Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jiří Novák

Systém pro distribuované výpočty

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Vyskočil, Ph.D.

Studijní program: Informatika, Správa počítačových systémů

2008

Děkuji RNDr. Jiřímu Vyskočilovi, Ph.D. za trpělivé a soustavné vedení při zpracování bakalářské práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 12.12.2008 Jiří Novák

.....

Obsah

1 Úvod....6

2 Distribuované výpočty....7

2.1 Úvod....7

2.2 SETI @ HOME....8

2.3 PVM (Parallel Virtual Machine)....8

3 Popis navrženého systému....9

3.1 Základní vlastnosti....9

3.2 Vstupy....10

3.2.1 Problémy spojené s využitím dodaných DLL....11

3.3 Procesy....11

3.3.1 P1 - centrální serverový proces....11

3.3.2 P2 - serverový rozdělovací proces....12

3.3.3 P3 - zákaznický klientský proces....12

3.3.4 P4 - centrální klientský proces....12

3.3.5 P5 - klientský výpočetní proces....12

3.4 Plánovací algoritmus....13

3.4.1 Úvod....13

3.4.2 Algoritmus....13

3.5 Spojení a zprávy....16

3.5.1 Seznam definovaných zpráv....16

3.5.2 Spojení....17

3.6 API....18

3.6.1 Struktury....18

3.6.2 Funkce serverové DLL....19

3.6.3 Funkce klientské DLL....19

4 Porovnání s existujícími systémy....20
4.1 Porovnání se SETI @ HOME....20
4.2 Porovnání s PVM....20
4.3 Výhody a nevýhody navrženého řešení....21
5 Možnosti použití....22
5.1 SETI @ HOME na navrženém systému....22
5.2 Vnitrofiremní výpočet....22
6 Závěr, možnosti dalšího pokračování....24
6.1 Závěr....24
6.2 Možnosti pokračování....24
6.2.1 Zvýšení spolehlivosti....25
6.2.2 Zvýšení výkonu....25
6.2.3 Rozšíření funkčnosti....25
7 Literatura....28

Název práce: Systém pro distribuované výpočty

Autor: Jiří Novák

Katedra (ústav): Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Vyskočil, Ph.D.

e-mail vedoucího: Jiri.Vyskocil@mff.cuni.cz

Abstrakt: Práce se zabývá řešením úloh prostřednictvím systémů pro distribuované výpočty. Na úvod seznamuje čtenáře se základními principy těchto systémů a představuje některé běžně používané systémy. Dále je pak navržen vlastní systém pro distribuované výpočty. Tento systém je popsán včetně struktury, procesů, navazovaných spojení a posílaných zpráv. Dále je provedeno porovnání navrženého systému s existujícími systémy a vyhodnoceny jeho výhody a nevýhody. Nakonec je uvedeno několik myšlenek pro další možná vylepšení a rozšíření navrženého systému.

Klíčová slova: distribuovaný, paralelní, výpočet, grid, Internet

Title: System for Distributed Computing

Author: Jiří Novák

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Vyskočil, Ph.D.

Supervisor's e-mail address: Jiri.Vyskocil@mff.cuni.cz

Abstract: This thesis is about solving problems using systems for distributed computing. In the beginning, it provides the reader brief introduction to the area of systems for distributed computing and introduces some of the existing systems. Further there is a proposal of a new distributed computing system. Design of this system is described thoroughly including the processes, network connections and messages which are used for communication between the processes. The system is then compared to existing implementations. Advantages and disadvantages are discussed. At the end there are some ideas how the system can be improved in the future.

Keywords: distributed, parallel, computing, grid, Internet

1 Úvod

Cílem této bakalářské práce je navrhnout systém pro provádění distribuovaných výpočtů zejména v prostředí Internetu. Systém by měl být použitelný pro úlohy dělitelné na menší, samostatně řešitelné části. Měl by umožňovat běh více úloh paralelně.

Distribuovaný výpočet v prostředí Internetu přináší další problémy a požadavaky:

- Problémy s konektivitou – systém by měl být schopný se zotavit z výpadku spojení mezi stroji.
- Problémy se spolehlivostí hardwaru a softwaru na výpočetních strojích – vzhledem k nekontrolovatelnému stavu výpočetních strojů je nutné, aby systém byl odolný vůči pádu výpočetních procesů.
- Uživatelé – uživatelé jsou z větší části laikové, takže instalace a provoz na jejich strojích by měly být co nejjednodušší.

Dále by mělo být ověřeno, že navržený systém je pro tento účel použitelný.

Rozvržení do kapitol

V druhé kapitole se seznámíme s tím, co jsou to distribuované výpočty a stručně si představíme některé existující implementace.

Ve třetí kapitole čtenář najde podrobný popis navrženého systému.

Ve čtvrté kapitole provádím porovnání systému s jinými systémy.

V páté kapitole popisuji možnosti využití.

V závěru uvádím několik možností, jak by šlo systém dále vylepšovat.

2 Distribuované výpočty

2.1 Úvod

Definice: *Distribuovaný systém* je takový systém propojení množiny nezávislých uzlů, který poskytuje uživateli dojem jednotného systému.

O distribuovaných systémech blíže pojednává například [9]. Pod distribuované systémy zahrnujeme množství různých systémů, mezi jejichž hlavní cíle patří přdevším vysoký výpočetní výkon a vysoká spolehlivost. Toho lze dosáhnout mnoha různými způsoby.

