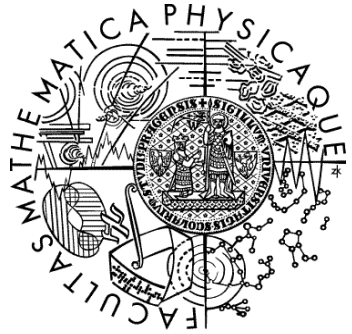
Univerzita Karlova v Prahe
Matematicko-fyzikálna fakulta

BAKALÁRSKA PRÁCA



Róbert Šišaj

Distribúcia výpočtov

Inštitút teoretickej informatiky - centrum mladej vedy

Vedúci bakalárskej práce: Mgr. Zdeněk Dvořák
Študijný program: Informatika, Programovanie

2008

Chcel by som poďakovať všetkým, ktorí mi pomáhali pri písaní tejto bakalárskej práce, či už cennými radami alebo nepriamo poskytnutím vlastného počítača pre testovanie aplikácie. Zvlášť ďakujem svojmu vedúcemu Mgr. Zdeňkovi Dvořákovi a RNDr. Filipovi Zavoralovi, Ph.D. za usmerňovanie a konzultácie ohľadom náplne práce.

Prehlasujem, že som svoju prácu napísal samostatne a výhradne s použitím uvedených prameňov. Súhlasím so zapožičiavaním a zverejňovaním práce.

V Prahe dňa Róbert Šišaj

Názov práce: Distibúcia výpočtov
Autor: Róbert Šišaj
Ústav: Institut teoretické informatiky - CMV, Univerzita Karlova v Praze
Vedúci bakalárskej práce: Mgr. Zdeněk Dvořák, Institut teoretické informatiky - CMV, Univerzita Karlova v Praze, Malostranské náměstí 25, 118 00 Praha 1
e-mail vedúceho: rakdver at kam.mff.cuni.cz
Abstrakt: Cieľom práce je vytvorenie systému pre distribúciu výpočtov pre menšie siete LAN. Systém by mal tvoriť platformu pre rôzne aplikácie, ktorým umožní paralelné spracovanie výpočtovo náročných úloh (hľadanie prvočísel, renderovanie videosekvencií, úpravy fotografií, ...). Čitateľ sa zoznámi s tematikou distribuovaných systémov, dozvie sa o ich výhodách i nevýhodách a osvojí si základné pojmy z tejto oblasti. Súčasťou práce je porovnanie s podobnými existujúcimi systémami (SETI@home, distributed.net). Zahrnutá je aj analýza slabín navrhovaného systému a prípadné námety na jeho vylepšenie do budúcnosti.

Kľúčové slová: Distribuované výpočty, distribuované systémy, paralelné spracovanie

Title: Distributed computing
Author: Róbert Šišaj
Department: Institute for Theoretical Computer Science, Charles University, Prague
Supervisor: Mgr. Zdeněk Dvořák, Institute for Theoretical Computer Science, Charles University, Prague, Malostranské náměstí 25, 118 00 Praha 1
Supervisor's e-mail address: rakdver at kam.mff.cuni.cz
Abstract: The aim of this work is creating a system for distributed computing in smaller LAN networks. The system should provide a platform for various applications, which will be able to run parallel processing of computationally expensive tasks (e.g. searching for prime numbers, rendering video sequences, editing photographs, ...). The reader will learn about distributed systems, about their advantages and disadvantages and will get knowledge of definitions and key words from this area. The work also makes a comparison of this system and a few existing systems (SETI@home, distributed.net). An analysis of weak parts of the proposed system is included in this paper and also some ideas and thoughts on some improvements of the system in the future.

Keywords: Distributed computing, distributed systems, parallel processing

Obsah

1 Úvod

Hlavným cieľom tejto práce je návrh a implementácia programu na využitie voľnej výpočtovej kapacity pre paralelne spracovateľné úlohy v menšej sieti LAN. Program bude založený na architektúre typu klient - server a bude určený pre menší počet počítačov (rádovo desiatky strojov). Predpokladá sa takmer homogénne prostredie v sieti (počítače budú veľmi vyrovnané, čo sa týka výkonu - napríklad sieť bežných kancelárskych počítačov). Jeden server by mal obsluhovať viacero klientov, preto hlavným limitom na množstvo klientov bude zrejme množstvo práce spojené so zadávaním úloh, spravovaním výsledkov a sieťovou komunikáciou (a, samozrejme, výkon samotného serveru). V prípade, že sa prekročí tento limit, zostanú niektorí klienti nevyťažení a výpočtová kapacita siete sa nevyužije naplno.

Program bude tvoriť rozhranie a logiku distribuovaného systému (prijímanie úloh od aplikácií, správa a prideľovanie jednotlivých úloh, skladanie hotových výsledkov, predávanie hotových výsledkov aplikáciám, komunikácia medzi dvoma uzlami, ...). Cez toto rozhranie budú ostatné aplikácie schopné využívať voľnú výpočtovú kapacitu v sieti pre svoje výpočty. Program musí podporovať ľubovoľnú aplikáciu, ktorá spĺňa predpísané rozhranie. Výsledný softvér by mal umožňovať jednoduché úpravy počas behu programu, ako je napríklad zmena algoritmu pre prideľovanie úloh.

Súčasťou práce je aj diskusia o vhodnosti alebo nevhodnosti riešení použitých pri implementácii a porovnanie týchto riešení, prípadne programu, s existujúcimi systémami pre distribuované výpočty. Čitateľ tu získa náhľad na danú problematiku, osvojí si základné pojmy z tejto oblasti a dozvie sa rôzne informácie, ktoré úzko súvisia s distribuovanými systémami, napríklad problémy s bezpečnosťou.

Štruktúra práce:

V **kapitole 2** sú vysvetlené základné pojmy a definície používané v distribuovaných výpočtoch, tiež sú tu načrtnuté výhody a nevýhody distribuovaných systémov.

Kapitola 3 je zameraná na návrh implementácie programu, sú tu diskutované apriorné predpoklady a obmedzenia, z ktorých následne vychádza implementácia.

Kapitola 4 je venovaná popisu implementácie a použitých riešení.

Kapitola 5 hodnotí výsledok a uvádza ďalšie možné vylepšenia programu.

V **kapitole 6** je porovnanie programu s existujúcimi riešeniami.

Kapitola 7 tvorí záver celej práce.

2 Distribuované výpočty

2.1 Čo sú distribuované výpočty

Distribuované výpočty sú jednou z metód spracovania dát počítačom, kedy rôzne časti programu bežia súčasne na dvoch alebo viacerých počítačoch, ktoré spolu komunikujú cez sieť. Distribuované výpočty môžeme klasifikovať ako špeciálny prípad paralelného spracovania dát. Paralelné spracovanie je širší pojem, pod ktorým chápeme akékoľvek spracovanie dát na dvoch alebo viacerých výpočtových jednotkách. Najčastejšie sa však týmto pojmom označuje spracovanie dát na viacerých procesoroch, ktoré sú súčasťou jedného počítača.
Obidva spôsoby spracovania potrebujú program, ktorý je schopný paralelizovať výpočet, teda rozdeliť ho na menšie časti, ktoré môžu byť spracované súčasne. Pri distribuovanom spracovaní je navyše potreba vziať do úvahy aj fakt, že výpočet môže prebiehať v rôznom prostredí a na odlišných architektúrach (počítače môžu obsahovať iné operačné systémy, súborové systémy alebo hardwarové prvky).

Pre distribuované výpočty sú najvhodnejšie rozsiahlejšie úlohy, ktorých spracovanie je časovo náročné a ktoré sa dajú rozdeliť na mnoho malých podúloh. Podúlohy sú distribuované do siete počítačov, takže každý počítač spracuje niekoľko podúloh a pošle naspäť výsledok. Čiastočné výsledky sú zhromažďované na jedno miesto a na konci sa z nich skompletuje finálny výsledok.

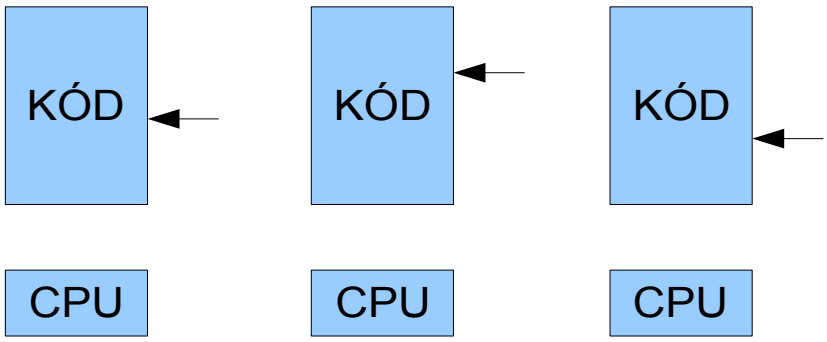
Distribuované výpočty sú prirodzeným dôsledkom vzniku počítačových sietí pre komunikáciu medzi počítačmi. Sieťovanie, anglicky *networking*, vzniklo primárne kvôli potrebe výmeny informácii, avšak len čo sa počítače stali dostupné aj pre bežných ľudí a sieťovanie spoľahlivé a rýchle, objavila sa celkom jednoduchá myšlienka: Postaviť jeden superpočítač s vysokým výpočtovým výkonom je veľmi nákladné, vyrobí a predá sa vždy len zopár kusov pre špecializované inštitúcie. Čo

takto skúsiť pospájať sieťou mnoho slabších strojov, ktoré sa vyrábajú v masovom merítku a sú teda lacné a ľahko dostupné, a pomocou vhodného softvéru ich zorganizovať tak, aby pracovali ako jeden „virtuálny stroj“?

Podľa Amdahlovho zákona je jasné, že takéto spracovanie je menej účinné, než laická optimistická predstava (*N* rovnakých počítačov zapojených spolu = *N*-krát rýchlejšie spracovanie). Spotreba energie moderného procesora rastie viac ako lineárne vzhľadom k frekvencii, na ktorej beží (pri rovnakej použitej technológii), preto z energetického hľadiska začína byť ekonomické nechať bežať paralelne viac slabších procesorov než jeden s vysokým taktom.

2.2 Definície a pojmy

Niektoré pojmy bývajú často použité nepresne alebo v nesprávnom kontexte, iné sa odlišujú tak jemne, že bývajú často zamieňané (grid computing vs distributed computing), preto sa pokúsim vysvetliť tieto pojmy.



Obrázok 1: Príklad paralelného výpočtu

Parallel computing

označuje súčasné vykonávanie časti kódu bežiaceho vo viacerých ištanciách na viacerých procesoroch za účelom zrýchlenia výpočtu.

Predstavme si program hľadajúci prvočísla, na troch výpočtových jednotkách je spustený rovnaký kód ale s inými počiatočnými hodnotami premenných, ktorý overí, či dané číslo je prvočíslo (Obrázok 1). Šípky ukazujú pozíciu vo vykonávanom kóde, všetky tri procesory pracujú nezávisle. Výpočet sa teoreticky môže zrýchliť až trikrát.

Myšlienka urýchlenia spočíva v skutočnosti, že väčšina problémov sa zvyčajne dá

rozdeliť na menšie časti, ktoré sa môžu spracovať nezávisle a následne sa z čiastočných výsledkov poskladá riešenie.

Distributed computing

je spôsob počítačového spracovania, kedy výpočet prebieha aspoň na dvoch počítačoch, ktoré spolu komunikujú cez sieť, stretneme sa preto tiež s označením network computing. Zvyčajne na každom uzle beží iná časť programu alebo výpočtu. Jednotlivé uzly sú často od seba geograficky veľmi vzdialené. Podľa charakteru uzlov môžeme distribuované výpočty ďalej rozdeliť na grid computing, cluster computing, volunteer computing a utility computing.

Grid computing

je typ distribuovaného spracovania, kedy je „virtuálny superpočítač“ tvorený zvyčajne výpočtovou kapacitou v rámci jednej organizácie alebo medzi viacerými organizáciami, ako sú napríklad univerzity, výskumné strediská alebo firmy.

Uzly v sieti tvoria zhluky, jeden zhluk je fyzicky tvorený skupinou počítačov spojených rýchlou sieťou LAN, prípadne to je jeden cluster alebo jeden superpočítač. Na rozdiel od cluster computing nie sú všetky počítače tak tesne prepojené, pretože zhluky môžu byť z rôznych organizácií, a teda aj na rôznych lokalitách. Všetky počítače zvyčajne spravujú vyškolení pracovníci príslušných organizácií a sú permanentne pripojené k sieti. Nepredpokladá sa umýselný škodca v sieti.

Cluster computing

označuje skupinu počítačov (cluster), ktorá spolupracuje navzájom tak úzko, že z mnohých pohľadov o nej môžeme hovoriť ako o jednom stroji. Počítače sú obvykle navzájom prepojené rýchlou sieťou LAN. Použitie je širšie, pretože cluster sa môže využiť nielen na zvýšenie výpočtového výkonu, ale aj na zvýšenie robustnosti nejakej sieťovej služby - napríklad WWW server môže byť interne tvorený viacerými počítačmi ako cluster a pri výpadku alebo mechanickej údržbe jedného z nich si užívateľ nevšimne žiadny rozdiel oproti normálnej prevádzke.

Volunteer computing

je spôsob distribuovaného počítania, princíp spočíva v tom, že majiteľ počítača, napríklad bežný domáci používateľ, dobrovoľne zapojí voľné zdroje svojho počítača (hlavne procesorový čas a pamäť) do distribuovaného systému. Týmto sa stáva súčasťou jedného alebo viacerých projektov, väčšinou zameraných na výskum, najčastejšie z akademickej sféry [1]. Grid computing sa odlišuje tým, že výpočtové

kapacity poskytujú zväčša organizácie a zároveň počítače tvoria väčšie zhluky. Veľkú časť počítačov v sieti poskytujú jednotlivci, ale mnoho počítačov zapájajú aj vzdelávacie centrá alebo firmy [6]. Na rozdiel od grid computingu sa predpokladá, že medzi dobrovoľníkmi sa môžu vyskytnúť aj škodcovia, ktorí sa budú usilovať o podvrhnutie falošného výsledku alebo inak narušovať výpočet.

