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 APLIKACE ZALOŽENÁ NA TECHNOLOGII JAVA RMI

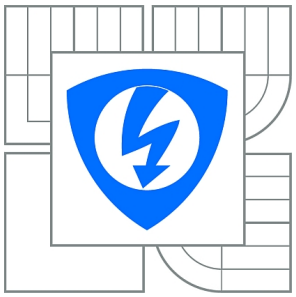
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN SALÁŠEK
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 APLIKACE ZALOŽENÁ NA TECHNOLOGII JAVA RMI

CLIENT-SERVER APPLICATION BASED ON JAVA RMI

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

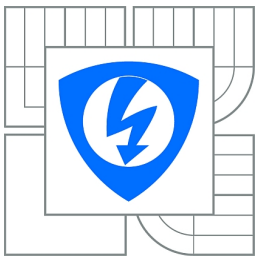
AUTOR PRÁCE — JAN SALÁŠEK

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:	Jan Salášek	***ID:***	146945
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Klient-server aplikace založená na technologii JAVA RMI

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 Java RMI a SOAP. 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] Grosso, W.; Java RMI. Java Series. O'Reilly Media; 1st edition. October 29, 2001. s. 576. ISBN-13: 978-1565924529.

[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

Bakalářská práce shrnuje možnosti pro vytváření distribuovaných systémů využitelných v platformě Java. Zabývá se zrychlením výpočtu pomocí paralelizace a distribuovaného zpracování dat.

KLÍČOVÁ SLOVA

Java RMI, SOAP, Java IDL, CORBA, paralelizace, genetické algoritmy, problém batohu

ABSTRACT

Bachelor thesis summarizes the possibilites for creating distributed systems for use in Java. It deals with acceleration of calculation using parallelization and distributed data processing.

KEYWORDS

Java RMI, SOAP, Java IDL, CORBA, parallelization, genetic algorithms, knapsack problem

SALÁŠEK, Jan *Klient-server aplikace založená na technologii JAVA RMI*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 40 s. Vedoucí práce byl Ing. Jan Karásek,

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Klient-server aplikace založená na technologii JAVA RMI“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu Ing. Janu Karáskovi za odborné vedení, konzultace, trpělivost a podnětné 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

PODĚKOVÁNÍ

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

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Bakalářská práce se shrnuje možnosti pro vytváření distribuovaných systémů využitelných v platformě Java. Zabývá se řešením problému batohu pomocí genetického algoritmu a zrychlením tohoto algoritmu pomocí paralelizace.

Byly nastudovány vlastnosti a použití technologie Java RMI a dalších technologií pro aplikace klient-server, paralelizaci a genetické algoritmy. Byla vytvořena jednoduchá aplikace pro předvedení technologie Java RMI. Pomocí paralelizace v rámci jednoho počítače byl zrychlen genetický algoritmus pro řešení problému batohu. Byla přidána možnost distribuovaní výpočtu pomocí technologii SOAP a Java RMI.

První část práce obsahuje rešerši síťové komunikace a distribuovaných systémů. Je zde seznámení s jednotlivými technologiemi použitelnými v platformě Java a porovnání technologií Java RMI a CORBA a porovnání technologií SOAP a Java RMI. V druhé části je popis paralelizace a jejích výhod a nevýhod. V třetí části je seznámení s genetickými algoritmy. Čtvrtá část obsahuje popis objektů a metod použitých v implementaci paralelizace. Pátá část obsahuje popis rozšíření o možnost distribuovaného výpočtu. V následujících částech byl zjišťován vliv paralelizace a distribuování výpočtu na rychlost výpočtu.

1 REŠERŠE SÍŤOVÉ KOMUNIKACE A DISTRIBUOVANÝCH SYSTÉMŮ

Síťová komunikace a distribuované systémy mají v dnešní době velký význam.

1.1 Síťová komunikace

Pro síťovou komunikaci vzniklo několik modelů. Zde se budu zabývat pouze počítačovými sítěmi. Pro popis sítě je používán referenční model ISO/Propojení otevřených systémů (anglicky Open Systems Interconnection) (OSI). Reálné použití popisuje několik modelů. Nejpoužívanějším a také jediným, kterému Java rozumí, je model Transmission Control Protocol (TCP)/Internet Protocol (IP).[1][2][5]

1.1.1 Referenční model ISO/OSI

Referenční model ISO/OSI je teoretický základ pro veřejné počítačové sítě. Skládá se ze sedmi vrstev. Každá vrstva komunikuje logicky s odpovídající vrstvou ve vodorovné rovině, ve skutečnosti se sousedními vrstvami na svislé rovině.[1][2][5]

Fyzická vrstva (Physical Layer)

Fyzická vrstva je nejnižší vrstva, která mění data na posloupnost bitů. Určuje mechanické a elektrické vlastnosti komponent.[1][2][5]

Spojová vrstva (Data Link Layer)

Druhá vrstva zajišťuje přenos rámců. Může být spolehlivá i nespolehlivá. V případě spolehlivé varianty oznamuje odesilateli špatně přenesené rámce a žádá opětovné přenešení celého rámce. Musí také poznat, kde rámec začíná a kde končí. Na této vrstvě pracují zařízení jako například přepínače a mosty.[1][2][5]

Síťová vrstva (Network Layer)

Síťová vrstva je zodpovědná za přenos dat v komplexní síti. Stará se o nalezení co nejlepší cesty v síti. Datová jednotka, kterou přenáší, se nazývá paket. Síťová služba může být buď se spojením, nebo bez spojení. V síťové vrstvě jsou sdruženy funkce, které umožňují překlenout rozdílné vlastnosti dílčích úseků trasy, které mohou být založeny na různých technologiích.[1][2][5]

Transportní vrstva (Transport Layer)

Transportní vrstva zajišťuje přenos dat mezi koncovými uzly. Jejím hlavním úkolem je poskytnou vyšším vrstvám kvalitu služeb, kterou potřebují. Transportní vrstva stojí mezi poskytovateli přenosových služeb a jejich uživateli. Je to také poslední vrstva, která může měnit nespojový charakter na spojový a naopak.[1][2][5]

Relační vrstva (Session Layer)

Umožňuje vytvoření a ukončení relačního spojení a oznamování výjimečných stavů.[1][2][5]

Prezentační vrstva (Presentation Layer)

Tato vrstva slouží k interpretování přenášených dat. Může docházet ke změně kódování, kompresi nebo šifrování.[1][2][5]

Aplikační vrstva (Application Layer)

Protokoly aplikační vrstvy zajišťují komunikaci mezi aplikačními procesy ve spojení s funkcemi operačního systému, který tyto procesy podporují. Aplikační vrstva poskytuje aplikačním procesům přístup ke komunikačním prostředkům.[1][2][5]

1.1.2 TCP/IP

Komunikace po síti probíhá pomocí soustavy protokolů označované TCP/IP. Komunikaci popisuje vrstvový model TCP/IP.[1][2][5]

Vrstva síťového rozhraní (Network Interface Layer)

Tato vrstva je nejnižší. Není v TCP/IP blíže specifikována, protože závisí na konkrétním přenosové technologii. Řídí přístup k mediu a posílání paketů. Může být jednoduchá, nebo obsahovat složitý linkový protokol.[1][2][5]

Internetová vrstva (Internet Layer)