Propojení na fyzické úrovni

Do systémů propojených na fyzické úrovni jde zahrnout vícejádrové a víceprocesorové počítače a superpočítače. Výhodou takových systémů je naprostá kontrola nad výpočetními prostředky. Nevýhodou je, zejména při vysokých výkonech, poměrně vysoká cena za jednotku výpočetního výkonu. Zástupcem takového systému je například v současnosti nejvýkonnější superpočítač IBM Roadrunner (viz. [4]) dosahující výpočetního výkonu 1100 Tflops (bilionů operací za sekundu).

Clustery

Cluster je skupina propojených počítačů, které spolu úzce spolupracují na řešení zadaných úloh. Clustery dosahují vysokých výpočetních výkonů při přiměřené ceně za jednotku výkonu. Jsou ale závislé na rychlém spojení každého výpočetního uzlu s každým. Tím pádem jsou použitelné zejména pro systémy běžící v rámci jedné společnosti.

Grid

Grid (viz [10]) je skupina propojených počítačů, která se liší od clusteru tím, že řeší části úlohy nezávisle na sobě. Komunikace tedy probíhá jen mezi klienty a serverem, nikoliv mezi klienty navzájem. Typickým příkladem gridu je projekt SETI @ HOME, který si blíže představíme v další podkapitole. Výhodou gridu je možnost snadno zapojit do výpočtů velké množství uživatelů připojených přes Internet.

V následujících podkapitolách uvedu několik existujících systémů pro distribuované výpočty.

2.2 SETI @ HOME

Jedním z nejznámějších projektů z oblasti distribuovaných výpočtů je SETI @ HOME (viz [2]). Cílem tohoto projektu je najít v datech zaznamenaných z radioteleskopu nějaké signály, které by naznačovaly existenci mimozemských civilizací.

Technicky vzato jde o aplikaci poměrně jednoduchých matematických postupů na velmi velké množství dat, s tím že výpočty jednotlivých částí jsou na sobě nezávislé. Úloha je tedy distribuovaný výpočet s potřebou velkého počtu výpočetních jednotek a bez potřeby vzájemné komunikace mezi nimi – ideální struktura pro takovou úlohu je grid.

Systém se skládal ze serveru a výpočetního programu, který běžel jako spořič obrazovky na mnoha počítačích, z nichž každý zkoumal přidělenou část dat.

2.3 PVM (Parallel Virtual Machine)

PVM (viz [3], [5], [6]) je systém, který umožňuje, aby byla skupina počítačů (se stejnými nebo různými operačními systémy a architekturami) používána jako jeden velký paralelní počítač. Skládá se z několika nástrojů a knihoven.

Základní jednotkou výpočtu je *task*, který zhruba odpovídá unixovému procesu.

Běžící systém se skládá z množiny počítačů propojených sítí, na kterých běží jednotlivé tasky. PVM knihovny poskytují taskům API pro spouštění dalších tasků a předávání zpráv mezi běžícími tasky.

Při spouštění nové úlohy je potřeba její soubory rozkopírovat na jednotlivé stroje.

Tento systém podporuje intenzivní komunikaci mezi tasky, je tedy vhodný pro nasazení do clusteru.