Utility computing

znamená v podstate prenájom výpočtovej a úložnej kapacity. Ide o platenú službu, kedy zákazník potrebuje krátkodobo veľmi vysoké prostriedky, ale neoplatí sa mu kupovať si vlastné zariadenia, preto si prenajme výpočtový výkon na určitú dobu. Zároveň odpadá čakacia doba na zostavenie a nainštalovanie hardvéru a softvéru a tiež akákoľvek údržba.

2.3 Výhody

Ponúka sa porovnanie medzi špičkovým superpočítačom a „virtuálnym“ strojom zloženým z bežných počítačov komunikujúcich cez sieť. Asi najväčšou výhodou distribuovaného spracovania je fakt, že každý uzol v sieti je vlastne bežný počítač, ktorého výroba je masová, a teda výsledná cena nízka. Vyvinúť a zostaviť superpočítač je náročné, a preto si ho nemôže každý kúpiť. Množstvo výpočtových jednotiek v superpočítači generuje pri plnom výkone nemalé teplo a je treba použiť sofistikované metódy chladenia, aby stroj mohol vôbec fungovať. Na druhú stranu distribuované výpočty prebiehajú na fyzicky oddelených strojoch (najväčšie projekty majú uzly po celom svete), a teda problém s chladením nie je nutné riešiť. To isté platí o dodávaní dostatočnej elektrickej energie. Rozširovanie distribuovaného systému je relatívne jednoduchá záležitosť, v drvivej väčšine prípadov stačí pridať nový uzol - počítač do siete. Zato rozširovanie superpočítača je už z princípu veľmi limitované - obsahuje totiž to najlepšie a najvýkonnejšie, čo je v tej dobe práve dostupné, a je stavaný tak, aby to plnou mierou dokázal využiť. Ani programovať pre superpočítač nemôže každý, je potreba poznať všetky technické detaily stroja a vedieť to náležite využiť. Ale ak sa problém dá dobre paralelizovať, tj. každý uzol má vlastné dáta, ktoré spracuje nezávisle od ostatných, potom programovanie pre distribuovaný systém je vlastne programovanie pre bežný počítač a odpadajú komplikácie s konkurenčnými vláknami, zdieľanou pamäťou alebo viacerými inštanciami toho istého programu bežiacimi v rovnaký čas. Výhodou je tiež veľká

robustnosť systému, geograficky sú jednotlivé uzly od seba aj kilometre vzdialené, a preto limit pre robustnosť tvorí zvyčajne sieťové spojenie - napríklad veľký požiar môže fyzicky zlikvidovať len niekoľko málo uzlov ale ostatným sa nestane nič, ani ich napájanie nie je nijako ohrozené. Superpočítač sa však nachádza v jednej lokalite a poškodenie menšej časti znamená zvyčajne výpadok celého stroja.

2.4 Nevýhody

Veľkou nevýhodou distribuovaného systému je fakt, že množstvo výpočtových jednotiek, ktoré ho tvoria, môže navzájom komunikovať len žalostne pomaly pomocou siete. Takisto úložisko dát celého systému je roztrúsené v jednotlivých uzloch. To limituje použitie systému len na určité typy úloh, kedy každý uzol môže konať svoju prácu nezávisle od ostatných a bez potreby komunikácie s okolím. Ak by mali uzly priebežne komunikovať, alebo sa často synchronizovať medzi sebou, mohol by výpočet v extrémnom prípade prebiehať dokonca pomalšie než na pôvodnom systéme. S rastúcim počtom uzlov stúpa pravdepodobnosť výpadku niektorého z nich, preto treba vždy predpokladať určitú nespoľahlivosť. Zabezpečenie však stojí určité prostriedky a znižuje teoreticky dosiahnuteľný maximálny výkon.

Dobrovoľníci, ktorí darujú prebytočnú výpočtovú kapacitu svojho počítača, určite časom spozorujú nárast spotreby elektrickej energie, pretože procesor bude väčšinu času zaťažený prácou a bude tiež produkovať viac tepla. Systém tvorený na základe dobrovoľníkov musí počítať so škodcami, teda že sa nájde počítač, ktorý vedome alebo nevedome posiela naspäť podvrhnutý alebo chybný výsledok a prehlasuje ho za korektný.

Každému, kto pracuje v obore okolo počítačovej bezpečnosti musí behať mráz na chrbte pri predstave, že na jeho vlastnom počítači beží cudzí program, ktorý navyše z času na čas sám od seba komunikuje po sieti - a to je presne to, čo tvorí základ distribuovaných výpočtov. Práve preto si žiadna organizácia, ktorá chce využívať distribuovaný systém, nemôže dovoliť podceniť súvisiace bezpečnostné riziká a musí sa tejto otázke venovať vo zvýšenej miere. S trochou paranoje si nie je ťažké vymyslieť aj temnejšie scenáre. Od tých jemnejších, kedy organizácia po získaní dôvery veľkého počtu počítačov na nich nechá tajne spustiť softvér, ktorý bude špehovať užívateľov (na nátlak nejakej vlády, alebo na nátlak zo strany

protipirátskych organizácií), až po tie horšie, kedy organizáciu infiltruje skupina hackerov a vypustí vlastný škodlivý program, ktorý sa z takto širokej základne môže bleskovo rozšíriť po celom svete. Určitá hrozba tu existuje, autor však nemá informácie o žiadnom útoku s podobným charakterom, ani o žiadnom nepočul. Pre užívateľa by malo byť samozrejmosťou, že zníži prípadné riziko obmedzením prostriedkov pre distribuovaný systém (pokiaľ mu to operačný systém umožní), napríklad zakáže prístup mimo vyhradený priestor na pevnom disku a pod.

2.5 Superpočítač zadarmo

Bleskový rozvoj Internetu a neustály kolotoč zlepšovania technológií a následného znižovania cien elektroniky umožnil situáciu, v ktorej sa nachádzame dnes - počítač a Internet má skoro každý. Máloktorý z bežných osobných počítačov je naplno vyťažený počas celej doby, čo je zapnutý. Pri normálnej kancelárskej práci je počítač väčšinu času bez práce.

Ak pospájame množstvo takýchto počítačov a vyplníme ich voľný čas užitočnou prácou, získame veľmi slušnú výpočtovú kapacitu, ktorá sa aspoň pri niektorých typoch výpočtov dá výkonnostne porovnať so superpočítačmi ako je napríklad Blue Gene (Obrázok 2). Teoretický výkon Blue Gene bol pôvodne 360 teraFLOPS, po vylepšení v roku 2007 stúpol na hodnotu 596 teraFLOPS [9]. Projekt BOINC mal k 3.5.2008 výkon vyšší ako 1000 teraFLOPS. Prvý, kto prekonal hranicu 1 petaFLOPS (1000 teraFLOPS), sa stal 16. septembra 2007 projekt Folding@Home. Začiatkom mája 2008 ten istý projekt pokoril svojím výkonom hranicu 2 petaFLOPS, čím sa stal opäť prvým výpočtovým systémom na svete, ktorému sa to podarilo [10].

Obrázok 2: Blue Gene/L

Veľa ľudí je ochotných pomáhať vede a výskumu aspoň tým, že darujú voľný čas svojho počítača, a potom je takýto superpočítač vlastne úplne zadarmo. Ako to funguje? Počítač pracuje na distribuovaných výpočtoch vtedy, keď je človek na obede, mimo pracovnú dobu, napríklad v noci, ale aj počas užívateľovej práce na počítači, kedy sa inštrukcie určené pre výpočet zaraďujú priebežne medzi inštrukcie, ktoré odpovedajú užívateľovej činnosti. Program využívajúci voľný čas procesora sa môže úplne jednoducho nastaviť tak, aby bežal s najnižšou prioritou, a teda užívateľ pracuje s počítačom normálne ako inokedy a program sa dostane k práci vtedy, keď počítač zaháľa - číta dáta z pomalého média, keď čaká na vstup z klávesnice a podobne.

2.6 Dôsledky

Zvýšená záťaž procesora má samozrejme svoju odozvu, ktorú si užívateľ môže všimnúť napríklad na zvýšenej spotrebe elektrickej energie (Tabuľka 1). Základné faktory, ktoré ovplyvňujú množstvo takto spotrebovanej energie:

- časové obdobie, ktoré počítač denne strávi touto extra prácou
- energetická spotreba počítača pri plnej záťaži
- úroveň správy napájania počítača, šetrenie energie odpojením nečinných diskov a pod.

Stav počítača (24 h/deň)	Typická spotreba	Energia spotrebovaná za mesiac
Vypnutý	0 wattov	0 kWh
Nečinný	100 wattov	73 kWh
Aktívny	150 wattov	110 kWh

Tabuľka 1: Odhad energetickej spotreby počítača za mesiac

Možnosti kompenzácie zvýšených energetických nárokov sú celkom pestré. Ak chce užívateľ ponechať počítač zapnutý len kvôli distribuovaným výpočtom, určite môže pri odchode od počítača pokojne vypnúť monitor. Novšie monitory už majú síce slušnú úsporu v stave stand-by, ale skutočne vypnutému monitoru sa tým nikdy nevyrovnajú. Ideálne je mať monitor v takej zásuvke, na ktorej sa dá skutočne prerušiť dodávka elektrickej energie, pretože každý vypnutý spotrebič, ktorý je

zapojený do elektrickej siete, odoberá časť energie a mení ju na teplo. Ďalšiu úsporu prinesie zlepšená správa napájania počítača, ktorá sa dá nastaviť v operačnom systéme. Aktívne by mali zostať len prostriedky, ktoré sú potrebné pre chod počítača a pre samotný výpočet, zvyšok zariadení môže byť odpojený alebo so zníženým napájaním. To sa týka napríklad USB zariadení (webkamery, tlačiarne a pod.) ale tiež pevných diskov, mechaník či zvukovej karty. Nie je nutné ponechávať počítač zapnutý len kvôli tomu, aby jeho zdroje mohli byť využité v distribuovanom systéme, darovanie voľných cyklov procesora počas bežnej práce s počítačom je často dostatočný prínos.

Logicky je počítač oproti obyklému stavu viac zaťažený, a preto produkuje aj viac tepla. Za normálnych podmienok jeho chladiaci systém určite zvláda účinne chladiť dôležité komponenty, ale vyplnením voľných chvíľ procesora sa situácia zhoršuje. Usadzovanie prachu na vetracie otvory, do chladičov a na samotné zariadenia vnútri počítača zabraňuje prístupu vzduchu a tým znižuje účinnosť chladenia. Užívateľ by preto mal častejšie a hlavne pravidelne kontrolovať stav vo vnútri počítača a prevádzať údržbu. Nepatrne zvýšená je tiež pravdepodobnosť poškodenia výpočtovej jednotky z dôvodu opotrebovania, ale pri správnom chladení je rozdiel takmer nulový. Navyše program, ktorý riadi výpočty, môže v spolupráci s operačným systémom znížiť pracovnú záťaž, aby nedošlo k prehriatiu. Takisto je možné nastaviť tento program tak, aby počítač intenzívne pracoval v čase, kedy dodávateľ poskytuje elektrickú energiu lacnejšie.

2.7 Zhrnutie

Výhody distribuovaných výpočtov:

- dobrá škálovateľnosť
- vysoký výkon za nízke náklady
- vysoká robustnosť

Nevýhody distribuovaných výpočtov:

- pomalé spojenie medzi uzlami
- nie je možné zaručiť bezpečnosť uzlov v sieti
- nevhodné pre určité typy úloh

3 Návrh implementácie

3.1 Predpoklady

Celkom iste sa dajú spraviť určité predpoklady o prostredí, v ktorom aplikácia pobeží. Aj z týchto predpokladov bude vychádzať návrh implementácie programu. Prostredie, v ktorom bude aplikácia najčastejšie používaná, budú pravdepodobne menšie kancelárske priestory, kde sa nachádza viacero osobných (stolných) počítačov s operačným systémom Windows XP. Tieto sú spojené rýchlou lokálnou sieťou. Aplikácia sa bude týkať vždy jednej skupiny počítačov, na ktorých bude spustená nasledovným spôsobom: jeden počítač v skupine spustí program v roli serveru, zvyšné počítače spustia program v úlohe klienta, ktorý sa pripojí k tomuto serveru. Veľkosť takejto skupiny (resp. počet klientov na jeden server) bola diskutovaná v kapitole 1 - Úvod, pôjde rádovo o desiatky strojov. Každá takáto skupina bude nezávislá, t.j. v rámci jednej lokálnej siete môžu vedľa seba súčasne fungovať viaceré skupiny (medzi skupinami nebude prebiehať žiadna komunikácia). Myšlienka prepojenia týchto nezávislých skupín je diskutovaná v kapitole 5 - Možné vylepšenia - Rozšíriteľnosť pre väčšie siete.

Program bude spúšťaný v čase, kedy s počítačmi nebude nikto pracovať, čiže napríklad počas obedňajšej pauzy alebo mimo pracovnú dobu. Náročnosť jednotlivých prideľovaných úloh sa predpokladá primeraná, klient by mal byť schopný spracovať jednu úlohu v priebehu pár minút prípadne niekoľko málo hodín. Úlohy by mali byť výpočtovo náročnejšie než réžia spojená s posielaním dát po sieti a s rozdeľovaním úloh na menšie pracovné jednotky, prípadne spájaním čiastkových výsledkov do jedného.