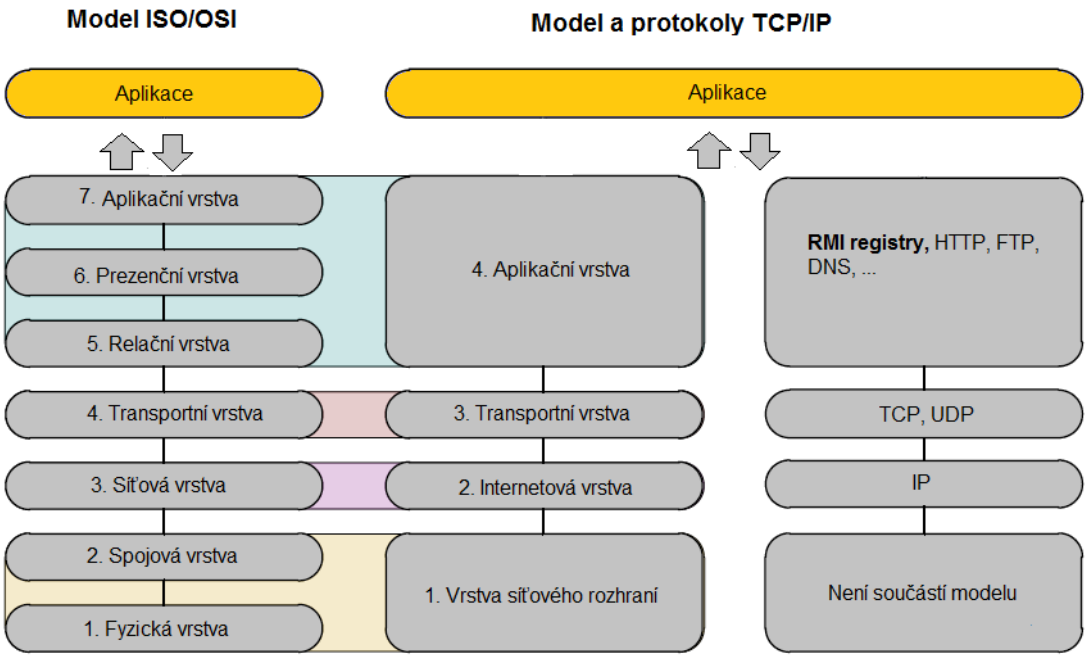
Druhá vrstva je často označovaná IP vrstva, protože je řízena protokolem IP. V současnosti se používají verze 4 a 6. Na této vrstvě se adresuje IP adresou. Hlavní rozdíl mezi verzemi je délka adresy. Verze 4 má adresu dlouhou čtyři bajty a adresy mohou být typů unicast, multicast a broadcast. Verze 6 má adresu dlouhou sto dvacet osm bajtů a mohou být typů unicast, multicast a anycast. Úkolem této vrstvy je, aby se jednotlivé pakety dostaly od odesílatele až ke svému skutečnému příjemci, obvykle přes mezilehlá zařízení. Na této úrovni je využívána nespolehlivá služba.[1][2][5]

Transportní vrstva (Transport Layer)

Třetí vrstva je nejčastěji řízena protokolem TCP (Transmission Control Protocol), v některých případech protokolem User Datagram Protocol (UDP). V této vrstvě se k adresaci používá číslo portu. Číslo portu určuje, které aplikaci se mají data doručit. Protokol TCP se označuje jako spolehlivý, protože má algoritmy na navázání spojení a kontrolu přenášených dat. Protokol UDP je nespolehlivý. Nezaručuje ani doručení ani pořadí došlých datagramů. Vyhoda tohoto protokolu je vyšší rychlost oproti protokolu TCP na úkor ztrátovosti.[1][2][5]

Aplikační vrstva (Application Layer)

Nejvyšší vrstva, entitami jsou jednotlivé aplikace. Používá se zde mnoho protokolů (Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Internet InterORB Protocol (IIOP), ...). Pro Javu RMI jsou důležité Java Object Serialization, RMI registry.[1][2][5]



Obr. 1.1: Vrstvový model ISO/OSI a TCP/IP[1][2][5]

1.1.3 Síťové třídy v JDK

V Javě se požívá TCP i UDP. Třídy *URL*, *URLConnection*, *Socket* a *ServerSocket* používají protokol TCP. Třídy *DatagramPacket*, *DatagramSocket* a *MulticastSocket*

používají protokol UDP.[1][2][5]

1.2 Java RMI

Remote Method Invocation, vzdálené volání metod (RMI) je jádro Java API a knihovní třída, která dovoluje Java programům běžet v jednom virtuálním stroji a volat metody z objektů, které běží na jiném virtuálním stroji. Tyto virtuální stroje mohou běžet na různých fyzických strojích. V podstatě je možné, aby část programu běžela na místním a část na vzdáleném počítači. RMI tvoří iluzi, že program běží na jednom počítači v jedné pamětí a přitom je bajtkód na dvou místech. Všechny implementační detaily jsou skryty. Síťové chyby jsou zastoupeny jako RemoteExceptions. Každý vzdálený objekt implementuje alespoň jedno vzdálené rozhraní, které deklaruje, které metody vzdálených objektů mohou být volány vzdáleným systémem.[1][2]

1.2.1 Java autentizační a autorizační služby

Obecně jsou veřejné služby přístupné všem. Omezení přístupu lze dodat pomocí Java Authentication and Authorization Service, Java autentizační a autorizační služby (JAAS). JAAS je abstraktní rozhraní, které může být konfigurováno pro různé služby a různá data.[1][2]

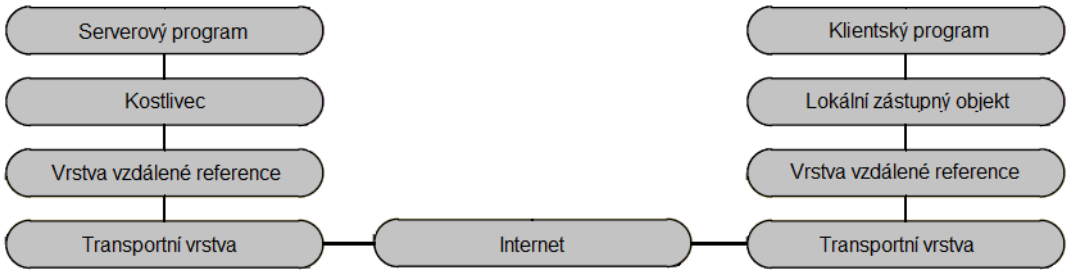
1.2.2 Serializace objektů

V případě chodu na jednom virtuálním stroji, pokud je objekt poslán metodě nebo má být přijat od metody, je poslán pouze odkaz, ve většině současných implementacích Javy ukazatel na ukazatel umístění objektu v paměti. Pokud metoda běží na jiném stroji, není to možné. Jsou dva způsoby, jak řešit tento problém. První možnost je konvertovat objekt do sekvence bajtů a tuto sekvenci poslat vzdálenému stroji. Vzdálený stroj obnoví ze sekvence bajtů objekt. Změna této kopie není automaticky reflektována v původním objektu. Druhá možnost je použít specialní referenci k objektu. Když vzdálený stroj zavolá metodu, zavolá tuto referenci, volání se vrací zpět na místní stroj, kde vytvoří původní objekt. Změny objektu se projeví na obou koncích spojení, protože oba konce sdílejí ten samý objekt. Konverze objektů může být složitá, protože objekty mohou odkazovat na další objekty, které je potřeba kopírovat. Serializovaný objekt může být také zapsán na disk a znovu obnoven.[1][2]
Z bezpečnostních důvodů nemohou být všechny objekty Javy serializovány. Všechny primitivní typy mohou být serializovány. Běžné objekty mohou být serializovány,

jen pokud je implementováno rozhraní *java.io.Serializable*. Základní typy, které implementují toto rozhraní, jsou *String* a *Component*. *Vector* je serializovatelný, jen pokud jsou serializovatelné všechny objekty, které obsahuje.[1][2]

