Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný odbor: Počítačové inžinierstvo
Študijný program: Počítačové systémy a siete

Bc. Peter Fillo, Bc. Marek Kočár, Bc. Jakub Krištofik,
Bc. Miroslav Kropáček, Bc. Martin Kvasnička

Podpora spravovania distribuovaných výpočtov

Tímový projekt

Vedúci projektu: Ing. Dušan Bernát
December 2006

Obsah

Zoznam obrázkov

1 Úvod

Počítače sa už neodmysliteľne zaradili medzi stavebné kamene dnešného sveta. S ich príchodom ľudia začali objavovať dovtedy netušené rýchlosti počítania a spracovania dát. Avšak prvé počítače boli pomerne veľké a ťažké. Preto niektorí dnes nad výrokom publikovaným v roku 1949 odborným časopisom Popular Mechanics len neveriacky krútia hlavami:

> Počítače budú v budúcnosti vážiť nie viac ako 1.5 tony.

Hmotnosť a hlavne výkonnosť dnešných počítačov je dnes v úplne iných dimenziách. Avšak zároveň s počítačmi úspešne vyriešenými problémami sa začali objavovať nové, ktoré požadovali stále viac prostriedkov. Čoskoro začalo byť jasné, že existujú problémy, ktoré sú ďaleko za hranicami jednotlivých počítačov. Počítače sa tak začali všemožne spájať za účelom zvýšenia ich výpočtového výkonu alebo poskytovanej pamäte. A tak sa kruh uzavrel. Znova tu máme superpočítače v obrovských halách a vážiace viac ako 1.5 tony. Ktovie ako sa na tieto monštrá budú dívať ďalšie generácie? Problém riadenia distribuovaného výpočtu je teda pomerne starý. Existuje už množstvo riešení, ktoré ho s väčšími alebo menšími úspechmi rieši a. Naším zámerom v tomto projekte bude navrhnúť ďalšie z nich. Uvedomujeme si, že tento cieľ nie je jednoduchý, ale aj preto sme si ho vytýčili.

1.1 Cieľ projektu

Hlavným cieľom tohto projektu je spraviť aplikáciu na podporu spracovania distribuovaných výpočtov. Je potrebné analyzovať možnosti a potreby systému pre spravovanie používateľských účtov (s rôznymi oprávneniami) a výpočtových úloh na zdieľaných prostriedkoch. Treba navrhnúť systém, ktorý používateľovi umožní vložiť zdrojový kód úlohy, tento skompilovať, spúšťať na zvolených uzloch, sledovať stav systému a ďalšie s tým súvisiace funkcie. Riešenie by malo byť implementované v podobe webového rozhrania k uvedeným funkciám.

Výstupom nášho snaženia by teda mala byť webová aplikácia, pomocou ktorej bude možné riadiť a monitorovať distribuovaný výpočet v prostredí niektorého operačného systému.

1.2 Spôsob realizácie

Na realizáciu zadaného cieľa, resp. cieľov vymedzených v kapitole 1 je možné použiť viacero metód a prostriekov, pričom na každú úlohu sa dá pozerať z viacerých uhlov pohľadu. Pozrime sa na niektoré z nich.

Prvá otázka, ktorá pravdepodobne čitateľa napadne je, aký stupeň paralelizmu vlastne naša aplikácia môže poskytovať. V zásade sú tieto možnosti:

1. Skript, ktorý beží nezávisle od ostatných, berie nejaké vstupné a produkuje nejaké výstupné údaje, pričom na každom počítači budeme dodávať rôzne vstupné údaje.

   - Výhody: okamžité spustenie, jednoduchá implementácia.
   - Nevýhody: závislosť na prostredí (nie všade musí existovať interpreter daného skriptu).

2. Skompilovaná aplikácia, ktorá sa chová v podstate ako skript, tzn. berie nejaké vstupy a produkuje nejaké výstupy.

   - Výhody: vyššia rýchlosť, vačšia flexibilita oproti skriptom.
   - Nevýhody: závislosť na nainštalovaných knižniciach, platforme a operačnom systéme.

3. Aplikácia v zdrojovej forme, tzn. najprv sa na danom uzle skompiluje a až potom spustí

   - Výhody: viď predošlý bod, ale získame naviac ešte možnosť spúšťať aplikáciu na rôznych systémoch a platformách (za podmienky zachovania niektorých pravidiel, napr. normy „ANSI C“ a pod.).
   - Nevýhody: závislosť na jednom programovacom jazyku, nainštalovaných knižniciach a operačnom systéme.

Ako už bolo načrtnuté, môžeme uvažovať buď o samostatne bežiacich programoch, ktoré budú počítať časti úlohy, ale o samotné rozdeľovanie týchto častí sa bude starať používateľ. Druhou možnosťou je program chápať ako časť nejakého celku, kde každý takýto program je riadený iným komponentom v rámci systému (nadriadený uzol, susedný uzol, atď.)

Pokiaľ ide o samotný programovací jazyk či skript a spôsob jeho implementácie, existuje viacero možností:

- Vlastný skriptovací resp. programovací jazyk s natívnou podporou paralelných výpočtov.
- Existujúci programovací jazyk (napr. C, Java, Python, atď.) a vlastná knižnica pre tento jazyk, ktorá by sprostredkovala rozhranie pre paralelné výpočty.
- Existujúca knižnica, ktorá poskytuje rozhranie v niektorom z existujúcich programovacích jazykov.

Nie je ťažké si predstaviť, že obtiažnosť realizácie rastie s implementáciou vlastných riešení. Vzhľadom na obmedzené časové obdobie, ako aj cieľ projektu (poskytnúť funkčný a v praxi nasaditeľný nástroj) sme sa rozhodli pre cestu existujúceho programovacieho jazyka a existujúcej knižnice na paralelné výpočty.

1.3 Prehľad dokumentu

V kapitole 1 sa hovorí o cieľoch tohto projektu, sú tu načrtnuté niektoré možné spôsoby realizácie, ich výhody a nevýhody.

V kapitole 2 je uvedená analýza projektu. Venuje sa skúmaniu existujúcich technológií, ktoré priamo, alebo nepriamo (ako možné systémové a programové prostriedky) súvisia s podporou distribuovaných výpočtov. Tiež je možné nájsť v tejto kapitole prehľad podobných riešení.

V kapitole 3 je uvedená špecifikácia požiadaviek. Zvolili sme tu niekoľko pohľadov na riešený problém. Venujeme sa tu používateľským požiadavkám, tak ako by ich vnímal objednávateľ systému. Ďalej sa venujeme technickým prostriedkom a technológiám ktoré plánujeme využiť pri vytváraní nášho projektu. A na koniec sa venujeme všade prítomnej otázke bezpečnosti.

V kapitole 4 je uvedený návrh všetký podstatných častí vytváraného sytému, opisuje sa tu systém a jeho komponenty. Ako ďalšie opisujeme systém z pohľadu prípadov použitia, kde sa snažíme ukázať možnosti použitia systému z pozície tých, ktorý sa dostávajú do kontaktu so systémom. Je tu uvedený model údajov, tak ako budú ukladané v systéme. Ďalej predstavujeme návrh vnútorného manažéra, ktorý tvorí jadro celého navrhovaného systému. Ide vlastne o akúsi vnútornú logiku a činnosti, ktoré celý systém definujú, pretože má na starosti spravovanie a riadenie distribuovaných výpočtov. Nakoniec uvádzame skutočné, fyzické rozmiestnenie hardvéru a jeho vzájomné prepojenie, ako aj jeho vzťahy so softvérom.

2 Analýza

V kapitole analýzy sa budeme venovať skúmaniu existujúcich technológií, ktoré priamo, alebo nepriamo súvisia s podporou spravovania distribuovaných výpočtov. Okrem technológií kapitola obsahuje aj analýzu existujúcich riešení.

2.1 Programovacie jazyky

2.1.1 Java Server Pages - JSP

Čo sú JSP

JSP sú nástroj pre vytváranie dynamických HTML (alebo XML, ...) stránok založený na jazyku Java, funkčnosťou veľmi podobný ASP alebo PHP. Ide vlastne o HTML stránky, do ktorých sa pomocou špeciálnych značiek (JSP tags) vloží kód v Jave - ten sa vykonáva pri vybavovaní požiadavky na strane serveru. Táto technológia bola vyvinutá spoločnosťou Sun Microsystems.

Na rozdiel od PHP a ASP (resp. VBSkriptu a JSkriptu, ktorý ASP používajú ku skriptovaniu), ktoré nemajú takmer žiadnu typovú kontrolu, JSP majú silnú typovú kontrolu z Javy. Druhou hlavnou odlišnosťou je kompilácia stránok do tzv. servletov, čo sú špeciálne triedy v jazyku Java, ktoré potom komunikují s Web serverem. JSP by sa teda dali tiež charakterizovať ako nástroj na písanie servletov.

Pretože servlety sú triedy v jazyku Java, môžu využívať dalšie triedy, ktoré neboli písané pomocou JSP a pochopiteľne tiež - ako v celej Jave - jednoducho používať už skôr napísané triedy. To umožňuje rýchly vývoj aplikácií a čiastočne tiež oddelenie designu od výkonného kódu.

JSP tiež umožňujú používanie "sessions", teda možnosť uchovania dát konkrétneho používateľa medzi viacerými HTTP požiadavkami na Web serveri.