3.2 Návrh implementácie

Na základe predpokladov je možné návrh vytvoriť nasledovne:

Operačný systém:

Windows XP, momentálne jeden z najpoužívanejších OS v kanceláriách, navyše má autor s týmto OS najviac skúseností

Programovací jazyk:

C++, WinAPI

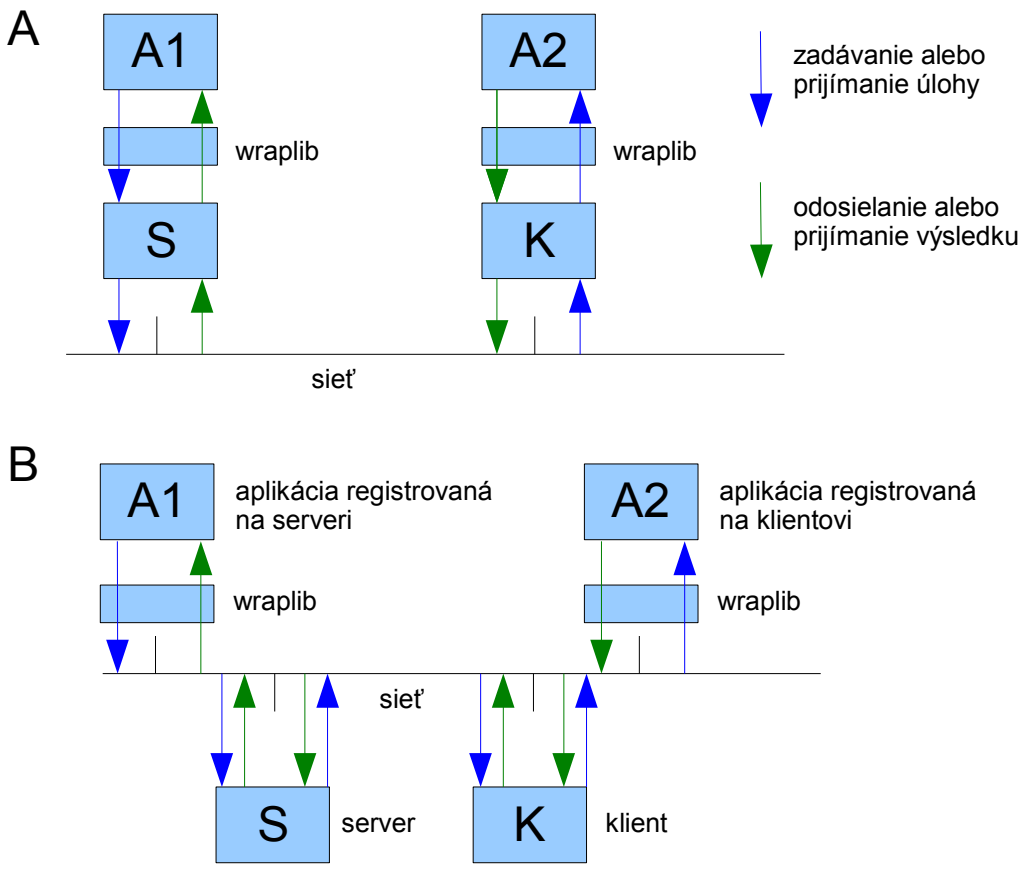
Sieťovanie:

TCP/IP, Windows Socket

GUI:

GUI pre zobrazovanie informácií o systéme (server: zoznam klientov, ich vlastnosti, zoznam úloh na spracovanie, atď, klient: stav spojenia so serverom, prípadne aktuálne riešená úloha, nejaká štatistika o riešených úlohách)

Program bude obsahovať klientskú časť i serverovskú. Pri spustení programu sa pomocou prepínača rozhodne, či program bude vykonávať prácu klienta alebo serveru. Je to výhodné, pretože sa dá očakávať, že počítač, ktorý je momentálne klientom, môže mať pri ďalšom spustení úlohu serveru. Navyše klient i server zdieľajú mnohé funkcie, takže údržba kódu bude takto jednoduchšia.
Aplikácia, ktorá chce využívať služby programu, sa pomocou rozhrania najskôr zaregistruje (ak sa chce účastniť na výpočte, tak sa registruje na klientovi, ak chce zadávať úlohu na výpočet, registruje sa na serveri). Tým získa prístup k distribuovanému systému. Komunikácia medzi aplikáciou a klientom/serverom bude zabezpečená pomocou dynamicky linkovanej knižnice s implementáciou vlastného riešenia posielania správ. Aplikácia môže komunikovať s programom len v rámci jedného stroja (Obrázok 3A). Výpočet jednej podúlohy vyžaduje len dva sieťové prenosy, použitie zdieľanej pamäte ušetrí zbytočné kopírovanie dát medzi aplikáciou a programom. Pri komunikácii aplikácie s programom cez sieťové rozhranie môže dochádzať k zvýšenej záťaži na sieťové prostriedky a k celkovému zníženiu výkonu distribuovaného systému (Obrázok 3B). Výpočet v tomto prípade vyžaduje šesť sieťových prenosov, dáta musia byť kopírované pri predávaní medzi aplikáciou a serverom/klientom. Program musí podporovať viacero aplikácií súčasne, dostupné výpočtové prostriedky budú pre tieto aplikácie zdieľané (prebehne časť výpočtu prvej aplikácie, potom časť výpočtu druhej, atď).



Obrázok 3A, 3B: Typický pracovný diagram distribuovaného výpočtu

3.3 Spoľahlivosť výpočtov

Spoľahlivosť počítačov v sieti je považovaná za postačujúcu, teda ak klient vráti výsledok, je tento považovaný za správny. Sieťový protokol sám zabezpečuje spoľahlivosť prenosu dát. Pri komunikácii odosielateľ pošle údaj o veľkosti správy a správu, ak sa prijímateľovi zhoduje dĺžka prijatej správy s týmto údajom, je prenos považovaný za úspešný. Ak by sme očakávali, že procesor na klientskom počítači môže robiť numerické chyby vo výpočtoch, potrebovali by sme znížiť pravdepodobnosť chybného výsledku. Jedna z možností, ako to dosiahnuť, je napríklad poslať každú úlohu trom odlišným klientom a porovnať ich výsledky - ak sú aspoň dva rovnaké, výsledok je správny. Tým, samozrejme, výrazne klesá výkon systému. Server bude pravdepodobne bežať na rovnakom hardvéri ako klienti, ale nebude prevádzať tak intenzívne výpočty, preto je riziko chyby na serveri menšie. V tomto návrhu implementácie sa s chybovosťou výpočtov neráta. Aplikácie, ktoré

požadujú vyššiu pravdepodobnosť správnosti výsledku, si ju musia zaistiť samy, napríklad viacnásobným zadaním danej úlohy (to ale nevylučuje prípad, že server pošle všetky opakované úlohy na ten stroj, ktorý ich spočíta chybne, len sa znižuje pravdepodobnosť tejto situácie). Možné vylepšenie je prezentované v časti 5 Zhodnotenie výsledku.

3.4 Bezpečnosť

Prostredie, v ktorom bude program spúšťaný, by malo byť podľa predpokladov bezpečné. Ak je pracovné prostredie súčasťou nejakej firmy, potom prípadný útočník znamená závažné narušenie firemnej bezpečnosti. Server i klient budú mať v sebe implementovaný komunikačný protokol, ktorý definuje typy posielaných správ. Pokiaľ príjemca zistí, že správa nedodržiava predpísaný formát, vygeneruje záznam do logu a správu zahodí. Odosielateľ môže odoslať takisto len presne definované správy. Týmto by sa nemalo stať, že prijatie nejakej zákernej správy uvedie program do nedefinovaného stavu, alebo zablokuje jeho činnosť. Dáta určené pre výpočet (resp. výsledok nejakého výpočtu) nie sú interpretované, len sa predávajú vhodnému príjemcovi. Podrobnejšie informácie sú v časti 4.1 Komunikácia.

4 Popis implementácie

Program dostal pracovný názov **Anthill** (angl. *mravenisko*), čo by malo v prenesenom význame symbolizovať: aj keď jeden mravec sám osebe je drobné stvorenie (~ málo výkonný počítač), správna organizácia mnohých mravcov je schopná bez problémov vybudovať aj obrovské mravenisko (~ vyriešiť zložitý výpočet presahujúci možnosti jedného stroja).

4.1 Komunikácia

4.1.1 Komunikácia medzi klientom a serverom

Program Anthill používa na komunikáciu medzi klientom a serverom sieťový protokol TCP/IP. Každý dôležitý prenos vyžaduje na konci potvrdenie o úspešnom prenose od druhej strany. Napríklad keď sa klient prihlasuje na server, pošle serveru svoje údaje a server musí potvrdiť, či prenos prebehol v poriadku. Ak klient dostane kladnú odpoveď od serveru, môže užívateľovi oznámiť úspešné pripojenie na server. Za nedôležitý prenos sa považuje napríklad oznámenie klienta, že si užívateľ praje okamžite ukončiť jeho činnosť, napríklad pri vypnutí počítača. Vtedy klient nečaká na potvrdenie prenosu, hneď po oznámení ukončí svoj beh, preto server ani na tento prenos neodpovedá. Typy správ a ich význam sú popísané v Tabuľke 2.

Správa	**Odosielateľ**	**Prijímateľ**	**Význam**	**Vyžaduje odpoveď**
WANT_TO_JOIN	klient	server	Klient sa chce prihlásiť na server	áno
APP_LIST	klient	server	Klient posiela zoznam uňho registrovaných aplikácií	áno
MUST_LEAVE	klient	server	Klient oznamuje ukončenie behu	nie
DONT_WANT_NEW_JOBS	klient	server	Klient už nechce ďalšie úlohy	nie
SEND_JOB	server	klient	Server posiela zadanie úlohy	áno
SEND_RESULT	klient	server	Klient posiela výsledok výpočtu	áno
STOP_WORK	server	klient	Server prerušuje výpočet	nie
PING	server	klient	Server zisťuje dostupnosť klienta	áno

Tabuľka 2: Komunikácia medzi klientom a serverom - typy správ a ich význam

Prenos, ktorý má byť ukončený odpoveďou z druhej strany, ale tá nepríde, je považovaný za neúspešný. Táto situácia je chápaná ako problém v sieti a nie je potreba ju špeciálne riešiť. Tu sú rozpísané jednotlivé prípady, kedy nedorazí odpoveď od prijímateľa (variant **A** označuje prípad, kedy prenos prebehol v poriadku, variant **B** označuje prípad, kedy príjemca zistí chybu v prenose - v oboch prípadoch sa správa s potvrdením/zamietnutím prenosu stratí v sieti):

WANT_TO_JOIN

A: klient je úspešne prihlásený na serveri, ale sám o tom nevie. Užívateľovi oznámi, že nedostal potvrdenie a užívateľ sa môže pokúsiť o nové pripojenie. Server zistí, že daný klient sa pokúša znovu pripojiť bez toho, že sa predtým odhlásil, a preto vypíše varovanie do logu. Po úspešnom potvrdení klientovi je systém v rovnakom stave, ako keby sa prihlásenie podarilo na prvýkrát.

B: server zaloguje správu o neúspešnom prenose, svoj stav nezmení. Klient informuje užívateľa o neúspešnom prihlásení a užívateľ sa môže pokúsiť o nové pripojenie. Po úspešnom potvrdení klientovi je systém v rovnakom stave, ako keby sa prihlásenie podarilo na prvýkrát.

APP_LIST

A: klient oznámi užívateľovi, že nedostal potvrdenie. Ten môže buď znovu odoslať zoznam aplikácií ručne, alebo stačí, ak sa na klientovi registruje ďalšia aplikácia (to vyvolá automatický export zoznamu aplikácií). Server je pripravený na opätovné prijatie zoznamu aplikácií, s ktorými klient spolupracuje, pretože predpokladá, že ten sa môže časom meniť.

B: situácia je z pohľadu klienta identická (až na zobrazenú hlášku pre užívateľa), z pohľadu serveru však môže byť daný klient nepoužiteľný. Ak server nemá pri tomto klientovi poznačenú žiadnu registrovanú aplikáciu, potom predpokladá, že klient nedokáže momentálne spracovať žiadnu úlohu, a teda nie je dôvod mu čokoľvek posielať.

MUST_LEAVE

A: všetko je v poriadku, odpoveď sa aj tak na túto správu neposiela.

B: server si stále myslí, že klient je živý, a môže mu posielať nové správy.

Zaslaním správy môže detekovať problém a má právo prehlásiť klienta za "mŕtveho" a zrušiť s ním spojenie (v súčasnosti ešte nie je implementované - server sám od seba nikdy spojenie neruší). "Mŕtvi" klienti môžu jemne spomaľovať algoritmy na výber stroja, ktorému sa posiela úloha na výpočet, niektoré naivné algoritmy by mohli zlyhať, ak sa server nebude automaticky zbavovať takýchto klientov (môže sa stať, že naivný algoritmus vždy vyberie "mŕtveho" klienta ako najvhodnejšieho kandidáta).

DONT_WANT_NEW_ JOBS

A: všetko je v poriadku, odpoveď na túto správu sa neposiela.

B: všetko je v poriadku, až kým klient neodošle výsledok (ak bol uprostred výpočtu) a neskončí. Vtedy nastáva rovnaká situácia ako v prípade MUST_LEAVE B (server si stále myslí, že klient je pripojený).

SEND_JOB

A: Server po určitom čase rozhodne, že odpoveď sa stratila, a nastaví si klientský počitač opäť ako voľný, ten však pracuje na výpočte. Táto nekonzistencia sa opraví až pri ďalšom prenose - buď server bude posielať zadanie novej úlohy (klient pošle odpoveď STILL_BUSY, po ktorej si server opraví informáciu o stave klienta), alebo klient stihne poslať riešenie úlohy.

B: klient zistí chybu v prenose, oznámi to do logu, pošle serveru správu o chybnom prenose a ďalej sa o nič nestará. Server po určitom čase rozhodne, že odpoveď sa stratila, a nastaví si klientský počitač opäť ako voľný.

SEND_RESULT

A: klient nedostane potvrdenie o úspešnom prenose, ohlási možnú chybu do logu a pokračuje ako keby prenos dopadol úspešne.

B: súčasná implementácia nie je na tento prípad optimalizovaná. Napriek tomu, že by veľmi pravdepodobne stačilo prenos zopakovať, sa výsledok zahodí a výpočet sa časom zopakuje, možno na inom stroji. Podrobnejšie informácie o pracovnom cykle sú v kapitole 4.2.

STOP_WORK

A: všetko je v poriadku, odpoveď na túto správu sa neposiela.