3 Popis navrženého systému

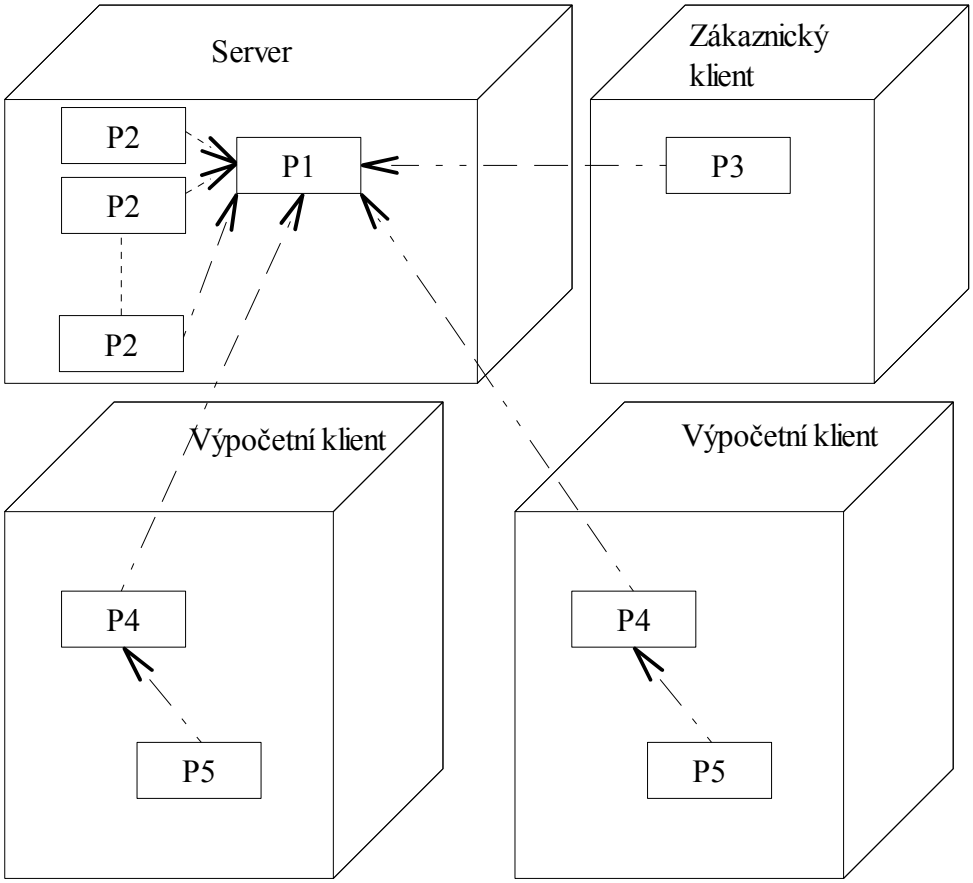
3.1 Základní vlastnosti

Systém je určen zejména pro běh v prostředí Internetu, zvolená struktura tedy odpovídá gridu. Vzhledem k tržním podílům operačních systémů ([7]) jsem se rozhodl navrhnout systém pro běh na 32-bitovém operačním systému Microsoft Windows XP. Mělo by ale být možné systém upravit pro jakoukoliv platformu, která podporuje vlákna, synchronizaci mezi vlákny, dynamicky linkované knihovny a TCP/IP přístupné přes BSD socket API nebo podobné rozhraní. Systém je implementován v jazyce C v prostředí Microsoft Visual C++.

Systém se skládá z pěti druhů procesů. Pro zvýšení spolehlivosti spolu jednotlivé instance procesů komunikují přes TCP. K síťové komunikaci je používána knihovna Winsock ve verzi 1.1 ([8]).

- P1 - centrální serverový proces
- P2 - serverový rozdělovací proces
- P3 - zákaznický klientský proces
- P4 - centrální klientský proces
- P5 - klientský výpočetní proces

Na následujícím diagramu je znázorněn systém, do kterého jsou zapojeni dva výpočetní klienti a ve kterém běží několik rozdělovacích procesů. Šipky znázorňují síťová spojení a směr jejich navazování. Spojení jsou trvale navázána po celou dobu běhu procesů. Kvádry znázorňují jednotlivé počítače. Server je vždy jen jeden, zákaznických klientů a výpočetních klientů může být libovolné množství. Na serveru běží vždy jen jeden proces P1, počet procesů P2 je stejný jako počet aktuálně zadaných úloh.



Obr (1)

3.2 Vstupy

Ten, kdo zadává úlohu do systému (dále zákazník), dodá celkem tři soubory.

- DLL, která zařizuje rozdělování úlohy na části (dále serverová DLL)
- DLL, která takové části řeší (dále klientská DLL)
- datový soubor obsahující zadání úlohy (dále vstupní data)

Tyto soubory jsou pak distribuovány příslušným procesům a po vyřešení úlohy smazány.

3.2.1 Problémy spojené s využitím dodaných DLL

Tím, že systém používá již zkompilované soubory od zákazníka, vzniká několik potenciálních problémů.

Knihovna s chybným kódem

Knihovna může obsahovat nějaký chybný kód, který bude způsobovat pády výpočtu. Takovým kódem může být například špatná práce s paměťovými ukazateli, která vede k ukončení procesu operačním systémem. Aby se předešlo pádu celého systému, byly vytvořeny procesy P2 a P5, ve kterých běží veškeré rozdělování, spojování a výpočty.

Záměrně škodlivá knihovna

Systém není odolný vůči záměrně škodlivé knihovně, která bude provádět například mazání souborů z disku.

Pokud tedy úlohy zadávají zákazníci, kteří nejsou důvěryhodní, může být vhodné od nich požadovat zdrojové kódy, ty zkontrolovat a zkompilovat.

3.3 Procesy

3.3.1 P1 - centrální serverový proces

Úkoly procesu P1

- rozdělování práce mezi klienty
- spouštění P2 pro zadávané úlohy
- udržování seznamu úloh
- udržování seznamu klientů
- sledování času věnovaného úlohám
- distribuce DLL klientům

P1 je centrální proces, ke kterému se připojují procesy P2, P3 a P4. Běží pouze v jedné instanci na stroji s předem známou IP adresou. Je to řídící proces celého systému.

3.3.2 P2 - serverový rozdělovací proces

Úkoly procesu P2

- generování částí úlohy
- zpracování částečných výsledků

- držení výsledků po skončení výpočtu

P2 existuje v jedné instanci pro každou řešenou úlohu. Do P2 se po startu linkuje serverová DLL. P2 musí běžet na stejném počítači jako P1. Výsledky jsou po ukončení výpočtu uloženy do souboru na určené místo. P2 se po startu připojuje na P1.

3.3.3 P3 - zákaznický klientský proces

Úkoly procesu P3

- zadání úlohy do systému
- zjištění výsledků

P3 posílá soubory do systému a určuje prioritu úlohy. Po odeslání dostane identifikátor výpočtu a čeká na výsledky. Během čekání se může odpojit. V takovém případě pro zjišťování výsledků používá identifikátor výpočtu a uživatelské jméno. Instance P3 nejsou omezeny.