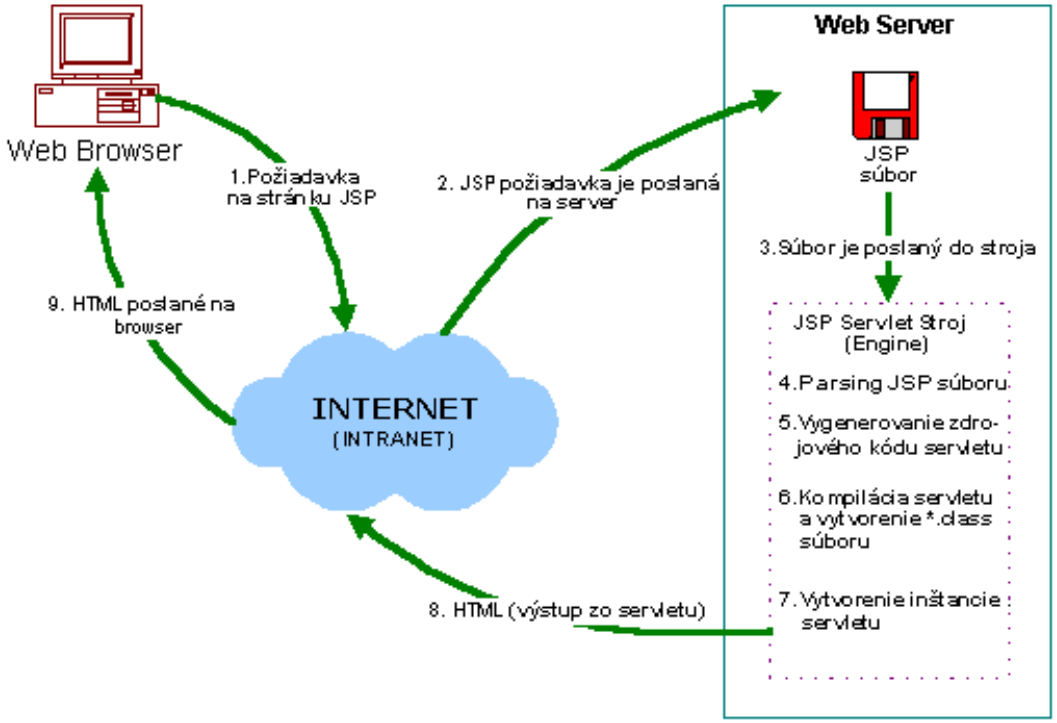
V súčasnosti, veľa webstránok funguje na princípe dynamického splnenia požiadavky klienta. Databáza je veľmi dobrý prostriedok na ukladanie dát používateľov a iných informácií. JDBC poskytuje excelentnú formu konektivity v heterogénnom databázovom prostredí. Použitím JSP a JDBC je veľmi jednoduché vyvíjať databázové web aplikácie. Java je taktiež známa svojim heslom "napíš raz, spusti kdekoľvek". JSP stránky sú platformovo nezávislé a portovateľné na ktorúkoľvek platformu.

Princíp činnosti

Zdrojový kód stránky JSP je prekladaný na servlet a následne kompilovaný. Výsledkom je servlet, ktorý generuje HTML. To je poslané klientovi, ako odpoveď na jeho požiadavku. Detailný postup je opísaný pod nasledujúcim obrázkom:

1. Používateľ požiada o webovú stránku, ktorá bola vytvorená ako JSP. Klient vytvorí požiadavku (request) a smeruje ju cez sieť na server.

2. Požiadavka je nasmerovaná na príslušný web server.



Obr. 1: Architektúra JSP

3. Web server zistí, že požadovaný súbor je špeciálny, pretože má koncovku ”jsp”. Preto presmeruje JSP súbor do JSP Servlet stroja.

4. V prípade, že tento súbor bol volaný prvýkrát, JSP stroj ho prekontroluje. V opačnom prípade pokračuje bodom 7.

5. V ďalšom kroku je vygenerovaný špeciálny servlet, vytvorený na základe súboru JSP. Všetko statické HTML je uloťené v out.println() príkazoch.

6. Zdrojový kód servletu je skompilovaný a je vytvorený ”.class” súbor.

7. Je vytvorená inštancia servletu a zavolajú sa metódy init() a service().

8. Výstupom zo servletu je HTML, ktoré je poslané cez sieť.

9. Používateľovi sa zobrazia výsledky.

Výhody oproti PHP

- možnosť používania Java API v JSP
- využívanie, programovanie vlastných tagov
- distribuovanie aplikácie bez nutnosti ukazovania zdrojových textov
- programovanie v JSP umožňuje zvýšenú prenositeľnosť aplikácií a multiplatformovosť

- využívanie JAVA beans - znovupoužiteľných komponentov
- možnosť použitia Enterprise Java beans na zabezpečenie perzistenice objektov, zjednodušuje programovanie medzi aplikačnou a dátovou vrstvou
- technológiu Hibernate pre mapovanie objektov do relačnej databázy

Softvér na vývoj a testovanie JSP

Na testovanie stránok JSP je treba mať server, ktorý implementuje špecifikáciu Java servlet 2.1 alebo 2.2 a Java Server Pages 1.0 alebo 1.1. V prípade, že už používate niektorý z komerčných aplikačných serverov, napr. BEA WebLogic, IBM WebSphere, je možné, že máte všetko čo potrebujete. Samotné písanie stránok JSP je totiž možné aj v notepade. Väčšina z nás však siahne po svojom obľúbenom editore HTML. Ale je dobré ak máte pre potreby testovania jednoduchý a nenáročný server, ktorý si môžete nainštalovať na svojom počítači. Takýchto serverov je niekoľko:

- Apache Tomcat: Tomcat je referenčná implementácia Servlet 2.3 a JSP 1.2. Je rýchly a nenáročný na výkon, ale treba vyvinúť väčšie úsilie v prípade konfigurácie tohto servera. Je možné ho použiť samostatne (standalone), alebo ho integrovať do servera Apache.
- Macromedia JRun: JRun je možné integrovať do MS IIS, MS Personal Web Server, StarNine WebSTAR a podobne. Obmedzená verzia je zdarma, komerčná poskytuje ďalšie možnosti.
- Gefion Software LiteWebServer (LWS): LWS je malý server, ktorý vychádza z Tomcatu. Podporuje špecifikáciu Servlety 2.3 a JSP 1.2.

2.1.2 PHP

PHP[1] je skriptovací jazyk, určený predovšetkým pre programovanie dynamických internetových stránok. PHP je jedným z najpoužívanejších spôsobov tvorby dynamicky generovaných www stránok.

História PHP

V roku 1994 si Rasmus Lerdof vytvoril v Perle jednoduchý systém pre evidovanie prístupu k jeho stránkam. Pretože neustále spúšťanie interpretera Perlu veľmi zaťažovalo WWW server, prepísal autor systém do jazyka C. Napriek tomu, že bol celý systém určený pre Rasmusovo použitie, zapáčil sa aj ostatným používateľom servera a začali ho používať. Systém sa stal obľúbeným a používalo ho stále viac používateľov. Tí prichádzali s požiadavkami na vylepšenie celého systému. Autor preto systém rozšíril, doplnil dokumentáciu a uvoľnil ho pod názvom „Personal Home Page Tools“.

Okrem tohto vytvoril aj nástroj, ktorý umožňoval začlenenie SQL požiadaviek do stránok, vytváranie formulárov, a zobrazovanie výsledkov SQL požiadaviek. Prelomová verzia PHP 3.0, v ktorej bol systém zrýchlený a rozšírený o mnoho ďalších funkcií. PHP 3.0 bola prvá verzia aj pod OS Windows. Obsah skratky PHP úplne

[1] skratka znamená PHP: Hypertext Preprocessor, pôvodne bola: Personal Home Page

stratil svoj pôvodný význam a nahradil ho nový hypertextový preprocesor PHP. V máji 2000 bolo vydané PHP 4, ktorého jadro tvoril nový Zend Engine 1.0. V júli 2004 bola vydaná verzia PHP5 s jadrom Zend Engine 2.

Vlastnosti jazyka PHP

PHP sa najčastejšie začleňuje priamo do štruktúry jazyka HTML. Tu je príklad:

```
<html>
   <head>
       <title>Príklad</title>
   </head>
   <body>
       <?php
       echo "Ahoj, ja som PHP skript!";
       ?>
   </body>
</html>
```

Je to veľká výhoda, pretože môžete spraviť iba tú časť html-stránky dynamickú, ktorú naozaj potrebujete. Zvyšok necháte štandardný html kód.

Sú tri hlavné oblasti, kde sa PHP používa.

- Skriptovanie zo strany servera. Toto je najtradičnejšie a hlavné cieľové pole pre PHP.
- Skriptovanie príkazového riadku. PHP skript môžete urobiť tak, aby bežal aj bez servera alebo prehliadača www-stránok.
- Písanie klientských GUI aplikácii. PHP nie je možno najlepším jazykom na tvorbu desktop aplikácii s grafickým používateľským rozhraním, ale existuje aj táto možnosť, pomocou knižnice PHP-GTK.

PHP skripty sú vykonávané na strane servera. K používateľovi je prenášaný až výsledok ich činnosti. Syntax jazyka kombinuje niekoľko programovacích jazykov (Perl, C, Pascal a Java). PHP je nezávislý na platforme, skripty fungujú bez úprav na rôznych operačných systémoch. Obsahuje rozsiahle knižnice funkcií pre spracovanie textu, grafiky, práce so súbormi, prístup k väčšine databázových systémov (MySQL, PostgreSQL, Oracle, ODBC). Podporuje veľa internetových protokolov (http, SMTP, SNMP, FTP, IMAP, POP3, LDAP, ...)

Jazyk je typovo dynamický, to znamená, že dátový typ premennej sa určí v okamihu priraďovania hodnoty.

PHP sa stalo veľmi obľúbeným predovšetkým vďaka jednoduchosti použitia a tomu, že kombinuje vlastnosti viacerých programovacích jazykov a necháva tak programátorovi čiastočnú slobodu v syntaxi. V kombinácii s databázovým serverom (predovšetkým s MySQL, alebo PostgresSQL) a webovým serverom Apache je často využívané k tvorbe webových aplikácii.

Vo verzii 5 sa výrazne zlepšil prístup k objektovo orientovanému programovaniu. Prístup je veľmi podobný ako v jazyku Java.

PHP je alternatívou k systémom: Microsoft ASP,VBScript,JScript, Sun Microsystems JSP/Java alebo k CGI/Perl

2.1.3 Perl

Perl je programovací jazyk, ktorý navrhol Larry Wall. Prvy krát bol oficialne vydaný v roku 1987. Perl si „prepožičal“ niektoré funkcie z ostatných programovacích jazykov, ako sú: jazyk C, shell-skript sh, AWK, sed a Lisp.

História

Larry Wall začal vyvíjať Perl v roku 1987, keď pracoval ako programátor v Unisys. 18. decembra 1987 vydal verziu 1.0. Jazyk sa veľmi rýchlo rozšíril v priebehu pár rokov. V roku 1988 bola vydaná verzia 2, ktorá dokázala lepšie pracovať s regulárnymi výrazmi, a v roku 1989 bola vydaná verzia 3, ktorá mala podporu pre binárne dáta.

Do roku 1991 bola jediná dokumentácia dlhá, stále narastajúca manuálová stránka. V roku 1991 bola vydaná publikácia „Programming Perl“, ktorá sa stala základnou príručkou pre Perl. V rovnakom čase prišla aj verzia 4, ktorá nemala nejaké podstatné zmeny, ale bola to verzia, pre ktorú bola vydaná príručka.

Perl prešiel rôznymi úpravami až do roku 1993, kedy bola vydaná posledná štvorková verzia – Perl 4.036. V tomto momente začal Larry Wall pracovať na verzií 5.