B: klient netuší, že by mal prerušiť výpočet. Môžu nastať dve zaujímavé situácie: 1) server zadáva novú úlohu skôr ako klient dokončí výpočet, vtedy klient odpovie STILL_BUSY a počíta ďalej. 2) klient odovzdáva riešenie úlohy, ale tá sa s najväčšou pravdepodobnosťou nebude nachádzať vo fronte odoslaných zadaní úloh, a preto server tento výsledok zahodí. Podrobnejšie informácie o pracovnom cykle sú v kapitole 4.2.

PING

A, B: v oboch prípadoch ide o závažnú indikáciu nedostupnosti klienta, pretože ak sieť nie je schopná doručiť dve krátke správy (PING a odpoveď klienta), pravdepodobne zlyhá aj prenos zadania úlohy či výsledku.

Typy odpovedí a ich význam:

TRANSFER_OK: prenos prebehol úspešne

TRANSFER_FAILED: prenos zlyhal pravdepodobne kvôli problémom v sieti

STILL_BUSY: prenos prebehol úspešne, ale prijímateľ momentálne nemá dostatok voľných prostriedkov na spracovanie požiadavku. Prenos je možné po čase zopakovať.

4.1.2 Komunikácia medzi užívateľskou aplikáciou a programom Anthill

Na komunikáciu medzi užívateľskými aplikáciami a Anthillom bola implementovaná dynamicky linkovaná knižnica s pracovným názvom Wraplib. Tá definuje rozhranie, ktoré umožňuje komunikáciu pomocou správ medzi aplikáciou a Anthillom v rámci jedného počítača. To dovoľuje použiť zdieľanú pamäť a šetrí zbytočné kopírovanie dát.

Posielané dáta sú uchovávané v dočasnom súbore (temporary file). Operačný systém udržuje podľa dokumentácie obsah takéhoto súboru priamo v operačnej pamäti bez toho, že by komunikoval s pevným diskom. Až v prípade, že veľkosť dočasného súboru je príliš veľká alebo nie je dostatok voľnej pamäte, uloží operačný systém tento súbor na pevný disk tak ako bežné súbory.

Keď chce aplikácia poslať napríklad zadanie úlohy, zapíše potrebné dáta do takéhoto

dočasného súboru, vyplní štruktúru správy pre Wraplib a správu odošle. Wraplib pomocou systémového volania požiada operačný systém o duplikáciu odkazu (handle) na tento súbor tak, aby príjemca mohol z tohto súboru čítať a zároveň tento odkaz odosielateľovi zruší. Takto má Anthill dáta priamo prístupné na ďalšie čítanie. Typy správ a ich význam sú popísané v Tabuľke 3.

Správa	**Odosielateľ**	**Prijímateľ**	**Význam**	**Vyžaduje odpoveď**
APP_REGISTER	aplikácia	server/klient	Aplikácia sa chce zaregistrovať	áno
APP_UNREGISTER	aplikácia	server/klient	Aplikácia chce ukončiť spoluprácu	áno
APP_ADD_JOB	aplikácia	server	Aplikácia posiela zadanie novej úlohy	áno
APP_SEND_RESULT	aplikácia	klient	Aplikácia odovzdáva výsledok	áno
APP_NOT_NOW	aplikácia	server	Aplikácia momentálne nechce prevziať výsledok	áno
APP_RESULT_READY	server	aplikácia	Server chce odovzdať výsledok aplikácii	áno

Tabuľka 3: Komunikácia medzi klientom a serverom - typy správ a ich význam

Typy odpovedí a ich význam:

TRANSFER_OK: prenos prebehol úspešne

TRANSFER_FAILED: prenos zlyhal pravdepodobne kvôli nesprávnemu formátu dát (pri zadávaní novej úlohy musí aplikácia dodržať definovaný formát).

4.2 Pracovný cyklus

4.2.1 Reprezentácia úlohy

Anthill udržuje na serveri tieto informácie o každej úlohe, ktorá bola prijatá od registrovanej aplikácie cez rozhranie Wraplib, až kým nie je úloha spracovaná a výsledok odoslaný späť aplikácii:

- interné identifikačné číslo, jedinečné pre každú úlohu
- názov aplikácie, ktorá zadala úlohu
- identifikačné číslo úlohy - pre účely zadávajúcej aplikácie, Anthill túto položku ignoruje
- typ úlohy - môže byť nápomocné pri sofistikovanejších algoritmoch na výber

stroja podľa dostupných zdrojov (napríklad úloha vyžadujúca náročný grafický výpočet môže byť odoslaná na počítač so špecializovanou grafickou kartou)

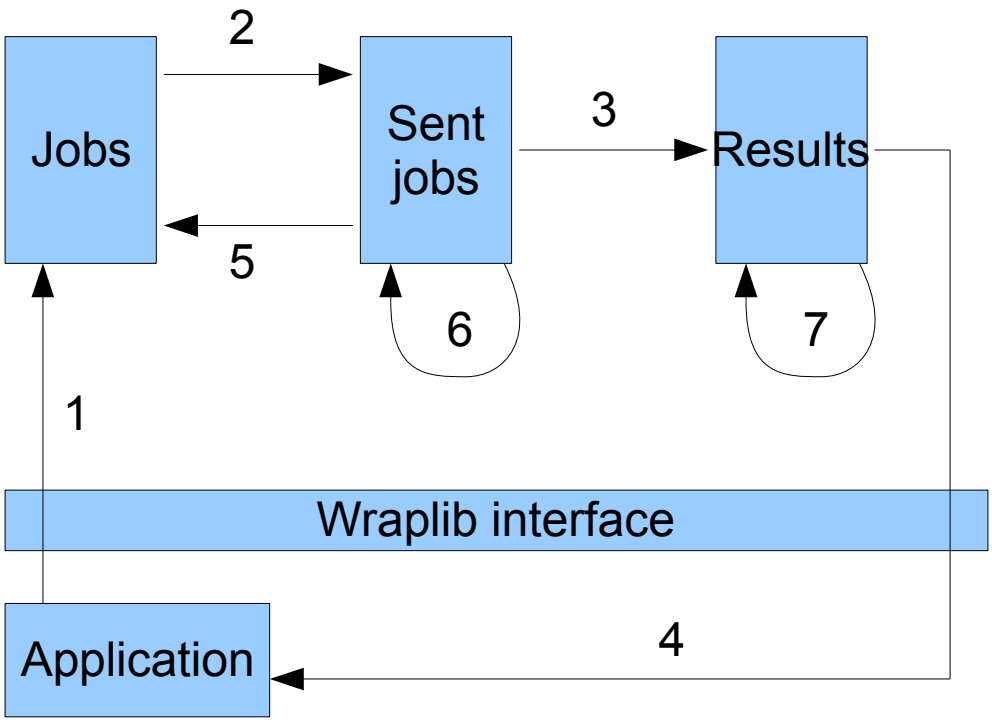
- veľkosť dát v bajtoch
- dáta úlohy

4.2.2 Reprezentácia klientského počítača

Server udržuje tieto informácie o každom klientovi:

- stav - pracujúci/voľný
- celkový počet úloh, ktoré boli klientovi zadané
- celkový počet úspešne vyriešených a odovzdaných úloh
- spoľahlivosť; podiel odovzdaných a zadaných úloh
- čas, koľko trvalo spracovanie poslednej úlohy
- najkratší čas, koľko trvalo spracovanie nejakej úlohy
- najdlhší čas, koľko trvalo spracovanie nejakej úlohy
- priemerný čas, koľko trvá spracovanie jednej úlohy
- aktuálny zoznam registrovaných aplikácií

4.2.3 Pracovný diagram serveru



Obrázok 4: Životný cyklus jednej úlohy na serveri

Na Obrázku 4 je schematicky znázornený priebeh výpočtu jednej úlohy od zadania až po odovzdanie výsledku naspäť aplikácii.

Aby server mohol spustiť distribuované spracovanie úloh, musí prijať aspoň jednu úlohu cez rozhranie Wraplib (šípka číslo 1). Jedna alebo viacero registrovaných aplikácií pošle správou APP_ADD_JOB zadanie úlohy. Úlohy je možné pridávať aj počas bežiaceho výpočtu, ale predpokladá sa, že distribuovaný systém ich bude stíhať spracovať (tj. nebudú sa hromadiť nespracované úlohy, pretože by to časom mohlo viesť k vyčerpaniu systémových prostriedkov).

Zadania úloh server udržuje vo fronte Jobs (na obrázku hore vľavo), ktorá je typu FIFO. Na pokyn užívateľa server spustí distribuovaný výpočet. Vezme prvú úlohu z fronty Jobs a dá ju ako parameter algoritmu na výber klienta (podrobnosti o algoritme sú v časti 4.5). Algoritmus vyberie vhodného klienta, server presunie úlohu do fronty Sent jobs (na obrázku hore vstrede, tiež je typu FIFO), kde sú úlohy, ktoré už boli vybrané na odoslanie (šípka číslo 2). Potom odošle zadanie tejto úlohy klientovi a poznačí si, že klient pracuje.

Server vyberá ďalšie úlohy z Jobs a rovnakým spôsobom ich distribuuje klientom v sieti, až kým nie sú všetci klienti vyťažení. Potom sa nachvíľu uspí a priebežne kontroluje, či niektorý klient neskončil výpočet. Keď klient pošle výsledok k niektorej úlohe, server skontroluje podľa interného identifikačného čísla úlohy, či existuje úloha s takým číslom vo fronte Sent jobs. Ak tam taká nie je, pravdepodobne niektorý iný klient stihol skôr odovzdať jej riešenie, a teda je všetko v poriadku. Ak sa tam nachádza úloha s takým číslom a riešenie bolo úspešne doručené, server presunie túto úlohu (šípka číslo 3) do fronty Results (hore vpravo, takisto typu FIFO). Zadanie úlohy už nie je potrebné, preto je nahradené prijatým riešením úlohy.

Vo fronte Results zostáva riešenie úlohy až dovtedy, kým sa serveru nepodarí doručiť výsledok naspäť aplikácii (šípka číslo 4). Server oznámi pomocou správy APP_RESULT_READY aplikácii, že už má pripravený výsledok úlohy. Aplikácia môže výsledok dočasne odmietnuť pomocou APP_NOT_NOW, na čo server zareaguje tým, že výsledok zaradí opäť na koniec fronty Results (šípka číslo 7). Aplikáciu znovu kontaktuje novou správou, až keď sa opäť dostane vo fronte na začiatok. Hneď ako aplikácia prijme výsledok, vymaže server úlohu z fronty Results a tým sa výpočet tejto čiastkovej úlohy považuje za ukončený.

Šípkou číslo 5 je naznačený presun všetkých úloh z fronty Sent jobs naspäť na

koniec fronty Jobs. To sa udeje jedine na príkaz užívateľa, keď dá zastaviť výpočet. Server vtedy oznámi všetkým pracujúcim klientom, aby prestali počítať. Klienti zahodia rozpracované úlohy a čakajú. Užívateľ môže potom výpočet opätovne spustiť.

Keď sú vyčerpané všetky úlohy z fronty Jobs, existuje voľný klient a zároveň fronta Sent jobs nie je prázdna, pošle server prvú úlohu z fronty Sent jobs a opäť ju zaradí na koniec fronty (šípka číslo 6). Je možné (a v prípade poslednej úlohy aj veľmi pravdepodobné), že niektorá úloha bude rozoslaná viacerým klientom. To však nevadí, server je pripravený na konkurenčné odovzdávanie výsledku ("zvíťazí" klient, ktorý odovzdá riešenie ako prvý). Celý výpočet končí vo chvíli, keď sú fronty Jobs a Sent jobs prázdne, užívateľovi sa zobrazí hláška o celkovom čase trvania výpočtu. Potom môže ešte prebiehať odovzdávanie výsledkov aplikáciám, ale novo prijaté zadania úloh už nie sú distribuované do siete. Nový výpočet sa spustí opäť až na užívateľov pokyn.

4.3 Registrácia aplikácií

Každá aplikácia, ktorá chce používať služby programu Anthill, sa musí najskôr zaregistrovať. Klient po pripojení na server odošle svoj zoznam registrovaných aplikácií. Zadané úlohy majú vo svojej reprezentácii obsiahnutý aj názov aplikácie, ktorá ich má spracovať. Server môže na základe týchto infromácií vybrať klienta, ktorý je schopný spracovať danú úlohu a podľa toho ju distribuuje do siete. Ak už aplikácia nechce spolupracovať s Anthillom, mala by sa odregistrovať.

4.4 Plánovací algoritmus

Anthill je navrhnutý tak, aby bolo možné zmeniť plánovací algoritmus serveru počas behu programu na voľné experimentovanie a testovanie efektivity rôznych algoritmov.

Vstup plánovacieho algoritmu:

- aktuálny zoznam pripojených klientov a úloha, ktorú je potreba spracovať.

Výstup algoritmu:

- klient, ktorému sa daná úloha pošle, alebo oznámenie, že nie je možné nájsť vhodného klienta pre túto úlohu.

Súčasná verzia programu obsahuje tri základné algoritmy na výber klienta.

1) Naivný algoritmus: postupne od začiatku prechádza vstupný zoznam klientov a vráti prvý voľný stroj, ktorý dokáže úlohu spracovať.

- výhody: jednoduchý, rýchly, minimálna záťaž na server
- nevýhody: nevyužíva žiadnu informáciu o klientoch (úspešnosť vyriešených úloh, časy spracovania...)

2) Výber najspoľahlivejšieho: vstupný zoznam klientov utriedi zostupne podľa ich spoľahlivosti (pomer odovzdaných riešení k zadaným úlohám) a hľadá prvý voľný stroj, ktorý dokáže úlohu spracovať.

- výhody: znižuje záťaž prostriedkov distribuovaného systému a nepriamo aj premávku v sieti
- nevýhody: najspoľahlivejší stroj nemusí byť najrýchlejší, celkový výpočet môže trvať dlhšie

3) Výber podľa najkratšieho času spracovania: vstupný zoznam klientov utriedi zostupne podľa času, koľko im trvalo spracovanie poslednej úlohy, a hľadá prvý voľný stroj, ktorý dokáže úlohu spracovať.