3.3.4 P4 - centrální klientský proces

Úkoly procesu P4

- spouštění P5
- ukončování P5

Na příkaz od P1 provádí spouštění P5. Zajišťuje stažení klientských DLL. Zprostředkovává komunikaci mezi P1 a P5. Po startu se připojuje na P1.

3.3.5 P5 - klientský výpočetní proces

Úkoly procesu P5

- výpočet částí úlohy

P5 počítá části úlohy, které generuje P2. Do P5 se linkuje klientská DLL. P5 má maximálně jednu instanci na každém výpočetním klientovi. P5 musí běžet na stejném počítači jako P4. Po startu se P5 připojuje k P4.

3.4 Plánovací algoritmus

3.4.1 Úvod

Rozdělování výpočetního výkonu mezi úlohy je realizováno spouštěním a ukončováním procesů P5 na klientech. Proces P5 nemůže být ukončen dříve, než spočítá přidělenou část úlohy, plánování tedy není preemptivní. Rozdělení výkonu je přepočítáno pokaždé, když P1 dostane zprávu RESULTS, když se k ní připojí nový proces P4 nebo když skončí nějaká úloha. Výpočetní výkon je mezi úlohy rozdělován podle priority, a to exponenciálně. Úloha s o 1 vyšší prioritou tedy dostane 2-krát (konstanta PRIORITY_POWER_BASE) vyšší výkon. Ke spouštění nebo ukončování procesů P5 dochází jen tehdy, pokud se vypočtené rozdělení výkonu liší od aktuálního alespoň o 50% (konstanta TASKING_TRESHOLD) u některé z úloh. Příliš nízké nastavení této konstanty může způsobit příliš časté střídání úloh na klientech a tím i snížení výkonu. Veškeré rozdělovací výpočty provádí P1.

3.4.2 Algoritmus

Události vedoucí k přepočtu

Každou událost, která vyvolá přepočítávací algoritmus nazveme tik. Jsou to tyto události:

- připojení nového výpočetního klienta
- ukončení výpočtu úlohy, zpracovává se jako několik prvních událostí
- přijetí částečných výsledků

Vznik nové úlohy není takovou událostí, protože nemůže být předčasně ukončen žádný P5.

Relace

Pro potřeby plánování si systém udržuje následující relace:

HISTORIE(tik_id, klient_id, uloha_id), která pro tik a klienta určuje, která úloha byla na daném klientovi od daného tiku řešena.

KLIENT(klient_id) - seznam klientů

ULOHA, která pro danou úlohu určuje prioritu a uživatele, který ji zadal.

TIK(cas), která udává časy tiků

Z těchto relací jsou vyřazovány události starší než TICKS_REMEMBERED tiků a výpočetní klienti, kteří už se před tou dobou odpojili.

Krok algoritmu

1. Zaznamenej tik do relace TIK
2. V relaci HISTORIE sečti čas všech dvojic klient - tik pro každou úlohu, která ještě není ukončená. Výsledek bude relace STRAVENO(uloha_id, cas), která udává pro každou úlohu jaký čas byl stráven jejím řešením během poslední doby.
3. Posčítej časy v relaci STRAVENO. Výsledek bude číslo CELKEM_STRAVENO.
4. Vyděl číslem CELKEM_STRAVENO časy v relaci STRAVENO. Výsledek bude relace STRAVENO1(uloha_id, cast_casu)
5. Pro každou neukončenou úlohu z relace ULOHA umocni PRIORITY_POWER_BASE na prioritu. Výsledek bude relace MA_STRAVIT(uloha_id, hodnota)
6. Posčítej hodnoty v relaci MA_STRAVIT. Výsledkem bude číslo CELKEM_MA_STRAVIT.
7. Vyděl číslem CELKEM_MA_STRAVIT hodnoty v relaci MA_STRAVIT. Výsledek je relace MA_STRAVIT1(uloha_id, cast_casu).
8. Vytvoř relaci HODNOCENI(uloha_id, schazi), kde *schazi* spočítáš jako 100 - 100*(STRAVENO1.cast_casu / MA_STRAVIT1.cast_casu)

Zde nastává dělení podle toho, jestli zpracováváme událost typu 1 nebo 2.

Typ 1 (připojení nového výpočetního klienta)

1. Vezmi z relace HODNOCENI hodnotu *uloha_id* s nejvyšší hodnotou *schazi*.

2. Spusť na klientovi P5 řešící tuto úlohu. Toto je provedeno posláním zprávy FILES.
3. Pošli procesu P2, který přísluší k této úloze, zprávu PART_OF_PROBLEM_WANTED.
4. Proveď záznam do relace HISTORIE.

Typ 2 (přijetí částečných výsledků)

1. Najdi v relaci HODNOCENI hodnotu *uloha_id* s nejvyšší hodnotou *schazi*.
2. Je hodnota *schazi* větší než TASKING_TRESHOLD?

ANO: klient bude řešit jinou úlohu než doteď

1. Pošli příslušnému P4 zprávu FILES se souborem od vybrané úlohy.
2. Pošli příslušnému P2 zprávu PART_OF_PROBLEM_WANTED.
3. Proveď záznam do relace HISTORIE.

NE: klient bude řešit stejnou úlohu jako doteď

1. Pošli příslušnému P2 zprávu PART_OF_PROBLEM_WANTED.
2. Proveď záznam do relace HISTORIE.

3.5 Spojení a zprávy

3.5.1 Seznam definovaných zpráv

Zde je seznam zpráv posílaných v rámci systému včetně odesílatele a příjemce.