1.2.3 Vrstvový model RMI

Pro programátora se komunikace jeví, jakoby klient komunikoval přímo se serverem. Ve skutečnosti komunikuje se zástupným objektem vzdáleného objektu. Zástupný objekt (stub) je umístěn u klienta. Zástupný objekt posílá komunikaci přes vrstvu vzdálené reference (remote reference layer). Tato vrstva komunikuje s transportní vrstvou (transport layer). Transportní vrstva na straně klienta přenese data přes internet k transportní vrstvě serveru. Transportní vrstva serveru komunikuje s vrstvou vzdálené reference, která komunikuje s částí serveru zvanou kostra (skeleton). Kostra komunikuje se samotným serverem. Servery, napsané v Javě 1.2 a pozdější, mohou mít vynechanou vrstvu kostry. V opačném směru, ze serveru na klienta, je postup obrácený. Logický přenos dat klient - server a server - klient je vodorovný a skutečný svislý.[1][2]



Obr. 1.2: Vrstvový model Java RMI[1][2]

1.2.4 Získávání URL vzdáleného objektu

Před zavoláním vzdáleného objektu je potřeba získat referenci. Pro získání této reference se klient zeptá registru podle jména. Klient dostane Uniform Resource Locator, jednotný lokátor zdrojů (URL). Registr je něco na způsob mini-DNS.[1][2]

Ve skutečnosti klient volá pouze metody v místním zástupném objektu. Zástupný objekt je místní objekt, který implementuje vzdálené rozhraní vzdáleného objektu. To dává zdání, že klient volá vzdálenou metodu, ale ve skutečnosti volá ekvivalentní metodu v zástupném objektu. Zástupné objekty jsou umístěny v klientském virtuálním stroji na místě skutečných objektů a metod vzdáleného serveru. Všechny rozhraní a třídy pro RMI jsou obsaženy v několika balíčcích. Nejpoužívanější jsou balíčky *java.rmi*, *java.rmi.server*, *java.rmi.registry*. Vzdáleně mohou být volány pouze metody deklarované ve vzdáleném rozhraní. Vzdálené rozhraní je potomkem rozhraní *java.rmi.Remote*. V tomto rozhraní deklarujeme vzdáleně volané metody. Každá metoda musí vyhazovat výjimku *java.rmi.RemoteException*. Parametry a návratové hodnoty mohou být jakéhokoli typu, který je možno serializovat. Primitivní datové typy je možno používat bez problémů.

1.2.5 Implementace vzdálené služby – vzdálený objekt

Je třeba vytvořit implementaci vzdáleného rozhraní, to je třídu, která bude součástí serveru. Tato třída bude implementovat rozhraní definující vzdálené služby a musí mít konstruktor vyhazující výjimku *java.rmi.RemoteException*. Třída může obsahovat jakékoli další metody, ty však nebude možno volat přímo přes RMI. Pokud nepotřebujeme tuto třídu napsat jako potomka jiné třídy, použijeme dědičnost od třídy *java.rmi.server.UnicastRemoteObject*. V případě, že chceme, aby třída dědila od jiného předka, než je *UnicastRemoteObject*, musí konstruktor třídy obsahovat tento kód: *UnicastRemoteObject.exportObject(this)*; Volání této metody (či volání konstruktoru předka, v případě dědění od třídy *UnicasteRemoteObject*) zahájí čekání tohoto objektu na volání metody klientem. V okamžiku, kdy klient požádá server o službu, je pro zpracování příchozího požadavku vytvořeno samostatné vlákno. Službu je tedy možno poskytovat více klientům najednou.[1][2]

1.2.6 Balíček *java.rmi*

Java.rmi package obsahuje třídy, které jsou viděny klientem. Oba, klient i server, musí importovat *java.rmi*. Balíček obsahuje jedno rozhraní, tři třídy a užitečné výjimky[1][2]

Rozhraní *Remote*

Rozhraní *Remote* se používá u objektů, které mají být vzdálené. Nedeklaruje žádné metody. Vzdálené metody obvykle implementují podtřídu rozhraní *Remote*, která deklaruje nějaké metody. Metody, které jsou deklarovány v rozhraní, jsou metody, které mohou být volány vzdáleně.[1][2]

Třída *Naming*

Třída *java.rmi.Naming* komunikuje s registry běžícími na serveru. Třída Naming má pět veřejných metod:

- *list( )* - Získá všechna jména z registry.
- *lookup( )* - Najde vzdálený objekt určený URL.
- *bind( )* - Sváže jména se vzdáleným objektem.
- *rebind( )* - Sváže jméno s jiným vzdáleným objektem
- *unbind( )* - Odstraní jméno z registry

[1][2]

1.2.7 Třída RMISecurityManager

Klient načte zástupný objekt z potenciálně nedůvěryhodného serveru. V tomto případě je vztah mezi klientem a zástupným objektem podobný jako vztah mezi prohlížečem a appletem. Ačkoliv by zástupný objekt měl pouze předávat argumenty a návratové hodnoty přes síť, je to třída, která obsahuje metody, které mohou dělat cokoliv. Podobně jako applety používají třídu *AppletSecurityManager*, aplikace používají třídu *RMISecurityManager* proti zástupným objektům.[1][2]

1.2.8 Vzdálené výjimky

Vzdálené metody závisí na mnoha věcech, které neovládají. Například to jsou stavy sítě a dalších nezbytných služeb jako DNS. Není nijak zaručeno, že nenastane chyba během provádění metody. Všechny vzdálené metody musí být deklarovány s "*throw RemoteException*".[1][2]

1.3 Java IDL

Java Object Management Group Interface Definition Language, jazyk pro popis rozhraní (IDL), umožnila využití technologie Common Object Request Broker Architecture (CORBA) v platformě Java. Kompatibilitu s platformou Java poskytuje

na základě standardů interoperability a konektivity. Java IDL umožňuje distribuované webové aplikace Java transparentně vyvolat operace vzdálené síťové služby využívající průmyslový standard IDL a IIOP definované společností Object Management Group. Runtime součásti zahrnují Java ORB pro distribuované výpočty pomocí IIOP komunikace.[3]

1.4 Java RMI-IIOP

Java RMI přes IIOP byla vyvinuta společnostmi Sun a IBM. Java RMI přes IIOP kombinuje nejlepší vlastnosti Javy RMI s nejlepšími vlastnostmi CORBA. Distribuované aplikace byly vyvinuty tak, aby kompletně pracovaly v jazyce Java. Při použití RMI - IIOP k vývoji rozdistribuovaných Java aplikací není potřeba se učit IDL nebo mapování. Podobně jako RMI i RMI - IIOP dovoluje posílat jakékoliv serializovatelné objekty a datové typy. Podobně jako CORBA i RMI - IIOP je založeno na otevřeném standardu. Podobně jako CORBA i RMI - IIOP používá IIOP jako komunikační protokol. IIOP jednoduše integruje stávající aplikace a komponenty napsané v jiných jazycích podporujích CORBA s komponenty běžícími na platformě Java. RMI - IIOP je založeno na dvou specifikacích Object Management Group (OMG):

- Java Language Mapping to OMG IDL Specification[7]
- CORBA/IIOP 2.3.1 Specification, formal/99-10-07[8]

S RMI - IIOP programátor může psát vzdálená rozhraní v Javě a implementovat je použitím pouze technologie Java a Java RMI. Tato rozhraní mohou být implementována v jakémkoliv jiném jazyce, který podporuje OMG a vydavatel dodává ORB pro tento jazyk. Podobně klient může být napsán v jiném jazyku, který používá IDL odvozené z technologie Java. Použitím RMI - IIOP může být objekt poslán jako reference i jako hodnota přes IIOP.