Perl 5 bol vydaný 17. októbra 1994. Bol vytvorený pre rôzné platformy. Bol prepísaný prekladač (interpreter), a bolo pridaných veľa funkcii, ako napríklad objekty, vzťahy (references), balíčky a moduly.

26. októbra 1995 bol vydaný CPAN (Comprehensive Perl Archive Network). CPAN je zbierka webových stránok, ktoré archivujú a distribujú Perlovské zdrojové súbory, binárne súbory, dokumentácie, skripy a moduly.

Perl je dodnes aktívne udržiavaný. Bolo pridaných niekoľko dôležitých funkcií, bola dorobená podpora pre Unicode, vlákna, podpora pre objektovo orientované programovanie a ďalšie funkcie. V súčasnosti v roku 2006 posledná stabilná verzia je 5.8.8.

Vlatnosti Perl-u

Perl je programovací jazyk, pôvodne navrhnutý pre manipuláciu s textom. Dnes sa používa v širokom rozsahu, na správu systému, na vyvíjanie webových, sieťových a grafických aplikácii.

Perl je orientovaný na jednoduché použitie, efektivitu a robustnosť. Podporuje rôzne programovacie paradigmy ako sú procedurálne, objektovo orientované a funkcionálne programovanie. Má automatickú správu pamäti, vstavanú podporu práce s textom a veľkú zbierku rôzných externých modulov.

Pomocou perlu je tiež možné vytvárať shell-skripty. Podobné ako Unix-ovské shell-y, aj Perl má veľa vstavaných funkcií ako je napríklad triedenie, alebo prístup k systémovému vybaveniu.

Struktúra Perl-u je odvodená z jazyka C. Perl prebral zoznamy z Lisp-u, asociatívne polia z AWK a regulárne výrazy zo sed-u. Tieto funkcie umožňujú jednoduchú syntaktickú analýzu, narábanie s textom a s dátami.

Vo všetkých verziách Perl-u je automatické definovanie typu premenných a automatická správa pamäti. Prekladač vie typ a požiadavky na pamäť každého dátového objektu v programe a alokuje a uvoľňuje miesto pre nich, tak ako je to potrebné.

Použitie

Perl začal byť používaný na vytváranie CGI skriptov. Je známy ako jeden z troch (Perl, Python a PHP) najpoužívanejších skriptovacích jazykov pre webovské aplikácie. Veľké projekty vytvorené v Perl-e sú: Slash, IMDb, UseModWiki. Z najnavštevovanejších web stránok sú to: amazon.com a ticketmaster.com.

2.2 Paralelné prostredie

2.2.1 Parallel Virtual Machine (PVM)

`Parallel Virtual Machine`[1] (ďalej `PVM`) je softvérový balík, ktorý umožňuje vykonávať paralelné výpočty. Tento systém vytvára jeden virtuálny paralelný počítač zo skupiny heterogénnych, alebo homogénnych počítačov. Ako formu komunikácie používa `PVM` zasielanie správ. Pod pojmom množina heterogénnych počítačov rozumieme množinu počítačov, ktoré majú rôznu platformu, alebo rôzny operačný systém.

História

„Projekt `PVM` vznikol v lete roku 1989 v *Oak Ridge National Laboratory*. Prototyp systému, `PVM 1.0`, vytvoril *Vaidy Sunderam* a *Al Geist* pre interné potreby inštitútu. Táto verzia nebola verejnosti k dispozícii. Verziu `PVM 2` vytvorila a zverejnila Univerzita Tenessee v marci roku 1991. Odvtedy sa začal systém `PVM` používať v mnohých vedeckých výskumoch. Vďaka spätnej väzbe od používateľov a po celej rade zmien `PVM 2.1 - 2.4`, bol celý systém kompletne prerobený. Vo februári roku 1993 bola zverejnená verzia 3“ [2]. V súčastnosti je projekt k dispozícii vo verzii 3.4.5.

Charakteristika systému

`PVM` je systém, ktorý vytvára jeden virtuálny paralelný počítač nad množinou viacerých počítačov. Týmto spôsobom získa systém väčší výpočtový výkon a z neho vyplývajúcu vyššiu rýchlosť. `PVM` využíva na komunikáciu medzi uzlami (počítačmi) zasielanie správ. Zasielanie správ je typická forma komunikácie v multipočítačových systémoch. V multiprocesorových systémoch je typická forma komunikácie komunikácia prostredníctvom zdieľanej pamäte.

Pojem heterogénnosti sa nespája iba s architektúrou a operačným systémom ale aj s pojmami ako:

- formát údajov,
- rýchlosť výpočtu,
- vyťaženie uzla,
- vyťaženie siete.

So všetkými týmito formami heterogénnosti uzlov, sa systém vyrovnáva pomocou zasielania správ. `PVM` dokonca eviduje vyťaženie uzlov a na jeho základe spúšťa,

resp. nespúšťa na daných uzloch úlohy.

Na vytvorenie virtuálneho počítača je potrebné zkonfigurovať PVM na všetkých počítačoch, ktoré ho majú tvoriť. Na každom z nich musí existovať proces - démon. V prípade aktuálnej verzie má démon názov pvmd3. Komunikácia démonov si vyžaduje prepojenie počítačovou sieťou. Na komunikáciu sa v súčastnosti používa zväčša protokol SSH. Pomocou konzoly a programu pvm možno meniť konfiguráciu systému PVM. Jednou zo základných funkcií je pridávanie a odoberanie uzlov. Distribuovaný výpočet sa realizuje pomocou procesov. Démoni zabezpečujú vytváranie a ukončovanie procesov, komunikáciu medzi nimi, atď.

PVM poskytuje programátorom funkcie, pomocou ktorých môžu riadiť a realizovať samotný distribuovaný výpočet. Tieto funkcie sú implementované v knižnici. Pri kompilácii stačí túto knižnicu prilinkovať. Mechanizmus zasielania správ obsahuje možnosť zabalovania údajov. Ak chce programátor zaslať niekoľko rôznych údajov, jednoducho ich uloží do zásobníka[2]. Vrstvenie údajov do zásobníka počíta okrem premenných známych údajových typov aj s možnosťou vkladania vlastných štruktúr a polí rôznych údajových typov. Systém PVM podporuje programovacie jazyky C, a Fortran. Okrem týchto programovacích jazykov existuje aj implementácia v programovacom jazyku Perl. Konkrétne ide o modul CPAN - Parallel-Pvm [3]. Modul prepisuje použitie knižničných funkcií z jazyka C do jazyka Perl. Okrem bežných funkcií, obsahuje modul funkcie na pridávanie a odoberanie uzlov, spúštanie démona pvmd, sprístupňuje informácie o konfigurácii systému, atď. Systém PVM má aj grafické rozhranie známe pod označením xpvm.
t Xpvm je program, ktorý umožňuje diagnostikovať komunikáciu medzi uzlami, odhaľovať chyby, meniť konfiguráciu systému, spúšťať distribuované výpočty, atď.

Systém PVM je vhodný na použitie v projekte spravovania distribuovaných výpočtov, pretože:

- slúži na realizáciu paralelných výpočtov,
- vytvára jeden virtuálny paralelný počítač,
- má podporu heterogénnych uzlov,
- podporuje programovacie jazyky C, Fortran,
- je zadarmo.

2.2.2 The Message Passing Interface (MPI) standard

The Message Passing Interface standard[4] (ďalej MPI) slúži na štandardizáciu programov, ktoré využívajú zasielanie správ. Cieľom projektu je vytvoriť praktický, prenositeľný, efektívny a flexibilný štandardný prístup na zasielanie správ. Na vývoji štandardu participuje príspevkami a komentármi okolo štyridsať spoločností.

[2]V tomto prípade ide o frontu správ, nakoľko sa pri vyberaní údajov dodržuje poradie v akom boli údaje do frontu vložené

História

Vývoj MPI možno datovať od roku 1992. V roku 1994 vznikla prvá verzia MPI 1.0, o rok neskôr verzia 1.1. Rozdiel v týchto verziách bol podľa zdroja [5] nepatrný.

Väčšou zmenou bol štandard MPI-2. Ten už nezahŕňal iba opravy predošlých verzii, ale zaoberal sa aj novými oblasťami.

Impulzom pre vznik štandardu bolo, že každý výrobca Massively Parallel Processor (MPP) vyvíjal vlastné rozhranie na zasielanie správ. Potreba štandardizovať zasielanie správ, medziprocesorovú komunikáciu, teda vzišla z neprenositeľnosti implementovaných aplikácií. Tvorcovia MPI mali možnosť vybrať niektorú implementáciu a prehlásiť ju za štandard. Rozhodli sa však vytvoriť nový štandard na základe kombinácii najlepších existujúcich implementácií. A tento nový štandard implementovať v podobe knižnice.

Charakteristika systému

Rovnako ako PVM využíva MPI na komunikáciu medzi počítačmi zasielanie správ a poskytuje programátorom funkcie, ktoré sú implementované v knižnici. MPI podporuje programovacie jazyky C, C++ a Fortran.

Štandard MPI-2 obsahuje okrem opráv chýb nižších verzií aj nasledujúce nové oblasti:

- komunikácia bod-bod,
- virtuálna topológia siete,
- vytváranie a manažment procesov,
- jednostranná komunikácia,
- pokročilé skupinové operácie,
- vstupno/výstupné zariadenia a porty,
- ďalšie jazykové väzby,
- atď.

Oblasti, ktoré implementuje MPI-2 sú približne rovnaké ako oblasti PVM. Návrh, aj implementácia, sú však rozdielne. Pokiaľ ide o implementáciu MPI-2, treba upresniť zopár faktov. Implementácia tohoto štandardu nebola realizovaná pod označením MPI-2. Najznámejšie implementácie sú MPICH, a ANL/MSU MPI. Oficiálna implementácia pod označením MPI-2 neexistuje.

2.2.3 Porovanie PVM a MPI

Porovnaní PVM a MPI už bolo napísaných niekoľko. MPI prišlo s verziou 1.0 v čase, keď už systém PVM mal niekoľko tisíc používateľov. Za vývojom MPI stojí vyšše 40 spoločností, medzi ktoré nepatria iba firmy, ale aj univerzity a vývojové pracoviská. Medzi zvučné mená, ktoré vývoj MPI podporujú, sú napríklad IBM, Intel,

HP, Silicon Graphics. S príchodom MPI-2 sa rozdiely v implementácii medzi PVM a MPI prehĺbily. Napriek veľkej podpore a snahe štandardizovať zasielanie správ má PVM stále svoje miesto na trhu. Na vývoji zasielania správ sa naďalej pracuje. Pravidelne sa organizujú konferencie za účasti oboch strán táborov. Konferencie nesú názov pvMpi.