Název	Kdo odesílá	Komu odesílá
PART_OF_PROBLEM	P2	P1
QUERY	P3	P1
PROBLEM	P3	P1
PARTIAL_SOLUTION	P4	P1
PARTIAL_SOLUTION	P1	P2
PART_OF_PROBLEM_WANTED	P1	P2
PROBLEM_ID	P1	P3
SOLUTION	P1	P3
FILES	P1	P4
PART_OF_PROBLEM	P1	P4
PARTIAL_SOLUTION	P5	P4
PART_OF_PROBLEM	P4	P5
TERMINATE	P1	P1
PROBLEM_ID	P2	P1
P2_NOT_CONNECTED	P1	P3
P5_DISCONNECTED	P4	P1
P4_DISCONNECTED	P1	P2

Tab (1)

3.5.2 Spojení

P2 – P1

Spojení mezi rozdělovacím klientem a centrálním serverem. Toto spojení navazuje P2. Na začátku pošle P2 identifikátor úlohy (zpráva PROBLEM_ID), aby si P1 mohl přiřadit navázané spojení k úloze. Dále se po tomto spojení posílají následující zprávy:

- PART_OF_PROBLEM – touto zprávou posílá P2 nově vygenerovanou část zadání. Zároveň tato zpráva obsah informaci o tom, na kolika výpočetních klientech by rozdělovací proces chtěl, aby byla odesílaná část zadání řešena.
- PARTIAL_SOLUTION – touto zprávou posílá P1 spočítanou část řešení.
- PART_OF_PROBLEM_WANTED – touto zprávou posílá P1 žádost o vygenerování nového kusu zadání.
- P4_DISCONNECTED – touto zprávou P1 uvědomuje P2 o tom, že při výpočtu zadané části úlohy došlo k problému a zadaná část nebyla vyřešena.

P3 – P1

Spojení mezi zákaznickým klientem a centrálním serverem. Toto spojení navazuje P3. Při použití v síti Internet je nutné omezit síťovými prostředky (například firewallem) navazování tohoto spojení jen na důvěryhodné klienty. Spojení je navazováno ve dvou případech:

Zadání úlohy do systému

V tomto případě posílá P3 zprávu PROBLEM, ve které jsou obsaženy obě DLL a datový soubor. P1 v takovém případě:

1. uloží soubory
2. vytvoří novou úlohu ve svých strukturách
3. přiřadí nové úloze *id*
4. spustí rozdělovací P2 proces
5. počká na jeho připojení
6. odešle na P3 zprávu PROBLEM_ID

V případě, že se P2 z nějakého důvodu nepřipojí, pošle P1 na P3 zprávu P2_NOT_CONNECTED a P3 poté ohlásí uživateli, že zadání nebylo úspěšné.

Zjišťování stavu úlohy

P3 posílá zprávu QUERY s *id* úlohy, P1 odpovídá zprávou SOLUTION, ve které buďto oznámí, že úloha ještě není vyřešena, nebo odešle datový soubor s výsledky.

P4 – P1

Spojení mezi centrálním klientským a centrálním serverovým procesem. Toto spojení navazuje P4 hned po svém spuštění. Přenášené zprávy jsou tyto:

- PARTIAL_SOLUTION – touto zprávou P4 posílá spočítanou část řešení.
- FILES – touto zprávou P1 posílá klientskou DLL, která má být přilinkována do nově spuštěného P5 procesu. P5 je spuštěn procesem P4.
- PART_OF_PROBLEM – touto zprávou P1 posílá část zadání k vyřešení.
- P5_DISCONNECTED – touto zprávou P4 upozorňuje na to, že P5 se odpojil a tím pádem nebyla zadaná část zadání vyřešena.

P5 – P4

Spojení mezi centrálním klientským a výpočetním klientským procesem. Toto spojení navazuje P5 po svém spuštění. Přenášené zprávy jsou tyto:

- PART_OF_PROBLEM – touto zprávou P4 posílá část zadání k vyřešení.
- PARTIAL_SOLUTION – touto zprávou P5 posílá řešení zadané části zadání

P1 – P1

Spojení mezi jednotlivými vlákny procesu P1 je navazováno jen při ukončování běhu celého systému. Těmito spojeními jsou ukončována čekání prováděná funkcí *accept()*. Jediná zpráva posílaná po tomto spojení je TERMINATE.

3.6 API

3.6.1 Struktury

V API se používá jen jedna nová struktura *kus_dat*.

```
typedef struct
{
      char * data;
      int length;
} kus_dat;
```

3.6.2 Funkce serverové DLL

int prisel_vysledek(kus_dat vysledek, int id)

Tato funkce je volána, pokud od P1 přišla část řešení. Jejím úkolem je zapracování příchozí části do celkového řešení. Funkce vrací nulu, pokud úloha ještě není vyřešená, a jedničku pokud zapracováním právě příchozí části řešení byla úloha vyřešena celá.