RMI - IIOP je dostupné stažením platformy Java 2SE ve verzi 1.3 a 1.4. RMI - IIOP IDL kompilátor, rmic, by měl být ve složce bin instalace J2SDK. Je potřeba použít RMI - IIOP kompilátor s parametrem -iiop pro generování lokálního zástupného objektu, kostry a vazeb pro vzdálený objekt používající IIOP.[4][6]

1.5 Porovnání Java RMI a CORBA

Porovnání RMI a CORBA nedává odpověď na to, co je lepší. Vlastnosti těchto dvou technologií je určují pro různé situace.

1.5.1 Java RMI

Pro

- Portovatelnost pro mnoho platforem.
- Může dostat nový kód do jiného virtuálního stroje.
- Vývojáři v Javě mohou mít zkušenosti s RMI už od JDK1.02.
- Existující systémy mohou být snadno přetvořeny pro RMI.

[1][2]

Proti

- Vázána pouze na platformy podporující platformu Java.
- Spouštění vzdáleného kódu je bezpečnostní riziko. Zabezpečení omezuje funkčnost.
- Spolupracuje pouze se systémy napsanými pro platformu Java.

[1][2]

1.5.2 CORBA

Pro

- Služby mohou být napsány v mnoha různých jazycích, spuštěny na mnoha platformách.
- S IDL je rozhraní jasně oddělené od implementace. Programátor může vytvářet mnoho implementací s jedním rozhraním.
- CORBA podporuje primitivní datové typy a mnoho datových struktur jako parametr.
- CORBA je ideální pro použití se stávajícími systémy. Zajistí, aby aplikace byly přístupné i v budoucnu.
- Jednoduchý způsob, jak spojovat objekty a systémy.
- Systémy mají často vyšší výkon.

[1][2]

Proti

- Služba musí používat další jazyk pro rozhraní (IDL), který se musí programátor naučit.
- IDL tvoří v nástrojích mapovaného jazyka lokální zástupné objekty založené na rozhraní, některé nástroje nemusejí integrovat změny v existujícím kódu.
- CORBA nepodporuje přenos kódu a objektů.
- Budoucnost je nepředvídatelná. Pokud CORBA nedosáhne dostatečného rozšíření, může se stát zastaralou.
- Specifikace CORBA se stále vyvíjí. Je potřeba se stále učit.

[1][2]

1.6 SOAP

Simple Object Access Protocol (SOAP) je ptotokol pro výměnu správ ve tvaru XML. Pro přenos se na aplikační vrstvě nejčastěji používá HTTP, méně často Simple Mail Transfer Protocol (SMTP). Mohu být použity i další protokoly. Správa je tvořena obálkou (envelope). Obálka se skládá z hlavičky (head)a těla (body). Nemusí nutně obsahovat obě části. Za obálkou může být příloha (attachments). [11][12]

1.7 Porovnání SOAP a JAVA RMI

1.7.1 Java RMI

Pro

- Může dostat nový kód do jiného virtuálního stroje.
- Vývojáři v Javě mohou mít zkušenosti s RMI už od JDK1.02.
- Existující systémy mohou být snadno přetvořeny pro RMI.
- Jednoduché použití z hlediska programátora.

[1][2]

Proti

- Vázána pouze na platformy podporující platformu Java.
- Spouštění vzdáleného kódu je bezpečnostní riziko. Zabezpečení omezuje funkčnost.
- Spolupracuje pouze se systémy napsanými pro platformu Java.

[1][2]

1.7.2 SOAP

Pro

- Služby mohou být napsány v mnoha různých jazycích, spuštěny na mnoha platformách.
- Se SOAP je rozhraní jasně oddělené od implementace. Programátor může vytvářet mnoho implementací s jedním rozhraním.
- Způsob, jak spojovat objekty a systémy.
- Možnost přenosu pomocí HTTP. HTTP obvykle není omezeno firewallem.
- Správa je snadno čitelná člověkem.

[11][12]

Proti

- SOAP nepodporuje přenos kódu a objektů.
- Nutnost přeformátovat data do formátu XML.
- Textová zpráva je náročnější na přenesená data.

[11][12]

2 GENETICKÉ ALGORITMY

2.1 Genetické algoritmy

Genetický algoritmus patří mezi evoluční algoritmy. Je to heuristický postup, který uplatňuje principy evoluční biologie. Nachází uplatnění v případech, kdy je obtížné nebo nemožné nalézt deterministický algoritmus. Jsou používány principy dědičnosti, mutace, křížení a přežití nejlepšího. Z biologie byly přeneseny některé termíny. Pracujeme s populací jedinců o určité velikosti. Jedinec je reprezentován genotypem. Pro potřeby genetických algoritmů na rozdíl od eukaryotních organismů je genotyp složen pouze z jednoho chromozomu a tedy se podobá spíše prokaryotám. Chromozom se dělí na geny. Geny jsou uspořádány lineárně. Každý gen zde má na rozdíl od přírody přesně určené místo na chromozomu. Toto místo je pro každý chromozom v celé populaci stejné. Aby mohl gen určovat nějakou vlastnost, může nabývat různých hodnot. Těmto hodnotám se říká alely.[10]

2.2 Problém batohu

Problém batohu je názorný příklad pro řešení pomocí genetického algoritmu. Do batohu omezené nosnosti se musí vejít obsah s co nejvyšší hodnotou. Nosnost batohu je předem určená. Obsah batohu je podmnožina vybraná z množiny předem určených prvků. Každý prvek představuje jednu věc, kterou lze vložit do batohu a má určenou hmotnost a hodnotu. Není možné do batohu vložit jen část předmětu.[10]

3 JEDNODUCHÁ APLIKACE KLIENT-SERVER

Pro předvedení technologie Java RMI jsem vytvořil jednoduchou kalkulačku. Aplikace vychází z příkladu ze stránek Oraclu.[9] Kalkulačka umí sčítat, odčítat, násobit a dělit. Kalkulačka má podobu konzolové aplikace. V klientské části aplikace se kalkulačka zeptá na první číslo, pak na druhé číslo a na konec na druh operace. Čísla mohou být desetinná. Po zadání všech hodnot se tyto hodnoty zašlou na serverovou část aplikace. Serverová část provede výpočet. Výsledek vypíše do konzolového okna serverové části, aby bylo vidět, že komunikace opravdu probíhá, a výsledek také zašle klientské části. Klientská část výsledek vypíše do svého konzolového okna.