- výhody: pri rovnomernej obtiažnosti úloh (spracovanie každej úlohy jedným strojom trvá približne rovnako) minimalizuje celkový čas výpočtu
- nevýhody: pokiaľ neplatí podmienka o rovnomernej obtiažnosti úloh, čas výpočtu môže trvať dlhšie

Algoritmy sú implementované ako samostatné moduly. Je možné implementovať nový algoritmus a buď nahradiť jeden zo súčasných, alebo ho pridať ako ďalší (momentálne Anthill nie je celkom pripravený na pridávanie nových algoritmov, napríklad konfiguračný dialóg programu zobrazuje v ponuke len tri algoritmy a musel by byť takisto upravený).

5 Zhodnotenie výsledku a možné vylepšenia

5.1 Zhodnotenie

Autor vytvoril pilotnú aplikáciu na hľadanie prvočísel (pracovný názov Primes - nachádza sa na priloženom CD), ktorá slúži hlavne na demonštráciu a na odskúšanie programu Anthill. Test s touto aplikáciou vyšiel uspokojivo, ukázal, že Anthill je funkčný a spĺňa požiadavky nastolené v špecifikácii. Ďalej bola vytvorená modifikácia aplikácie Primes, hľadajúca čísla, ktoré sú zároveň druhou mocninou nejakého prvočísla. Táto modifikácia má pracovný názov Squares (k dispozícii na CD).

S použitím Primes a Squares bolo úspešne overené, že Anthill dokáže súčasne spolupracovať s rôznymi aplikáciami, ktoré zadávajú úlohy na spracovanie.

Testovanie prebiehalo na štyroch PC, kde bol Anthill spustený nasledovne:

1 server, registrované aplikácie Primes a Squares

1 klient s registrovanou aplikáciou Primes

1 klient s registrovanou aplikáciou Squares

1 klient s registrovanými aplikáciami Primes a Squares

Na serveri bolo vytvorených rádovo 1000 zmiešaných podúloh (Primes a Squares konkurenčne zadávali jednotlivé podúlohy), zadanie jednej podúlohy tvoril interval, v ktorom bolo treba hľadať čísla s danou vlastnosťou. Riešením bol zoznam čísel patriaci do príslušného intervalu.

Správnosť výsledkov bola overená porovnaním zoznamov vygenerovaných pri distribuovanom výpočte so zoznamom čísel vygenerovaných priamo samotnými aplikáciami Primes a Squares na rovnakých intervaloch.

Ďalej bol vykonaná séria jednoduchých výkonnostných testov programu Anthill (Tabuľka 4). Testová úloha bola zadaná nasledovne: vygenerovať zoznam prvočísel, nachádzajúcich sa v intervale <20 000 000, 40 000 000). Na výpočet bola použitá aplikácia Primes. Pre distribuovaný výpočet bol interval rozdelený na 20 podintervalov s rozsahom 1 000 000. Každý podinterval tvoril zadanie jednej úlohy.

V teste boli použité dva počítače, jeden slabší s jednojadrovým CPU (stroj A), druhý silnejší s dvojjadrovým CPU (stroj B).

Číslo testu	Popis testu	Čas výpočtu (s)
1	nedistribuovaný výpočet na A	129
2	nedistribuovaný výpočet na B	86
3	1 server na B, 1 klient na A	141
4	1 server na A, 1 klient na B	105
5	1 server na A, 2 klienti na B	49
6	1 server a 1 klient na A, 2 klienti na B	40
7	1 server a 2 klienti na B, 1 klient na A	36

Tabuľka 4: Prehľad výkonnostných testov

Z nameraných údajov je možné odvodiť niektoré zaujímavé súvislosti. Porovnaním časov nedistribuovaného výpočtu a simulovaného nedistribuovaného výpočtu (server na fyzicky inom stroji, jediný klient na stroji simulujúcom nedistribuovaný výpočet) zistíme, že použitie Anthillu znamená v prípade slabšieho stroja A spomalenie výpočtu o 9,3%, v prípade stroja B dokonca spomalenie o 22,1%. Je to spôsobené tým, že nedistribuovaný výpočet aplikáciou Primes intenzívne využíva CPU po celý čas, ale pri simulácii nedistribuovaného výpočtu Anthillom nie je CPU využité po celý čas výpočtu naplno, kvôli komunikácii medzi aplikáciou a Anthillom i medzi klientom a serverom. Navyše výsledkom celého výpočtu je v tomto prípade približne 10 MB dát (výsledné čísla sú uchovávané v textovej podobe). Odosielanie týchto dát zaberie určitý čas, počas ktorého výpočet vôbec nepostupuje. Sieťové prenosy zaberú obom strojom približne rovnaký čas, čo v percentuálnom vyjadrení vychádza horšie pre rýchlejší stroj B.

Paradoxne testy číslo 4 a 5 ukazujú viac ako dvojnásobné urýchlenie výpočtu, keď na stroji B pracuje jeden klient na každom jadre CPU. Očakávané zrýchlenie by sa malo blížiť k dvojnásobku, ale v praxi túto hodnotu dokonca prekračuje. Tento jav veľmi pravdepodobne súvisí s optimalizáciou použitého hardvéru a OS na vyššiu záťaž (napríklad sieťová karta musí posielať/prijímať dáta dvojnásobnou rýchlosťou, preto pravdepodobne dostáva vyššiu prioritu od OS, čím sa skracuje čas potrebný na jeden prenos v teste 5 oproti prenosom v teste číslo 4, a teda umožňuje prekonať očakávané urýchlenie).

Najvyšší výkon bol dosiahnutý v teste číslo 7, kedy s použitím troch výpočtových jednotiek (jeden jednojadrový a jeden dvojjadrový CPU) došlo k zrýchleniu výpočtu o 58% oproti času, ktorý bol dosiahnutý pri výpočte prevádzanom len aplikáciou Primes na rýchlejšom stroji B. Teoretická hranica urýchlenia v tomto prípade bola

32,25 sekundy (ak uvažujeme, že máme k dispozícii tri stroje, jeden vyrieši celý výpočet za čas dosiahnutý v teste číslo 1 a dva stroje dospejú k riešeniu každý za čas dosiahnutý v teste číslo 2). Výpočet prevedený s použitím Anthillu je teda o 11,6% pomalší než táto teoretická hodnota, čo je spôsobené hlavne komunikáciou po sieti a nutnosťou čakať na nové dáta.

5.2 Možné vylepšenia

Tu je niekoľko návrhov na zlepšenie a spríjemnenie práce s programom:

Flexibilný plánovací algoritmus

Program je možné upraviť tak, aby aplikácia zadávajúca úlohu mohla vopred určiť, aký algoritmus na prideľovanie má byť použitý pri rozhodovaní. Aplikácia vyžadujúca rýchle spracovanie by volila algoritmus uprednostňujúci počítače, ktoré najrýchlejšie vracajú výsledok a iná aplikácia by mohla vyžadovať, aby jej úlohy spracovávali stroje s najlepšou spoľahlivosťou.

Odstránenie limitu na totožnosť zadávajúcej a pracovnej aplikácie

Súčasná implementácia vyžaduje, aby zadávajúca aplikácia a aplikácia, ktorá má úlohu skutočne riešiť, boli z pohľadu Anthillu identické (pri registrácii sa musia identifikovať presne rovnako). Anthill môže byť rozšírený tak, aby pri registrácii každá aplikácia zadala detailnejší popis toho, čo dokáže spracovať (napríklad podporované formáty dát, podporované operácie s dátami, implementované algoritmy,...). Server by na základe týchto informácií mal viac možností na výber klientov (pokiaľ sú na dvoch klientoch rôzne aplikácie, ktoré však obidve zvládajú požadovaný výpočet, potom sú v tomto smere z pohľadu serveru identické).

Možnosť priority pre jednotlivé úlohy

Ku každému záznamu o zadanej úlohe by pribudla položka určujúca prioritu spracovania. Plánovací algoritmus môže uprednostniť úlohy s vyššou prioritou a zaradiť ich na spracovanie skôr. Riešenie priority by mohlo vyzerať napríklad takto: server bude udržovať tri zoznamy úloh na spracovanie, jeden s vysokou prioritou, druhý s normálnou a tretí s nízkou prioritou. Užívateľ by mohol nastaviť v akom pomere sa majú spracovávať úlohy z týchto zoznamov. Napríklad pomer 6:3:1 by znamenal, že z každých desiatich naplánovaných úloh by bolo 6 zo zoznamu s vysokou prioritou, 3 zo zoznamu s normálnou prioritou a 1 zo zoznamu s nízkou prioritou (za predpokladu, že každý zoznam obsahuje dostatočný počet úloh).

Spresnenie údajov o klientských počítačoch

Záznam o klientoch sa dá doplniť o položky ako sú: operačný systém, voľné miesto na pevnom disku, veľkosť RAM pamäte, počet a typ procesorov, atď. Na základe týchto údajov môže algoritmus efektívnejšie naplánovať prácu klientov, môže sa vyhnúť situácii, kedy pošle úlohu klientovi, ktorý nemá dostatok prostriedkov na jej spracovanie.

Zaručenie správnosti výsledkov

Pri zadávaní úlohy bude môcť aplikácia určiť, akú veľkú redundanciu pre výpočet požaduje. Napríklad pri spracovaní fotky nie je prípadná drobná chyba v hodnote pixelu rušivá, stačí vyriešiť každú úlohu iba raz, ale numerický výpočet, kde závisí na presnosti výpočtu, sa bez redundancie neobíde. Aplikácia pri zadávaní úlohy špecifikuje požiadavok, na koľko odlišných strojov má byť úloha odoslaná. Server prijímajúci úlohu bude môcť požiadavok odmietnuť s tým, že nemá postačujúci počet klientov. Aplikácia by tiež mohla nejakým príznakom určiť, že chce len prvých N výsledkov, pokiaľ chce čo najrýchlejšie spracovanie úlohy, tak vezme prvý hotový výsledok a ostatné už nepotrebuje.

Rozšíriteľnosť pre väčšie siete

Program popísaný v tejto práci má určitý limit na množstvo svojich klientov a teda aj na veľkosť siete, v ktorej sa dá rozumne použiť. Nemal by byť však problém vytvoriť vo väčšej sieti niekoľko menších skupín, každá bude obsahovať jeden server a N klientov. Tieto skupiny by pracovali nezávisle a dosiahla by sa tak vyššia vyťaženosť sieťovej kapacity. Nevýhodou je, že nie všetky typy úloh sa dajú spracovať úplne nezávisle. Program by sa dal rozšíriť na hierarchický distribouvaný systém napríklad tak, že každý server by mohol fungovať v 2 režimoch: buď len ako server (koreň stromu), alebo ako server a klient v jednom (vnútorný uzol). S takýmto rozšírením by nemal byť problém pokryť akúkoľvek veľkú sieť pomocou stromu, ktorého listy budú tvoriť klienti (tí budú vykonávať skutočnú výpočtovú prácu), vnútorné uzly budú mať na starosti rozdrobenie prijatej úlohy na menšie časti (ktoré budú distribuované smerom k listom) a zároveň spätné poskladanie výsledku (ten bude propagovaný smerom ku koreňu).

Ďalším možným riešením je vytvorenie nejakej komunikácie medzi servermi (oproti predchádzajúcemu riešeniu to zníži pomer medzi počtom serverov a klientov, ale zároveň zvýši záťaž na servery).

6 Porovnanie s existujúcimi riešeniami

6.1 Výber porovnávaných riešení

Pre porovnanie sa ponúkajú najstarší projekt distributed.net, ktorý vznikol v roku 1997 a momentálne asi najznámejší a najrozšírenejší projekt BOINC. Distributed.net nie je z bezpečnostných dôvodov úplne open source projekt [5], detailné informácie o komunikácii v sieti nie sú voľne dostupné, preto je porovnanie s týmto projektom obmedzenejšie než s BOINC, ktorý pochádza z akademickej sféry a je plne otvorený.

6.2 Podpora OS a rôznych platforiem

BOINC (Berkeley Open Infrastructure for Network Computing) je open source softvér, ktorý tvorí prostredie pre hosťovanie najrôznejších projektov využívajúcich distribuované výpočty. Každý projekt je úplne nezávislý od ostatných, má vlastné dátové servery, webové stránky i databázy. BOINC je určený pre dobrovoľníkov, ktorí chcú venovať svoju voľnú výpočtovú kapacitu pre výskum. Klientský program umožňuje, aby si užívateľ vybral z ponuky aktuálnych projektov tie, ktoré ho zaujímajú a ktorých sa chce zúčastniť, a určil spôsob, akým budú využité jeho prostriedky na riešenie úloh z týchto projektov. Softvér podporuje rôzne platformy a operačné systémy, takže je dostupný takmer pre každého. Projekt BOINC odštartoval v roku 2002 a čerpal zo skúseností z projektu SETIhome (spustený v roku 1999), obidva projekty vznikli na University of California, Berkeley, USA.

Projekt distributed.net vznikol začiatkom roku 1997 a odvtedy sa úspešne zúčastnil mnohých súťaží v lámaní šifier podporovanými organizáciou RSA Labs. Okrem toho pomohol vyriešiť niektoré zaujímavé matematické problémy (napríklad problém optimálneho Golombovho pravítka). Projekt podporuje mnoho operačných systémov, na oficiálnych stránkach sú prístupné balíky s klientskými programami v binárnej podobe pre každý z nich.

Anthill v súčasnosti podporuje len jeden operačný systém, väčšina firiem využíva možnosť hromadných licencií na produkty i hardware, a preto sa predpokladá takmer homogénne prostredie v sieti.

6.3 Správa aplikácií

BOINC používa vylepšenú správu aplikácií. Keď pracovníci v rámci projektu pozmenia časť programu, ktorá je dôležitá pre výpočet, napríklad vylepšia algoritmus

na spracovávanie dát alebo opravia nejakú chybu, nie je potreba, aby si užívatelia manuálne sťahovali novú verziu tohto programu. BOINC sám zistí, že je dostupná novšia verzia aplikácie, stiahne ju zo serveru, a teda užívateľ sa nemusí o nič starať.

Distributed.net spolieha na záujem dobrovoľníkov, ktorých raz za čas na svojich stránkach upozornia na zmenu verzie klienta (buď po oprave nejakej chyby, úprave algoritmu alebo spustení nového projektu). Po prechode na novšiu verziu sú akékoľvek výsledky od starších klientov ignorované.