*int dalsi_kus(kus_dat * data, int id, int * pocet_p4)*

Tato funkce je volána v případě, že systém chce počítat další část dané úlohy. Parametr id obsahuje identifikátor přiřazený vygenerované části zadání a je používán, aby funkce *prisel_vysledek()* mohla spárovat části zadání a řešení. Funkce vrací nulu, pokud úloha má ještě další části zadání, a jedničku pokud je nemá. Výstupní parametr slouží k předání informace o tom, na kolika výpočetních klientech má tato část zadání běžet.

*void dealokuj_kus_dat(kus_dat * data)*

Tato funkce provádí dealokaci paměti alokované ve funkcích serverové DLL. Její použití je nutné z toho důvodu, že DLL mají oddělený heap a uvolnění pomocí prostředků hlavního programu není možné použít.

void selhani_vypoctu(int id)

Tato funkce je volána pokud dojde na výpočetním klientovi k selhání procesu P4 nebo P5, který právě řešil část zadání s daným id.

3.6.3 Funkce klientské DLL

*int vypocet(kus_dat * vystup, kus_dat vstup)*

Tato funkce provádí výpočet jedné části zadání.

*void dealokuj_kus_dat(kus_dat * data)*

Tato funkce provádí dealokaci paměti alokované ve funkcích klientské DLL.

4 Porovnání s existujícími systémy

4.1 Porovnání se SETI @ HOME

Původní systém SETI @ HOME byl motivací a inspirací pro tuto práci. Navržený systém se od něj liší především svoji schopností řešit úlohy, které nejsou předem známy a také tím, že může řešit více úloh najednou. Tomu také odpovídá struktura systému – existuje samostatný proces P2, který zařizuje rozdělování zadání a spojování výsledků vždy pro jednu úlohu.

Během tvorby této práce přešlo SETI @ HOME na nový systém BOINC (viz [1]), který rovněž umožňuje řešení více různých úloh a je tak poměrně blízký mnou navrženému řešení.

Od systému BOINC se navržený systém liší například v tom, že o přiřazení úloh na klienty rozhoduje systém, nikoliv nastavení klientů. Výhodou pro systém u tohoto přístupu je, že dokáže řešit i nepopulární úlohy, kterým by jinak výpočetní klienti nepřidělovali dostatek výkonu. Zároveň umožňuje rychlý start zpracování úlohy – vzhledem k tomu, že nová úloha neběží na žádných výpočetních klientech, bude jí plánovač přidělovat většinu nových a uvolněných výpočetních klientů, dokud nebude mít odpovídající podíl na využívaném výpočetním výkonu.

4.2 Porovnání s PVM

- PVM je systém, který je z uživatelského hlediska podobný jednomu stroji s velkým počtem procesorů bez sdílené paměti, Jednotlivé *tasky* běží nezávisle a pouze si posílají zprávy.
- PVM neposkytuje prostředky pro zvýšenou spolehlivost.
- Při vývoji aplikací určených k běhu v systému PVM je potřeba využívat PVM knihovny.

Výhodou navrhovaného řešení je vyšší spolehlivost a jednodušší práce zadavatele úlohy. Nevýhodou je obtížnější řešení v případě úloh, které nejsou dobře dělitelné na nezávislé části a při kterých je potřeba interaktivní komunikace mezi výpočetními jednotkami.

4.3 Výhody a nevýhody navrženého řešení

Jednoduchost pro zadavatele úloh

Zadavatel úlohy se při vývoji knihoven nemusí starat o spouštění a ukončování procesů, síťovou komunikaci, posílání zpráv, synchronizaci. Podstatné je pouze aby úloha splňovala podmínky zadání – řešitelnost částí zadání nezávisle na ostatních částech.

Nezávislost na programovacím jazyku

Knihovny, které se dodávají jako součást vstupu, jsou již zkompilované. Je tedy možné je vytvářet v kterémkoliv jazyce, který umožňuje vytváření dynamicky linkovaných knihoven. Systém v tomto směru nemá žádná omezení.

5 Možnosti použití

5.1 SETI @ HOME na navrženém systému

Pomocí navrženého systému by šlo snadno napodobit funkčnost původního systému SETI @ HOME.

Klientská DLL by, stejně jako klient v původním systému, uměla zpracovat jednotlivé kusy záznamu rádiových vln.

Serverová DLL by shromažďovala výsledky zpracování od jednotlivých klientů.

Data ke zpracování v případě SETI @ HOME přicházejí průběžně, zatímco systém podporuje jen zadávání úlohy a zjišťování výsledků vcelku. Tento problém se dá řešit dvěma způsoby.

- Kouskováním úlohy – vznikala by nová úloha vždy pro nějaké období přijatých dat a takové úlohy by se řešily odděleně. V takovém případě by bylo vhodné starším úlohám dávat větší prioritu, aby nedocházelo k vyhladovění starších úloh.
- Obejitím P3 procesu pro zadávání a zjišťování výsledků – rozdělovací proces by mohl přímo komunikovat s nějakým serverem, který by mu poskytoval nová příchozí data a odebíral si výsledky. Tento postup přináší jednu výhodu. SETI @ HOME je úloha, která hledá informace v datech a výsledek není závislý na zpracování všech vstupních dat. Takovéto řešení by tedy umožnilo oznámení dosaženého výsledku dříve než by všechna data byla zpracována.

