VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

KLIENT-SERVER APPLIKACE ZALOŽENÁ NA TECHNOLOGII CORBA

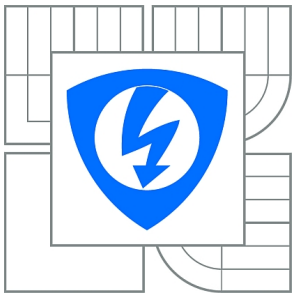
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MAJTÁN
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

KLIENT-SERVER APPLIKACE ZALOŽENÁ NA TECHNOLOGII CORBA

CLIENT-SERVER APPLICATION BASED ON CORBA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

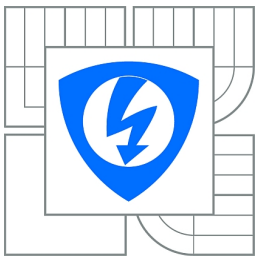
AUTOR PRÁCE — MARTIN MAJTÁN

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KARÁSEK

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Martin Majtán	***ID:***	147426
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Klient-server applikace založená na technologii CORBA

POKYNY PRO VYPRACOVÁNÍ:

Náplní bakalářské práce bude zpracování rešerše týkající se síťových komunikačních technologií používaných při vývoji aplikací typu klient-server v programovacím jazyce JAVA. Student provede srovnání výhod a nevýhod nastudovaných technologií a dále se zaměří na technologie CORBA a Hessian. V praktické části student nejprve provede návrh paralelizace genetického algoritmu pro problém batohu, který implementuje a srovná se zpracováním algoritmu v jednom vlákně. Dále student vytvoří dva distribuované modely algoritmu na základě znalostí z první části práce a jejich výsledky srovná s předcházejícími. Klientská část bude schopna spustit výpočet algoritmu na N serverech v M vláknech. Výsledky budou vhodně prezentovány v tabulkách a grafech.

DOPORUČENÁ LITERATURA:

[1] Vogel, A.; Duddy, K.; Java Programming with CORBA. Willey; 2nd edition. February, 1998. s. 528. ISBN-13: 978-0471247654.

[2] Harold, E. R.; Java Network Programming. Java Series. O'Reilly Media; 3rd edition. October, 2004. s. 504. ISBN-13: 978-0596007218.

[3] Oaks, S.; Java Threads. Java Series. O'Reilly Media; 3rd edition. September 17, 2004. s. 362. ISBN-13: 978-0596007829

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Ing. Jan Karásek
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

ABSTRAKT

Bakalárska práca sa zaoberá problematikou klient-server aplikácií a technológiami používanými na softvérovú implementáciu klient-server aplikácií v programovacom jazyku Java. Cieľom bakalárskej práce je paralelizácia genetického algoritmu pre problém batohu a vytvorenie dvoch distribuovaných modelov pre technológie CORBA a Hessian. V teoretickej časti práce sú popísané základné pojmy týkajúce sa sieťovej komunikácie, vysvetlenie klient-server modelu sieťovej komunikácie, sú rozobraté technológie Java RMI, CORBA a Hessian. V práci je popísaný paralelný a distribuovaný model spracovania dát, genetický algoritmus a jeho využitie na vyriešenie problému naplnenia batohu. V praktickej časti je vytvorený paralelný a distribuovaný model genetického algoritmu pre problém naplnenia batohu s použitím technológií CORBA a Hessian. V práci je spravené porovnanie týchto modelov z hľadiska času výpočtu. Výsledky merania času genetického algoritmu sú v tabuľkách.

KĽÚČOVÉ SLOVÁ

Java RMI, CORBA, Hessian, klient-server aplikácia, genetický algoritmus, paralelizácia, distribuované výpočty

ABSTRACT

Bachelor thesis deals with client-server applications and software technologies used to implement client-server applications in the Java programming language. The goal of bachelor thesis is the parallelization of genetic algorithm for knapsack problem and create two distributed models for technology CORBA and Hessian. In the teoretical part of the thesis are describes the basic concept of network communication, explanation client-server model of network communication, there are discussed technologies Java RMI, CORBA and Hessian. In the thesis is described the parallel and the distributed model of data processing, genetic algorithm and its use to solve the knapsack problem. In the practical part is created parallel and distributed model of a genetic algorithm for knapsack problem using technology CORBA and Hessian. In the thesis is done comparison of parallel model and distributed models in terms of calculation time. Results of measurement time are displayed in tables.

KEYWORDS

Java RMI, CORBA, Hessian, client-server application, genetic algorithm, parallelization, distributed computing

MAJTÁN, Martin *Klient-server applikace založená na technologii CORBA*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 54 s. Vedúci práce bol Ing. Jan Karásek,

PREHLÁSENIE

Prehlasujem, že som svoju bakalársku prácu na tému „Klient-server applikace založená na technologii CORBA“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánovi Ing. Janovi Karáskovi, za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Distribuované systémy tvoria dôležitú časť informačných systémov, náchadzajú široké uplatnenie telekomunikáciach a sieťových aplikáciách, v priemysle, vo vede a výzkume na modelovanie zložitých problémov. Distribuované systémy sa stali nástrojom k riešeniu zložitých problémov, ktoré vyžadujú veľký výpočtový výkon a tiež pomáhajú zredukovať čas a náklady na vyriešenie problému.

V tejto práci sú rozobraté technológie CORBA, Java RMI a protokol Hessian, ktoré umožnujú vytvárať jednoduché klient-server aplikácie až po zložité distribuované systémy v programovacom jazyku Java. Cieľom bakalárskej práce bolo vytvorenie paralelného a distribuovaného modelu genetického algoritmu pre problém naplnenia batohu v programovacom jazyku Java. Distribuovaný model genetického algoritmu je vytvorený pomocou technológie CORBA a Hessian. Na vytvorenie paralelného a distribuovaného modelu genetického algoritmu bol použitý zdrojový kód genetického algoritmu ktotý mi poskytol Ing. Jan Karásek.

Celá práca je rozdelená do piatich kapitôl. Prvá kapitola je venovaná histórii sieťovej komunikácie a sú v nej vysvetlené základné a často používané pojmy týkajúce sa sieťovej komunikácie. Ďalej je v prvej kapitole popísaný klient-server model sieťovej komunikácie a sú tam zhrnuté výhody a nevýhody tohoto modelu. V druhej kapitole sú rozobraté technológie CORBA, Java RMI a protokol Hessian a popísané výhody a nevýhody týchto technológií. V tretej kapitole je popísaný paralelný a distribuovaný model spracovávania dát. V štrvtej kapitole je popísaný genetický algoritmus a vysvetlenie jeho funkcie, problém naplnenia batohu a aplikovanie genetického algoritmu na vyriešenie problému naplnenia batohu. Ďalej je v tejto kapitole vysvetlená paralelizácia a distribúcia genetického algoritmu. Piata kapitola sa venuje výsledkom práce. Je tu popísaný spôsob merania času genetického algoritmu a nastavenie parametrov genetického algoritmu počas merania. Ďalej sú tu vysvetlené časti zdrojových kódov genetického algoritmu ktoré rozdelujú inicializáciu populácie v prípade paralelného modelu do viacerých vlákien a v prípade distribuovaného modelu na viacej počítačov. V piatej kapitole je tiež spravené meranie času jednovláknového spracovania, paralelného vo viacerých vláknach a distribuovaného na viacerých počítačoch. Namerané hodnoty sú v tabuľkách.

1 SIEŤOVÁ KOMUNIKÁCIA

1.1 História sieťovej komunikácie

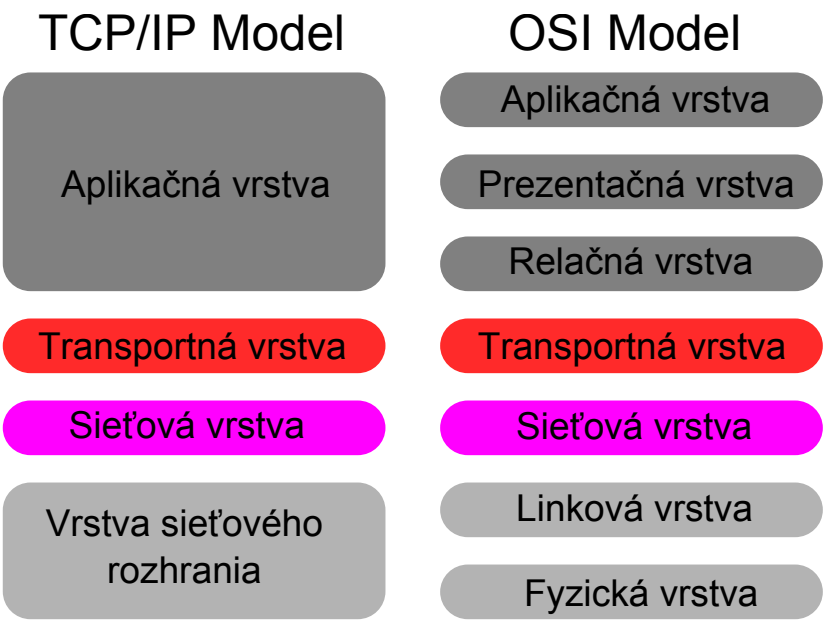
Začiatky sieťovej komunikácie začínajú v roku 1969, v tomto roku výskumná agentúra Advanced Research Projects Agency (ARPA) vytvorila prvú experimentálnu počítačovú sieť nazývanú ARPA Network (ARPANET), ktorá pracovala na princípe prepínania paketov a prepojovala 4 počítače. Na komunikáciu bol použitý protokol Network Control Protocol (NCP). Do tejto siete sa postupne pripájaly ďalšie počítače a v roku 1973 ARPANET prenikol aj do Európy. V roku 1983 sa prestal používať protokol NCP, ktorý bol nahradený Transmission Control Protocol/Internet Protocol (TCP/IP) ten sa používa dodnes. S nástupom lokálnych sietí sa ARPANET stále častejšie využíval iba ako hlavná sieť pre prenos na veľké vzdialenosti medzi lokálnymi sieťami, túto jeho funkciu postupne prebrala National Science Foundation Network (NSFNET) a v roku 1990 bola sieť ARPANET zrušená.[6]

V roku 1970 začaly dva projekty nezávisle na sebe s rovnakým cieľom, vytvoriť zjednotený štandard pre architektúru sieťových systémov. Jeden bol spravovaný International Organization for Standardization (ISO), druhý spravovala International Telegraph and Telephone Consultative Committee (CCITT). Tieto dve organizácie vytvorily dokumenty, ktoré definovaly podobné sieťové modely. Tieto dva dokumenty boli v roku 1983 spojené aby vytvorili štandard nazývaný The Basic Reference Model for Open Systems Interconnection, často nazývaný iba ako OSI Reference Model. V roku 1984 bol tento štandard prijatý ako ISO 7498 [8]. OSI Reference Model bol pôvodne vytvorený ako základ pre navrhovanie univerzálnych protokolov nazývaných OSI Protocol Suite. Tento model nikdy nedosiahol úspech, ale stal sa užitočným nástrojom pre vzdelávanie v oblasti sieťových technológií. Obr. 1.1 znázorňuje porovnanie dvoch sieťových architektúr, TCP/IP model a ISO/OSI model podľa[9]. Konkrétna realizácia TCP/IP modelu je popísaná v kapitole 1.2.1. [27]

1.2 Základné pojmy

1.2.1 Protokol

Sieťový protokol definuje súbor pravidiel, algoritmov správ a ďalších mechanizmov, ktoré umožnia softvéru a hardvéru komunikovať. Protokol zvyčajne opisuje prostriedky pre komunikáciu medzi dvomi alebo viacerými zariadeniami, ktoré pracujú na rovnakej vrstve OSI modelu.



Obr. 1.1: Porovnanie TCP/IP modelu a ISO/OSI modelu.

Internet Protocol (IP) je zakladným protokolom pracujúcim na sieťovej vrstve používaným v počítačových sieťach. Jednou z úloh IP je adresovanie datagramov aby mohli byť doručené do spravných sietí . Ďalšou úlohou IP je zapuzdrovanie dát. IP príjma dáta z transportnej vrstvy a zapuzdruje ich do IP datagramov pred posielaním. Maximálna veľkosť rámca sa v sieti, ktorá používa IP môže líšiť, preto IP zahŕňa schopnosť fragmentácie IP datagramov. IP tiež podporuje priame a nepriame smerovanie IP datagramov. Priame sa využíva v lokálnych sieťach medzi zariadeniami v jednej sieti. Nepriame smerovanie sa využíva pri prenose dát medzi rôznymi sieťami. IP je často používaný spoločne s TCP. V súčastnosti sa používajú dve verzie IP, Internet Protocol version 4 (IPv4) a Internet Protocol version 6 (IPv6).[12]

Transmission Control Protokol (TCP) je ďalším zo základných protokolov ktoré sa využívajú na komunikáciu v počítačových sieťach, pracuje na transportnej vrstve, dáta sú prenášané formou paketov. TCP využíva veľa rôznych služieb, preto je dôležité aby TCP multiplexoval dáta získane z rôznych procesov, aby mohli byť zaslané pomocou protokolov na sieťovej vrtsve. TCP je spojovo orientovaný protokol, obsahuje sadu postupov ako vytvárať, riadiť a ukončovať spojenie medzi zariadeniami. Medzi výhody TCP patrí spôsob spojovej komunikácie a garantované doručenie dát. Nevýhodou je malé oneskorenie prenosu dát spôsobené vytváraním spojenia.[7]

User Datagram Protokol (UDP) je ďalším protokolom, ktorý sa často používa v počítačových sieťach a rovnako ako TCP aj UDP pracuje na transportnej vrstve. UDP poskytuje nespojovanú komunikáciu, dáta sú prenášané formou datagramov. Hlavnou výhodou UDP je, že sa nemusí vytvárať spojenie, prenos dát je rýchlejší.