30. mája 1996 vyšlo porovnanie autorov *G.A Geist, J. A. Kohl, P. M. Papadopoulos* s názvom PVM and MPI : a Comparison of Features - PVM a MPI: Porovnanie funkcií [6]. Autori sa pri porovnaní zameriavajú na situácie, kedy je vhodnejšie použiť PVM a kedy MPI. Tomuto porovnaniu sa nemá význam bližšie venovať, pretože porovnáva staršie verzie projektov.

Na stránkach projektu MPI sú uverejnené porovnania [7, 8] medzi týmto štandardom a PVM. Už podľa názvov[3] je zrejmé, že autori v projektoch našli veľké rozdiely. Jeden z rozdielov je, že MPI obsahuje možnosť vytvárať virtuálnu topológiu siete. Ďalším, väčším, rozdielom je prístup k lokálnym prostriedkom počítačov.

„PVM a MPI sa často porovnávajú. Tieto porovnania často začínajú nevysloveným predpokladom, že PVM a MPI predstavujú rôzne riešenia toho istého problému. V skutočnosti tieto dva systémy často riešia rôzne problémy“ [7].

Autori tiež tvrdia, že v situáciách, kedy je typ problému rovnaký, sa výsledky použitia týchto systémov rôznia. Je to prirodzene spôsobené rôznymi cieľmi, a rôznou implementáciou projektov. Porovnávanie funkcií rad za radom môže byť podľa autorov zavádzajúce, pretože oba systémy používajú rovnaké výrazy pre úplne rozdielne koncepty implementácie.

Zaoberať sa konkrétnymi rozdielmi medzi týmito projektami nemá zmysel. Dôležité je, na aký účel paralelné spracovanie potrebujeme a aké prostriedky máme k dispozícii. Z pohľadu vytvárania rozhrania pre spravovanie distribuovaných výpočtov sú rozdiely v implementácii nepodstatné.

Oba systémy sú vhodné na vytvorenie prostredia na distribuované výpočty. Z hľadiska vytvorenia rozhrania na spravovanie distribuovaných výpočtov sú tieto systémy použiteľné a rovnocenné. Jedným z kritérií pre použitie týchto systémov je ich komunikácia s rozhraním spravovania. Keďže komunikácia medzi procesom systému a webovým rozhraním nie je problematická, oba systémy spĺňajú toto kritérium.

2.3 Databázové systémy

2.3.1 MySQL

MySQL je viacvláknový, viacpoužívateľský SQL Databázový systém. MySQL Server je dostupný ako voľne šíriteľný softvér pod GNU GPL licenciou. Existuje tiež platená verzia MySQL Enterprise, ktorá je pre firmy. a dvojitá licencia (dual-license), pre prípady použitia, ktoré nie sú v zhode s GPL.

[3] Why are PVM and MPI So Different - Prečo sú PVM a MPI tak rozdielne, PVM and MPI are completely different - PVM a MPI sú úplne rozdielne

MySQL vlastní a sponzoruje Švédska firma MySQL AB, ktorá vlastní práva na väčšinu zdrojových kódov. Spoločnosť vyvíja a poskytuje aktualizácie pre systém, predáva podporu a servis, tak ako platené verzie MySQL.

Existujú programy pre používateľov, ktoré sú napísané v rôznych programovacích jazykoch na prístup do MySQL databázy. Sú to napríklad C, C++, C#, Borland Delphi, Smalltalk, Java, PHP, Perl a mnoho ďalších.

MySQL je veľmi populárny pre webové aplikácie, ako napríklad MediaWiki, alebo Drupal, a patrí do skupín LAMP, MAMP a WAMP (Linux/Mac/Windows-Apache-MySQL-PHP/Prel/Python). Jeho popularita s webovými aplikáciami je spojená s popularitou PHP, ktoré je často spájané s MySQL

Spravovanie MySQL databázy je možné buď z príkazového riadku, použitím grafických aplikácii, alebo web aplikácii. Známa voľne dostupná web aplikácia je phpmyadmin.

MySQL dokáže pracovať na rôznych platformách sú to napríklad: Linux, MacOS X, BSD, Windows, a mnoho ďalších.

Zdrojové kódy MySQL sú napísané v jazyku C. Syntakticky analyzátor pre jazyk SQL používa yacc[4] a lexer[5]

Skoršie verzie MySQL mali len niektoré črty zo štandardných relačných databázových systémov (ďalej RDB) a súčasným verziám stále chýba veľa vlastností iných RDB. Toto kritizuje veľa databázových expertov, a spochybňujú, že MySQL patrí do rodiny RDB. Veľa vecí pre ktoré bolo MySQL v minulosti kritizované boli pridané v novších verziách.

Funkcie MySQL a obmedzenia

MySQL podporuje indexovanie pomocou binárneho stromu, vnorené príkazy podporuje iba pri príkazoch INSERT a REPLACE. MySQL nevykoná príkaz:

```
SELECT * FROM tabulka1 WHERE id IN (SELECT id FROM tabulka2)
```

Je možné vykonať príkaz pomocou LEFT JOIN, alebo celý príkaz preprogramovať ručne, ale je to komplikovanejšie.

Pre bezpečnosť má v sebe MySQL vytvorený systém používateľov a práv. Prístup do databázy je chránený heslom, pričom každý používateľ môže mať definovanú množinu operácii, na ktoré má oprávnenie ich vykonávať. To znamená, že niekto môže iba čítať, iný zapisovať či meniť štruktúru tabuliek. Pre zvýšenie ochrany sa hesla prenášajú iba v zašifrovanej podobe.

Limity veľkosti tabuliek boli do verzie 3.22 nastavené na 4GB. Od verzie 3.23 sa limit posunul na 2^{63} bajtov, čo predstavuje 8 miliónov terabajtov. V praxi to znamená, že MySQL je do objemu dát obmedzované iba operačným systémom.

Nevýhodou MySQL je takmer nulová kontrola zložitejších reintegračných obmedzení. Väzby medzi tabuľkami si musíte pamätať sami a kontrolovať, aby ste vzťahy neporušili. Pri definovaní novej tabuľky MySQL číta konštrukciu FOREIGN KEY, ale nezohľadňuje ju pri práci s tabuľkami.

Najväčšou zbraňou MySQL je rýchlosť. Oproti PostgreSQL je niekoľko krát rýchlejší. Autori sami hovoria, že práve rýchlosť je ich hlavnou prioritou a že kvôli nej sa ani nechystajú implementovať niektoré funkcie.

[4]Yacc je program, ktorý sa pokladá za štandardný syntaktický analyzátor na unixových systémoch.

[5]Lexer je program, ktorý robí lexikálnu analýzu.

2.3.2 PostgreSQL

PostgreSQL je výkonný „open source“ relačný databázový systém. Je vyvíjaný viac ako 15 rokov a má overenú architektúru, ktorá získala silnú reputáciu v spoľahlivosti, v integrite dát a v správnom fungovaní. Je to asi najprepracovanejší „open-source“ databázový systém. Je možné spúšťať PostgreSQL na väčšine operačných systémoch ako Linux, UNIX, Mac OS X a Windows. Je plne zhodný s ACID[6] a podporuje „foreign keys, joins, views, triggers, and stored procedures“. Podporuje tiež väčšinu SQL 92 a SQL99 dátových typov. Má programové prostredie pre C/C++, Java, .Net, Perl, Python, ODBC a iné a dobrú dokumentáciu.

Za 15 rokov dostalo PostgreSQL niekoľko ocenení, ako napríklad „Linux New Media Award“, „The Linux Journal Editors' Choice Award“ za najlepší databázový systém.

Funkcie a obmedzenia

PostgreSQL má rôzne zaujímavé funkcie, ktoré sú opísané v [16]. Veľkosť databázy je neobmedzená, maximálna veľkosť tabuľky je 32TB, maximálna veľkosť riadka 1,6TB, maximálna veľkosť poľa je 1 GB. Maximálny počet riadkov v tabuľke je neobmedzený, maximálny počet stĺpcov v tabuľke je od 250 do 1600 v závislosti od typu stĺpca. Počet indexov je tiež neobmedzený.

PostgreSQL má plnú podporu vnorených príkazov. Integrácia dát zahŕňa primárne kľúče, cudzie kľúče, s aktualizáciou/mazaním, obmedzenia na kontrolu, unikátnosť, ne-nulovú hodnotu. Vývoj je už tak ďaleko, že zamykanie tabuliek nahradili systémom pre viac verzií dát. V praxi to znamená, že zatiaľ čo jedno vlákno niektoré záznamy prepisuje, iné môže bez prerušenia z rovnakej tabuľky čítať konzistentné dáta.

Bezpečnostná politika PostgreSQL je na vyššej úrovni ako má MySQL. Okrem klasického prístupu chráneného heslom, umožňuje PostgreSQL použitie autentifikácie pomocou Kerberosu, alebo takzvaného "Identifikačného protokolu"

Niektoré funkcie PostgreSQL aj prevzal, je to napríklad auto-inkrementácia, LIMIT/OFFSET, čo umožňuje vracať výpisy po častiach. Podporuje aj zmiešane, unikátne, čiastočné a funkcionálne indexovanie, pri ktorom môžete použiť niektoré z B-stromu, R-stromu, hash, GiST[7] metódy.

Ďalšou výhodou je dedenie tabuliek, pravidlá systému a udalosti databázy. Dedenie tabuliek dáva objektovo orientovanú časť do vytvárania tabuliek, umožňuje navrhovateľom databáz odvodzovať nové tabuľky zo starých. PostgreSQL podporuje aj viacnásobné dedenie. Je možné vytvoriť si aj vlastné dátové typy.

Existuje aj takzvaný dotazovo prepisovací systém, ktorý umožňuje vytvoriť pravidlá, ktoré vykonajú určitú operáciu, napríklad vypíšu konkrétnu tabuľku.

Systém udalostí je systém na medziprocesovú komunikáciu, v ktorom môžu byť posielané správy a udalosti medzi klientami, ktorí používajú príkazy „LISTEN“ a „NOTIFY“. Tento systém umožňuje jednoduchú komunikáciu rovný s rovným, ale aj zložitú koordináciu na udalosti databázy. Používatelia PostgreSQL takto môžu monitorovať udalosti databázy ako sú vkladanie, mazanie, upravovanie tabuliek.