Aplikácie, ktoré chcú používať Anthill, je potreba manuálne nahrať alebo nainštalovať ako na server, tak aj na klientov. Anthill je plánovaný pre malý počet počítačov, obsluha ktorých ešte nie je mimo ľudských síl. Napríklad BOINC využíva viac ako 550 000 klientov (stav ku dňu 15.5.2008 - [2]). Automatizácia distribúcie aplikácií ako i updatov klientov má nespornú výhodu v tom, že nie je potreba čakať, kým užívateľ zareaguje na výzvu. Navyše časté manuálne updatovanie môže po čase odradiť menej zainteresovaných ľudí a tým znížiť celkový výkon distribuovaného systému.

6.4 Komunikácia a práca programu

Všetku komunikáciu iniciuje BOINC klient, aby bolo spojenie možné aj v prípade, keď užívateľ používa pre pripojenie k internetu firewall alebo NAT [6].

Klient distributed.net využíva TCP/IP na spojenie so servermi, spojenie iniciuje opäť klient. Distributed.net umožňuje zapojiť do projektu aj klientov, ktorí nie sú pripojení k sieti, ale je potrebných niekoľko zásahov užívateľom, a preto sa dá predpokladať, že pôjde len o minimálny počet takých klientov (je potreba, aby užívateľ pomocou jednoduchej komunikácie e-mailom získal vstupné dáta pre klienta, tie nahral na stroj, kde pobeží klient a po jeho skončení odoslal výstup opäť ako e-mailovú prílohu).

V prípade Anthillu môže začať komunikáciu aj klient, aj server, obidva počítače by mali byť v jednej lokálnej sieti, kde nie je problém s obojsmernou komunikáciou.

BOINC klient si sám pýta prácu od plánovacieho serveru. Ten mu pridelí ďalšie inštrukcie podľa toho, aké prostriedky má klient k dispozícii (RAM, diskový priestor, sieťové pripojenie atď.). BOINC klient periodicky meria tieto prostriedky a nahlasuje tieto údaje plánovaciemu serveru každého projektu, ktorého sa účastní. Na základe

inštrukcií od plánovacieho serveru si klient stiahne z dátového serveru balíček aplikácií a vstupných dát k nim a začne pracovať. V balíčku môžu byť úlohy z viacerých projektov, na ktorých sa chce užívateľ účastniť. Klient sa potom môže odpojiť od siete a až po spracovaní výsledkov opäť naviaže spojenie so serverom a odošle výsledky. Prvoradé je, aby klienti vrátili výsledky včas, nie je až tak nutné, aby bol CPU každého klienta vyťažený na maximum. Kvôli tomu je na základe výkonu počítača a nastavenia využitia prostriedkov spravený odhad, kedy má byť výsledok hotový. Tento termín musí klient dodržať, pokiaľ je to možné, tak priebežne informuje plánovací server o stave spracovania. Ten môže rozhodnúť aj o zrušení práce, ak má podozrenie na chybu alebo večný cyklus u klienta. Priebežné zisťovanie stavu spracovania je dôležité aj pre užívateľa, aby mohol sledovať, ako prispieva k projektu, spolu s ďalšími motivačnými štatistikami je to významný prvok slúžiaci na udržanie záujmu užívateľa zotrvať v projekte.

Distributed.net zámerne nezverejňuje detaily ohľadom komunikácie, zabezpečenia integrity a spoľahlivosti výsledkov z obavy, že by sa začali výraznejšou mierou vyskytovať škodcovia. Na oficiálnych stránkach sú diskutované rôzne možnosti ako by sa dala zabezpečiť spoľahlivosť a zároveň aj slabiny týchto riešení. Aj naďalej považujú utajenie detailov za jednu z najlepších stratégií [8].

Anthill klient sa po spustení prihlási na serveri. Server má na starosti len malý počet klientov, preto je všetka logika umiestnená na ňom. Server má zoznam klientov, ktorí sú naň pripojení a sám vyberie klienta, ktorému pridelí prácu. Tohto klienta si označí ako pracujúceho, až kým mu nevráti výsledok. Medzitým spolu priebežne nekomunikujú, okrem situácie, kedy buď klient oznamuje, že užívateľ chce okamžite skončiť, alebo server zastaví prácu, prípadne sa vypína úplne. Na rozdiel od BOINCu Anthill nezisťuje vôbec nič o počítači, na ktorom je spustený. Vedie si však štatistiku o pracujúcich klientoch, kde sú tieto údaje: čas, koľko trvalo spracovanie poslednej úlohy, priemerný čas spracovania jednej úlohy, minimálny a maximálny čas spracovania jednej úlohy, počet zadaných úloh, počet úspešne odovzdaných výsledkov a spoľahlivosť klienta (podiel odovzdaných výsledkov a zadaných úloh). Na základe týchto údajov sa môže rozhodovať pri plánovaní práce. Na rozdiel od BOINC a distributed.net je možné plánovací algoritmus meniť počas behu Anthillu.

6.5 Podpora rôznych aplikácií

BOINC používa time-slicing, čo je spôsob, ako využiť výkon počítača pre viac projektov naraz v takom pomere, ako chce vlastník či užívateľ počítača. Celkový čas práce klienta je rozdelený na menšie celky, napríklad po hodine. Ak užívateľ preferuje projekt A pred projektom B v pomere 2:1, potom úloha projektu A sa bude spracovávať 2 hodiny, potom sa medzivýsledok uloží a pokračuje práca na úlohe projektu B po dobu 1 hodiny, uloží sa medzivýsledok a tento cyklus sa opakuje. Time-slicing má aj ďalšiu prednosť, pri veľmi náročných úlohách (predikovanie klímy) by bežný počítač spracovával jednu úlohu aj mesiac a ostatné úlohy by sa po celý ten čas nedostali k spracovaniu. BOINC priebežne ukladá spracovaný výsledok v takzvaných checkpointoch, tj uloží medzivýsledky v takom momente výpočtu, kedy je to možné [7].

Distributed.net používa vždy len špecializovaných klientov, ktorí sa zúčastňujú len na jedinom projekte. Každý klient obsahuje algoritmus optimalizovaný na výpočet, čo zvyšuje výkon distribuovaného systému, ale znižuje jeho univerzálnosť na minimum.

Anthill dokáže spolupracovať s viacerými aplikáciami súčasne. Server vždy posiela klientovi len jednu úlohu. Očakáva sa, že úlohy nie sú samy o sebe príliš časovo náročné na výpočet, preto nie je implementovaný žiadny mechanizmus na ukladanie medzivýsledkov. Úlohu musí klient spracovať a odovzdať vcelku, ak je výpočet prerušený kedykoľvek pred odovzdaním výsledku, celá námaha klienta bola zbytočná.

7 Záver

Podarilo sa implementovať funkčný softvér, ktorý spĺňa špecifikované požiadavky. Navyše boli vytvorené dve pilotné aplikácie, ktoré slúžia na otestovanie a predvedenie funkčnosti Anthillu a demonštráciu princípov distribuovaných systémov. Práca zdôvodňuje použité riešenia a obsahuje porovnanie s existujúcimi systémami. Prezentuje tiež návrhy na možné vylepšenia programu do budúcnosti.

Distribuované sytémy skrývajú v sebe nepochybne obrovský potenciál. Dôkazom toho je, okrem už spomínaných pozitívných projektov, aj veľký nárast botnetov, čo nie sú nič iné, len obrovské siete počítačov schopných vykonávať koordinovanú činnosť. Bohužiaľ tento potenciál si všimli už aj nekalé živly, ako sú spammeri, a využívajú ho na masívne rozosielanie nevyžiadanej pošty a podobne. Také veľké množstvo strojov môže v nesprávnych rukách fungovať dokonca ako zbraň - použitie botnetu na DDoS útok môže v okamihu zrušiť prístup na napadnutý server, či už firemný, univerzitný alebo aj vládny.

Našťastie však prevláda využitie distribuovaných systémov na pozitívne ciele a verme, že tomu tak bude aj naďalej.

Literatúra

[1] University of California, Berkeley, "Volunteer computing", 2007, dostupné na adrese http://boinc.berkeley.edu/trac/wiki/VolunteerComputing

[2] http://en.wikipedia.org/wiki/Boinc

[3] http://en.wikipedia.org/wiki/Grid_computing

[4] Kirk Pearson, "What is Distributed Computing?", 2002, dostupné na adrese www.distributedcomputing.info

[5] http://www.distributed.net/source/

[6] D. P. Anderson, G. Fedak, "The Computational and Storage Potential of Volunteer Computing," *ccgrid*, pp. 73-80, Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006

[7] D.P. Anderson, C. Christensen, and B. Allen. "Designing a Runtime System for Volunteer Computing", Supercomputing '06 (The International Conference for High Performance Computing, Networking, Storage and Analysis), Tampa, November 2006.

[8] Jeff Lawson, "Operational code authentication", 1999, dostupné na adrese http://www.distributed.net/source/opcodeauth.html

[9] http://en.wikipedia.org/wiki/Blue_Gene

[10] http://en.wikipedia.org/wiki/Grid_computing#Fastest_virtual_supercomputers

Prílohy

Príloha A: Obsah CD

\bin - tento adresár obsahuje binárne spustiteľné súbory:
- Anthill.exe - program Anthill
- Primes.exe - aplikácia Primes
- Squares.exe - aplikácia Squares
- wraplib.dll - dynamická knižnica wraplib

\doc - adresár obsahuje dokumentáciu programu

\Primes - obsahuje stručný návod na použitie aplikácie Primes

\Squares - obsahuje stručný návod na použitie aplikácie Squares

\program_doc - obsahuje programátorskú dokumentáciu k Anthillu, prehľad o rozsahu jednotlivých dokumentov je v Prílohe C: Programátorská dokumentácia

\testing - obsahuje stručný návod na simuláciu serveru a viacerých klientov na jednom stroji
- obsahuje pomocné batch súbory, pomocou ktorých sa dá uľahčiť opakované spúšťanie Anthillu a aplikácií Primes, Squares; vhodné pre testovacie účely

Anthill_program_doc - úvodný dokument k programátorskej dokumentácii Anthillu

Anthill_User_Guide - užívateľská príručka k Anthillu, je obsiahnutá aj v Prílohe B: Užívateľská príručka

\src - adresár obsahuje kompletné zdrojové kódy k programom Anthill, Primes, Squares a ku knižnici wraplib

\testing - adresár obsahuje ukážkové testovacie prostredie schopné bežať aj na jednom fyzickom stroji (je nutné upraviť IP adresy v nastavení klientov, aby bolo možné spojenie so serverom)

Distribucia_vypoctov - kompletný text bakalárskej práce aj s prílohami

Príloha B: Užívateľská príručka

Anthill

užívateľská dokumentácia

Prehlásenie

Anthill je freewarový program.
Smie byť voľne šírený a upravovaný.
Použitie je na vlastné riziko, autor neberie na seba žiadnu zodpovednosť!

Autor: Róbert Šišaj

Úvod:

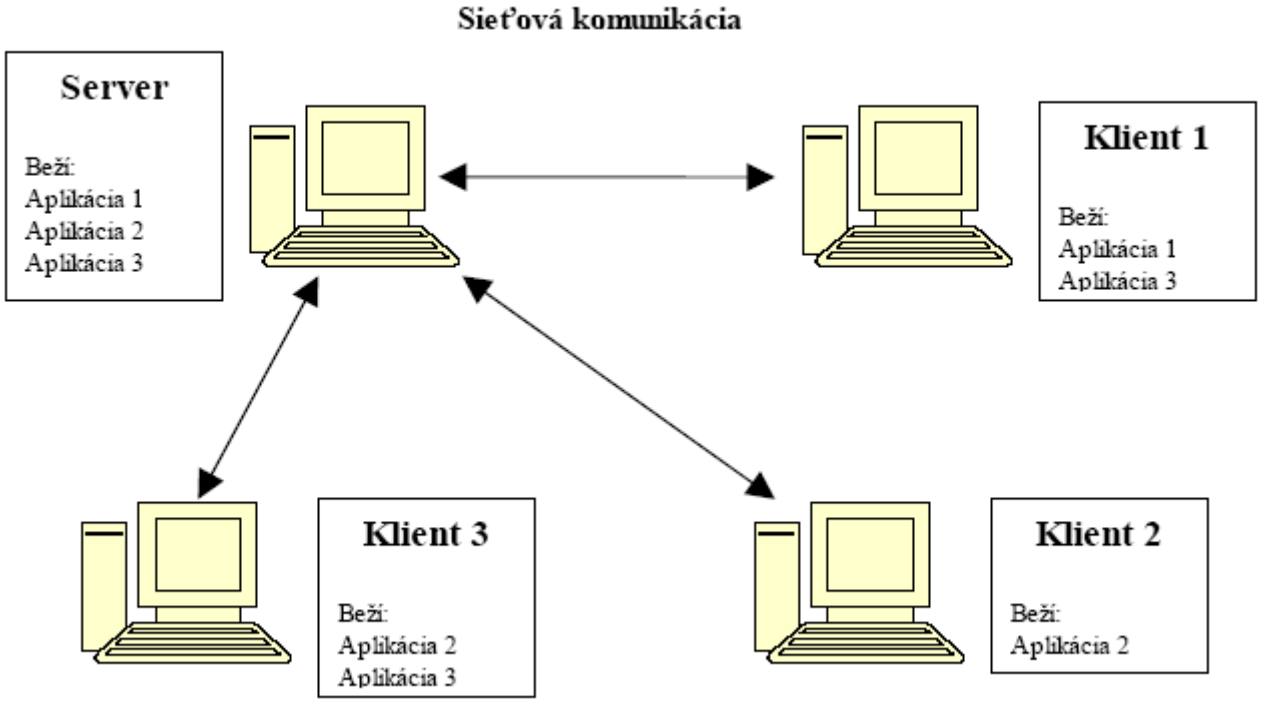
Anthill je jednoduchá aplikácia typu klient/server, ktorá poskytuje menšie distribuované výpočtové prostredie. Je navrhnutá pre menšiu sieť počítačov a umožňuje využívať voľnú výpočtovú kapacitu na spracovanie náročnejších úloh. Podporuje viac rôznych aplikácií, ktoré tak môžu súčasne využívať možnosti rovnakej siete pre svoje účely. Program vyžaduje operačný systém Windows a využíva jeho interface WinAPI.