Nevýhodou UDP je negarantované doručenie dát.[7]

Transmission Control Protocol/Internet Protocol (TCP/IP) je rodina protokolov, ktorá obsahuje protokoly pre komunikáciu v počítačových sieťach a v súčastnosti je najpoužívanejšou sadou protokolov. Architektúra TCP/IP podľa [10]:

1. Vrstva sieťového rozhrania, umožňuje prístup k fyzickému médiu. Je špecifikovaná pre každú sieť, napr.: Ethernet, Token ring a iné.
2. Sieťová vrstva, zaisťuje sieťovú adresáciu, smerovanie a predávanie datagramov. Je implementovaná vo všetkých prvkoch siete. Na sieťovej vrstve pracujú protokoly IP, ARP, ICMP, IPSEC, IGMP a iné.
3. Transportná vrstva, je implementovaná až v koncových zariadeniach a umožňuje prispôsobiť chovanie siete individuálnym potrebám aplikácie. Poskytuje prenos dát pomocou protokolov TCP a UDP.
4. Aplikačná vrstva, túto vrstvu používajú programy ktoré využívajú prenos dát po sieti pre užívateľov, napr.: FTP, HTTP, DHCP.

1.2.2 Sieťové rozhranie

Sieťové rozhranie je bod prepojenia medzi počítačom a súkromnou alebo verejnou sieťou. Sieťové rozhranie je zvyčajne sieťová karta, ale nemusí mať iba fyzickú podobu, sieťové rozhranie môže byť implementované ako softvér. [4]

1.2.3 IP adresa

IP adresa je číslo, ktoré jednoznačne identifikuje sieťové rozhranie v počítačovej sieti, ktoré používajú na komunikáciu IP. V súčastnosti sa používajú dva druhy IP adries, ktoré používaju IPv4 a IPv6. IPv4 používa 32bitovú adresu rozdelenú na štyri oktety po ôsmich bitoch. Druhý typ IP adresy používa IPv6, je to 128bitová adresa. Na zápis sa používa osem skupín so štyrmi hexadecimálnymi znakmi. [12]

1.2.4 Port

Klientské zariadenia komunikujúce cez sieť majú väčšinou jedno fyzické pripojenie (sieťová karta). Cez toto pripojenie sa prenášajú všetky dáta z rôznych aplikácii určené pre konkrétne zariadenie. Aby toto zariadenie vedelo ktoré dáta má priradiť ktorej aplikácii, všetky dáta majú adresu zariadenia kam majú byť doručené a číslo portu. Porty sú identifikované 16bitovým číslom. Číslo portu používajú TCP a UDP protokoly na doručenie dát do správnej aplikácie. Rozsah všetkých portov je 0-65 535, porty s rozsahom 0-1023 sú rezervované pre často používané aplikácie napr.: HTTP (80), FTP (20), SMTP (25), DNS (53), Telnet (23), POP3 (110). [11]

1.2.5 Socket

Socket je jeden koncový bod obojsmernej komunikácie medzi dvoma programami. Socket je naviazaný na číslo portu takže TCP protokol môže identifikovať aplikáciu kam majú byť dáta doručené. [19]

1.2.6 Uniform Resource Identifier

Uniform Resource Identifier (URI) alebo jednotný identifikátor zdroja je textový reťazec s definovanou štruktúrou, ktorý slúži k presnej špecifikácii zdroja informácii. Hlavným účelom je použitie v počítačových sieťach. Jednotný identifikátor zdroja podľa [13] má dve časti:

- Uniform Resource Locator (URL), jednotný lokátor zdroja definuje doménovú adresu serveru, umiestenenie zdroja na serveri a protokol ktorým je možné pristupovať k informáciám na serveri.
- Uniform Resource Name (URN), jednotný identifikátor mena slúži ako trvalý identifikátor zdroja informácií. Jedná sa o identifikáciu zdroja informácií v počítačových sieťach, ktorá jednoznačne určí umiestnenie zdroja bez ohľadu na dómenu alebo server na ktorom sa nachádza.

Najľahší spôsob ako vytvoriť URL je vytvoriť ho z textovej podoby, ktorá je dobre zrozumitelná pre ľudí. Väčšinou sa jedná o absolútne vytvorené URL adresy. Absolútna URL adresa obsahuje všetky informácie potrebné k nájdeniu súboru. Relatívne vytvorená URL adresa neobsahuje celú cestu k súboru ale napríklad len cestu k priečinku. Na vytváranie URL adries so špeciálnymi znakmi sa využíva Uniform Resource Identifier (URI). Keď adresa obsahuje špeciálny znak, najskôr sa z tejto adresy vytvorí URI, pri vytváraní URI sa zakódujú špeciálne znaky v adrese a potom sa URI prevedie na URL adresu. [18]

1.3 Klient-server aplikácie

Pojem klient-server označuje často používaný model sieťovej komunikácie. V tomto modeli sú dva typy zariadení, klient a server. Každé s týchto zariadení je optimalizované vykonávať iný typ úlohy. Klient-server model môže byť použitý na lokálnych sieťach alebo na Internete.

Zariadenia typu klient sú väčšinou osobné počítače, ktoré cez sieť odosielajú požiadavky a prijímajú na ne odpovede. Nepotrebujú veľký výkon ani veľa pamäte, väčšinou obsahujú jednoduchý softvér pretože ich hlavnou úlohou je zobrazovať informácie prijaté zo serveru, napr. zobrazenie webovej stránky.

Klienti sa ďalej delia na tenkých a tučných. Tenký klient je počítač alebo program, ktorý pri plnení svojej úlohy závisí na inom počítači, najčastejšie to býva server. Tenkí klienti sa často vyskytujú ako súčasť väčšej počítačovej infraštruktúry, kde viac klientov zdieľa výpočty s rovnakým serverom. Častým typom tenkého klienta je terminál, ktorý poskytuje iba grafické rozhranie a webový prehliadač, ostatné funkcie ako operačný systém poskytuje server. Medzi výhody tenkého klienta patrí malá cena, jednoduchosť a fakt, že nepotrebuje synchronizovať dáta pretože tie sú stále na serveri. Nevýhodou je, že tenký klient vyžaduje stále pripojenie na server, v opačnom prípade má veľmi obmedzené funkcie.[14]

Tučný klient je opakom tenkého klienta, tučný klient poskytuje funkcie nezávisle na serveri. Pri používaní tučných klientov nemusí mať server taký veľký výkon ako pri použití tenkého klienta. Tučný klient nevyžaduje stále pripojenie na server, je vhodný na multimediálne aplikácie. Výhodou tučného klienta je schopnosť samostatne spracovávať informácie a nezávislosť na serveri, nevýhodou je väčšia cena, väčšie nároky na výkon a v niektorých prípadoch nutnosť synchronizácie dát zo serverom.[15]

Zariadenie ktoré v sieti zastupujú úlohu serveru sú náročné na výpočtový výkon a pamäť, pretože musia spracovávať veľké množstvo dát v reálnom čase, napr. počítanie zložitých operácií, ktoré by sa na osobných počítačoch spracovávali príliš dlho. Servery majú optimalizovaný softvér na tieto úlohy a maju veľké množstvo pamäte takže tiež dokážu uchovávať veľké databázy. Server príjma požiadavky od klientov, spracuje ich a výsledky posiela naspäť klientom.

V klient-server modeli je server centralizovaný systém. Výhody tohoto systému sú v rýchlejšom spracovávaní informácií na strane serveru. Nevýhoda je že, keď sa server pokazí alebo sa preruší spojenie zo serverom tak užívatelia, ktorí mali uložené dáta na serveri k nim stratia prístup. [5]

2 SIEŤOVÁ KOMUNIKÁCIA V JAVE

Táto časť práce sa zaoberá technológiami, ktoré sú v súčastnej dobe najpoužívanejšie pre sieťovú komunikáciu so zameraním na programovací jazyk Java a porovnaním týchto technológií.

2.1 Java Remote Method Invocation

Java Remote Method Invocation (RMI) umožnuje vzdialenú komunikáciu medzi programami napisanými v Jave. Java RMI dovoluje objektu bežiacemu v jednom Java Virtual Machine (JVM) vyvolať metódy na objekte v inom JVM. Aplikácie vytvorené pomocou RMI sa často skladájú z dvoch častí. Program na serveri vytvorí objekty, na tieto objekty vytvorí referencie a sprístupní ich pre klientské programy a čaká na klientov, ktorí chcú vyvolať metódy na týchto objektoch. Klientský program získa odkaz na jeden alebo viac vzdialených objektov na serveri a potom na nich môže vyvolávať metódy. Technológia RMI poskytuje mechanizmus ktorým môžu server a klient komunikovať medzi sebou a odovzdávať si informácie. Tieto aplikácie sa niekedy nazývaju ako objektovo distribuované aplikácie.[16]

Objektovo distribuované aplikácie na správne fungovanie potrebujú:

- Lokalizovať vzdialené objekty. Aplikácie môžu využívať rôzne spôsoby na získanie referencií na vzdialené objekty. Aplikácia môže zaregistrovať vzdialený objekt pomocou RMI registry.
- Komunikovať zo vzdialenými objektami. Detaily komunikácie medzi vzdialenými objektami má na starosti RMI ,takže pre programátora vyzerá komunikácia medzi vzdialenými objektami ako normálne volanie metód v Jave.
- Musia poznať definície tried, ktoré chcú vyvolávať. RMI umožnuje objektom prechádzať medzi dvoma JVM, to poskytuje mechanizmus pre načítanie definicií tried objektov podobne ako prenos dát objektov.

2.1.1 Výhody dynamického načítania kodu

Unikátnou vlastnosťou RMI je schopnosť načítať definíciu objektu triedy aj keď trieda nie je definovaná na strane klienta v JVM. Správanie objektov bolo prístupné iba v jednom JVM, teraz môžu byť prenesené do iného JVM alebo do vzdialeného. RMI posiela objekty do skutočných tried, takže spravánie objektov sa nezmení keď sú odoslané do iného JVM. Táto funkcia umožňuje zaviesť nové spôsoby správania objektov do vzdialeného JVM. Týmto spôsobom sa dynamicky rozširuje správanie aplikácie.[16]

2.1.2 Vzdialené rozhrania, objekty a metódy

Distribuované aplikácie vytvorené pomocou RMI sa rovnako ako iné aplikácie v Jave skladajú z tried a rozhraní. Triedy implementujú metódy deklarované v rozhraniach. V distribuovaných aplikáciach sú niektoré implementácie umiestnené vo JVM. Objekty s metódami, ktoré môžu byť vyvolané cez JVM sa volajú vzdialené objekty.

Objekt sa stáva vzdialeným keď implementuje vzdialené rozhranie, ktoré má nasledujúce vlastnosti:

- Vzdialené rozhranie rozširuje rozhranie java.rmi.Remote, toto rozhranie slúži na identifikáciu rozhrania, ktorého metódy sa môžu vyvolať na vzdielenom JVM. Každý objekt, ktorý je vzdialený musí implementovať toto rozhranie. Iba metódy ktoré sú uvedené vo vzdialenom rozhraní sú prístupné na diaľku.
- Každá metóda rozhrania deklaruje java.rmi.RemoteException v klauzule throws.

RMI zaobchádza zo vzdialenými objektami rozdielne ako s lokálnymi objektami, keď objekt prejde z jedného JVM do druhého. Namiesto vytvorenia kópie implementovaného objektu na strane klienta v JVM, RMI pošle stub pre vzdialený objekt. Stub pôsobí ako lokálny zástupca pre vzdialený objekt, pre klienta je to odkaz na vzdialený objekt. Klient vyvolá metódu na lokálnom stube, ktorý je zodpovedný za vykonanie vyvolanej metódy na vzdialenom objekte. Stub implementuje na objekt rovnaké rozhranie ako má vzdialený objekt. Táto vlastnosť umožňuje stubu zmeniť rozhranie na také, ktoré je implementované na vzdialenom objekte. Iba metódy definované vo vzdialenom rozhraní sú prístupné na zavolanie zo strany klienta.[16]

2.1.3 Návrh a implementácia aplikácie

Definovanie vzdialeného rozhrania stanovuje metódy ktoré môžu byť na diaľku vyvolané klientom. Návrh týchto rozhraní zahŕňa určenie typov objektov, ktoré budú použité ako parametre a navratové hodnoty pre metódy. Objekty musia implementovať jedno alebo viac rozhraní. Objekt môže zahŕňať implementáciu iných rozhraní a metód ktoré sú prístupné iba lokálne. Klientské aplikácie ktoré používajú vzdialené objekty môžu vykonávať metódy až po definovaní vzdialeného rozhrania.[16]

2.1.4 Java RMI-Internet Inter-ORB Protocol

Java RMI-Internet Inter-ORB Protocol (RMI-IIOP) kombinuje vlastnosti technológie Java RMI s technológiou CORBA. Podobne ako Java RMI aj RMI-IIOP urychľuje vývoj distribuovaných aplikácií a dovoľuje vývojárom kompletne pracovať v jazyku Java.