[6] Atomicity, Consistency, Isolation, and Durability

[7] GiST (Generalized Search Tree) – je pokročilý systém, ktorý dáva dokopy rôzne sortovacie a vyhľadávacie algoritmy. Sú to B-strom, B+-strom, R-strom, čiastočné spočítavacie stromy, usporiadané B+-stromy a mnoho ďalších.

Podpora v PostgreSQL existuje aj pre národné lokalizácie. Systém dokáže čítať rôzne kódovania. Pokiaľ server a klient používajú rôzne kódovania, PostgreSQL dokáže automaticky toto kódovanie prevádzať.

Zdrojový kód je dostupný pod viac liberálnou open-source licenciou – BSD licencia. Licencia dáva slobodu na používanie, menenie a distribuovanie PostgreSQL v ľubovoľnej forme akú chcete, aj zatvorený zdrojový kód. Hocijaké zmeny spravíte, tak je to vaše a môžete si s tým robiť čo chcete.

2.4 Rozbor platformovo závislých komponentov

Táto kapitola si dáva za cieľ poskytnúť čitateľovi základný prehľad o možnostiach a problémoch súvisiacich s behom a používaním nášho systému na rôznych platformách.

2.4.1 Implementačné prostredie

Zadanie nešpecifikuje, resp. neodporúča žiadny konkrétny operačný systém či platformu, na ktorej by mala naša aplikácia bežať. Naskýta sa teda otázka, z čoho sa dá vyberať a aké výhody a nevýhody z daného výberu plynú.

Najprv si však zadefinujme, čo vlastne od daného systému požadujeme na beh našej aplikácie:

- databázu (zoznam používateľov, oprávnení, hesiel, ...)
- web server (zobrazovanie ponuky, interakcia s používateľom, spúšťanie skriptov)
- kompiláciu bez asistencie používateľa (cez príkazový riadok resp. skript)
- nástroj na jednoduché prideľovanie procesorového času (pre distribuované výpočty)

Existujúci stav

Výstup tohto projektu by mala byť aplikácia na spracovanie distribuovaných výpočtov. Jej hlavné nasadenie sa očakáva predovšetkým na pôde FIIT STU v prostredí s centrálnym serverom a podriadenými uzlami so zdieľaným súborovým systémom. Konkrétne ide o:

- centrálny server s operačným systémom FreeBSD
- uzly s operačným systémom GNU/Linux
- zdieľanie domovských adresárov cez "Network File System"
- 10 Mbps sieť

Výber implementačného prostredia teda do značnej miery závisí aj na už existujúcom technickom vybavení.

Microsoft Windows

Na domácich počítačoch momentálne najrozšírenejší systém. Firma Microsoft poskytuje komplexné riešenia ako v desktopovej (Windows 9x, Windows XP), tak aj serverovej oblasti (Windows NT, Windows 2000, Windows 2003 Server; web server Microsoft Internet Information Server, databázový server Microsoft SQL Server). Taktiež je k dispozícii mohutný vývojársky balík (Microsoft Visual Studio 2003/2005).

Ide teda o možnosť nainštalovať si kompletné riešenie od jednej firmy, kde je istý predpoklad na vzájomnú kompatibilitu a jednotnosť konfigurácie.

Proti použitiu tohto riešenia však hovorí viacero faktorov:

- cena (aj keď školy a študenti dostávajú výrazné zľavy, stále ide o platené produkty)
- uzavretosť (nie je možné si jednoducho systém rozširovať či meniť)
- nekompatibilita (web server či databáza pracuje len pod OS Windows a samotná komunikácia s inými systémami tiež nie je bezproblémová)
- nároky na hardvér (systém by mal byť schopný fungovať aj na priemernej konfigurácii)
- nároky na administráciu (systém je často náchylný na chyby či vírusy)
- veľmi limitované možnosti skriptovania a automatizácie kompilovania bez ďalších nástrojov (nezriedka opäť platených a/alebo uzavretých)

Z uvedeného výpisu vyplýva, že toto riešenie je pre naše nasadenie nevyhovujúce, hlavne keď si uvedomíme podmienky, do akých má byť naša aplikácia nasadená – počítače v učebni (učebniach), na ktorých doteraz bežal dobre nakonfigurovaný OS Linux, v ktorom sa aj napriek postaršiemu technickému vybaveniu dalo bezproblémovo pracovať. Ďalšia stránka veci sú aj financie, je potrebné zakúpiť licenciu na web server, SQL server, vývojové postredie, na každý počítač by bolo treba kúpiť samotný operačný systém atď.

GNU/Linux

Slobodný operačný systém vyvíjaný od roku 1991 dobrovoľníkmi ako aj firmami po celom svete. Zväčša nasadzovaný v serverovej oblasti pre jeho stabilitu, výkon a univerzálnosť (môže fungovať od mikrovlnky až po serverové polia o niekoľko tisíc počítačoch).

Samotný systém tvorí len jadro (kernel), ku ktorému sa pridávajú ďalšie aplikácie, spravidla založené na podobnej filozofii ako samotný kernel (univerzálnosť a slobodné podmienky na šírenie/modifikovanie).

Vzhľadom na spomínané príjemné vlastnosti veľmi rýchlo vznikla iniciatíva ponúkať spolu s kernelom už aj hotové riešenia na správu databáz či prevádzku web servera. Najčastejšie sa môžeme stretnúť s riešením *LAMP* - systém založený na Linuxe, web serveri Apache, databáza MySQL a skriptovacom jazyku PHP či Perl (viď inú kapitolu tohto dokumentu pre bližší popis). Samozrejme, Apache či MySQL sa dá prevádzkovať aj pod OS Windows, ale tu už nemusí byť inštalácia ako

aj samotná prevádzka až taká bezproblémová. Ako sa môžeme dočítať na inom mieste tohto dokumentu, MySQL nie je jediné možné databázové riešenie – existuje minimálne ešte jedno, PostgreSQL, ktoré sa vlastnosťami dokáže MySQL plne vyrovnať resp. ho aj vo všeličom prekonať. Samozrejmosťou je bezproblémový beh na GNU/Linux.

FreeBSD

V podstate by sa o tomto systéme dalo povedať to isté, čo o GNU/Linuxe (podobná licencia na šírenie/modifikovanie, univerzálnosť, stabilita, aplikácie). Je tu spomenutý hlavne preto, lebo hlavný server na distribuované výpočty beží práve na tomto operačnom systéme.

Výber vhodného prostredia

Ako bolo spomenuté, OS Windows je pre naše účely a možnosti značne nevyhovujúci. Za zváženie stoja teda dve možnosti: GNU/Linux a FreeBSD.

Za Linux by hovorila skúsenosť členov tímu s týmto systémom ako aj množstvo distribúcii, špecializujúcich sa na riešenia typu *LAMP*.

Za FreeBSD by hovoril už spomínaný fakt, že na hlavnom serveri už FreeBSD bezproblémovo beží a nie je dôvod to meniť.

2.4.2 Proces kompilácie a binárna kompatibilita

Druhým aspektom, ktorý treba pri analýze zobrať do úvahy, je heterogénnosť systémov, ktoré sa pri distribuovanom výpočte môžu vyskytnúť. Zadanie síce explicitne nevyžaduje podporu nehomogénneho prostredia, ale po diskusii v tíme, ako aj s našim vedúcim sme dospeli k záveru, že takáto podpora by bola veľmi užitočná vec.

O akých systémoch hovoríme? OS Windows sme (aj) pre jeho uzavretosť a nekompatiblitu vylúčili, ale stále je tu veľa unix-like systémov, ktoré majú veľa spoločných vlastností, ale samozrejme každý z nich aj nejaké tie rozdiely od tých ostatných. Reč je o GNU/Linux, Open/Net/FreeBSD, Solaris, QNX, ...

Tieto systémy majú spoločné základy v Unixe, takže do istej miery sú medzi sebou (zdrojovo, nie strojovo) kompatibilné. To znamená, že ak nepoužívame špecifické vlastnosti toho-ktorého systému, tak by teoreticky nemal byť problém jeden a ten istý zdrojový kód preložiť a spustiť na každom z týchto systémov. Problémy sa môžu vyskytnúť, ak napríklad na danom systéme neexistuje nami zadefinvaný kompilátor, ale iba nejaký jeho ekvivalent (môže používať iné prepínače či parametre ako argumenty pri spúšťaní), ak sa niektoré deklarácie (v prípade jazyka C napríklad) nachádzajú na iných miestach (v iných hlavičkových súboroch, v iných adresároch a pod) alebo daný systém nepodporuje nejakú vlastnosť, ktorú požadujeme (vlákna, zdieľané knižnice, ...). Istým riešením sú nastroje autotools, automake a autoconf ktoré umožňujú tvorbu skriptov na zisťovanie tých „správnych“ ciest a súborov.

Druhá vec je samotný hardvér, na ktorom daný systém beží – v dnešnej dobe sú vo veľkej prevahe počítače s procesorom x86 kompatibilným, ale nie je problém sa stretnúť (hlavne v akademickom prostredí) aj s inými procesormi – napr. Alpha či procesory od Sun Microsystems. Najväčší problém (okrem rozdielnej inštrukčnej sady) predstavuje rozdielne kódovanie bajtov v pamäti – tzv. “big endian” a “little endian” kódovanie, ktoré sú navzájom nekompatibilné. Z toho dôvodu, ak by naša

aplikácia umožňovala pracovať aj s binárnymi vstupmi/výstupmi, treba myslieť aj na tento fakt. Tu treba spomenúť veľmi príjemnú vlastnosť aplikácie *PVM*, ktorá tieto problémy rieši za nás (teda na úrovni komunikácie a predávaní dát, o zvyšok sa samozrejme musí postarať buď používateľ alebo naša aplikácia).

2.5 Podobné riešenia

Existuje mnoho riešení pre podporu distribuovaných výpočtov. Navzájom sa odlišujú oblasťami a možnosťami svojho uplatnenia. Pomocou systémov na distribuované výpočty môžeme lámať šifry, predpovedať klimatologické zmeny alebo hľadať prvočísla, pričom existuje ešte mnoho iných oblastí uplatnenia.

2.5.1 BOINC

Azda najznámejší systém pre distribuované výpočty je `BOINC`[8] `- Berkeley Open Infrastructure for Network Computing`, ktorý sa do povedomia verejnosti dostal hlavne vďaka projektu *Seti@home*[9].