Čo sú distribuované výpočty:

Distribuované výpočty je metóda spracovania dát, pri ktorom rôzne časti programu súčasne bežia aspoň na dvoch počítačoch, ktoré spolu vzájomne komunikujú cez sieť. Distribuované spracovanie je typ paralelného výpočtu. Zvyčajne má výbornú škálovateľnosť a poskytuje značný výkon. Samozrejme, má aj určité nevýhody, nie je vhodný na všetky typy výpočtov. Výborné uplatnenie má distribuované spracovanie, ak sa dáta dajú spracovať po menších častiach a nezávisle od seba. – projekt SETI@home je výborným príkladom. Každý stroj zúčastnený sa projekte obdrží svoju porciu dát, ktoré je potreba analyzovať. Vo chvíľach, kedy je normálne procesor nevyťažený (užívateľ napríklad píše textový dokument a podobne) môže analyzovať dáta bez toho, aby obmedzoval užívateľa pri bežnej práci s počítačom. Po spracovaní balíku dát odošle výsledky späť na domovský server SETI a dostane novú dávku na spracovanie. Zapojením veľkého množstva počítačov do tohto projektu je možné analyzovať obrovské kvantum dát prakticky za nulové náklady – pretože ľudia, ktorí sa projektu účastnia, sú dobrovoľníci, ktorí "darujú" voľné cykly svojich CPU na výskumné účely.

Cieľ programu Anthill:

Anthill neprichádza s novou myšlienkou, distribuované spracovanie je dobre známe a má už svoju históriu. Anthill je program vytvárajúci prostredie pre distribuované spracovanie v menšej sieti počítačov. Umožňuje spracovanie úloh prichádzajúcich od rôznych aplikácií súčasne. Program bol vytvorený hlavne pre vzdelávacie a experimentálne účely (hlavne čo sa týka plánovania úloh a optimalizácie využitia dostupných sieťových prostriedkov). Napríklad ak sa v sieti nachádza stroj, ktorý má špeciálny hardvér na spracovanie grafických dát, je možné upraviť plánovací algoritmus, aby v spracovaní úloh týkajúcich sa grafických výpočtov uprednostňoval tento špeciálny stroj oproti ostatným.



Popis inštalácie

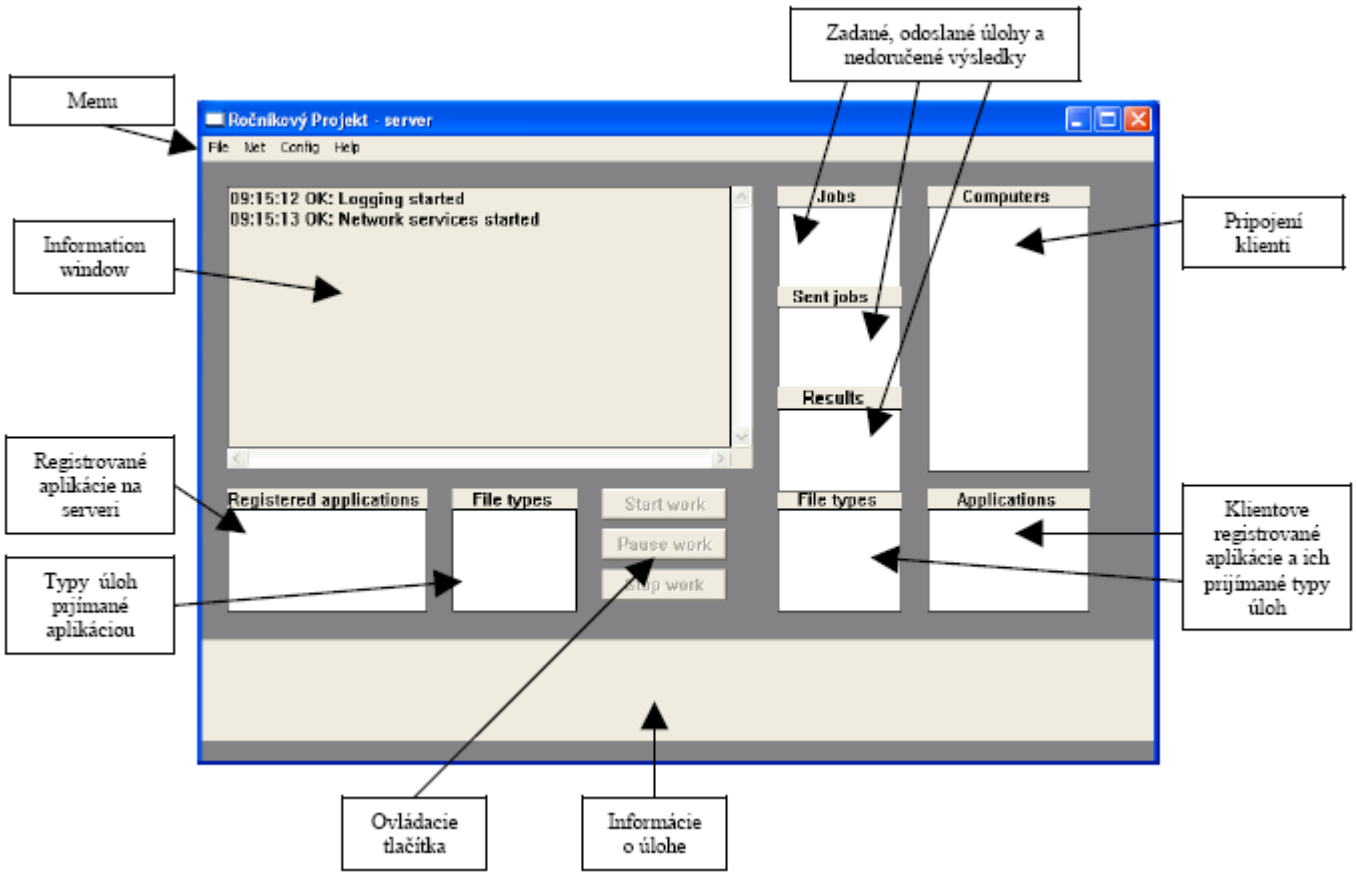
Anthill nie je potreba inštalovať. Pre správnu činnosť je nutné, aby Anthill a všetky aplikácie, ktoré sa budú registrovať, používali rovnakú inštanciu knižnice wraplib. Buď sa budú nachádzať v rovnakom adresári (anthill, aplikácie i wraplib), alebo wraplib musí byť dostupný v ceste zahrnutej v systémovej premennej PATH (napríklad C:\Windows\system32). V prípade, že sa wraplib nachádza v rovnakom adresári ako Anthill, má táto lokálna kópia prednosť a Anthill využije jej služby.

Pri prvom spustení je potreba povoliť prístup k sieťovým službám (nastavnie firewallu ak treba). Je možné, že pri prvom spustení sa zobrazí varovná správa o nedostupnej konfigurácii (ak sa súbor config.txt nenachádzal v rovnakom adresári ako Anthill).

Popis ovládania programu - SERVER

Hlavné okno:

Nachádza sa tu informačné okno, menšie okná, kde sa zobrazujú informácie o registrovaných aplikáciách, klientoch a úlohách a tiež ovládacie tlačítka a menu.



Information window:

Zobrazuje všetky dôležité udalosti a informácie o behu programu.

Registered applications:

Zoznam aplikácií, ktoré sú registrované na serveri. Len tieto aplikácie môžu zadávať úlohy na spracovanie a prijímať výsledky.

Applications:

Zoznam aplikácií, ktoré sú registrované u aktuálne označeného klienta v okne Computers. Len tieto aplikácie sa smú učastniť výpočtového procesu a prijímať úlohy a odovzdávať výsledky.

File types (both):

Sú dve okná s týmto názvom, ich pozícia však definuje ich význam. Ľavé okno je bližšie k oknu Registered applications, preto patrí k nemu, zatiaľčo pravé okno je bližšie k oknu Applications a tvorí s ním logickú dvojicu.

Keď si užívateľ praje zobraziť viac informácií o registrovanej aplikácii, vyberie príslušnú aplikáciu v okne Registered applications alebo Applications window. Správne okno File types potom zobrazí typy súborov alebo úloh, ktoré daná aplikácia podporuje (prijíma, zadáva, dokáže spracovať).

Jobs, Sent jobs, Results:

Jobs okno zobrazuje zoznam úloh, ktoré budú postupne spracované. Okno Sent jobs obsahuje zoznam úloh, ktoré už boli vybrané k spracovaniu a odoslané klientom. Results okno zobrazuje zoznam úloh, ktoré už boli úspešne vyriešené a ich výsledky čakajú na odovzdanie aplikáciám.

Task information:

Upresňuje informácie o úlohe aktuálne vybranej v Jobs, Sent jobs alebo Results okne.

Computers:

Zobrazuje aktuálny zoznam pripojených klientov. Iba títo klienti sa účastnia výpočtového procesu.

Start work, Stop work, Pause work:

Funkciu týchto tlačítok prezrádzajú ich názvy – spúšťajú, pozastavujú alebo úplne zastavia celý výpočtový proces.

Menu:

File:

Exit: Ukončuje beh aplikácie. Klávesová skratka ALT + F4.

Net:

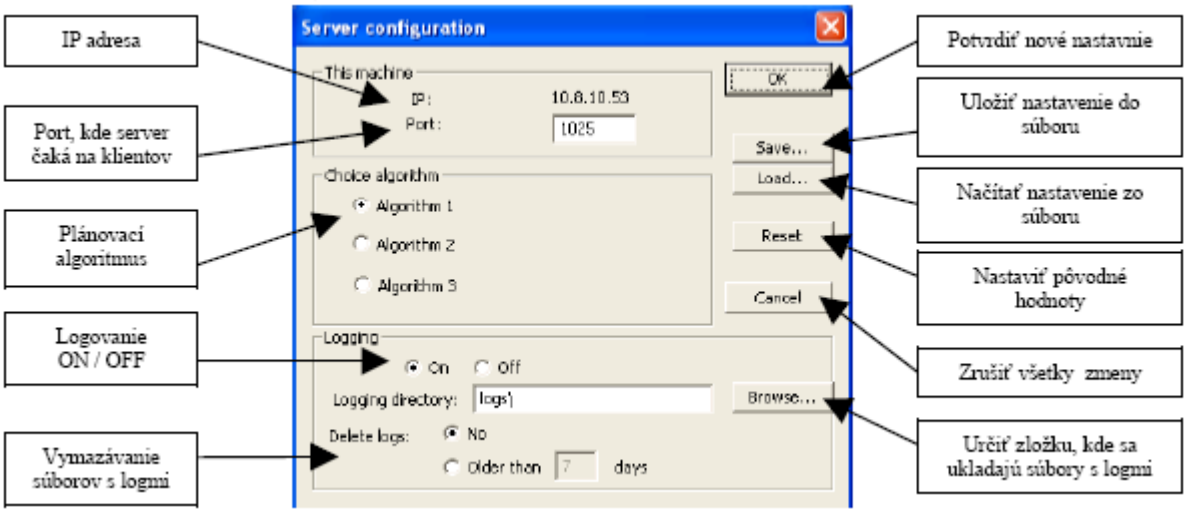
Check network: Server otestuje spojenie so svojimi klientami, aby vedel, či všetci ešte odpovedajú. Klávesová skratka CTRL + N.

Config: Otvorí dialóg, v ktorom sa dá nastaviť konfigurácia programu. Skratka CTRL + F.

Help: Zobrazí nápovedu. Klávesová skratka F1.

Config dialog:

V tomto dialógu je možné pozmeniť chovanie programu.



This machine:

IP: Vlastná IP adresa, hodnota sa v tomto dialógu nedá zmeniť (pod Windows XP sa dá zmeniť v Start\Control panels\Network connections\ ).

Port: Port, na ktorom server čaká na všetku príchodziu komunikáciu od klientov. Túto hodnotu je možné zmeniť aj z príkazového riadku pomocou prepínača -listport (viď Použitie pre ďalšie detaily).

Choice algorithm: Plánovací algoritmus na výber klienta, ktorý spracuje úlohu. Toto nastavenie môže ovplyvniť efektivitu a dĺžku výpočtového procesu.

Algorithm 1: Vyberie prvý voľný stroj zo zoznamu, ktorý je schopný spracovať úlohu. Naivný algoritmus, ale najrýchlejší.

Algorithm 2: Utriedi zoznam klientov podľa spoľahlivosti (pomer medzi zadanými úlohami a odovzdanými výsledkami) a vyberie prvý voľný stroj.

Algorithm 3: Utriedi zoznam klientov podľa času, koľko trvalo spracovanie poslednej úlohy a vyberie prvý voľny stroj.

Logging:

On/Off: Zapne alebo vypne zápis logov do súboru.

Logging directory: Všetky súbory s logmi sa budú ukladať do tejto zložky.

Browse: Otvorí dialóg na výber zložky pre súbory s logmi.

Delete logs:

No: Žiadne súbory s logmi nebudú zmazané.

Older than X days: Pri štarte Anthillu sa zmažú všetky súbory s logmi staršie ako X dní.

OK: Potvrdí a aplikuje všetky aktuálne nastavenia (toto zahŕňa aj zmazanie súborov s logmi, ak sa na to vzťahujú nové nastavenia).

Save: Otvorí dialóg pre uloženie nastavení do súboru.

Load: Otvorí dialóg pre načítanie nastavení zo súboru.