RMI-IIOP podobne ako CORBA používa IIOP ako komunikačný protokol. S technológiou RMI-IIOP môžu vývojári vytvárať vzdialené rozhrania v jazyku Java a potom ich môžu používať s Java API. RMI-IIOP je pre vývojárov, ktorí chcú programy s RMI rozhraniami, ale chcú použiť IIOP ako základný komunikačný protokol.[17]

2.1.5 Java RMI výhody a nevýhody

RMI má významné vlastnosti ktoré CORBA nemá. Je to schopnosť poslať nové objekty (kód a dáta) v sieti pre iné JVM. RMI je dostupné od verzie JDK 1.02, takže veľa vývojárov je oboznámených s touto technológiou. Organizácie už zaviedli systémy založené na tejto technológii. Hlavnou nevýhodou je, že RMI je obmedzené iba na JVM a nemôže komunikovať s inými jazykmi.[1]

Výhody Java RMI

- Prenositeľná na veľa platforiem.
- RMI môže zaviesť nový kód do cudzích JVM.
- Veľa Java vývojárov už má skúsenosti s RMI.
- Existujúce systémy môžu používať RMI, náklady a čas potrebný na prevedenie na nový systém môžu byť príliš vysoké.[1]

Nevýhody Java RMI

- RMI funguje iba na platformách s podporou Javy.
- Bezpečnostné hrozby vzdialeného spúšťania kódu, obmedzenie funkčnosti presadzované bezpečnostnými pravidlami.
- Náročnosť naučenia sa RMI je pre vývojárov, ktorí nemajú žiadnu skúsenosť s RMI je porovnateľná ako s technológiou CORBA.
- RMI môže pracovať iba s Java systémami, nepodporuje staršie systémy napísané v jazykoch C++, Fortran, Cobol a iné.[1]

2.2 Common Object Request Broker Architecture

Object Managment Group (OMG) je organizácia ktorá vyvíja štandard Common Object Request Broker Architecture (CORBA). Bola založená v roku 1989 skupinou spoločností s cieľom spojiť dve technológie: vzdialené volanie procedúr a objektovú orientáciu. OMG vytvorila kompletnú infraštruktúru pre distribuované výpočty, Object Managment Architecture (OMA). CORBA špecifikácia, ktorá je hlavnou časťou OMA, opisuje základný princíp vytvárania a volania objektovo orientovaných vzdialených procedúr.

Rozhranie na CORBA objekty je definované pomocou Interface Definition Language (IDL). IDL je deklaratívny jazyk. Výhodou použitia IDL je, že je optimalizované tak aby sa prispôsobilo rôznym programovacím jazykom a syntax môže byť rozšírená podľa potrieb distribuovaných systémov.

CORBA na komunikácie cez sieť používa Internet Inter-ORB Protokol (IIOP). Je to štandard pre komunikáciu CORBA aplikácií cez TCI/IP. Hlavnou výhodou IIOP je nezávislosť medzi rôznymi vydavateľmi. Klient a server môžu byť vytvorený rôznymi vydavateľmi, stále je zaistená kompatibilita komunikácie medzi klientom a serverom. [2]

2.2.1 API Špecifikácia

API špecifikácia technológie CORBA pre programovací jazyk Java obsahuje 5 balíčkov. Programy vytvorené v Jave pomocou technológie CORBA musia implementovať tieto balíčky. Týchto 5 balíčkov umožňuje programom komunikovať medzi sebou, posielať a prijímať informácie a vytvárať programy ktoré slúžia ako klienti a serveri.

org.omg.CORBA

Poskytuje mapovanie CORBA API do Javy, obsahuje implementovanú triedu **ORB**, takže programátori môžu používať všetky funkcie Object Request Broker-u (ORB). Metóda ORB spracováva vyvolané metódy medzi klientom a serverom. Väčšina z toho, čo robí ORB je úplne transparentné pre užívateľov a hlavná časť balíčku CORBA sa skladá z tried požívanych ORB-om v pozadí. Výsledkom je, že väčšina programátorov využíva len malú časť z tohoto balíčku.[20]

org.omg.CosNaming

Tento balíček obsahuje dve verejné rozhrania a niekoľko pomocných tried. Verejné rozhrania:

1. NamingContext, poskytuje hlavné funkcie pre pomenovanie služby.

2. BindingIterator, poskytuje prostriedky na iteráciu zoznamu názvov alebo referencií objektov.

Tieto dve rozhrania poskytujú prostriedky na pripojenie a odpojenie mien a referencií na objekty.[21]

org.omg.PortableServer

Poskytuje triedy a rozhrania pre vytváranie serveru, zabezpečuje, aby boli prenosné medzi rôznymi vydávateľmi. Triedy v tomto balíčku, ktoré končia príponou PolicyValue poskytujú hodnoty použité pre volanie metódy *createPOA()*. Pomocné triedy ktoré sú generované pre všetky typy definované užívateľom v rozhraní IDL potrebujú statické metódy k manipulovaniu s týmyto typmi. Je iba jedna metóda medzi pomocnými metódami ktorú programátor používa, je to metóda *narrow()*. Iba Java rozhranie mapované z IDL rozhrania bude mať pomocnú triedu ktorá zahŕňa metódu *narrow()*, takže v tomto balíčku obsahujú metódu *narrow()* iba tieto triedy:

- ForwardRequestHelper
- ServantActivatorHelper
- ServantLocatorHelper

Portable Object Adaptor (POA) triedy slúžia na implementáciu ServantActivator alebo ServantLocator.

Väčšinu z toho čo robí balíček PortableServer je pre užívatela transparentné, takže programátori používajú len niektoré rozhrania z tohoto balíčku. Ostatné rozhrania zabezpečujú implementáciu ORB-u.[22]

org.omg.PortableInterceptor

Poskytuje mechanizmus pre registráciu ORB-u, vďaka čomu môže služba ORB zachytávať požiadavky alebo odpovede.

Sú tri typy zachytávačov ktoré môžu byť zaregistrované podľa [23]:

- IORInterceptor - Používa sa na vytvorenie označených komponentov v profiloch v rámci jednej Interoperable Object Reference (IOR).
- ClientRequestInterceptor - Zachytáva požiadavky alebo odpovede pomocou ORB-u na strane klienta.
- ServerRequestIntreceptor - Zachytáva požiadavky alebo odpovede pomocou ORB-u na strane serveru.

org.omg.DynamicAny

Tento balíček obsahuje triedy a rozhrania, ktoré umožňujú posielanie dát medzi klientom a serverom. Any hodnoty môžu byť dynamicky poslané a zostrojene z DynAny objektov. DynAny objekt je spojeny s hodnotou dát, ktoré odpovedajú hodnote vloženej do objektu Any. [24]

2.2.2 CORBA výhody a nevýhody

CORBA je dôležitá vo veľkých organizáciach, kde medzi sebou musí komunikovať veľa rozdielnych systémov. CORBA poskytuje prepojenie medzi jedným jazykom a platformou s inými jazykmi a platformami. CORBA tiež poskytuje zvýšenie výkonu oproti RMI.[1]

Výhody technológie CORBA

- Služby môžu byť vytvorené v rozdielnych jazykoch na rozdielnych platformách a môže sa k nim pristupovať pomocou rôznych jazykov, ktoré podporujú IDL mapovanie.
- S IDL je rozhranie oddelené od implementácie, vývojári môžu vytvárať rozdielné implementácie založené na rovnakom rozhraní.
- CORBA podporuje primitívne dátové typy a veľa dátových štruktúr.
- CORBA je ideálna na použitie so staršími systémami, dokáže zabezpečiť, že napísané aplikácie budú prístupné aj v budúcnosti.
- CORBA umožňuje ľahké prepojenie medzi objektami a systémami.
- Systémy vytvorené technológiou CORBA môžu ponúknuť lepší výkon.[1]

Nevýhody technológie CORBA

- Vytvorenie služieb vyžaduje použitie IDL. Implementácia a používanie služieb vyžaduje IDL mapovanie pre jazyk, v ktorom je vytvorené rozhranie.
- CORBA nepodporuje prenos objektov alebo kódu.
- Budúcnosť technológie CORBA je neistá, ak sa jej nepodarí dosiahnuť prijatie priemyslom a nebude pokračovať vývoj tak sa stane zastaralou.
- Špecifikácie technológie CORBA sa stále vyvíjajú.
- Všetky aplikácie nepotrebujú veľký výkon v reálnom čase, rýchlosť sa môže vymeniť za jednoduché použitie systému iba z Javy.[1]

2.3 Hessian

Hessian je komunikačný protokol ktorí vytvorila spoločnosť Caucho Technology, Inc a bol vydaný pod otvorenou (open source) licenciou. Hessian je kompaktný binárny protokol určený hlavne na komunikáciu medzi webovými službami, pretože knižnice protokolu Hessian majú veľkosť len stovky kilobajtov môže byť Hessian implementovaný aj na mobilných telefónoch. Pretože Hessian je binárny protokol je vhodný na odosielanie binárnych dát bez rozširovania protokolu ďalším príslušenstvom alebo ďalšími funkciami. Aplikácie komunikujúce cez Hessian vyžadujú iba dva balíčky:[30, 31]

1. com.caucho.hessian.server, tento balíček obsahuje triedy na vytváranie serverov ktoré komunikujú pomocou protokolu Hessian
2. com.caucho.hessian.client, tento balíček obsahuje triedy na vytváranie klientov ktorí komunikujú pomocou protokolu Hessian

Komunikáciu protokolu Hessian pomocou TCP protokolu zaisťuje trieda **HessianInput** a **HessianOutput**. Protokol Hessian môže byť implementovaný pomocou rôznych programovacích jazykov, napr. Java, C++, Python, PHP a iné.[30, 31]

Výhody protokolu Hessian

- Aplikácie komunikujúce cez Hessian môžu byť implementované v rôznych programovacích jazykoch, napr. Java, C++ a iné.
- Hessian je jazykovo nezávislý, podporuje tiež skriptovacie jyzyky napr. PHP.
- Hessian je jednoduchý protokol takže je jednoduché ho testovať.
- Podporuje prenos 8-bitových binárnych dát bez používania príloh.
- Podporuje šifrovanie a kompresiu dát.
- Hessian je vydaný pod otvorenou licenciou.
- Hessian môže byť implementovaný aj na mobilných telefónoch.
- Pri prenose dát je Hessian rýchlejší oproti protokolom ktoré sú založené na XML prenose dát.[32, 31]

Nevýhody protokolu Hessian

- Hessian má väčší čas odozvy pri prenose väčších dát (250 a viac prvkov v poli) oproti Java RMI.
- V porovnaní s technológiou CORBA, Hessian podporuje menej dátových typov. [33, 31]

2.4 Zhodnotenie

Z textu popísaného v kapitolách 1 a 2 bolo zistené, že na implementáciu klient-server aplikácie je lepšie použiť tučného klienta. Výhody tučného klienta sú hlavne v nezávislosti komunikácie medzi klientom a serverom, tučný klient dokáže vykonávať funkcie aj bez pripojenia na server. Nevýhody použitia tučného klienta sú väčšsie nároky na hardvér, softvér a nutnosť synchronizovať dáta užívateľov. V prípade tenkého klienta sa dáta vždy nachádzajú na servery. Vzhľadom na to, že väčšina dnes predávaných osobných počítačov alebo notebookov spĺňa požiadavky, ktoré musí tučný klient mať (schopnosť samostatne vykonávať úlohy), tak jedinou nevýhodou použitia tučného klienta je vyššia cena.

Pri porovnávaní technológií CORBA a Java RMI bolo zistené, že ako softvérovú implementáciu klient-server siete je lepšie použiť technológiu CORBA. Hlavnou výhodou použitia technológie CORBA je, že podporuje viac programovacích jazykov, server a klienti možu byť vytvorený pomocu rôznych programovacích jazykov, CORBA zabezpečuje komunikáciu medzi týmito aplikáciami. Táto vlastnosť sa využíva hlavne v organizáciach kde pracujú rôzne systémy vytvorené v rôznych jazykoch a je potrebné zabezpečiť vzájomnú komunikáciu medzi týmito sytémami. Pri použití Java RMI by museli všetky systémy, server aj klienti, byť vytvorený v jednom programovacom jazyku. Ďalšou výhodou je, že CORBA dokáže poskytnúť lepší výkon aplikácií ako Java RMI pri prenose veľkých objemov dát. CORBA potrebuje na prenos dát menej paketov ako Java RMI, to znamená, že komunikácia je rýchlejšia pri použití technológie CORBA. Nevýhodou použitia technológie CORBA je potreba naučiť sa IDL jazyk, ktorým sa vytvárajú rozhrania pre aplikácie. Vzhľadom na to, že IDL je deklaratívny jazyk, jeho naučenie nie je ťažké.