Systém BOINC vznikol v roku 2003 na University of California, Berkeley pod vedením Davida Andersona, ako softvérová platforma pre distribuované výpočty s použitím dobrovoľníckych výpočtových prostriedkov. Samotný projekt pod systémom BOINC pozostáva zo servera, ktorý prijíma a rozdeľuje dáta na spracovanie a klienta (čo môžu byť rôzne aplikácie). Klient príjíma dáta na spracovanie, poprípade odosiela spracované dáta na server. Server pracuje na Linuxovej platforme a používa Apache, PHP, a MySQL. BOINC poskytuje vlastnosti ktoré zjednodušujú vytvorenie a prevádzku projektov na distribuované výpočty[19]:

- Pružný aplikačný "framework" - aplikácie v bežných jazykoch ako C, C++, alebo Fortran môžu bežať vrámci systému BOINC s minimálnymi modifikáciami. Aplikácie môžu obsahovať viacero súborov. Nové verzie aplikácii sa môžu inštalovať automaticky.

- Bezpečnosť - BOINC je chránený proti niektorým typom útokov, napr. používa digitálne podpisy na autentifikáciu údajov.

- Použitie viacerých serverov - projekty môžu mať oddelené plánovacie a dátové servery, pričom z každého typu ich môže byť niekoľko.

- Otvorený zdrojový kód - BOINC je šírený pod licenciou Lesser GNU, ale samotné aplikácie nemusia byť *open source*.

- Podpora veľkých údajov - BOINC umožňuje prácu aplikáciam, ktoré vytvárajú alebo spotrebúvajú veľké množstvo údajov, poprípade potrebujú veľa operačnej pamäte. Rozdeľovanie a zber dát môže byť rozdelený na niekoľko serverov.

[8] http://boinc.berkeley.edu
[9] http://setiathome.berkeley.edu/

BOINC podporuje na strane klienta niekoľko platforiem (Mac OS X, Windows, GNU/Linux, Unix). Používatelia môžu pre svoje účty používať webové rozhranie. Klient má rôzne možnosti nastavenia, napr. frekvencia odovzdávania výsledkov, atď.

BOINC poskytuje dva možné spôsoby použitia ako realizovať výpočty. Prvý, asi najznámejší, sú dobrovoľnícke výpočty (`volunteer computing`)[17]. Pri tomto spôsobe existuje dohoda medzi dobrovoľníkmi, ktorý poskytujú svoje výpočtové prostriedky pre projekt, ktorý ich používa pre distribuované výpočty. Dobrovoľníci sú zvyčajne jednotlivci, ktorý majú počítač s Internetovým pripojením. Organizácie ako školy alebo firmy tiež často poskytujú svoje prostriedky. Projekty sú zvyčajne akademické (z univerzitného prostredia) a venujú sa vedeckým výpočtom. Dobrovoľníci sú anonymní (aj keď sú potrebné niektoré registračné údaje ako napr. email). Pre svoju anonymitu nie sú dobrovoľníci zodpovední za projekt, teda ak sa niektorý dobrovoľník "nespráva slušne", tak ho vrámci projektu nemožno potrestať. Dobrovoľníci musia dôverovať projektu v niektorých oblastiach:

- Dobrovoľníci dôverujú projektu, že aplikácie, ktoré poskytuje, nepoškodia ich počítač, resp. nenarušia ich súkromie.
- Dobrovoľníci dôverujú projektu, že ich pravdivo informuje o tom, čo jeho aplikácie spracovávajú a ako sa naloží s ich výsledkami.
- Dobrovoľníci dôverujú projektu, že vykonal všetky potrebné bezpečnostné opatrenia, aby prípadný útočník nezneužil projekt na svoje záškodnícke aktivity.

Hoci bol BOINC pôvodne vyvíjaný pre dobrovoľnícke výpočty, môže byť využívaný aj ako `desktop grid computing`[18]. Je to forma distribuovaných výpočtov, kde organizácia používa svoje existujúce osobné počítače na výpočet náročných úloh. Existuje niekoľko odlišností od výpočtov vytváraných na dobrovoľníckej báze.

- Výpočtovým prostriedkom možno dôverovať, teda sa nepredpokladá, že by výsledky boli nesprávne, či už zámerne alebo hardverovou chybou. Preto nie je dôležitá redutantnosť výpočtov.
- Nie je potreba "šetričov obrazoviek", pretože je možné aby výpočty bežali úplne neviditeľne a mimo kontrolu používateľa osobného počítača.
- Inštalácia klientov je zvyčajne automatická.

2.5.2 CONDOR

Systém CONDOR[10] umožňuje spravovať vyhradené výpočtové prostriedky, a tiež umožňuje využiť výpočtový výkon práve nevyužívaných pracovných staníc (cycle scavenging). CONDOR vznikol v roku 1988 pod vedením profesora Miron Livny na University of Wisconsin-Madison na základe predchádzajúcich projektov z oblasti distribuovaných výpočtov[20]. Je šírený pod *open souce* licenciu a podporuje mnoho

[10] http://www.cs.wisc.edu/condor/

architektúr (MacOSX, AIX, Tru64, UNIX, Irix, ,GNU/Linux, Windows).

CONDOR možno zaradiť do kategórie tzv. `High-Throughput Computing (HTC)`. Jedná sa o systémy poskytujúce dlhodobo veľký výpočtový výkon. Naproti tomu systémy z kategórie `High-Performance Computing (HPC)` poskytujú obrovské množstvo výpočtovej sily v malom časovom úseku. Výkon HPC systémov sa zvykne merať pomocou pohyblivej radovej čiarky za sekundu (FLOPS). Naproti tomu pre problémy riešené pomocou HTC sa hovorí o výpočtoch nie v sekundách, ale rádovo v mesiacoch alebo rokoch[21].

Myšlienka CODOR-a je jednoduchá. Používatelia nemusia modifikovať svoje programy, aby mohli používať CONDOR. Musia iba súhlasiť s tým, že budú patriť do siete CONDOR-u, kde si počítače navzájom medzi sebou vymieňajú správy. CONDOR striehne na pozadí siete, zahniezdený v počítači, ktorý sa nazýva *central manager*, ktorým sa môže stať ktorýkoľvek počítač v sieti, a na ktorom sa bude sledovať aktivita počítačov. Keď zistí, že niektorý z počítačov nie je dostatočne vyťažený, tak ho uchmatne a spustí na ňom úlohu. Keď sa používateľ rozhodne opäť použiť svoj počítač, tak CONDOR presunie úlohu niekam inam[22]. CONDOR poskytuje tradičnú funkcionalitu pre usporiadanie úloh do radov a plánovanie úloh, spolu s mnohými novými prvkami ako je napr. klasifikácia zdrojov. Systém CONDOR má niekoľko typických vlastností[23].

Vytváranie kontrolných bodov je proces, pri ktorom ukladáme stav programu tak, aby program mohol byť reštartovaný z tohoto stavu niekedy v neskoršom čase, typicky na inom mieste. Periodické vytváranie kontrolných bodov je hlavne vhodné pri dlho bežiacich úlohách, pričom v prípade poruchy stačí spustiť výpočet s posledného kontrolného bodu. Kontrolné body pomáhajú vytvárať určitý komfort pre používateľa, na ktorého počítači práve prebieha cycle scavenging. Akonáhle CONDOR zaregistruje, že používateľ je aktívny na počítači, okamžite ukončí výpočet, a daná úloha môže pokračovať od posledného kontrolného bodu na imom miste a v inom čase. Takýto používateľ nemusí potom čakať, kým systém dokončí výpočet a odošle výsledky, ale takmer okamžite (nejaká komunikácia tam medzi počítačom a central managerom samozrejme ešte musí prebehnúť) môže počítač používať, bez toho aby bol vlastne CONDOR-om obmedzovaný.

Migrácia úloh je umožnená vytváraním kontrolných bodov, vďaka čomu môže plánovač migrovať úlohy do iných zdrojov bez straty výsledkov z práve prebiehajúceho výpočtu.

Klasifikácia zdrojov *ClassAds*, umožňuje oznamovať voľné systémové prostriedky a dovoľuje úlohám sa o ne uchádzať, ak ich práve potrebujú.

Prostredie, v ktorom bude úloha spúšťaná, musí byť známe už v čase jej zaslania. CONDOR nazýva bežiace prostredie ako *universe*. Existuje podpora pre nasledovné prostredia:

- Štandartné - podporuje vytváranie kontrolných bodov, migráciu úloh, ale má niektoré obmedzenia, úloha musí byť linkovaná s knižnicou CONDOR-u.
- Vanilla - podporuje niekoľko služieb s malými obmedzeniami.
- PVM - programy napísané pre Parallel Virtual Machine.
- MPI - programy pre MPICH.

- Globus - pre zadávanie úloh pre Globus.
- Java - pre programy napísané pre Java Virtual Machine.

2.5.3 Grid MP

`Grid MP`[11] je komerčný produkt spoločnosti United Devices, ktorý ponúka riešenie pre vytvorenie rozsiahlej mnoho zdrojovej dynamickej virtuálnej infraštruktúry. Grid MP má niekoľko typických vlastností[27]

- "Provisioning". Rozdelenie zdrojov a používateľov do skupín, nastavenie privilégii.
- Virtualizácia. Jednoduché užívateľské rozhranie umožňuje prístup k celej infrštruktúre.
- Bezpečnosť. Má zabudovaných niekoľko bezpečtnostných opatrení ako šifrovanie dát prenášaných v infraštruktúre systému. Autentifikácia, aby iba oprávnený užívatelia mali prístup do systému. Digitálne podpisy aplikácií zabezpečujú, aby iba autorizované aplikácie mohli byť registrované a používané v systéme.
- Aplikačný "framework" pre rýchly vývoj programov.
- Optimalizácia rozloženia záťaže.
- Škálovateľnosť, manažovateľnosť a ochrana proti poruchám.
- Podpora mnohých platforiem - GNU/Linux (SuSE, RedHat, Fedora, Mandrake, Debian), Windows (98/ME,NT 4.0 (SP6 and up), 2000, 2003 CCE, XP, all on x86), Mac OS X v.10.2 10.3, AIX 4.3 5L 32-bit and 64-bit on IBM RS/6000, Solaris 9 (32-bit and 64-bit) on Sun SPARC).

Grid MP architektúra[28] pozostáva zo skupiny serverov, ktoré poskytujú služby *gridu* pre administrátorov, vývojárov a používateľov a súboru výpočtových prostriedkov pozostávajúci z osobných počítačov, pracovných staníc, "high-end" serverov a "clusterov". Grid MP servery poskytujú služby pre manažovanie zdrojov, ako výpočtových zariadení, údajov, aplikácií a záťaže používateľov. Prístup k týmto službám je možný pomocou webového rozhrania (*MP Grid Services Interface MGSI*). Servery majú nasledovné úlohy.