3.1 Třída *ComputeEngine*

Třída obsahuje metody *ComputeEngine*, *main* a *ExecuteTask*. Metoda *main* je spuštěna při spuštění serveru. Vytvoří objekt této třídy přístupný z klienta. Metoda *ComputeEngine* je konstruktor a pouze zavolá konstruktor předka. Metoda *ExecuteTask* spouští metodu *execute* z požadovaného objektu.

3.2 Třída *ComputePi*

Třída obsahuje pouze metodu *main*. Tato metoda je spuštěna při spuštění klienta. Metoda získá adresu objektu umístěného na serveru z registru. Přečte vstup od uživatele, odešle ho serveru, spustí výpočet na serveru a vypíše výsledek operace spočítaný serverem. Výpočet spouští pomocí metody *ExecuteTask* ze třídy *ComputeEngine*.

3.3 Třída *MathOperation*

Třída obsahuje metody *MathOperation* a *execute*. Metoda *MathOperation* je konstruktor a pouze naplňuje privátní proměnné metody. Metoda *execute* provede výpočet, vypíše výsledek do konzolového okna serveru a vrátí výsledek jako návratovou hodnotu. Tato metoda je spouštěna metodou *ExecuteTask* ze třídy *ComputeEngine*.

4 APLIKACE ŘEŠÍCÍ PROBLÉM BATOHU

Aplikace vychází z aplikace poskytnuté vedoucím práce. Výchozí aplikace využívala pouze jedno vlákno, a proto nebyl výpočet efektivní na počítačích schopných zpracovávat více vláken zároveň. Aplikace byla upravena na využití více vláken.

4.1 Třída *Execute*

Tato třída obsahuje pouze metodu *main*. Metoda *main* je spuštěna při spuštění aplikace. Metoda zadá jednotlivé geny a spustí evoluci. Pro potřeby měření potřebného času je evoluce spuštěna desetkrát a pak je vypsán aritmetický průměr těchto časů.

4.2 Třída *Evolver*

Tato třída je nejsložitější. Třída obsahuje privátní konstanty, kterými jsou určeny vlastnosti algoritmu a metody řídící průběh genetického algoritmu.

Význam konstant je následující:

- *POPULATION_SIZE* - velikost populace
- *MAX_GENERATIONS* - nejvyšší možný počet evolučních kroků (generací).
- *MAX_KNAPSACK_WEIGHT* - nosnost batohu.
- *MIN_KNAPSACK_PRICE* - nejnižší cena věcí v batohu.
- *CROSSOVER_RATE* - pravděpodobnost, že dojde ke křížení vyjádřená v procentech.
- *MUTATION_RATE* - pravděpodobnost, že dojde k mutaci vyjádřená v procentech.
- *ELITISM* - počet chromosomů, které budou vybrány na základě principu elitismu.
- *THREADS_NUMBER* - počet vláken, mezi které se rozdělí výpočetně náročné operace.

Evoluce se spustí zavoláním metody *evolve*. Tato metoda vytvoří prvotní populaci. Spustí se cyklus, který vytváří nové generace pomocí privátní metody *createNewPopulation()* a zjišťuje, zda není splněna podmínka pro ukončení cyklu. Po ukončení cyklu je vypsán výsledek. Metoda *createNewPopulation()* byla upravena tak, aby rozdělila algoritmus mezi několik vláken. Tato metoda nejprve spustí vlákno, které vybere chromosomy přeživších jedinců z předchozí populace do nové. Pak spustí vlákna vytvářející nové chromosomy křížením. Pak spustí vlákna vytvářející nové chromosomy mutací. Počet vláken pro vytváření nových chromosomů křížením je

dán konstantou *THREADS_NUMBER* a stejný počet je i vláken pro vytváření nových chromosomů mutací. Na konci počká na dokončení všech vláken.

4.3 Třída *Population*

Třída obsahuje seznam všech chromosomů a metody pro jejich inicializaci a vypsání. Zde jsem upravil metodu *initializePopulation*. Tato metoda původně přímo vytvářela náhodně prvotní generaci. Nyní spouští vlákna vytvářející novou populaci stejným způsobem. Počet vláken je dán konstantou *THREADS_NUMBER*. Každé vlákno vytvoří poměrnou část populace.

4.4 Třída *InitializePopulation*

Tato třída byla vytvořena pro vytváření prvotní populace. Obsahuje konstruktor a metodu *run*. Konstruktor naplní potřebné proměnné pro vlákno. Metoda *run* provede vlastní vytváření části nové populace. Tato metoda obsahuje upravený kód z metody *initializePopulation* ze třídy *Population* z původní aplikace.

4.5 Třídy *Chromosome* a *Gene*

Třída *Chromosome* obsahuje seznam genů. Třída *Gene* obsahuje vlastnosti jednotlivých genů.

4.6 Třída *ApplyElitism*

Tato třída byla vytvořena pro vybrání nejlepších chromosomů z předchozí populace. Obsahuje konstruktor a metodu *run*. Konstruktor naplní potřebné proměnné pro vlákno. Metoda *run* provede vlastní algoritmus. Tato metoda obsahuje upravený kód z metody *applyElitism* ze třídy *Evolver* z původní aplikace.

4.7 Třídy *ApplyCrossover* a *ApplyMutation*

Tyto třídy byly vytvořeny pro vytvoření jedinců křížením a mutací. Obsahuje konstruktor a metodu *run*. Konstruktor naplní potřebné proměnné pro vlákno. Metoda *run* provede vlastní algoritmus. Tato metoda obsahuje upravený kód z metod *applyCrossover* a *applyMutation* ze třídy *Evolver* z původní aplikace.

4.8 Třída *FitnessComparator*

Tato třída obsahuje metodu *compare*. Tato metoda porovnává dva chromosomy. Tato třída zůstala beze změny.

5 ROZŠÍŘENÍ APLIKACE ŘEŠÍCÍ PROBLÉM BAŤOHU

Aplikace byla rozšířena o možnost zpracovávat výpočet na více počítačích zároveň. Pro porovnáni byly využity technologie Java RMI a SOAP.

5.1 Třída *Computers*

V této třídě se nastavuje, zda má být výpočet zpracováván lokálně, nebo distribuovaně. V případě použití distribuovaného zpracování obsahuje nastavení adres použitých počítačů a zda má být použita technologie SOAP, nebo Java RMI.

5.2 Třídy *ApplyCrossoverDist*, *ApplyMutationDist* a *InitializePopulationDist*

Tyto jsou využívány místo tříd *ApplyCrossover*, *ApplyMutation* a *InitializePopulation* v případě, že je použita technologie JavaRMI. Třídy obsahují metodu *run*, která volá příslušnou vzdálenou metodu.

5.3 Třídy *ApplyCrossoverSOAP*, *ApplyMutationSOAP* a *InitializePopulationSOAP*

Tyto jsou využívány místo tříd *ApplyCrossover*, *ApplyMutation* a *InitializePopulation* v případě, že je použita technologie SOAP. Třídy obsahují metodu *run*, která zasílá zprávu příslušnému servletu.

5.4 Webový server a kontejner pro servlety

Jako server přijímající zprávy SOAP a kontejner pro servlety byl použit Apache Tomcat. Tento server byl spuštěn na všech počítačích provádějících výpočet distribuovaný pomocí technologie SOAP. Server předá zprávu servletu, servlet ji zpracuje a pošle zpět.