Pri porovnávaní komunikačných protokolov a technológií Hessian, RMI a CORBA bolo zistené že, Hessian má rovnako ako CORBA výhodu v podpore rôznych programovacích jazykoch, ale na rozdiel od technológie CORBA, Hessian podporuje aj skriptovacie jazyky, to znamená, že klienti a serveri komunikujúci pomocou protokolu Hessian nemusia byť vytvorený v rovnakom programovacom alebo skriptovacom jazyku. Ďaľšou výhodou Hessianu je možnosť implementácie tohoto protokolu na mobilných telefónoch oproti technológii CORBA. Hessian je vydaný s open source licenciou takže rôzne spoločnosti alebo jednotlivci si môžu protokol Hessian upraviť podľa seba tak aby im viac vyhovoval. Výhodou protokolu Hessian je, že je efektívnejší ako RMI pri prenose dát pretože má viac kompaktné kódovanie. Nevýhodou Hessianu je, že podporuje menší počet dátových typov ako CORBA a RMI.

3 SPRACOVANIE DÁT

Spracovanie dát je jednou z hlavných oblastí informačných systémom, využíva sa tiež vo vede, priemysle a v iných oblastiach. Medzi základné požiadavky na spracovanie dát patrí rýchlosť spracovávania dát. Táto požiadavka sa dá splniť použitím paralelného a distribuovaného spracovania dát. Táto kapitola sa venuje paralelnému a distribuovanému spracovaniu dát. Je tu vysvetlený paralelný model, základné podmienky pre použitie paralelného modelu a dva základné typy granularity ktoré sa využívajú v paralelnom modeli. V podkapitole ktorá sa venuje distribuovanému spracovaniu dát je vysvetlený distribuovaný model, základné body ktoré by mal každý distribuovaný systém spĺnať a rozdelenie distribuovaných systémov.

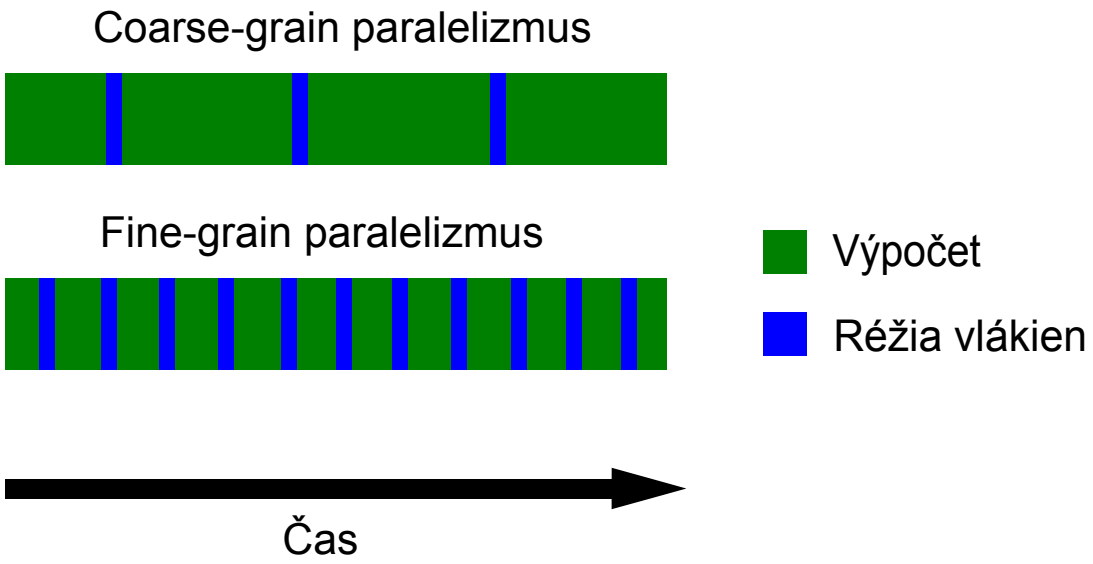
3.1 Paralelný model

V minulosti bol tradičný softvér vytvorený pre sériové spracovávanie informácií v jedno-jadrových procesoroch. V jednom momente procesor spracováva iba jednu inštrukciu, po jej dokončení začne s daľšou. Takýto spôsob nie je vhodný na vykonávanie veľkého počtu rovnakých inštrukcií, výsledok z jednej inštrukcie neovplyvňuje výsledok daľšej. S nástupom viac-jadrových procesorov sa prešlo na paralelné spracovávanie inštrukcií v procesore. Každé jadro procesoru môže spracovávať navzájom na sebe nezávislé inštrukcie čo výrazne zlepší rýchlosť aplikácií. Tento spôsob spracovávania informácií sa nazýva paralelný model.[29]

Základnou podmienkou použitia paralelného modelu je možnosť rozdeliť daný problém na menšie časti, ktoré sa budú jednotlivo spracovávať na rôznych jadrách procesoru. Pojem paralelizácia označuje proces, pri ktorom sa vykonáva viacej operácií zároveň. Hlavným účelom paralelizácie a použitia paralelného modelu je zredukovanie času potrebného na dokončenie výpočtu a efektívnejšie využitie viac jadrových procesorov. Paralelizácia prebieha tak, že rôzne programy bežia v rôznych vláknach, vlákna predstavujú nezávisle bežiace úlohy. Druhý spôsob paralelného spracovávania informácií je keď jeden procesor spracováva striedavo dve alebo viac vlákien, tento spôsob neprináša zlepšenie výkonu pretože procesorový čas je rozdelený pre viacero vlákien a procesor prepína medzi vykonávaním týchto vlákien.[29][36]

Pri návrhu paralelných programov je dôležité, aby bola záťaž vlákien rovnomerne rozdelená na všetky jadrá procesoru. V prípade neefektívneho rozdelenia záťaže musia vlákna ktoré dokončili výpočet čakať na najpomalšie vlákno, čím sa zhorší celkový výkon a efektivita programu. Ďalším dôležitým parametrom pri návrhu paralelného programu je pomer výpočtu a komunikácie, tento parameter sa tiež nazýva granularita (angl. granularity). Sú dva základné typy granularity pri paralelných výpočtoch:

- Fine-grain (jemná granularita), má nízky pomer výpočtov ku komunikácii, časti ktoré sa majú spracovať sú malé takže uľahčuje rovnomerné rozloženie záťaže vo vláknach. Tento spôsob má vysokú komunikačnú réžiu a v prípade, že časti ktoré sa majú spracovať sú až príliš malé môže nastať stav, kedy komunikácia a réžia vlákien trvá dlhšie ako samotný výpočet.[29]
- Coarse-grain (hrubá granularita) má vysoký pomer výpočtov ku komunikácii, časti ktoré sa majú spracovať sú väčšie. Tento spôsob má malú komunikačnú réžiu ale preto že časti na spracovanie sú väčšie je ťažšie rovnomerne rozdeliť záťaž rovnako na všetky vlákna.[29]



Obr. 3.1: Porovnanie Fine-grain a Coarse-grain paralelizovaného modelu.

Paralelné spracovávanie dát má široké využitie vo vede, medicíne, priemysle a v daľších oboroch kde je potreba spracovávať veľké množstvo dát, paralelné spracovávanie sa často využíva spoločne s distribuovaným modelom.

3.2 Distribuovaný model

Definícia distribuovaného modelu alebo systému: Distribuovaný systém je kolekcia nezávislých počítačov ktoré sa svojim užívateľom zobrazujú ako jeden koherentný systém. Dôležitými aspektami distribuovaného systému je, že distribuovaný systém sa skladá z počítačov ktoré sú navzájom nezávislé a môžu pracovať samostatne. Druhým aspektom je, že užívateľ si myslí, že pracuje s jedným systémom, to znamená, že všetky komponenty distribuovaného systému musia spolupracovať.[34][35]

Distribuovaný systém musí spĺnať niekoľko hlavných bodov:

- Prístupné zdroje, hlavným cieľom distribuovaného systému je uľahčiť užívatelom a aplikáciam prístup k vzdialeným zdrojom a zdielať ich efektívnym a kontrolovaným spôsobom.
- Dsitribuovaná transparentnosť, dôležitou vlastnosťou distribuovaného systému je ukryť fakt, že procesy a zdroje sú fyzicky rozdelené na počítačoch v distribuovanom systéme.
- Otvorenosť, otvorený distribuovaný systém je systém ktorí ponúka služby podľa štandardných pravidiel.
- Škálovateľnosť, systém môže byť škálovatelný podľa počtu užívateľov alebo počtu zdrojov, čo znamená, že je jednoduché pridávať a odoberať užívateľov a zdroje zo systému.[34]

Typy distribuovaných systémov:

- Distribuovaný výpočtový systém, využíva sa pre vysoký výpočtový výkon, tento systém sa ešte rozdeľuje na dva podsystémy:
  - Klastrový výpočtový systém, tento systém sa stal populárnym vďaka dobrému pomeru ceny a výkonu systému. V tomto systém beží jeden program paralelne na viacerých počítačoch. Charakteristickým rysom klastrových systémov je homogenita, vo väčšine prípadov sú v takomto systéme všetky počítače rovnaké, majú rovnaký operačný systém a sú pripojené v rovnakej sieti.
  - Gridový výpočtový systém, základným rysom gridového systému je heterogenita. V tomto systéme majú počítače rôzny hardvér, operačný systém, sú pripojené cez rôzne siete. Typické zdroje gridových systémov sú výpočtové servery (môžu byť klastrového typu), úložné zariadenia, v špeciálnych prípadoch môžu byť pripojené zariadenia ako napríklad vesmírne ďalekohlady. Hlavným cieľom gridových systémov je spolupráca rôznych organizácií.[34]
- Distribuovaný informačný systém, je ďalším dôležitým distribuovaným systémom ktorí sa nachádza v organizáciách ktoré pracujú s množstvom sieťových aplikácií. V týchto systémoch sú rôzne komponenty od seba oddelené ale potrebujú medzi sebou komunikovať a spolupracovať, napríklad komponenty databázy sú oddelené od komponentov ktoré spracúvajú dáta.[34]
- Distribuovaný pervazívny (všadeprítomný) systém, charekteristickým znakom distribuovaného výpočtového a informačného systému je stabilita vďaka kvalitnému sieťovému pripojeniu, charakteristyckým znakom pervazívneho systému je nestabilita zapríčinená tým, že v tomto systéme sú zariadenia ktoré majú

malé batérie, iba bezdrátové pripojenie, často sa jedná o mobilné telefóny.[34]

Distribuované systémy majú veľké využitie v telekomunikačných sieťach, telefónych a počítačových sieťach, sieťových apikáciách. Distribuované výpočty sa využívaju na vedecké a iné výpočty, najznámejší distribuovaný systém je Berkeley Open Infrastructure for Network Computing (BOINC) ktorí sa často používa na distribuované výpočty.

4 GENETICKÝ ALGORITMUS

Genetický algoritmus (Obr. 4.1) patrí do triedy evolučných algoritmov. Do triedy evolučných algoritmov ďalej patrí evolučné programovanie, evolučné stratégie a genetické programovanie. Evolučné algoritmy sú vyhľadávacie algoritmy založené na mechanizme prirodzeného výberu a princípoch genetiky, tieto princípi sú prevzaté z Darwinovej teórie a aplikované v informatike.[26]

Podľa Darwinovej teórie [28] v prírode v jednej populácii dochádza k boju jedincov o prežitie. Šancu na prežitie majú silnejší jedinci s lepšími vlastnosťami. Tento proces sa volá prirodzený výber. V informatike sa tento proces vykonáva pomocou tzv. fitness funkcie, jedná sa o funkciu ktorá hodnotí jednotlivých jedincov v populácii. Podľa hodnoty fitness funkcie sa v daľších krokoch robí výber jedincov pre novú populáciu, jedinci s vyššou hodnotou fitness funkcie majú väčšiu pravdepodobnosť výberu do novej populácie a jedinci s malou hodnotou fitness funckie majú malú pravdepobnosť výberu do novej populácie. V informatike sa vätšinou používa funkcia elitizmu, je to funkcia ktorá vždy skopíruje určitý počet jedincov do novej populácie s najvyššou hodnotou fitness funckie, čím je zaistené, že hodnota najlepšieho jedinca v populácii bude mať rastúci trend.[26]

Podľa Darwinovej teórie sa na populáciu uplatňujú dalšie procesy, je to náhodný genetický drift a reprodukčný proces. Náhodný genetický drift môžeme chápať ako náhodné javy u jedincov, ktorí ovplyvňujú daľšiu populáciu. Môže to byť náhodná mutácia génov jedinca alebo náhodná smrť jedinca pred reprodukciou. V informatike sa tento proces vykonáva funkciou, ktorá náhodne mení (mutuje) gény jedincov. V prípade, že by jednotlivé gény jedinca reprezentovali binárne čísla, tak by operátor mutácie znegoval náhodný gén chromozómu. Tabuľka 4.1 znázorňuje mutáciu chromozómu, v ktorom jednotlivé gény predstavujú binárne čísla, šedou farbou je vyznačený pôvodný a zmutovaný gén.[26]