- Autentifikácia používateľov a zariadení.
- Manažment údajov.
- Manažment úloh a zaťaženia.
- Plánovanie úloh pre volné zariadenia.

[11] http://www.ud.com/products/gridmp.php

Na každom zdroji v Grid MP systéme beží odľahčený MP agent, ktorý poskytuje kontrolovaný prístup k zariadeniu. Agent poskytuje prostredie, kde može bežať aplikácia prijatá od servera, v nevtieravom (neobmedzujúcom pre používateľa daného počítača), bezpečnom *sandbox* prostredí. Grid MP podporuje GNU/Linux, AIX, Windows, Mac, and Solaris na strane klientov. Systém poskytuje niekoľko možností pre prístup a riadenie zdrojov. Okrem webového rozhrania je možné používať aj príkazový riadok. Zdroje môžu byť zoskupené do skupín, koré môžu byť manažovateľné administrátorom, ktorý môže vytvoriť pravidlá o ich používaní. Tieto pravidlá môžu obsahovať typicky nasledovné položky[28]:

- Časy, kedy môžu byť zariadenia používané.
- Koľko diskového miesta môže byť použitého.
- Priority pre skupiny používateľov systému pre využívanie zdrojov.

Grid MP podporuje tri typy úloh[12]:

- Dávkové úlohy.
- Úlohy MPI.
- Paralelné úlohy. Podporuje hrubozrnný paralelizmus pre veľké úlohy, ktoré môžu byť rozdelené na niekoľko nezávislých podúloh.

2.5.4 Sun Grid Compute Utility

`Sun Grid Compute Utility`[12] je komerčná služba spoločnosti SUN microsystems, ktorá ponúka prístup k rozsiahlym počítačovým zdrojom formou jednoduchého webového rozhrania. Cena za využívanie tejto služby je stanovená na 1$ za jeden CPU za jednu hodinu. Je to vhodné pre spoločnosti[24]:

- ktoré malú cyklické požiadavky na výpočtovo náročné úlohy(denne, týždenne, mesačne, sezónne),
- pre potreby výpočtových prostriedkov len na určitý čas, napríklad na špeciálne projekty,
- ak výpočtové úlohy môžu bežať v dávkach,
- ak sú vyčerpané všetky dostupné výpočtové prostriedky, ale stále pretrváva potreba po vyššom výkone.

Sun Grid Compute Utility je zatiaľ poskytovaný iba pre obyvateľov USA, ale počíta sa aj s medzinárodnou podporou. Pozostáva zo serverov, ktoré sú osadené "Dual AMD Opteron" procesormi, pričom na jeden CPU(jadro) pripadá 4 GB RAM. Servery pracujú pod operačným systémom Solaris.

Je podporované paralelné prostredie MPICH (otvorená implementácia *Message Passing Interface*). Binárne spustiteľné súbory môžu byť buď 32-bitové alebo 64-bitové aplikácie vytvorené pod Solaris 10 OS, x64 Platform Edition. Tiež je možné, aby aplikácie boli vytvorené pre Java Virtual Machine. Pre používateľa prebieha celý proces v piatich krokoch[26]:

[12] http://www.network.com/

- Príprava a balenie. Všetky aplikácie musia pracovať správne pod JVM alebo Solaris 10 OS pre x64 na architektúre x86/x64. Všetky aplikácie, skripty (ktoré sa používajú na inicializáciu výpočtu) a údaje budú uložené do pracovného adresára. Je preto potrebné presvedčiť sa, či sú všetky referencie relatívne k danému adresáru, resp. podadresárom, a tiež, či všetky operácie sú adresované relatívne k domácemu adresáru. Dalej je potrebné, aby sa aplikácie vykonali až do konca pod kontrolou Solaris OS. Ešte je potrebné, aby neboli použité žiadne závislosti na konkrétne "Solaris user ID" a práva musia byť nastavené tak, aby dovolili vykonávanie náhodne priradeným "Solaris user ID". Veľkosť všetkých súborov nemôže prekročiť veľkosť používateľského konta a to je 10GB.

- Nahranie súborov, aplikácii a údajov. Prebieha pomocou "browsera" cez webov0 rozhranie.

- Vytvorenie opisu úlohy. Definovanie dát, programov a vytvorenie inicializačného skriptu.

- Plánovanie a spustenie. Vyberie sa opis úlohy, ktorý sa bude vykonávať. Po spustení je možné kontrolovať stav v ktorom sa výpočet nachádza (čakanie v rade, príprava, vykonávanie, úspešné ukončenie, neúspešné ukončenie, zrušenie). Užívateľ môže o jednotlivých etapách dostávať upozornenia vo forme emailu.

- Stiahnutie výsledkov. Po skončení výpočtov sú výsledky sprístupnené v jednom súbore vo formáte zip, ktorý si používateľ môže stiahnuť cez webové rozhranie.

3 Špecifikácia požiadaviek

V tejto kapitole sa budeme venovať špecifikácii požiadaviek na navrhovaný systém. Konkrétne sa budeme venovať špecifikácii požiadaviek z rôznych uhlov pohľadu. Prvým z nich bude pohľad používateľa, ktorý je možné zameniť s pohľadom objednávateľa systému. Ďalším pohľadom bude pohľad na technické prostriedky a technológie, pomocou ktorých budeme budovať systém spravovania distribuovaných výpočtov. Na koniec špecifikujeme bezpečnostné a nefunkcionálne požiadavky na systém. Bezpečnostná otázka je nemálo dôležitá, pretože hrozba prieniku je dnes všadeprítomná.

3.1 Špecifikácia používateľských požiadaviek

Systém má slúžiť ako podpora pre spravovanie distriuovaných výpočtov. Z požiadaviek vyplynulo, že táto podpora sa bude realizovať ako informačný systém. Rozhraním pre prístup k informačnému systému bude webová aplikácia. Bude to jediný spôsob prístupu používateľov k systému distribuovaných výpočtov. Výnimku bude tvoriť možnosť prístupu administrátora ku ktorejkoľvek časti pomocou štandardných prístupových metód, ktoré sa pri danej časti systému bežne používajú.

Základné používateľské požiadavky sú na správu rôznych častí systému. Sú to časti:

- Používateľske účty:
  Správa používateľských účtov zahŕňa pridávanie, odoberanie účtov, zmenu hesla, editáciu ostatných hodnôt parametrov.

- Projekty:
  Možnosť pridávania, odoberania, spravovania celých projektov distribuovaných výpočtov.

- Súbory:
  Možnosť spravovať súbory, ktoré patria do projektu. Spravovanie bude zahŕnať kopírovanie (`upload`) zdrojových súborov do systému spravovania.

- Uzly:
  Používateľ bude môcť vybrať uzly, ktoré budú výpočet realizovať. Privilegovaný používateľ bude môcť pridávať, odoberať uzly zo systému distribuovaných výpočtov. Taktiež bude editovať hodnoty parametrov, ktoré charakterizujú uzol. Napr. typ operačného systému, frekvencia procesora, veľkosť operačnej pamäte, atď.

- Výpočty:
  Používateľ bude môcť spúšťať, ukončovať distribuované výpočty.

- Štatistika:
  O uzloch a používateľskej aktivite sa budú viesť záznamy, štatistika. Tie budú

slúžiť privilegovanejším používateľom na identifikáciu prípadných problémov a ako zdroj informácií o činnosti systému.

- Limity:
  Limity budú slúžiť na obmedzovanie používateľov pri prístupe k výpočtovým prostriedkom na uzloch.

- Zálohovanie:
  Zálohovanie sa vzťahuje na celý systém podpory spravovania distribuovaných výpočtov. Najdôležitejšou časťou však bude vytváranie zálohy databázy informačného systému.

Ďalšie používateľské požiadavky, ako aj väčšia miera ich podrobností, sú spracované v kapitole návrhu.

3.2 Technické požiadavky

V súvislosti so stanovením špecifikácie požiadaviek nemožno zabudnúť na požiadavky a predstavy realizátorov projektu. Z diskusie vyplynuli nasledujúce predstavy o systéme a z nich vyvplývajúce požiadavky.

Základom prostredia distribuovaných výpočtov bude lokálna počítačová sieť, ktorá bude spájať počítače s operačným systémom `GNU/Linux` (ďalej klienti, alebo uzly). V pomyselnom centre systému bude server s operačným systémom `FreeBSD` (ďalej server). Kvôli efektívnosti zdieľania prístupu k informáciam zo strany klientov bude na serveri nainštalovaný `Network File System` (ďalej `NFS`). Ten bude zabezpečovať ľahší prístup uzlov k zdieľaným údajom.

Ďalšou požiadavkou na systém je zavádzanie operačného systému `GNU/Linux` zo servera s operačným systémom `FreeBSD` (`booting from LAN - Local Area Network`).

3.3 Požiadavavky na bezpečnosť

Základnou požiadavkou na bezpečnosť v prípade systému s viacerými používateľmi je spôsob identifikácie a autentifikácie používateľov. V systéme na spravovanie distribuovaných výpočtov sa bude používateľ autentifikovať prihlasovaním pomocou webovej aplikácie a databázy.

Pri riešení problému synchronizácie používateľských účtov v prostredí webovej aplikácie s operačnými systémami uzlov sme zvažovali dve alternatívy:

- vytvorenie samostatných používateľských účtov na každom uzle,
  V tomto prípade je realizácia zložitejšia, ale umožňuje identifikovať bežiace procesy používateľa na základe jeho identifikačného používateľského čisla, alebo mena.

- vytvorenie jedného používateľského účtu.
  Pri vytvorení jedného používateľského účtu by tento účet prepožičiaval svoju

identitu všetkým používateľom systému spravovania distribuovaných výpočtov. To by neumožňovalo administrátorovi operačného systému ľahko identifikovať pôvodcu nekorektného stavu operačného systému. Implementácia by však bola jednoduchšia ako v prvom prípade.

Rozhodli sme sa pre vytvorenie samostatných používateľských účtov. Toto riešenie je na realizáciu náročnejšie ako druhá alternatíva. Bude potrebné vytvoriť skript, ktorý bude používateľov do operačných systémov klientov pridávať a odoberať.