6 VLIV PARALELIZACE NA RYCHLOST VÝPOČTU

Měření byla provedena na počítači značky Acer s čtyřvláknovým mikroprocesorem Intel CORE i5 s taktovacím kmitočtem 2,27 GHz a s operační pamětí 4 GiB a na počítači značky HP s dvouvláknovým mikroprocesorem Intel CORE 2 DUO s taktovacím kmitočtem 1,8 GHz a s operační pamětí 2 GiB. Měření bylá provedena tak, že algoritmus proběhl desetkrát a byla zaznamenána průměrná hodnota. Pro lepší porovnání byl pro každou tabulku zvolen jednotný počet evolučních kroků. Pro počet evolučních kroků byla omezující velikost operační paměti počítače.

6.1 Porovnání podle počtu vláken

Měření bylo provedeno na počítači Acer. Počet evolučních kroků byl zvolen čtyřicet. Z tabulky 1 je vidět, že při nižších počtech jedinců má zrychlení rozložením mezi vlákna menší vliv než zpomalení réžií těchto vláken. Rozdíl není velký také proto, že i při zadání nižšího počtu vláken je algoritmus rozdělen mezi vlákna pro elitismus, křížení a mutaci. Zadat osm vláken nemá smysl.

Počet jedinců	Počet vláken	Průměrná doba běhu [s]
10	1	0,0373
10	2	0,0499
10	4	0,1005
10	8	0,1486
10^2	1	0,1056
10^2	2	0,1152
10^2	4	0,1354
10^2	8	0,1648
10^3	1	0,5802
10^3	2	0,3826
10^3	4	0,3600
10^3	8	0,4195
10^4	1	9,3976
10^4	2	9,1058
10^4	4	8,7031
10^4	8	9,1713

Tab. 6.1: Doba běhu na počítači Acer pro 40 evolučních kroků

6.2 Porovnání na dvou počítačích

Počet evolučních kroků byl s ohledem na počítač HP zvolen osmnáct. Rozdíl mezi počítači je znatelný až u větších populací. Protože se elitismus, křížení a mutace zpracovávají zvlášť v paralelních vláknech i při zadání jednoho vlákna, je rozdíl znatelný i pro jedno vlákno.

Počet jedinců	Počet vláken	Průměrná doba běhu [s]	
		Počítač Acer	Počítač HP
10	1	0,0228	0,0219
10	2	0,0301	0,0234
10	4	0,0416	0,0360
10	8	0,0576	0,0656
10^2	1	0,0471	0,0656
10^2	2	0,0585	0,0678
10^2	4	0,0644	0,0640
10^2	8	0,0807	0,0780
10^3	1	0,2067	0,5178
10^3	2	0,2108	0,4109
10^3	4	0,1960	0,4313
10^3	8	0,2345	0,4547
10^4	1	4,1807	9,2406
10^4	2	3,0566	11,6078
10^4	4	3,3573	13,7188
10^4	8	3,4383	13,2266

Tab. 6.2: Doba běhu na obou počítačích pro 18 evolučních kroků

7 VLIV ROZDĚLENÍ VÝPOČTU MEZI VÍCE POČÍTAČŮ

Měření bylo provedeno na šesti počítačích s čtyřvláknovými procesory Intel Core I5-2500 CPU @ 3,30Ghz a 8 GiB RAM. Každé měření bylo provedeno desetkrát. V tabulkách a grafech je uveden průměr z těchto měření.

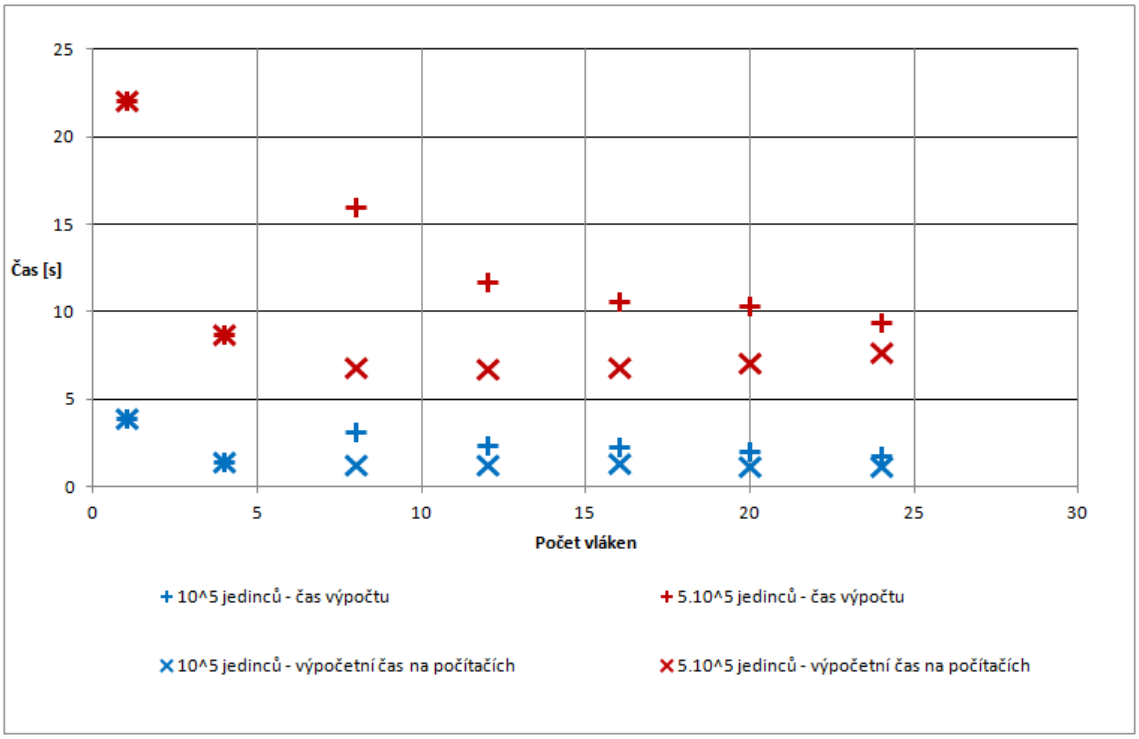
7.1 Java RMI

Aby neměla náhoda takový vliv na měření, byla nejprve změřena pouze inicializace populace. Hodnoty pro jeden počítač byly měřeny bez použití technologie Java RMI. Čas byl měřen jednak celkový (Čas výpočtu), jednak součet časů potřebných k výpočtu na jednotlivých počítačích (Výpočetní čas na počítačích).