Pôvodný chromozóm	1	0	1	0
Zmutovaný chromozóm	1	1	1	0

Tab. 4.1: Mutácia chromozómu

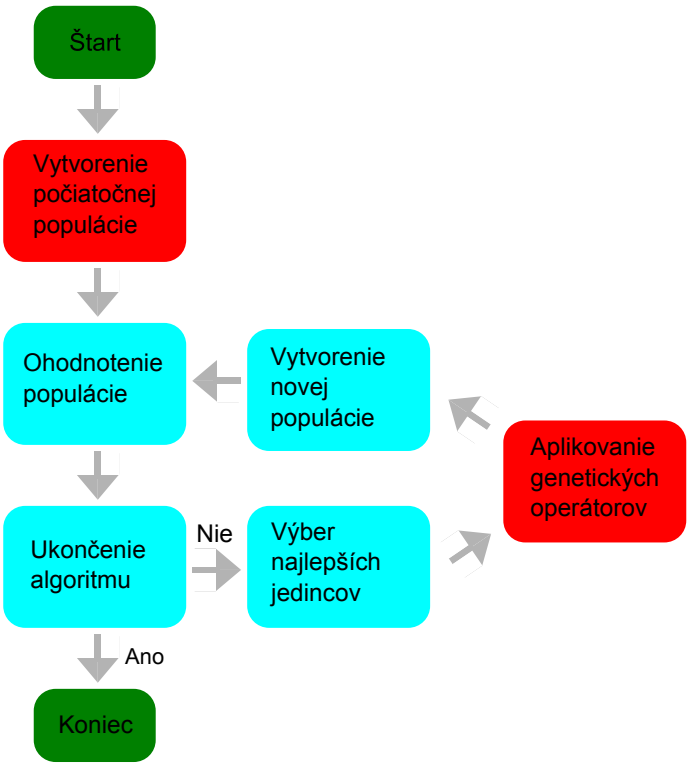
Posledým procesom, ktorý sa v prírode uplatňuje je reprodukčný proces. V tomto procesi dochádza k vytváraniu potomkov z rodičov. Potomkovia dedia gény po svojich rodičoch, tento proces sa vykonáva tak, že z každého rodiča sa náhodne vyberú gény, ktoré bude potomok dediť. V informatike tento proces vykonáva funkcia, ktorá náhodne robí kríženie génov dvoch jedincov, pri jedno-bodovom krížení sa náhodne

vygeneruje bod kríženia a podľa tohoto bodu sa z dvoch rodičov vytvoria dvaja potomkovia. Pri viac bodovom krížení sa vygeneruje viac bodov kríženia. Tabuľka 4.2 zobrazuje kríženie dvoch chromozómov, bod kríženia je medzi dvomy farbami, červená časť chromozómu od rodiča 1 a modrá časť chromozómu od rodiča 2 sa pri vytváraní potomkov vymenia čím sa vytvoria dva nové chromozómy.[26]

Rodič 1	1	1	0	0	0	1
Rodič 2	0	0	0	1	1	0
Potomok 1	1	1	0	1	1	0
Potomok 2	0	0	0	0	0	1

Tab. 4.2: Kríženie chromozómov

V teórii umelej inteligencie je genetický algoritmus proces postupného vylepšovania jedincov opakovaným aplikovaním genetických operátorov. Tento proces vedie k vytvoreniu populácie, ktorá najviac vyhovuje podmienkam, ktoré boli stanovené na začiatku algoritmu. Základným princípom genetických algoritmov je kopírovanie a vymienanie chromozómov. Chromozóm je skupina génov s pevne danou dĺžkou ktorý často reprezentuje jednotlivých jedincov v populácii.



Obr. 4.1: Všeobecný vývojový diagram genetického algoritmu.

Genetické algoritmy sa v praxi používajú na riešenie úloh optimalizácie, ďalej sa využívajú v technológii, výrobe a v priemyselnej automatizácii a ako alternatívne metódy učenia neurónových sietí. Uplatnenie genetických algoritmov je pri riešení problémov, ktorých riešenie sa nedá nájsť deterministickými a lineárnymi metódami alebo keď je priestor riešenia moc veľký.[25][26]

4.1 Problém naplnenia batohu

V tejto práci je ako praktická aplikácia genetického algoritmu spravený program, ktorý rieší problém naplnenia batohu. Jedná sa o problém v ktorom je potrebné naplniť batoh predmetami s rôznou cenou a hmotnosťou. Batoh má maximálnu nosnosť ktorá sa nemôže prekročiť, výsledný zoznam predmetov musí mať čo najväčšiu hmotnosť a zároveň čo najväčšiu cenu.

Jednotlivé predmety predstavujú gény v genetickom algoritme. Zoznamy predmetov s ktorými program pracuje tvoria chromozómy. Zoznamy chromozómov tvoria jednotlivé populácie, ktoré sa postupne vyvíjajú. Na začiatku programu sa náhodne vygeneruje populácia. Potom sa skontroluje či niektorí chromozóm spĺňa podmienky ukončenia algoritmu. Keď je chromozóm vyhovujúci algoritmus sa ukončí, ak nieje, na súčastnú populáciu sa aplikujú genetické operátory.

Ako prvý sa aplikuje elitizmus, ktorý skopíruje určitý počet najlepších chromozómov do ďalšej populácie. Ako ďalšia sa aplikuje mutácia, náhodne vybraté predmety sa zmutujú, takým spôsob, že keď sa nachádzajú v batohu tak sa z neho odstránia a naopak. Posledný genetický operátor, ktorý sa aplikuje je kríženie. Pred krížením sa vygeneruje bod kríženia a podľa tohoto bodu sa spraví kríženie dvoch chromozómov. Mutácia a kríženie sa aplikujú s určitou pravdepodobnosťou, na tomto nastavení závisí ako rýchlo bude algoritmus konvergovať k najlepšiemu riešeniu. Po aplikovaní genetických operátorov sa vytvorí nová populácia z potomkov a znovu sa skontroluje či niektorí chromozóm vyhovuje podmienkam ukončenia algoritmu. Algoritmus sa ukončí keď sú splnené podmienky a nájde sa vyhovujúcy chromozóm alebo keď sa dosiahne maximálneho počtu evolučných krokov.[26]

Trieda **Execute** slúži na spustenia programu, obsahuje metódu *main()* v ktorej sa vytvorí inštancia triedy **Evolver**, do tohoto objektu sa na začiatku vloží niekoľko predmetov, pri vkladaní predmetov sa zadávajú informácie či je predmet v batohu (na začiatku vždy nie je), názov predmetu, hmotnosť a cena. Z tejto triedy sa volá metóda *evolve()* v ktorej prebieha evolučný proces.

Trieda **Evolver** vykonáva evolučný proces v metóde *evolve()*. V tejto triede sa aplikujú genetické operátory v metóde *createNewPopulation()*, ktorá volá jednotlivé

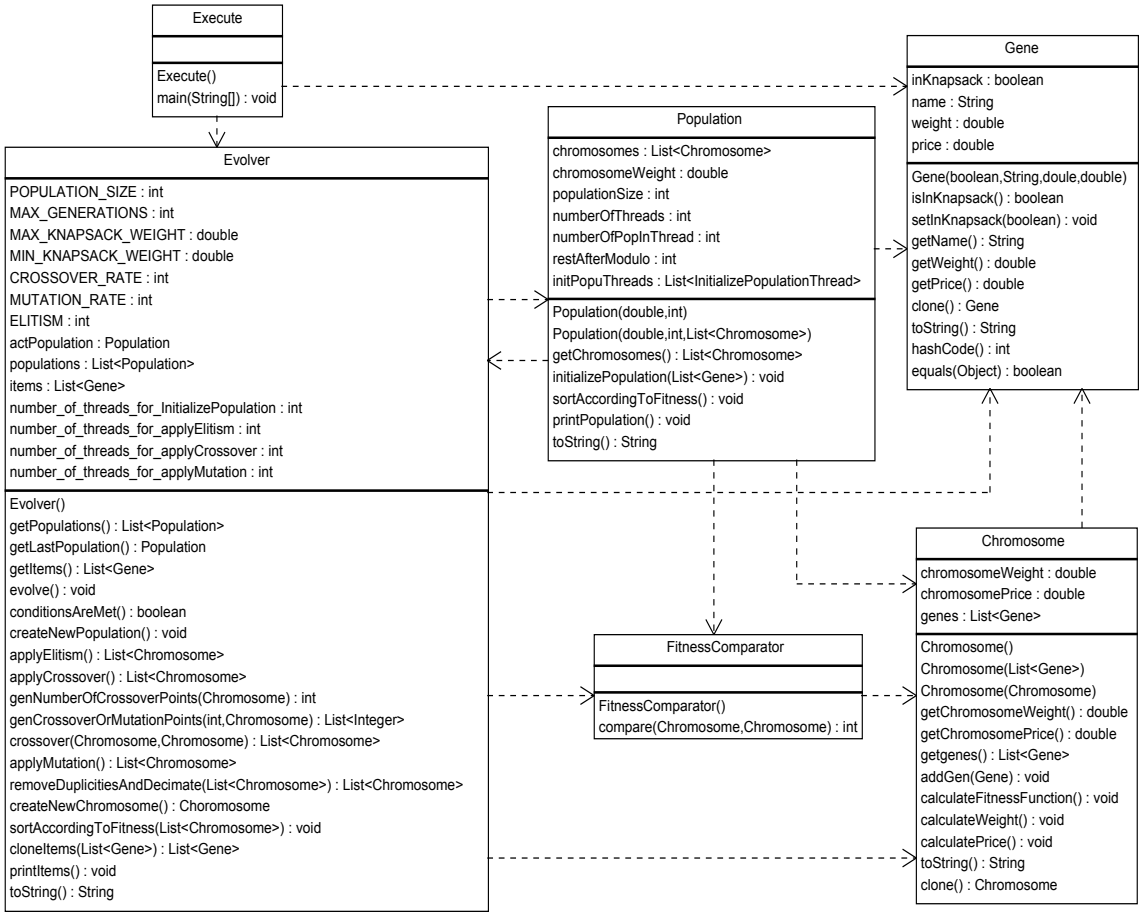
metódy na aplikáciu genetických operátorov.

Trieda **Population** obsahuje zoznam chromozómov, ktoré sa počas evolučných cyklov vyvíjajú. V tejto triede je metóda *InitializePopulation()*, ktorá náhodne vygeneruje počiatočnú populáciu o zadanej veľkosti.

Trieda **Gene** predstavuje gén chromozómu, v tomto prípade to sú jednotlivé predmety ktoré sa vkladajú do batohu.

Trieda **Chromosome** predstavuje zoznam génov, každý chromozóm predstavuje jednu variantu naplnenia batohu. V tejto triede prebieha výpočet fitness funkcie, ktorá počíta cenu predmetov v batohu, ideálna hodnota je maximálna cena.

Trieda **FitnessComparator** implementuje rozhranie Comparator<Chromosome>, metóda *compare()* určuje, ktorý z objektov predstavujúce chromozómy je vätší, menší alebo sú si rovné. V Obr. 4.2 sú zobrazené jednotlivé triedy v genetickom algoritme a ich závislosti, ktorá trieda ktorú volá.



Obr. 4.2: Diagram tried pôvodného genetického algoritmu.

4.2 Paralelizácia genetického algoritmu

Paralelizácia algoritmu, ktorý rieši problém naplnenia batohu je spravená na troch miestach, Obr. 4.1 znázorňuje vývojový diagram s paralelizovanými časťami označenými červenou farbou.

Trieda **InitializePopulationThread** slúži na paralelizáciu metódy *initializePopulation()*, ktorá na začiatku algoritmu vygeneruje zadaný počet populácií. Táto metóda vytváranie populácie paralelizuje do zadaného počtu vlákien tak, že v každom vlákne sa vytvára menší počet populácií a nakoniec sa z nich vytvorí jedna populácia o zadanej veľkosti. Trieda **InitializePopulationThread** obsahuje metódu *run()*, v ktorej prebieha samotné vytváranie populácie a metódu *getChromosomes()* pomocou, ktorej vlákno vráti vytvorenú populáciu do hlavného vlákna kde prebieha evolučný proces. Ako vstupné parametre pre vytvorenie vlákna je použitý zoznam génov z ktorých sa vytvára populácia, maximálna hmotnosť chromozómu a veľkosť populácie ktorá sa má vytvoriť vo vlákne.

Trieda **ApplyMutationThread** slúži na paralelizáciu metódy *applyMutation()*, táto metóda aplikuje mutáciu na populáciu. Je paralelizovaná rovnako ako metóda *applyElitism()*. Na začiatku sa vytvorí zadaný počet vlákien a v každom vlákne sa aplikuje mutácia na určitý počet chromozómov. Trieda **ApplyMutationThread** obsahuje metódy *run()* v ktorej prebieha mutácia chromozómov, metódu *genCrossoverOrMutationPoints()* v ktorej sa generujú body v ktorých sa zmutuje chromozóm a metódu *getOffsprings()*ktorá slúži na vrátenie potomkov po mutácii do hlavného vlákna. Ako vstupné parametre pri vytváraní vlákna je použité pole chromozómov na ktoré sa bude aplikovať mutácia a mutation rate, ktorí určuje s akou pravdepodobnosťou sa bude aplikovať mutácia.

Trieda **ApplyCrossoverThread** slúži na paralelizovanie metódy *applyCrossover()* ktorá aplikuje kríženie na populáciu a je paralelizovaná rovnako ako metóda *applyMutation()*. Na začiatku sa vytvorí zadaný počet vlákien a v každom vlákne sa aplikuje kríženie na malú časť populácie. Trieda **ApplyCrossoverThread** je podobná ako trieda **ApplyMutationThread** ale obsahuje navyše metódy *crossover()* a *genNumberOfCrossoverPoints()*. Sú to pomocné metódy na aplikovanie kríženia.