5.2 Vnitrofiremní výpočet

Vhodné prostředí pro použití navrženého systému může být i rozsáhlá firemní síť. V takové síti se běžně nacházejí stovky až tisíce počítačů, které jsou mimo pracovní dobu zcela nevytížené. Ani v pracovní době není jejich vytížení příliš vysoké, protože na nich běží jen běžné kancelářské aplikace. Taková síť je ideální pro instalaci systému pro distribuované výpočty. Obvykle poskytuje dobré spojení mezi jednotlivými stroji. Neklade nároky na zabezpečení, vzhledem k tomu, že komunikace obvykle probíhá jen po místní síti a pokud je například pobočka připojena vzdáleně, bývá to pomocí spojení šifrovaného na nižších síťových vrstvách.

Uveďme příklad: Maloobchodní řetězec, který zaměstnává v kancelářích několik stovek lidí, chce odhadnout zisk z připravovaných slevových akcí.

Firemní IT oddělení se rozhodne použít náš systém a nasadit ho na nepříliš využívané kancelářské počítače. Při tomto rozhodování může hrát roli i fakt, že takto se nedostanou žádná data ani odhadovací modely mimo firemní infrastrukturu.

Instalace systému na jednotlivé počítače je celkem jednoduchá. S instalací na velké množství počítačů může pomoci vhodné nastavení Windows domény.

Rozdělovací proces v takovém případě bude generovat části zadání skládající se z parametrů slevové akce a ze všech dalších údajů potřebných pro výpočet.

Výstupem z výpočetního klienta pak bude pro každou slevovou akci vydělaná/prodělaná částka.

Na výstupu systém dodává ke každé slevové akci i očekávaný zisk.

Navržený systém by tedy byl použitelný i pro řešení takovéto úlohy.

6 Závěr, možnosti dalšího pokračování

6.1 Závěr

V rámci této práce jsem čtenáře seznámil se základními pojmy a technologiemi z oblasti distribuovaných výpočtů. Navrhl jsem systém pro provádění distribuovaných výpočtů. Uvedl jsem také dvě jednoduché případové studie.

Navržený systém byl v téměř plném rozsahu naimplementován a vyzkoušen. Pracoval bez problémů na dvou platformách – Windows XP Professional 32bit na architektuře Intel x86, a Windows Vista Home Premium 64bit na architektuře Intel x64. V obou případech byly použity stejné binární soubory vytvořené 32-bitovým kompilátorem.

Použitelnost v prostředí Internetu

Nyní vyhodnotíme splnění požadavků na systém.

Schopnost řešit úlohy

- Systém je schopen předávat části zadání a řešení mezi svými procesy a zároveň voláním příslušných funkcí řešit části zadání a zapracovávat části řešení do celkového řešení.

Spolehlivost vůči pádům a odpojením

- V případě, že dojde k odpojení P5 od P4 nebo P4 od P1, zajistí systém opakované spuštění dané části zadání na dalším výpočetním klientovi.
- Ošetřování pádu centrálního procesu P1 není prakticky možné.
- Pád rozdělovacího procesu P2 způsobí ukončení výpočtu příslušné úlohy.

Jednoduchost pro výpočetní uživatele

- Uživatel si nakopíruje jen soubory p4.exe, p5.exe a konfigurační soubor s adresou serveru P1. Poté spustí p4.exe a od toho okamžiku se nemusí o nic dalšího starat.

Navržený systém tedy splňuje požadavky pro použití na velkém množství počítačů připojených přes Internet.

6.2 Možnosti pokračování

Systém poskytuje prostor pro další vylepšení. Některá z nich jsou blíže popsána v této podkapitole.

6.2.1 Zvýšení spolehlivosti

Systém v současné podobě je zaměřen především na vysoký výpočetní výkon. Bylo by možné doplnit nebo upravit některé postupy tak, aby byla zvýšena spolehlivost a zjednodušen vývoj knihoven u zákazníků.

Příklad: Pokud je v současné verzi počítána ta samá část zadání na několika výpočetních klientech, dostává rozdělovací proces P2 řešení od všech výpočetních klientů hned po jejich spočítání. Možnou změnou by bylo, pokud by systém počkal, než dostane výsledky od všech výpočetních klientů, porovnal je a v případě, že by dané procento výsledků bylo stejných, předal by je rozdělovacímu procesu.

6.2.2 Zvýšení výkonu

Současné řešení má také prostor pro další zvýšení výpočetního výkonu.

Lepší využití výpočetních klientů

- Pokud má výpočetní klient více jader nebo procesorů, mohl by řešit více úloh najednou.

Benchmarkování výpočetních klientů

- Na výpočetních klientech by se při startu spouštěl test výpočetního výkonu. K jeho výsledkům by poté mohl přihlížet plánovací algoritmus.

Optimalizace dodávaných DLL pro různé architektury a procesory

- Systém by udržoval informaci o tom, na jakých strojích běží výpočetní klienti a podle toho by jim posílal DLL optimalizované například pro procesory s podporou různých verzí SSE.

Využití možností UNIXu

- V případě implementace systému pro operační systém UNIX je možné pro komunikaci mezi procesy na stejném stroji (P1 – P2, P4 – P5) nahradit spojení přes TCP/IP využitím UNIXových socketů nebo sdílené paměti.