Počet jedinců	Počet počítačů	Počet vláken	Čas výpočtu	Výpočetní čas na počítačích
10^5	1	1	3,9269	3,9269
10^5	1	4	1,4717	1,4717
10^5	2	8	3,1381	1,2735
10^5	3	12	2,3784	1,2627
10^5	4	16	2,2723	1,3184
10^5	5	20	2,0398	1,182
10^5	6	24	1,7818	1,2041
5.10^5	1	1	22,089	22,089
5.10^5	1	4	8,7518	8,7518
5.10^5	2	8	15,9722	6,8523
5.10^5	3	12	11,7353	6,7866
5.10^5	4	16	10,6136	6,8717
5.10^5	3	12	10,3734	7,1207
5.10^5	4	16	09,4265	7,6735

Tab. 7.1: Doba inicializace v závislosti na počtu počítačů a velikosti populace

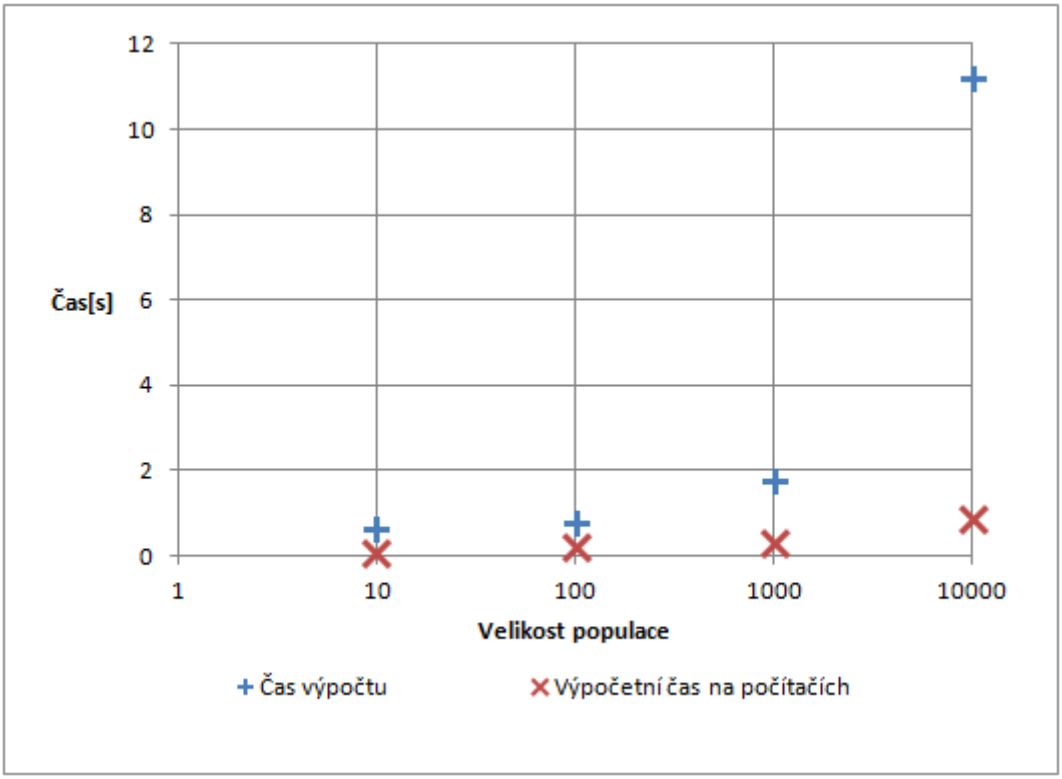
Dále bylo změřen výsledný čas evoluce. Měření bylo provedeno na šesti počítačích a na každém počítači ve čtyřech vláknech. Čas byl měřen jednak celkový (Čas výpočtu), jednak součet časů potřebných k výpočtu na jednotlivých počítačích (Výpočetní čas na počítačích).



Obr. 7.1: Graf závislosti času na počtu použitých vláken

Velikost populace	Čas výpočtu	Výpočetní čas na počítačích	Počet evolučních kroků
10	0,6322	0,0824	20
100	0,776	0,2226	20
1000	1,7543	0,3245	19,5
10000	11,2087	0,8598	18,5

Tab. 7.2: Doba běhu evoluce



Obr. 7.2: Graf závislosti doby evoluce na velikosti populace

7.2 SOAP

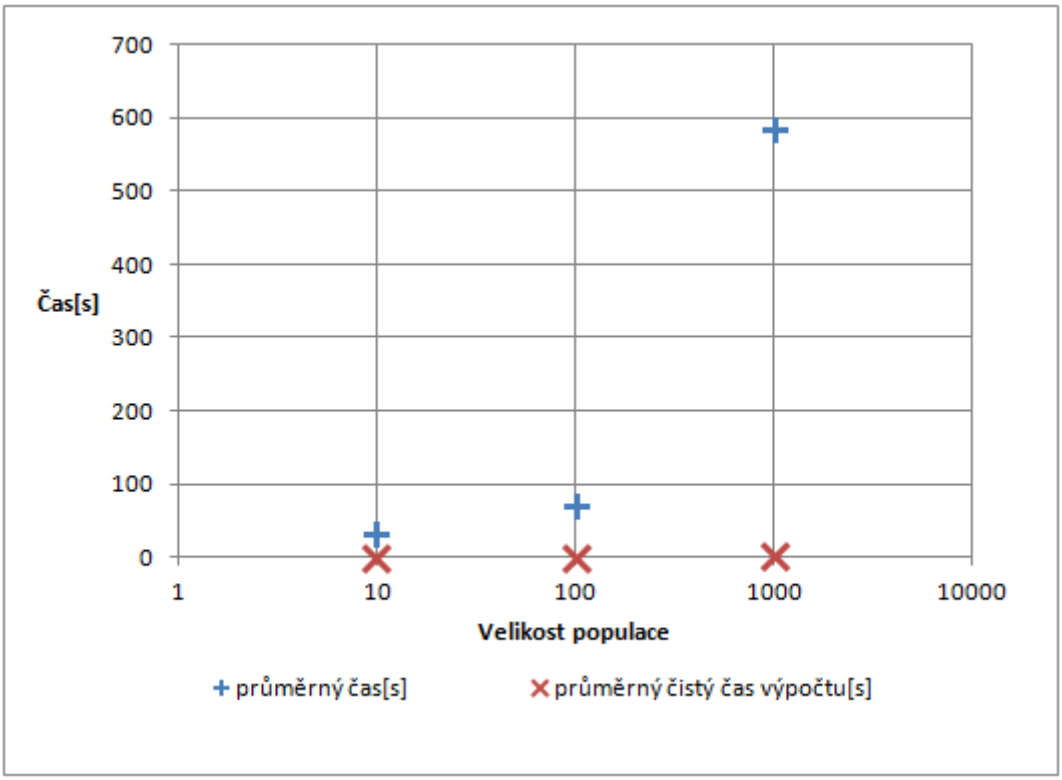
Protože při běhu programu s větší populací docházelo k chybám, byla změřena pouze doba evoluce pro menší počet jedinců. Podobně jako v předchozích případech je uváděn vždy průměr z deseti časů.