4.3 Distribúcia genetického algoritmu

Distribúcia genetického algoritmu na servery je spravená pomocou dvoch tried pre CORBA model (**InitializePopulationCORBA** a **ApplyMutationAndCrossoverCORBA**) a dvoch pre Hessian model (**InitializePopulationHessian** a **ApplyMutationAndCrossoverHessian**). Všetky tieto triedy rozširujú triedu **Thread**. Triedy pre inicializáciu populácie a aplikovanie mutácie a kríženia majú rovnakú

funkciu ale sú rozdelené pre rôzne technológie. Distribúcia genetického algoritmu je spravená pomocou vlákien, každé vlákno komunikuje s jedným serverom.

Trieda **InitializePopulationCORBA** a **InitializePopulationHessian** slúži na distribúciu inicializácie populácie na servery. Obsahuje dve metódy, metóda *getChromosomes()* slúži na vrátenie vytvorených chromozómov do hlavného vlákna a v metóde *run()* sa na serveri zavolá metóda *initiliazePopulation()*. Samotná distribúcia inicializácie populácie je spravená rovnako ako v paralelnom modeli, zadaná veľkosť populácie ktorá sa má vytvoriť sa rozdelí na menšie časti, každá menšia časť populácie sa potom inicializuje na jednotlivých serveroch.

Trieda **InitializePopulationHessian** a **ApplyMutationAndCrossoverHessian** slúži na aplikáciu genetických operátorov mutácie a kríženia na servery. Táto trieda obsahuje dve metódy, metóda *getChromosomes()* slúži na vrátenie chromozómov do hlavného vlákna, v metóde *run()* sa na serveri zavolá metóda *applyMutationAndCrossover* ktorá aplikuje operátory mutácie a kríženia na chromozómy. Aplikovanie mutácie a kríženia na serveroch sa robí podobne ako inicializácia populácie, populácia sa rozdelí na menšie časti, tie sa pošlú na servery na ktorích sa aplikuje mutácia a kríženie a vytvorení potomkovia sa vrátia späť klientovi.

5 VÝSLEDKY PRÁCE

V tejto časti práce sú vysvetlené časti zdrojového kódu pomocou ktorích je spravená paralelizácia a distribúcia genetického algoritmu. V tabuľkách sú zobrazené namerané časy genetického algoritmu pre paralelný a distribuovaný model. Distribuovaný model je implementovaný pomocou technológií CORBA a Hessian.

5.1 Meranie genetického algoritmu

Meranie genetického algoritmu sa robilo pre veľkosť populácie 100, 1 000, 10 000, 20 000 a 50 000 jedincov. Meranie paralelného modelu sa robilo pre jedno, dve, štyri, osem a šestnásť vlákien. Meranie distribuovaného modelu sa robilo na 6 počítačoch, jeden počítač slúžil ako klient, ostatné slúžili ako servery. Pri meraní distribuovaného modelu bolo na serveroch použité paralelné spracovanie na štyroch vláknach.

Meranie paralelného a distribuovaného modelu bolo robené na počítači s dvojjadrovým procesorom Intel i5 2500 2,3GHz s podporou spracovávania maximálne štyroch vlákien naraz a s 64-bitovým operačným systémom Microsoft Windows 7 Profesional a veľkosťou operačnej pamäte 8 GB.

Meranie prebiehalo pomocou triedy ktorá opakovane spúšťala genetický algoritmus a zapisovala namerané časy. Meranie sa robilo 10 krát pre paralelný aj pre obi dve technológie (CORBA a Hessian) distribuovaného modelu a pre veľkosti populácie 100, 1 000, 10 000, 20 000 a 50 000 jedincov. Meranie času sa robilo pomocou systémového času, ktorí sa na začiatku genetického algoritmu zapísal do jednej premennej, na konci sa čas zapísal do druhej a výsledný čas sa vypočítal ako rozdiel týchto časov.

Maximálny počet generácií	20
Maximálna hmotnosť predmetov v batohu	5 kg
Minimálna hmotnosť predmetov v batohu	4,8 kg
Minimálna cena predmetov v batohu	21 200 CZK
Pomer kríženia	30 %
Pomer mutácie	10 %
Hodnota elitizmu	5

Tab. 5.1: Nastavenie parametrov genetického algoritmu.

5.2 Paralelný model GA

Paralelizácia genetického algoritmu je spravená na troch miestach, inicializácia počiatočnej populácie, aplikovanie mutácie a kríženia. V tejto podkapitole je vysvetlená paralelizácia inicializácie populácie a tabuľke 5.2 sú namerané časy genetického algoritmu pre paralelný model. Paralelizácia aplikácie mutácie a kríženia je spravená rovnako ako inicializácia populácie.

Zdrojový kód inicializácie počiatočnej populácie:

```
chromosomes = new Chromosome[populationSize];
int numberOfPopInThread = 0;
int restAfterModulo = 0;
List<InitializePopulationThread> initPopuThreads =
  new ArrayList<InitializePopulationThread>();
```

Inicializácia premenných ktoré sa používajú pri inicializácii populácie.

```
switch( Configuration.useLocal ){
  case 0:{
    if(numberOfThreads <= populationSize){
      restAfterModulo = populationSize % numberOfThreads;
      numberOfPopInThread = (populationSize - restAfterModulo) /
        numberOfThreads;
```

Keď sa premenná `useLocal` rovná nule tak sa populácia inicializuje na počítači na ktorom sa spustil genetický algoritmus. Počítanie premenných `restAfterModulo` a `numberOfPopInThread` pomocou ktorých sa inicializácia populácie rozdelí do vlákien.

```
      if(restAfterModulo != 0){
        for(int i = 0; i < numberOfThreads; i++){
          if(restAfterModulo != 0){
            initPopuThreads.add(new InitializePopulationThread(items,
              chromosomeWeight, numberOfPopInThread + 1));
            restAfterModulo--;
            initPopuThreads.get(i).start();
          }else {
            initPopuThreads.add(new InitializePopulationThread(items,
```

```
            chromosomeWeight, numberOfPopInThread));
          initPopuThreads.get(i).start();
        }
      }
```

Keď sa premenná `restAfterModulo` nerovná nule tak rozdelenie populácie vo vláknach nebude rovnomerné. V niektorých vláknach sa bude vytvárať väčšia časť populácie. Vytvorenie vlákna so vstupnými parametrami a spustenie vlákna.

```
    }else{
      for(int i = 0; i < numberOfThreads; i++){
        initPopuThreads.add(new InitializePopulationThread(items,
          chromosomeWeight, numberOfPopInThread));
        initPopuThreads.get(i).start();
      }
    }
```

Keď sa premenná `restAfterModulo` rovná nule tak rozdelenie populácie vo vláknach bude rovnomerné. V každom vlákne sa bude vytvárať rovnaká časť populácie. Vytvorenie vlákna so vstupnými parametrami a spustenie vlákna.

```
  }else{
    numberOfThreads = populationSize;
    numberOfPopInThread = populationSize / numberOfThreads ;
    for(int i = 0; i < numberOfThreads; i++){
      initPopuThreads.add(new InitializePopulationThread(items,
        chromosomeWeight, numberOfPopInThread));
      initPopuThreads.get(i).start();
    }
  }
```

Keď je počet vlákien väčší ako veľkosť populácie tak sa počet vlákien nastaví na veľkosť populácie. Ďalej sa vytváranie populácie robí rovnako ako v predchádzajucich krokoch.

```
  int k = 0;
  for(int i = 0; i < numberOfThreads; i++){
    try {
      initPopuThreads.get(i).join();
```

```
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      for(int o = 0; o < initPopuThreads.get(i).getChromosomes().
        length; o++){
        chromosomes[k++] = new Chromosome(initPopuThreads.get(i).
          getChromosomes()[o]);
      }
    }
    initPopuThreads = null;
  }
  break;
```

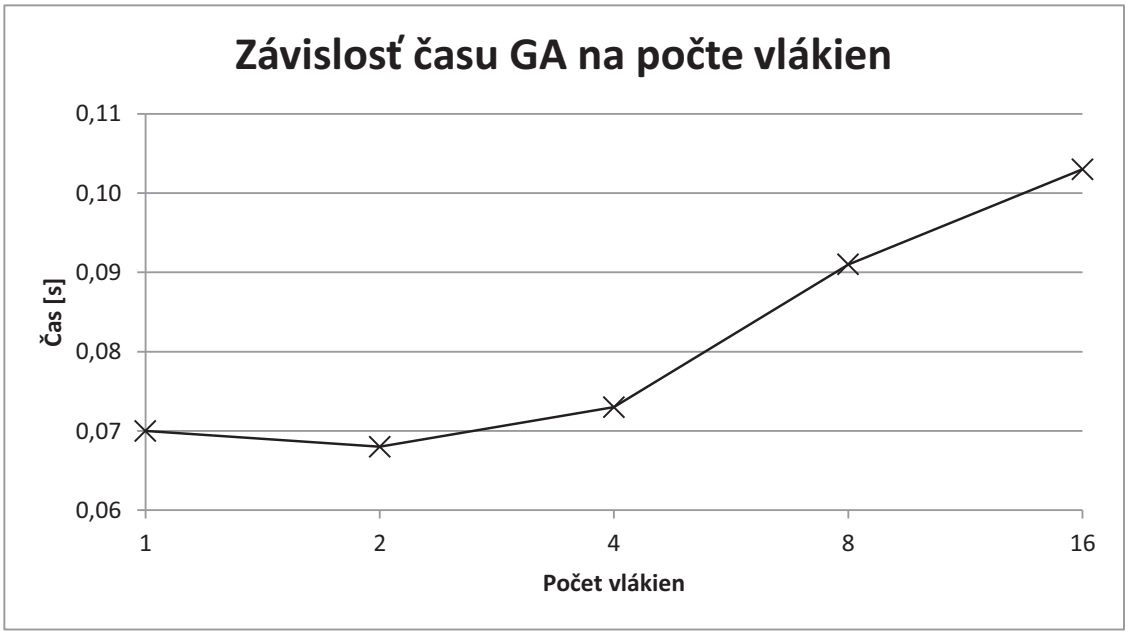
Program počká na ukončenie vytvárania populácii vo vláknach a potom z jednotlivých vlákien zkopíruje chromozómy do premennej `chromosomes`. Na konci vytvárania populácie sa do premennej `initPopuThreads` ktorá obsahovala vlákna v ktorích sa vytvárala populácia zapíše hodnota null, potom môže Garbage Collector (GC) uvoľniť použitú pamäť.

5.2.1 Meranie paralelného modelu

Veľkosť populácie	100	1 000	10 000	20 000	50 000
	Čas [s]				
Jednovlákonový model	0,070	0,521	6,285	16,003	62,427
Paralelný model: 2 vlákna	0,068	0,502	5,550	12,836	43,539
Paralelný model: 4 vlákna	0,073	0,505	5,320	12,356	37,532
Paralelný model: 8 vlákien	0,091	0,516	5,464	12,422	37,745
Paralelný model: 16 vlákien	0,103	0,549	5,457	12,458	38,064

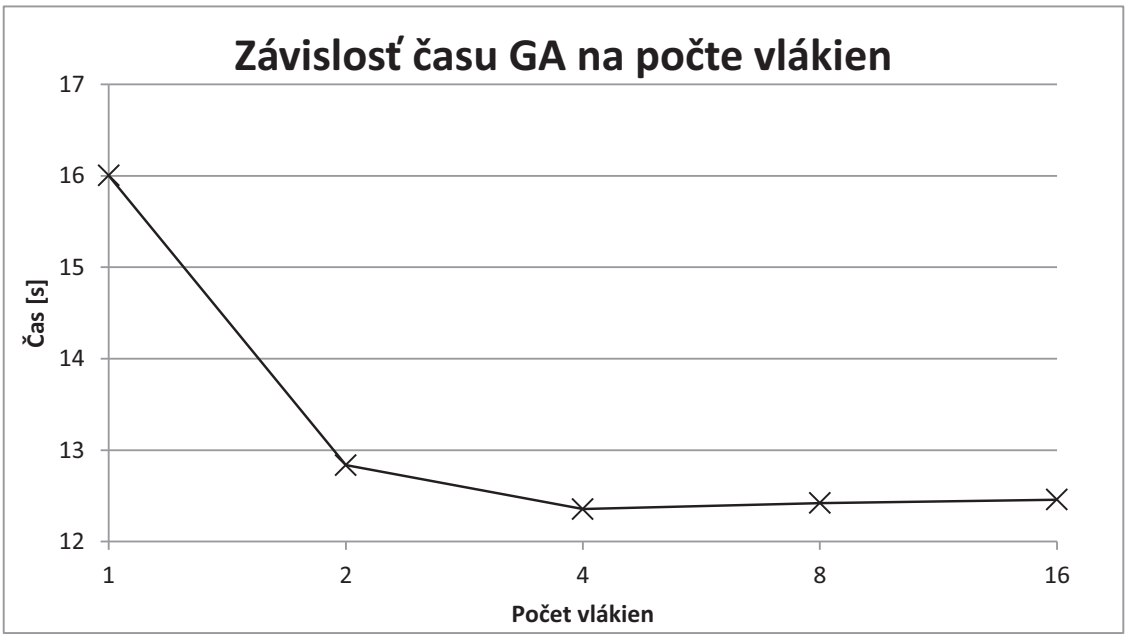
Tab. 5.2: Namerané hodnoty genetického algoritmu pre paralelný model

Tab. 5.2 zobrazuje namerané časy genetického algoritmu pre paralelný model. Pre veľkosť populácie 100 jedincov sa čas zlepšil len pre dve vlákna, pri použití viac vlákien bol čas väčší, je to spôsobené tým, že réžia vlákien trvá príliš dlho v pomere k výpočtu. Na obr.5.1 je graf závislosti času genetického algoritmu na počte vlákien pre 100 jedincov. Na obrázku je vidieť, že pri dvoch vláknach sa čas zlepšil ale ďalším prídavaním vlákien sa čas kvôli réžii vlákien zvyšoval. Pre veľkosť populácie 1 000 jedincov sa podarilo zredukovať čas pri použití dvoch, štyroch a ôsmich vláknach, pre šestnásť vlákien bol čas horší. Pri použití dvoch vlákien bol čas lepší o 3,07 %.