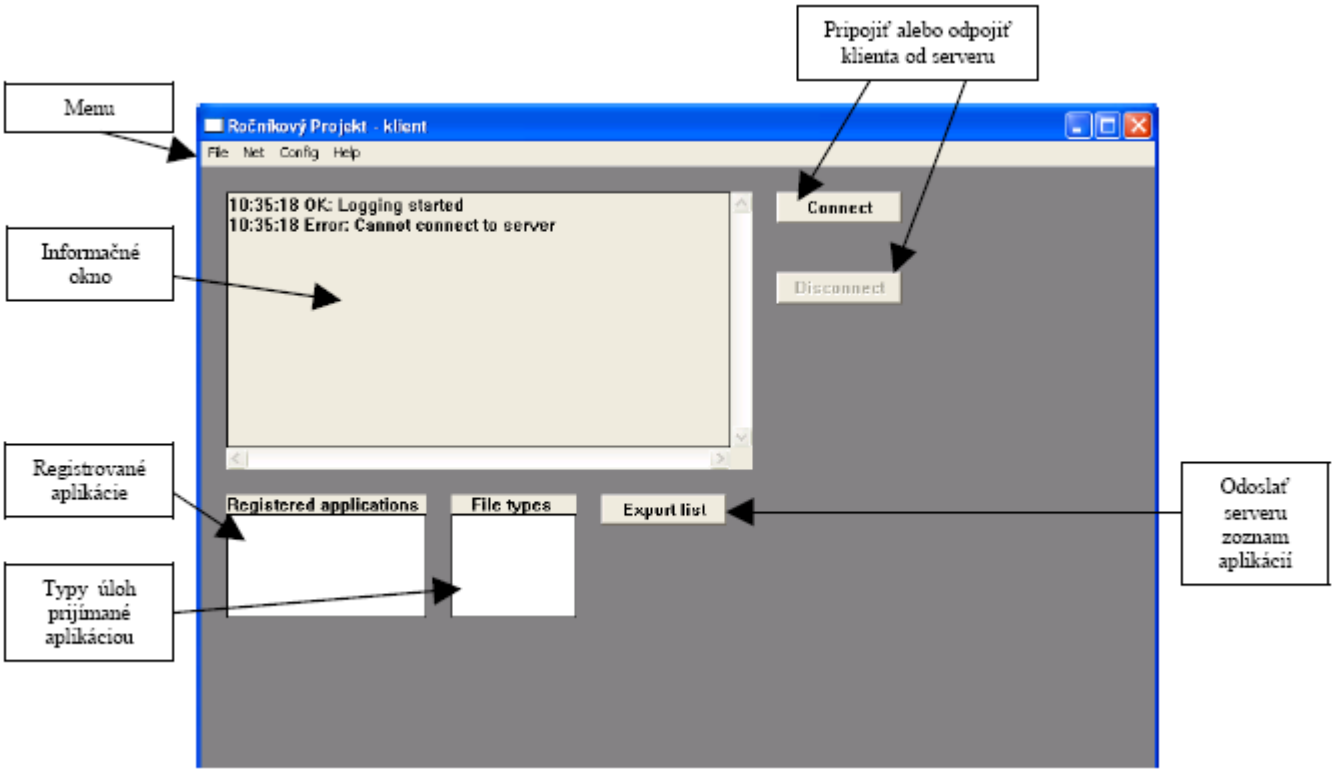
Reset: Nastaví všetky hodnoty na pôvodné nastavenie.

Cancel: Zruší všetky zmeny prevedené v tomto dialógu.

Popis ovládania programu - KLIENT

Hlavné okno:

Nachádza sa tu informačné okno, menšie okná, kde sa zobrazujú informácie o registrovaných aplikáciách a tiež tlačítka na ovládanie pripojenia na server a menu.



Information window:

Zobrazuje všetky dôležité informácie o behu programu.

Registered applications:

Zoznam registrovaných aplikácií u klienta. Iba tieto aplikácie sa môžu účastniť výpočtového procesu.

File types:

Ak chce užívateľ vidieť podrobnosti o registrovanej aplikácii, označí príslušnú aplikáciu v okne Registered applications. Okno File types potom zobrazí typy súborov alebo úloh, ktoré aplikácia podporuje.

Connect, Disconnect:

Tieto tlačítka umožňujú pripojiť a zaregistrovať klienta na serveri, resp. odpojiť a zrušiť registráciu klienta.

Export list:

Odošle na server aktuálny zoznam registrovaných aplikácií.

Menu:

File:

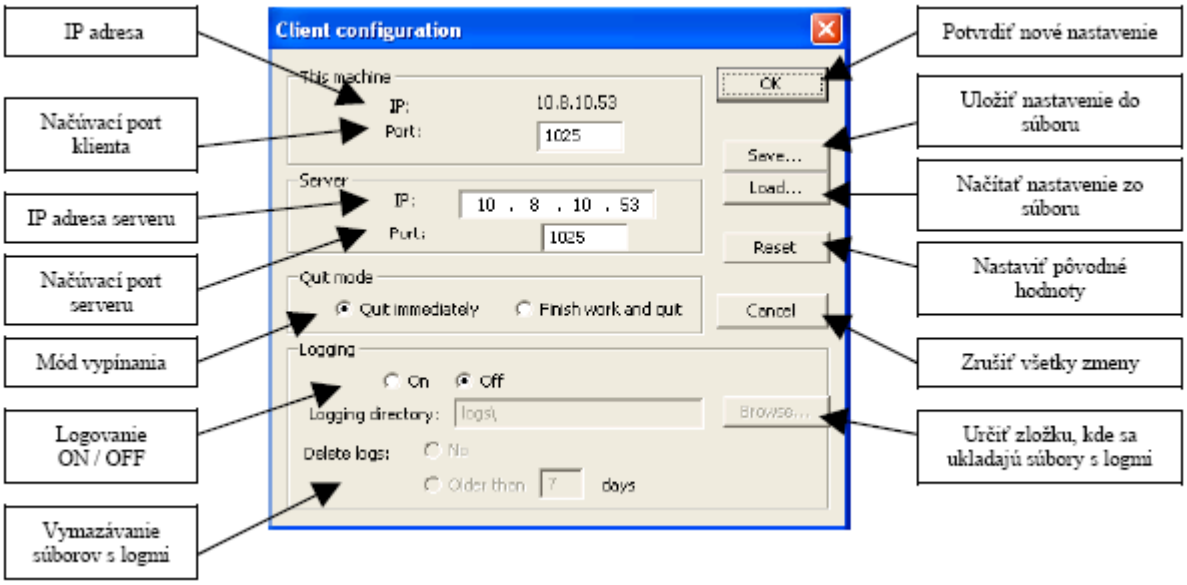
Exit: Ukončí beh aplikácie. klávesová skratka ALT + F4.

Config: Otvorí dialóg, v ktorom sa dá nastaviť konfigurácia programu. Skratka CTRL + F.

Help: Zobrazí nápovedu. Klávesová skratka F1.

Config dialog:

V tomto dialógu je možné pozmeniť chovanie programu.



This machine:

IP: Vlastná IP adresa, hodnota sa v tomto dialógu nedá zmeniť (pod Windows XP sa dá zmeniť v Start\Control panels\Network connections\ ).

Port: Port, na ktorom klient čaká všetku príchodziu komunikáciu. Túto hodnotu je možné zmeniť z príkazového riadka použitím prepínača -listport (viď Použitie pre ďalšie detaily).

Server:

IP: IP adresa serveru. Táto hodnota môže byť zmenená z príkazového riadku použitím prepínača -sendIP (viď Použitie pre ďalšie detaily).

Port: Port, na ktoro server očakáva príchodziu komunikáciu. Táto hodnota môže byť zmenená z príkazového riadku použitím prepínača -sendport (viď Použitie pre ďalšie detaily).

Quit mode:

Quit immediately: Toto nastavenie upravuje správanie sa klienta pri ukončení programu. Keď užívateľ zvolí možnosť Exit, klient oznámi serveru, že užívateľ si praje skončiť okamžite a že server od tohto klienta neobdrží žiadny výsledok (ak mal v čase ukončenia rozpracovanú nejakú úlohu). Ak je nejaký problém so sieťou, klient čaká maximálne 1 sekundu na to, aby svoje ukončenie oznámil serveru, po uplynutí tohto limitu skončí bez ohľadu na to, či sa spojenie podarilo alebo nie.

Finish work and quit: Toto nastavenie upravuje správanie sa klienta pri ukončení programu. Keď užívateľ zvolí možnosť Exit, klient oznámi serveru, že užívateľ si praje skončiť. V prípade, že klient má rozpracovanú nejakú úlohu, oznámi serveru, že výpočet dokončí a po odovzdaní výsledku okamžite skončí. Server už nebude tomuto klientovi posielať nové úlohy.

Logging:

On/Off: Zapne alebo vypne zápis logov do súboru.

Logging directory: Všetky súbory s logmi sa budú ukladať do tejto zložky.

Browse: Otvorí dialóg na výber zložky pre súbory s logmi

Delete logs:

No: Žiadne súbory s logmi nebudú zmazané.

Older than X days: Pri štarte Anthillu sa zmažú všetky súbory s logmi staršie ako X dní.

OK: Potvrdí a aplikuje všetky aktuálne nastavenia (toto zahŕňa aj zmazanie súborov s logmi, ak sa na to vzťahujú nové nastavenia).

Save: Otvorí dialóg pre uloženie nastavení do súboru.

Load: Otvorí dialóg pre načítanie nastavení zo súboru.

Reset: Nastaví všetky hodnoty na pôvodné nastavenie.

Cancel: Zruší všetky zmeny prevedené v tomto dialógu.

Súbory používané programom Anthill (rovnaký význam v roli serveru i klienta):

config.txt: Tento súbor obsahuje celú konfiguráciu, ktoré program potrebuje pre svoj beh. Je možné ju meniť pomocou ľubovoľného textového editora.

ListenPort: Port, kde Anthill čaká na príchodziu sieťovú komunikáciu od iných inštancií Anthillu.

ServerPort: Port, kde server čaká na komunikáciu. Ignorované v režime serveru.

ServerIPAddress: IP adresa serveru. Ignorované v režime serveru.

LoggingDirectory: Plná cesta k zložke, v ktorej majú byť ukladané súbory s logmi.

DeleteAfter: Ak je hodnota DeleteLogs nastavená na 1, potom hodnota DeleteAfter určuje počet dní, koľko budú logy uchovávané.

Logging: Hodnota 0 znamená vypnuté logovanie, hodnota 1 znamená, že logovanie je zapnuté.

DeleteLogs: Hodnota 0 znamená, že žiadny súbor s logmi nebude zmazaný. Hodnota 1 naopak znamená, že všetky súbory s logmi, ktoré sú staršie ako počet dní určený hodnotou DeleteAfter, budú vymazané (z aktuálnej zložky určenej hodnotou LoggingDirectory).

QuitMode: Platné iba v režime klienta. Hodnota 1 znamená, že spôsob ukončenia klienta je "okamžité ukončenie (Quit immediately)",

hodnota 2 znamená, že spôsob ukončenia je "dorátať a skončiť (Finish and quit)" (viď dialóg pre nastavenie klientskej konfigurácie pre ďalšie podrobnosti ohľadom QuitMode).

Algorithm: Platné iba v režime serveru. Hodnota vyjadruje číslo použitého plánovacieho algoritmu na výber klienta (viď dialóg pre nastavenie konfigurácie serveru pre ďalšie podrobnosti).

logging_index.txt: Pomocný súbor pre logovanie, uchováva aktuálne číslo nasledujúceho súboru, do ktorého budú zapisované logy. Je možné ho upraviť pomocou textového editoru. Platné hodnoty sú 0 až 999.

Použitie

Program Anthill je navrhnutý na použitie v menšej sieti počítačov. Jeden stroj s dobrým spojením má úlohu serveru, ostatné počítače sa môžu stať jeho klientmi.

Anthill môže fungovať v dvoch režimoch: ako server alebo v úlohe klienta. Pri spustení programu sa podľa prítomnosti alebo neprítomnosti prepínača -s určí, v ktorom režime bude táto inštancia bežať. Potom sa dá režim zmeniť jedine ukončením a novým spustením programu. Východzí režime je klientský, pre spustenie Anthillu v režime serveru je potreba použiť prepínač -s, viď Voľby.

Voľby:

-s	spustí Anthill v režime serveru
-sendIP1.2.3.4	nastaví IP adresu serveru na 1.2.3.4, platné len v klientskom režime
-sendport1234	nastaví odosielací port na 1234, platné len v klientskom režime
-listport1234	nastaví načúvací port na 1234, platné v oboch režimoch

Príklad:

Predpokladajme, že máme štyri stroje s IP adresami 192.168.1.10, 192.168.1.11, 192.168.1.12 a 192.168.1.13. Ako server zvolíme stroj s IP 192.168.1.10. Na tomto počítači spustíme aplikáciu pomocou príkazu "anthill -s". Zvyšné tri stroje môžu spustiť aplikáciu príkazom
"anthill -sendIP192.168.1.10". Odosielacie a načúvacie porty zostanú nastavené na východzie porty. Ďalšia možnosť je, že klienti budú spustený iba pomocou príkazu "anthill" a IP adresa serveru sa dodatočne nastaví pomocou konfiguračného dialógu na každom klientovi.

Verzia

1.0 (8.9.2007)

Príloha C: Programátorská dokumentácia

Programátorská dokumentácia pozostáva z niekoľkých dokumentov, ktoré sa nachádzajú na priloženom CD. Dokumenty popisujú tieto časti programu:

Accept_proc: popisuje procedúru AcceptProc(), ktorá je zodpovedná za prijímanie všetkej príchodzej komunikácie od serveru či klienta.

Accepting_results: dokumentuje proces prijímania výsledkov od klienta.

Dispatching_results: popisuje proces preposielania výsledku od Anthill serveru k aplikácii.

Networking: detailne popisuje sieťovú komunikáciu medzi klientom a serverom.

Work_proc: dokumentuje procedúru WorkProc(), ktorá tvorí jadro spracovania úloh, obsahuje logiku rozosielania úloh jednotlivým klientom

Wraplib: dokumentuje dynamickú knižnicu wraplib.dll spolu s použitým rozhraním

Wraplib_com: popisuje komunikáciu medzi Anthillom a ktoroukoľvek aplikáciou s využitím knižnice wraplib

Ďalšie informácie možno nájsť v zdrojových hlavičkových súboroch pri deklaráciách funkcií, štruktúr alebo tried.