Velikost populace	Čas výpočtu	Výpočetní čas na počítačích	Počet evolučních kroků
10	32,797	0,0329	20
100	71,1447	0,016	20
1000	585,5587	0,266	20

Tab. 7.3: Doba běhu evoluce

7.3 Porovnání SOAP a Java RMI

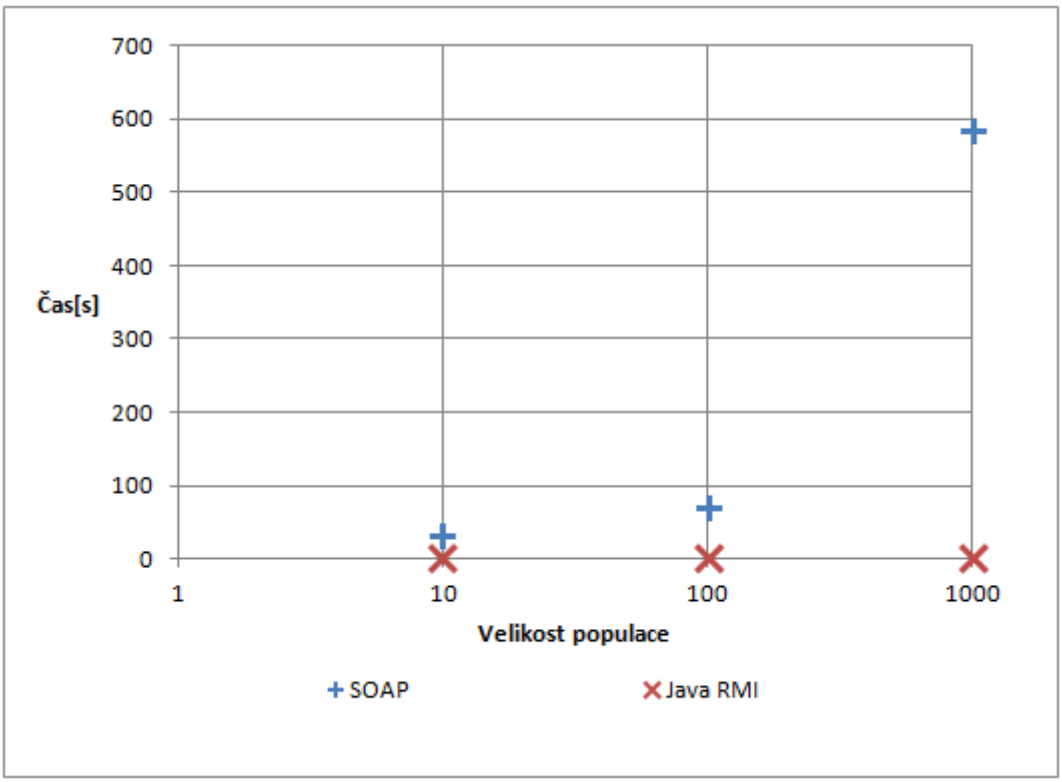
Porovnání naměřených výsledků



Obr. 7.3: Graf doby evoluce na velikosti populace

Velikost populace	Čas výpočtu SOAP	Čas výpočtu Java RMI
10	32,797	0,6322
100	71,1447	0,776
1000	585,5587	1,7543

Tab. 7.4: Porovnání SOAP a Java RMI



Obr. 7.4: Porovnání SOAP a Java RMI

8 ZÁVĚR

Byla vytvořena jednoduchá kalkulačka jako ukázka použití Javy RMI. Tato kalkulačka předvádí zadání vstupních hodnot na straně klienta, provedení operace na straně serveru a vypsání výsledku na obou stranách. Je zde vidět, že použití Javy RMI je jednoduché.

Byla obdržena aplikace řešící problém batohu od vedoucího práce. Tato aplikace byla zrychlena pomocí paralelizace. Byly porovnány doby běhu evoluce pří rozdělení průběhu algoritmu na různý počet vláken. Výsledné doby běhu jsou v tab. 5.1 a v tab. 5.2. Ukázalo se, že rozdělení do více vláken má význam až u větší populace. Hraniční velikost je zhruba tisíc jedinců.

Aplikace řešící problém batohu byla upravena pro distribuovaný výpočet. Při distribuování pomocí technologie Java RMI se ukázalo, že pro řešení problému baťohu je zpomalení přenosem dat významné. Toto je vidět v tab. 6.1 a obr. 6.1. Řešení problému baťohu vyžaduje velký přenos dat. S těmito daty provádí jednoduchou operaci. V případě, že by řešený problém potřeboval malý přenos dat a složité operace s daty, pravděpodobně by došlo ke zrychlení.

Výpočet distribuovaný pomocí SOAP nebyl úspěšný pro větší počet jedinců. Při menším počtu jedinců byl výpočet pomalý. Naměřené hodnoty jsou vidět v tab. 7.4 a obr. 7.4.

V porovnání SOAP a Java RMI má Java RMI výhodu v binárním přenosu dat a proto je rychlejší.

LITERATURA

[1] GROSSO, William. *Java RMI*. Beijing: O´Reilly, 2002, 545 s. ISBN 15-659-2452-5

[2] HAROLD, Elliotte Rusty. *Java network programming*. 3rd ed. Sebastopol, Calif.,: O'Reilly, c2005, xxii, 735 p. ISBN 05-960-0721-3.

[3] ORACLE CORPORATION. *Java IDL* [online]. 2011 [cit. 2013-11-19]. Dostupné z: <http://docs.oracle.com/javase/6/docs/technotes/guides/idl>.

[4] ORACLE CORPORATION. *Java RMI over IIOP* [online]. 2013 [cit. 2013-11-19]. Dostupné z: <http://docs.oracle.com/javase/7/docs/technotes/guides/rmi-iiop>.

[5] JEŘÁBEK, J. *Komunikační technologie*. Brno: Vysoké učení technické v Brně, 2013. s. 1-172. ISBN: 978-80-214-4713-4.

[6] ORACLE CORPORATION. *Java RMI over IIOP* [online]. [cit. 2013-11-19]. Dostupné z: <http://www.oracle.com/technetwork/java/rmi-iiop-139743.htm>

[7] OBJECT MANAGEMENT GROUP, Inc. *Java Language Mapping to OMG IDL Specification*. Dostupné z: <http://omg.org/cgi-bin/doc?ptc/00-01-06>

[8] OBJECT MANAGEMENT GROUP, Inc. *CORBA/IIOP 2.3.1 Specification, formal/99-10-07*. Říjen 1999. Dostupné z: <http://www.omg.org/cgi-bin/doc?formal/99-10-07.pdf>

[9] ORACLE CORPORATION. *The Java Tutorials: Trail: RMI*. [online]. [cit. 2013-10-28]. Dostupné z: <http://docs.oracle.com/javase/tutorial/rmi/index.html>

[10] KARÁSEK, Ing. Jan. *Teoretická informatika: Laboratorní úloha – Genetické algoritmy*.

[11] Latest SOAP versions. WORLD WIDE WEB CONSORTIUM. [online]. [cit. 2014-06-01]. Dostupné z: <http://www.w3.org/TR/soap/>

[12] Sun Java System Message Queue 4.3 Developer's Guide for Java Clients. ORACLE CORPORATION. [online]. [cit. 2014-06-01]. Dostupné z: <http://docs.oracle.com/cd/E19340-01/820-6767/index.html>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ISO Mezinárodní organizace pro normalizaci (anglicky International Organization for Standardization)

OSI Propojení otevřených systémů (anglicky Open Systems Interconnection)

TCP Transmission Control Protocol

UDP User Datagram Protocol

IP Internet Protocol

RMI Remote Method Invocation, vzdálené volání metod

JAAS Java Authentication and Authorization Service, Java autentizační a autorizační služby

URL Uniform Resource Locator, jednotný lokátor zdrojů

HTTP Hypertext Transfer Protocol

FTP File Transfer Protocol

SMTP Simple Mail Transfer Protocol

IIOP Internet InterORB Protocol

JDK Java Development Kit

IDL Object Management Group Interface Definition Language, jazyk pro popis rozhraní

CORBA Common Object Request Broker Architecture

OMG Object Management Group

SOAP Simple Object Access Protocol

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

- Elektronická verze bakalářské práce
- Zdrojové kódy jednoduché aplikace klient-server
- Zdrojové kódy aplikace řešící problém batohu