Obr. 5.1: Závislosť času GA na počte vlákien (veľkosť populácie 100)

Pri ôsmich vláknach bol čas horší kvôli zvýšenej réžii vlákien. Pre veľkosť populácie 10 000 jedincov sa podarilo čas zredukovať pri použití dvoch až šestnástich vlákien, najlepší čas bol pri štyroch vláknach o 15,35 %, pri ôsmich a šestnástich vláknach čas mierne stúpol. Pre veľkosť populácie 20 000 jedincov sa podarilo čas zredukovať pre všetky počty vlákien. Najlepší čas bol pri použití štyroch vlákien, oproti jednovláknovému spracovaniu sa čas zlepšil o 22,78 %. Na obr.5.2 je závislosť času genetického algoritmu na počte vlákien pre 20 000 jedincov. Na obrázku je vidieť ako sa čas zredukoval pre dve a štyri vlákna, ďalšie zvyšovanie počtu vlákien nemá žiadny vplyv na ďalšie zredukovanie času. S narastajúcim počtom vlákien čas mierne stúpol kvôli vyššej réžii vlákien. Pri veľkosti populácie 50 000 jedincov sa podarilo čas zredukovať pre všetky počty vlákien. Najlepší čas bol pri spracovaní v štyroch vláknach, precentuálne zlepšenie bolo 39,87 %. Pri ôsmich a šestnástich vláknach čas stúpol kvôli vyššej réžii vlákien.



Obr. 5.2: Závislosť času GA na počte vlákien (veľkosť populácie 20 000)

5.3 Distribuovaný model GA

Distribuovaný model genetického algoritmu je implementovaný pomocou technológie CORBA a Hessian. V genetickom algoritme sa inicializácia počiatočnej populácie, mutácia a kríženie spracúva na serveroch. V tejto podkapitole je vysvetlená inicializácia počiatočnej populácie na serveroch a v tabuľkách 5.3, 5.4 sú namerané časy genetického algoritmu pre dva distribuované modely (CORBA a Hessian). Aplikovanie mutácie a kríženia je spravené rovnako ako inicializácia populácie.

Zdrojový kód inicializácie populácie pre distribuovaný model CORBA:

```
case 1: {
int totalCores = 0;
for(int i = 0; i < Configuration.listOfDevices.size(); i++){
  totalCores += Configuration.listOfDevices.get(i).
    getNumberOfCores();
}
int resAfMo = populationSize % totalCores;
int numOfPopOnDevice = (populationSize - resAfMo) / totalCores;
List<InitializePopulationCORBA> initializePopulationCorbaList =
  new ArrayList<InitializePopulationCORBA>();
```

V prípade, že sa premenná `useLocal` rovná jednej tak sa inicializácia počiatočnej populácie bude vykonávať pomocou distribuovaného modelu CORBA. Do premennej `totalCores` sa uloží celkový počet jadier. Počítanie premenných `numOfPopOnDevice` a `resAfMo` pomocou ktorích sa rozdeluje inicializácia populácie na servery. Inicializácia zoznamu vlákien `initializePopulationCorbaList`, každé vlákno v tomto zozname komunikuje z jedným serverom.

```
if( resAfMo == 0 ){
  for(int i = 0; i < Configuration.listOfDevices.size(); i++){
    initializePopulationCorbaList.add(new InitializePopulationCORBA
      (gaClient.gaImplList.get(i), items, chromosomeWeight,
      numOfPopOnDevice * Configuration.listOfDevices.get(i).
      getNumberOfCores()));
    initializePopulationCorbaList.get(i).start();
  }
```

Keď sa premenná `resAfMo` rovná nule tak sa na každom serveri bude inicializovať rovnako veľká časť populácie. Vytvorenie vlákna ktoré komunikuje s jedným serverom, ako vstupné parametre pri vytváraní vlákna je použitá statická premenná zo zoznamu `gaImplList`, v tejto premennej sú informácie potrebné k tomu aby sa klient mohol pripojiť na server na ktorom sa bude robiť inicializácie. Ďalšie parametre sú predmety z ktorích sa vytvárajú chromozómy a súčin premenných `numOfPopOnDevice` a počet jadier serveru. tento súčin určuje jak veľká časť populácie sa bude incializovať na danom serveri. V spustenenom vlákne klient zavolá metódu *initilaizePopulation()* na serveri a ten vráti zoznam vytvorených chromozómov.

```
}else{
  for(int i = 0; i < Configuration.listOfDevices.size(); i++){
    if( resAfMo != 0){
      initializePopulationCorbaList.add(new
        InitializePopulationCORBA(gaClient.gaImplList.get(i),
          items, chromosomeWeight, numOfPopOnDevice *
          Configuration.listOfDevices.get(i).getNumberOfCores()
          + resAfMo));
      initializePopulationCorbaList.get(i).start();
      resAfMo = 0;
    }else{
      initializePopulationCorbaList.add(new
```

```
          InitializePopulationCORBA(gaClient.gaImplList.get(i),
          items, chromosomeWeight, numOfPopOnDevice *
          Configuration.listOfDevices.get(i).getNumberOfCores()));
        initializePopulationCorbaList.get(i).start();
      }
    }
  }
```

Keď sa premenná `resAfMo` nerovná nule tak sa ne jednom serveri bude vytvárať väčšia časť populácie.

```
int m = 0;
for(int i = 0; i < initializePopulationCorbaList.size(); i++){
  try {
    initializePopulationCorbaList.get(i).join();
    for( Chromosome ch : initializePopulationCorbaList.get(i).
      getChromosomes())
      chromosomes[m++] = new Chromosome(ch);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
initializePopulationCorbaList = null;
}
break;
```

Po dokončení inicializácie na všetkých serveroch sa vytvorené chromozómy skopírujú do premennej `chromosomes`.

5.3.1 Meranie distribuovaného modelu

Tab.5.3 zobrazuje namerané časy genetického algoritmu pre distribuovaný model CORBA. Pri meraní distribuovaného modelu sa zredukoval čas pre všetky veľkosti populácie a tiež bol čas lepší pri použití dvoch a viacerých počítačoch oproti spracovaniu na jednom počítači. Pre veľkosť populácie 100 jedincov sa najlepší čas dosiahol pri piatich počítačoch, zlepšenie bolo 35,83 %, pri pridávaní ďalších počítačov čas mierne stúpol, čo je spôsobené réžiou komunikácie. Pre veľkosť populácie 1 000 jedincov bol najlepší čas pri šiestich počítačoch lepší o 76,23 % oproti jednému počítaču. Pre veľkosť populácie 10 000 jedincov bol najlepší čas pri použití šiestich počítačov,

Veľkosť populácie	100	1 000	10 000	20 000	50 000
	Čas [s]				
Distribuovaný model: 1 počítač	0,360	4,953	39,986	87,105	117,409
Distribuovaný model: 2 počítače	0,260	2,783	24,763	53,497	75,014
Distribuovaný model: 3 počítače	0,243	2,256	19,810	40,135	74,082
Distribuovaný model: 4 počítače	0,235	2,045	16,764	35,121	60,898
Distribuovaný model: 5 počítačov	0,231	1,917	14,882	28,330	62,077
Distribuovaný model: 6 počítačov	0,232	1,177	14,190	27,895	62,008

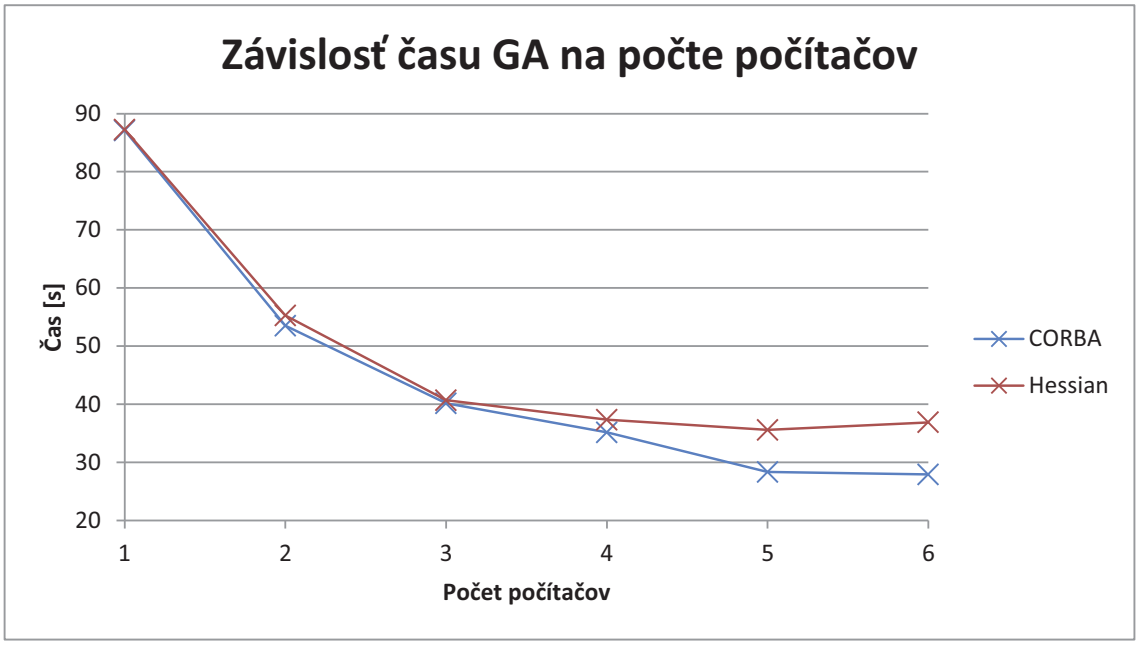
Tab. 5.3: Namerané hodnoty GA pre distribuovaný model CORBA

čas bol lepší o 64,51 % oproti jendému počítaču. Pre veľkosť populácie 20 000 jedincov bol najlepší čas pri šiestich počítačoch lepší o 67,97 % oproti jednému. Pre veľkosť populácie 50 000 jedincov sa dosiahol najlepší čas pri štyroch počítačoch, oproti jednému počítaču bol čas lepší o 48,13 %.

Veľkosť populácie	100	1 000	10 000	20 000	50 000
	Čas [s]				
Distribuovaný model: 1 počítač	0,548	3,805	37,971	87,242	234,356
Distribuovaný model: 2 počítače	0,367	2,596	23,666	55,268	136,850
Distribuovaný model: 3 počítače	0,307	1,926	19,607	40,675	118,084
Distribuovaný model: 4 počítače	0,325	1,803	17,179	37,304	94,876
Distribuovaný model: 5 počítačov	0,328	1,885	17,121	35,563	97,214
Distribuovaný model: 6 počítačov	0,342	1,900	16,697	36,875	100,452

Tab. 5.4: Namerané hodnoty GA pre distribuovaný model Hessian

Tab.5.4 zobrazuje namerané časy genetického algoritmu pre distribuovaný model Hessian. Pri meraní distribuovaného modlu Hessian sa podarili zredukovať čas pre všetky veľkosti populácie a tiež bol čas lepší pri použití dvoch a viacerých počítačoch oproti spracovaniu na jednom počítači. Pre veľkosť populácie 100 jedincov sa najlepší čas dosiahol pri troch počítačov, zlepšenie bolo 43,97 %, pri pridávaní ďalších počítačov čas mierne stúpol, čo je spôsobené réžiou komunikácie. Pre veľkosť populácie 1 000 jedincov bol najlepší čas pri štyroch počítačoch lepší o 52,61 % oproti jednému počítaču. Pre veľkosť populácie 10 000 jedincov bol najlepší čas pri použití šiestich počítačov, čas bol lepší o 56,02 % oproti jendému počítaču. Pre veľkosť populácie 20 000 jedincov bol najlepší čas pri piatich počítačoch lepší o 59,23 % oproti jednému. Pre veľkosť populácie 50 000 jedincov sa dosiahol najlepší čas pri



Obr. 5.3: Závislosť času GA na počte počítačov (veľkosť populácie 20 000)

štyroch počítačoch, oproti jednému počítaču bol čas lepší o 59,79 %. Z nameraných hodnôt časov v tab.5.3 a tab.5.4 je vidieť, že pri použití technológie CORBA a použití šiestich počítačov sa dosiahl lepších časov ako technológia Hessian. Na obr.5.3 je graf závislosti času genetického algoritmu na počte počítačoch použitých pri meraní distribuovaného modelu pre technológie CORBA a Hessian. Z grafu je vidieť že pri použití jedného až troch počítačov je čas približne rovnaký pre obi dve technológie. Pri použití štyroch až šiestich počítačov je vidieť, že technológia CORBA má lepší čas ako Hessian.