6.2.3 Rozšíření funkčnosti

Systém by mohl mít i další vylepšení:

Sdílená paměť

- Pokud by byl systém využíván i v síťových prostředích s nízkými latencemi (typicky sítě v rámci jedné firmy, budovy a podobně), bylo by možné využít sdílenou paměť přístupnou pro všechny procesy, s jednotným adresováním.
- Bylo by vhodné sdílenou paměť doplnit o synchronizaci.

Multiplatformnost

- Systém by mohl mít výpočetní klienty pro různé operační systémy a architektury.
- Zákazník by dodal výpočetní knihovny pro různé operační systémy a systém by je poté předával výpočetním klientům podle toho, na jakém operačním systému běží.

Algoritmus plánování úloh

- Plánovací algoritmus poskytuje velký prostor pro experimenty, zvláště při vysokém zatížení a velkém počtu výpočetních klientů. Ať už pro změnu konstant v navrženém algoritmu nebo pro implementaci úplně nového algoritmu.

Offline výpočet

- Výpočetní klient by se mohl dočasně odpojit od systému a po vyřešení své části zadání se opět připojit.

Bezpečnost

- Mohlo by být doplněno šifrování dat přenášených po síti.
- Byl by implementován systém uživatelů s tím, že každý výpočetní a zákaznický klient, který se připojuje, by se musel autentizovat.
- Pokud by byl systém zaměřen na maximální bezpečnost a spolehlivost, mohl by být doplněn o interpret DLL. Výpočet by poté běžel interpretovaně místo přímého běhu v operačním systému. Toto by umožnilo větší kontrolu nad výpočtem a omezení aktivit prováděných v DLL – například omezení přístupu k souborovému systému.

Typování přenášených dat

- V současné verzi jsou části zadání a výsledků přenášeny jako pole bytů. Pokud by se přenášená data skládala z otypovaných proměnných, bylo by možné například provádět automatické změny pořadí bytů v číselných proměnných při přenosu mezi různými architekturami.

Motivace uživatelů

- V případech, kdy chceme pro běh výpočetních klientů využít širokou internetovou veřejnost, by bylo vhodné uživatele motivovat k poskytnutí co nejvyššího výpočetního výkonu. Prostředky takové motivace by pak znamenaly i nutnost přidání další funkčnosti do systému.

- Příklad 1: žebříčky nejlepších uživatelů (poskytujících nejvíce výpočetního času) podle země, operačního systému, organizace a podobně by si vyžádaly nutnost vést dlouhodobé statistiky.

- Příklad 2: v případě, kdy má úloha nějaký konkrétní cíl jako třeba prolomit šifru nebo najít vzorek mimozemské komunikace, by určitě uživatele zajímalo, na čím stroji k tomu došlo. To by vyžadovalo sledování částí zadání a řešení včetně identifikačních údajů.

7 Literatura

[1] Berkeley Open Infrastructure for Network Computing http://boinc.berkeley.edu/

[2] SETI @ HOME http://setiathome.berkeley.edu/

[3] PVM http://www.csm.ornl.gov/pvm/

[4] Los Alamos Lab: High-Performance Computing: Roadrunner http://www.lanl.gov/roadrunner/

[5] Recent advances in parallel virtual machine and message passing interface: 13th European PVM/MPI User's Group Meeting, Bonn, Germany, September 17-20, 2006: proceedings / Bernd Mohr ... [et al.]

QA76.58.E973 2006

[6] Parallel Virtual Machine - EuroPVM'96: Third European PVM Conference, Munich, Germany, October, 7 - 9, 1996. Proceedings / by Arndt Bode, Jack Dongarra, Thomas Ludwig.

Berlin, Heidelberg : Springer Berlin Heidelberg, 1996.

ISBN 978-3-540-61779-2

[7] OS market share

http://marketshare.hitslink.com/os-market-share.aspx?qprid=9

[8] Windows sockets http://msdn.microsoft.com/en-us/library/ms740673.aspx

[9] Distributed computing : principles, algorithms, and systems / Ajay D. Kshemkalyani and Mukesh Singhal

Cambridge : Cambridge University Press, 2008

QA76.9.D5K74 2008

[10] Grid Computing: Experiment Management, Tool Integration, and Scientific Workflows / by Radu Prodan, Thomas Fahringer.

Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007.

ISBN 978-3-540-69261-4

8 Příloha

Na přiloženém DVD se nachází tato práce a zmíněná implementace systému vytvořená v Microsoft Visual C++ 2008 Express edition. Projekty, ze kterých se systém skládá, je možné v tomto prostředí kompilovat. Je ale nutné mít do MS Visual C++ nainstalovaný Windows SDK. Zdrojové kódy jsou v adresáři *zdrojove_kody*.

Na DVD dále je vzorová instalace systému. V adresáři *instalace\server* jsou soubory, které mají být na serveru, v adresáři *instalace\vypocetni_klient* jsou soubory, které mají být na výpočetním klientovi a v adresáři *instalace\zakaznicky_klient* jsou soubory, které mají být na zákaznickém klientovi.

V případě instalace nově zkompilovaných souborů je nutné tyto soubory nakopírovat do příslušných adresářů stejně jako ve vzorové instalaci.

Spuštění systému se provede spuštěním programu p1.exe bez parametrů. Výpočetní klienti p4.exe se také spouštějí bez parametrů. Zákaznický klient p3.exe se spouští s parametry podle toho, co má udělat – jestli zadat novou úlohu nebo zjistit stav existující úlohy.