,66	74,99	74,25	74,97	75,01	75,29	75,65	75,04	0,39
100	149,10	148,32	148,23	148,30	148,93	148,55	148,94	148,90	148,53	148,06	148,06	148,30	148,54	148,92	149,10	148,58	0,36

Tabulka 19: Pravděpodobnost přidělení části výpočtu konkrétní stanici

Zatížené stanice	Stanice 1	Stanice 2	Stanice 3	Stanice 4	Stanice 5	Stanice 6
0	0,160	0,133	0,193	0,167	0,173	0,173
1	0,033	0,187	0,207	0,187	0,200	0,187
2	0,033	0,040	0,253	0,227	0,207	0,240
3	0,040	0,067	0,033	0,267	0,300	0,293
4	0,053	0,067	0,060	0,080	0,393	0,347
5	0,113	0,093	0,120	0,127	0,093	0,453
6	0,160	0,153	0,193	0,187	0,167	0,140