6 ZÁVER

Cieľom bakalárskej práce bolo naštudovanie komunikačných technológii ktoré sa používajú na vytváranie klient-server aplikácií v programovacom jazyku Java a vytvorenie paralelného a distribuovaného modelu genetického algoritmu pre problém naplnenia batohu pre technológie CORBA a Hessian. V úvodných kapitolách sa práca venuje teoretickým poznatkom z oblasti sieťovej komunikácie, technológiám CORBA, Java RMI a Hessian ktoré sa používajú na vytváranie klient-server aplikácií, spracovaniu dát pomocou paralelného a ditribuovaného modelu a genetickým algoritmom a využitie genetického algoritmu na vyriešenie problému naplnenia batohu.

V praktickej časti práce sú vysvetlené časti zdrojových kódov, je tu spravené meranie pre paralelný a distribuovaný model. Paralelizácia a distribúcia genetického algoritmu je spravená pre metódy *initializePopulation()*, *applyMutation()* a *applyCrossover()*. Praktická časť práce je rozdelená do troch podkapitôl. V prvej podkapitole je popísaný spôsob merania genetického algoritmu, nastavenie parametrov genetického algoritmu pri meraní a hardvérová špecifikácia počítačov na ktorích sa robilo merania paralelného a distribuovaného modelu.

Druhá podkapitola sa venuje paralelnému modelu, je tu vysvetlená časť zdrojového kódu z metódy *initializePopulation()* ktorá paralelizuje inicializáciu populácie do viacerých vlákien. V ďalšej časti je spravené meranie genetického algoritmu pre 1, 2, 4, 8 a 16 vlákien, výsledky sú zobrazené v tabuľke a grafoch. Meranie ukázalo, že paralelizácia nezredukuje čas pre všetky veľkosti populácie, čo je spôsobené tým, že pre malé veľkosti populácie trvá réžia vlákien dlhšie ako samotný výpočet. Pri meraní sa tiež zistilo že je najlepšie použiť toľko vlákien koľko podporuje procesor, ďalšie zvyšovanie počtu vlákien čas viacej nezredukuje ale čas môže byť horší kvôli zvýšenej réžii viacerých vlákien.

Tretia podkapitola sa venuje distribuovanému modelu, je tu vysvetlená časť zdrojového kódu metódy z metódy *initializePopulation()* ktorá rozdeľuje inicializáciu populácie na viacej počítačov. Je tu spravené meranie distribuovaného modelu pre technológie CORBA a Hessian, pre obi dve technológie je meranie spravené pre 1, 2, 3, 4, 5 a 6 počítačov. Výsledky sú zobrazené v tabuľkách a grafe. Meraním distribuovaného modelu sa podarilo zredukovať čas pre všetky veľkosti populácie a pre dva až šesť počítačov oproti jednému. Meraním sa zistilo, že technológia CORBA má menšiu komunikačnú réžiu ako technológia Hessian a s použitím technológie CORBA sa dosiahlo lepších časov ako s technológiou Hessian.

LITERATÚRA

[1] REILLY, D. *Java RMI and CORBA, A comparison of two competing technologies* [online]. 2000, posledná aktualizácia 5. 6. 2006 [cit. 11. 8. 2013]. Dostupné z URL: <http://www.javacoffeebreak.com/articles/rmi_corba/>.

[2] BOLTON, F. *Pure CORBA*. USA: Sams Publishing, 2002. ISBN 0-672-31812-1.

[3] *The Java Tutorials* [online]. [cit. 11. 8. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/tutorial/index.html>.

[4] *What Is a Network Interface?* [online]. [cit. 11. 8. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/tutorial/networking/nifs/definition.html>.

[5] BRRADLEY, M. *Introduction to Client Server Networks* [online]. 2013, posledná aktualizácia 2013 [cit. 11. 8. 2013]. Dostupné z URL: <http://compnetworking.about.com/od/basicnetworkingfaqs/a/client-server.htm>.

[6] LEINER, B. *Brief History of the Internet* [online]. 2013, posledná aktualizácia 2013 [cit. 11. 8. 2013]. Dostupné z URL: <http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet>.

[7] KOZIEROK, Ch. *The TCP/IP Guide* [online]. 2013, posledná aktualizácia 2013 [cit. 11. 8. 2013]. Dostupné z URL: <http://www.tcpipguide.com/free/t_TCPIPTransportLayerProtocolsTransmissionControlPro.htm>.

[8] *OSI-Basic Reference Model: The Basic Model* [online]. [cit. 11. 15. 2013]. Dostupné z URL: <http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf>.

[9] *Internetworking Basics* [online]. [cit. 11. 16. 2013]. Dostupné z URL: <http://www.cisco.com/cpress/cc/td/cpress/fund/ith/ith01gb.htm>.

[10] KESSLER, G. *An Overview of TCP/IP Protocols and the Internet* [online]. Posledná aktualizácia 24. 4. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://www.garykessler.net/library/tcpip.html>.

[11] KOZIEROK, Ch. *TCP Common Applications and Server Port Assignments* [online]. Posledná aktualizácia 24. 4. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://www.tcpipguide.com/free/t_TCPCommonApplicationsandServerPortAssignments.htm>.

[12] KOZIEROK, Ch. *Internet Protocol Version 4* [online]. Posledná aktualizácia 24. 4. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://www.tcpipguide.com/free/t_InternetProtocolVersion4IPIPv4.htm>.

[13] *Uniform Resource Name* [online]. Posledná aktualizácia 28. 10. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://cs.wikipedia.org/wiki/Uniform_Resource_Name>.

[14] *Thin client* [online]. Posledná aktualizácia 28. 10. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://en.wikipedia.org/wiki/Thin_client>.

[15] *Fat client* [online]. Posledná aktualizácia 28. 10. 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://en.wikipedia.org/wiki/Fat_client>.

[16] *An Overview of RMI Applications* [online]. Posledná aktualizácia 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/tutorial/rmi/overview.html>.

[17] *RMI-IIOP Programmer's Guide* [online]. Posledná aktualizácia 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/7/docs/technotes/guides/rmi-iiop/rmi_iiop_pg.html>.

[18] *Creating a URL* [online]. Posledná aktualizácia 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/tutorial/networking/urls/creatingUrls.html>.

[19] *What Is a Socket?* [online]. Posledná aktualizácia 2013 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html>.

[20] *Package org.omg.CORBA* [online]. Posledná aktualizácia 2011 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/6/docs/api/org/omg/CORBA/package-summary.html>.

[21] *Package org.omg.CosNaming* [online]. Posledná aktualizácia 2011 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/6/docs/api/org/omg/CosNaming/package-summary.html>.

[22] *Package org.omg.PortableServer* [online]. Posledná aktualizácia 2011 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/6/docs/api/org/omg/PortableServer/package-summary.html>.

[23] *Package org.omg.PortableInterceptor* [online]. Posledná aktualizácia 2011 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/6/docs/api/org/omg/PortableInterceptor/package-summary.html>.

[24] *Package org.omg.DynamicAny* [online]. Posledná aktualizácia 2011 [cit. 11. 16. 2013]. Dostupné z URL: <http://docs.oracle.com/javase/6/docs/api/org/omg/DynamicAny/package-summary.html>.

[25] TEDA, J. *Genetické algoritmy a jejich aplikace v praxi* [online]. Posledná aktualizácia 26. 7. 2005, [cit. 30. 11. 2013]. Dostupné z URL: <http://programujte.com/clanek/2005072601-geneticke-algoritmy-a-jejich-aplikace-v-praxi/>.

[26] KARÁSEK, J. *Laboratorní úloha - Genetické algoritmy.* Posledná aktualizácia 2012 [cit. 30. 11. 2013].

[27] *The OSI Reference Model for Network Protocols*[online]. Posledná aktualizácia 22. 3. 2013 [cit. 29. 12. 2013]. Dostupné z URL: <http://www.hardwaresecrets.com/article/The-OSI-Reference-Model-for-Network-Protocols/431/4>.

[28] PETER, M. *Evolúcia a evolučná teória* [online]. Posledná aktualizácia 2012, [cit. 22. 12. 2013]. Dostupné z URL: <http://www.evolutionrevolution.eu/sk/>.

[29] Blaise, B. *Introduction to Parallel Computing* [online]. Posledná aktualizácia 2013, [cit. 28. 3. 2014]. Dostupné z URL: <https://computing.llnl.gov/tutorials/parallel_comp/>.

[30] *Resin Documentation* [online]. Posledná aktualizácia 2012, [cit. 3. 4. 2014]. Dostupné z URL: <http://hessian.caucho.com/doc/>.

[31] *Hessian 2.0 Serialization Protocol* [online]. Posledná aktualizácia 2012, [cit. 3. 4. 2014]. Dostupné z URL: <http://hessian.caucho.com/doc/hessian-serialization.html>.

[32] Gredler, D. *Java Remoting: Protocol Benchmarks* [online]. Posledná aktualizácia 2013, [cit. 3. 4. 2014]. Dostupné z URL: <http://daniel.gredler.net/2008/01/07/java-remoting-protocol-benchmarks/>.

[33] *Middleware remoting protocol migration* [online]. Posledná aktualizácia 2007, [cit. 3. 4. 2014]. Dostupné z URL: <http://blog.nominet.org.uk/tech/2007/03/13/middleware-remoting-protocol-migration/>.

[34] S. Tanenbaum, A., Van Steen, M. *Distributed Systems: Principles and Paradigms*. ISBN 0-13-239227-5.

[35] *Distributed Computing Overview* [online]. Posledná aktualizácia 2009, [cit. 21. 4. 2014]. Dostupné z URL: <`http://www.tedb.net/articles/DistributedComputingOverview.html`>.

[36] *Parallel Algorithms* [online]. Posledná aktualizácia 2009, [cit. 21. 4. 2014]. Dostupné z URL: <`https://www.cs.cmu.edu/~guyb/papers/BM04.pdf`>.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

ARPA Advanced Research Projects Agency

ARP Address Resolution Protocol

API Aplication program interface

BOINC Berkeley Open Infrastructure for Network Computing

CORBA Common Object Request Broker Architecture

CCITT International Telegraph and Telephone Consultative Committee

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

FTP File Transfer Protocol

GA Genetický algoritmus

GC Garbage Collector

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDL Interface Definition Language

IIOP Internet Inter-ORB Protocol

IGMP Internet Group Management Protocol

IOR Interoperable Object Reference

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IPSEC Internet Protocol Security

ISO Organization for Standardization

JDK Java Development Kit

JVM Java Virtual Machine

NPC Network Control Protocol

NSFNET National Science Foundation Network

OSI Open Systems Interconnection

OMA Object Managment Architecture

OMG Object Managment Group

ORB Object Request Broker

POA Portable Object Adaptor

POP3 Post Office Protocol

RMI Remote Method Invocation

SMTP Simple Mail Transfer Protocol

TCP Transmission Control Protokol

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Datagram Protokol

URI Uniform Resource Identifier

URL Uniform Resource Locator

ZOZNAM PRÍLOH

A PRÍLOHA CD

Na priloženom CD sú všetky zdrojové kódy spolu s elektronickou formou bakalárskej práce. Adresárová šruktúra prílohy obsahuje niekoľko priečinkov, v priečinku `lib` sú knižnice potrebné k spusteniu Hessian modelu. Priečinky `bin` , `src` a `target` obsahujú zdrojové kódy.

Adresárová štruktúra CD:

```
BP13_MajtanMartin
├── .settings
├── bin
│   ├── client
│   ├── clientHessian
│   ├── ga
│   ├── ga_thread
│   ├── orb.db
│   │   └── logs
│   ├── server
│   └── serverHessian
├── lib
│   ├── com
│   ├── junit
│   └── org
├── src
│   ├── client
│   ├── clientHessian
│   ├── ga
│   ├── ga_thread
│   ├── orb.db
│   │   └── logs
│   ├── server
│   └── serverHessian
└── target
    ├── classes
    │   ├── client
    │   ├── clientHessian
    │   ├── ga
    │   ├── ga_thread
    │   ├── orb.db
    │   │   └── logs
    │   ├── server
    │   └── serverHessian
    └── maven-archiver
```

Pri spustení distribuovaného modelu CORBA sa serverová časť pustí pomocou súboru server.bat ktorí sa nachádza v priečinku `src`. Klientská časť sa spustí s vý-

vojového prostredia kde sa tiež nastavia parametre genetického algoritmu v triede **Evolver**. Pri spustení distribuovaného modelu Hessian sa obi dve časti spustia z vývojového prostredia. Pre distribuované modely sa v triede **Execute** nastavujú počítače na ktorích bude prebiehať výpočet. V triede **Configuration** sa nastavuje aký model spracovania sa má použiť a počet vlákien pre paralelný model.