Na uzloch budú mať teda používalia svoje kontá. Z ohľadom na bezpečnosť sa však nebudú môcť na uzly prihlásiť zo vzdialených počítačov, ani priamo z terminálov. Zabezpečenie bude tvoriť blokácia spustenia príkazového riadku(ďalej `shell`). Namiesto cesty k obvyklému `shellu` (`/bin/sh`, `/bin/bash`, atď), bude cesta zapísaná ako napr. `/bin/false`. Keďže sa používateľ nebude môcť prihlásiť, nebude potrebné synchronizovať jeho heslo medzi informačným systémom a operačným systémom na každom uzle. Bude potrebné synchronizovať iba existenciu konta.

Ďalej sme zvažovali použitie kvót a limitov pre používateľov systému. Je zrejmé, že použitie limitov a kvót je nevyhnutné, kvôli snahe udržať stavy všetkých uzlov v medziach normálneho stavu. Kvóty sa budú sťahovať na súborové systémy operačných systémov. Limity sa budú môcť nastaviť pre veľkosť pamäte, ktorú dostane proces k dispozícii, na čas strávený vykonávaním v procesore, atď.

Prirodzene na serveri bude potrebné aplikovať pravidlá `firewall`-u.

Komunikácia medzi serverom a uzlami, v zmysle spúšťania programov, bude prebiehať pomocou programu `secure shell - ssh`. Ten slúži na vytvorenie šifrovaného spojenia medzi dvoma uzlami. Okrem iného, umožňuje spúšťať na vzdialených uzloch programy, ktoré dostane ako vstupné hodnoty parametrov.

Klienti budú pristupovať k webovej aplikácii pomocou internetových prehliadačov. Pri tom budú využívať protokol - `HyperText Transfer Protocol HTTP`. Pre zvýšenie bezpečnosti bude počas autentifikácie komunikácia medzi webovou aplikáciou a prehliadačmi klientov šifrovaná. V tomto prípade pôjde o komunikáciu pomocou protokolu `HyperText Transfer Protocol Secure` (`HTTPS`).

3.4 Nefunkcionálne požiadavky

Zabezpečenie nášho systému by malo byť rozdelené do dvoch častí. Prvá časť zahŕňa zálohovanie dát. Zálohovanie sa bude vykonávať automaticky v určených intervaloch. Okrem toho bude možné vykonávať zálohy aj manuálne. Druhá časť bude zameraná na viacúrovňové používateľské práva s autentifikáciou pomocou mena a hesla.

4 Návrh

Táto kapitola obsahuje návrh všetkých podstatných častí systému na spravovanie distribuovaných výpočtov. Obsahuje predstavu o systéme a jeho komponentoch, návrh používateľských rolí, model prípadov použitia, návrh modelu údajov, návrh vnútorného manažéra, a diagram prípadov použitia.

Prestava o systéme ako takom vychádza z myšlienky vytvoriť rozhranie pre používateľa na pohodlnejšie spravovanie distribuovaných výpočtov. Naša snaha je navrhnúť a implementovať systém, pri ktorom bude mať používateľ prístup k systému iba pomocou webového rozhrania. Všetky nástroje, ktoré budú obsluhovať systém na distribuované výpočty, budú pre používateľa skryté.

Základom celého systému, z pohľadu fyzických prostriedkov, bude vytvorenie počítačovej siete. Uzly siete budú samozrejme tvoriť počítače. Jeden z nich bude server s operačným systémom `FreeBSD`. Ostatné budú mať operačný systém `GNU/Linux` a budú sa zavádzať zo servera.

Na serveri bude nainštalovaný démon databázy `postgreSQL`, podpora `PHP` a démon programu `Apache`. Okrem týchto programov bude na serveri nainštalovaný systém na zdieľanie diskového priestoru `Network File System - NFS`. Keďže predpokladáme, že server bude pripojený k Internetu, na serveri nastavíme `firewall`. Ten bude zprístupňovať klientom iba tie služby, ktoré budú nezbytne potrebné pre projekt spravovania.

Navrhovaný systém sme rozdelili na tri základné časti:

- správa používateľských účtov
- interaktívne webové rozhranie s používateľom
- riadenie a správa distribuovaných výpočtov (ďalej vnútorný manažér)

Použitie systému najlepšie ilustrujú interakcie medzi jednotlivými časťami systému:

- webová časť interaktívne reaguje na používateľské požiadavky,
- webová časť zasiela vstupy vnútornému manažérarovi (ten je určený pre `UNIX OS`),
- vnútorný manažér spúšťa používateľské programy na zadaných uzloch,
- vnútorný manažér zbiera výsledky a poskytuje ich webovej časti,

- webová časť poskytuje výsledky používateľovi,
- webová časť komunikuje s databázou.

Používatelia systému sa budú prihlasovať na webovej stránke. Ich používateľské mená a heslá budú trvalo uložené v databáze. Pre väčšiu prehľadnosť systému sme navrhli, aby každý používateľ mal vlastný adresár na disku servera. Ten sa bude zhodovať s jeho používateľským menom. Každý používateľ bude môct vytvárať vlastné projekty. Súčasťou projektu budú súbory, ktoré si bude môcť používateľ uložiť na serveri. Kompilácia a samotné spúštanie sa bude riadiť pomocou súboru `makefile`[13]. Používateľ si bude môcť zvoliť, na ktorých uzloch chce výpočet spustiť. Na podnet používateľa webová aplikácia zašle vnútornému manažérovi údaje o používateľovi, projekte, projektových súboroch, zoznam uzlov, limity pre používateľa. Vnútorný manažér spustí pomocou programu `secure shell` na prvom uzle v zozname program, ktorý sme nazvali `wrapper`. Ten pomocou volania funkcie `change root - chroot` zmení pre proces kompilácie a spustenia koreňový adresár operačného systému na adresár projektu. Ďalej nastaví požadované limity. `Wrapper` ďalej pomocou volania `fork` vytvorí nový proces. Ten bude spúšťať kompiláciu s jediným parametrom, názvom `makefile` súboru. To či sa bude realizovať samotný výpočet, bude záležať na obsahu súboru `makefile`.

4.1 Používatelia

Základnou požiadavkou na náš systém je jeho webové rozhranie. Z toho logicky vyplýva schopnosť systému obsluhovať viacerých používateľov naraz v rovnakom čase. Aby sa zabránilo neadekvátnemu použitiu systému musia zákonite existovať pre používateľov isté jasne definované hranice ich schopností v systéme. Táto podkapitola sa preto zaoberá definovaním používateľských rolí v systéme a ich funkciami.

1. Bežný používateľ,
   - možnosť prihlásenia a odhlásenia,
   - nahratie súborov do systému (vytvorenie projektu),
   - výber uzlov, na ktorých bude spustený distribuovaný výpočet,
   - manipulácia so spustiteľným súborom (spustenie, zastavenie, zadávanie vstupov, stiahnutie výstupov),
   - informácie o uzloch (vyťaženosť procesora, dostupná pamäť, voľná disková kapacita, počet bežiacich procesov na uzle).
2. „Supervízor“,
   Zmyslom tejto role je čiastočne nahradiť administrátora systému. Budeme predpokladať, že administrátor sa síce bude pravidelne pripájať, ale nemôžeme predpokladať, že bude vždy k dispozícii.

[13] `Makefile` je pomocný súbor pre kompilátor, ktorý slúži na pokročilé riadenie kompilácie a spúšťanie binárnych súborov.

- správa uzlov (pridávanie/odoberanie uzlov, obmedzovanie používateľov na skupinu uzlov, obmedzovanie vyťaženosti uzlov, diskovej kapacity, pamäťovej kapacity),

- podrobná štatistika o uzloch (vyťaženosť uzlov jednotlivými používateľmi, prezeranie používateľských súborov (projektov), dĺžka trvania distribuovaných výpočtov na uzloch),

- „supervízorove“ zmeny v systéme budú mať len dočasný charakter, t.j. budú sa môcť vrátiť do stavu definovaného správcom,

- vytváranie nových používateľov a editovanie hodnôt parametrov u existujúcich používateľov (napr. nastavovanie kvóty na použitý diskový priestor)

- správa používateľských účtov: možnosť pridávať nových používateľov, definovať im role, vymazávať existujúcich používateľov a modifikovať používatľské účty,

3. Administrátor,
   Táto rola je pre funkčnosť celého systému nevyhnutá. Bez nej by prakticky nebola možná správa celého systému. Hlavnou úlohou administrátorov je dohliadať na funkčnosť a bezchybný chod celého systému.

   - potvrdzovanie zmien, ktoré vykonal „supervízor“,

   - možnosť návratu k starším nastaveniam (systém si ich bude pamätať v databáze),

   - prezeranie histórie (zmien v systéme, aktivita študentov (ich súbory a použité vstupy)),

   - správa používateľských účtov: možnosť pridávať nových používateľov, definovať im role, vymazávať existujúcich používateľov a modifikovať používatľské účty,

   - správa výpočtových uzlov v systéme: možnosť pridať alebo vymazať výpočtové uzly, ktoré bude systém evidovať. Uzly, ktoré budú v systéme zaregistrované (bude o nich systém vedieť), budú môcť byť využité pri spúšťaní distribuovaných výpočtov,

   - monitorovanie stavu systému: možnosť sledovať reálnu alebo súhrnnú vyťaženosť jednotlivých uzlov systému, sledovanie štatistiky o spustených výpočtoch.

Návrh hlavných funkcií systému

Funkcia	*Opis*	*Vstupy*	*Vstupy*
prihlásenie do systému	používateľ zadá meno a heslo a systém ho autentifikuje a autorizuje	meno a heslo	potvrdenie prihlásenia
vytváranie / mazanie používateľských účtov	supervízor vytvorí / vymaže používateľa a zadefinuje jeho rolu	meno používateľa	nový účet / potvrdenie o zrušení účtu
nahratie zdrojového súboru	používateľ nahrá `makefile` a súbory na kompiláciu, ktoré sa následne používajú na distribuovaný výpočet	zdrojové súbory, `makefile`	skompilovaný súbor
priradenie úloh vnútornému manažérovi	nahratie vstupných dát alebo argumentov príkazového riadku skompilovaného programu	vstupy a argumenty	spustenie výpočtu
zber výsledkov	vnútorný manažér zo všetkých uzlov zozbiera výsledky výpočtu	výsledky výpočtu z každého uzla	výsledky výpočtu zo všetkých uzlov
odhlásenie zo systému	odhlásenie používateľa zo systému a vytvorenie záznamov o jeho aktivite	záznamy o činnosti v databáze	záznamy o činnosti v databáze
zaznamenávanie činností	vnútorný manažér monitoruje činnosť, vstupné a výstupné dáta, nahraté súbory, vyťaženie pamäti, procesora	I/O data, nahraté súbory, štatistiky uzlov	nahraté údaje v databáze

4.2 Model prípadov použitia

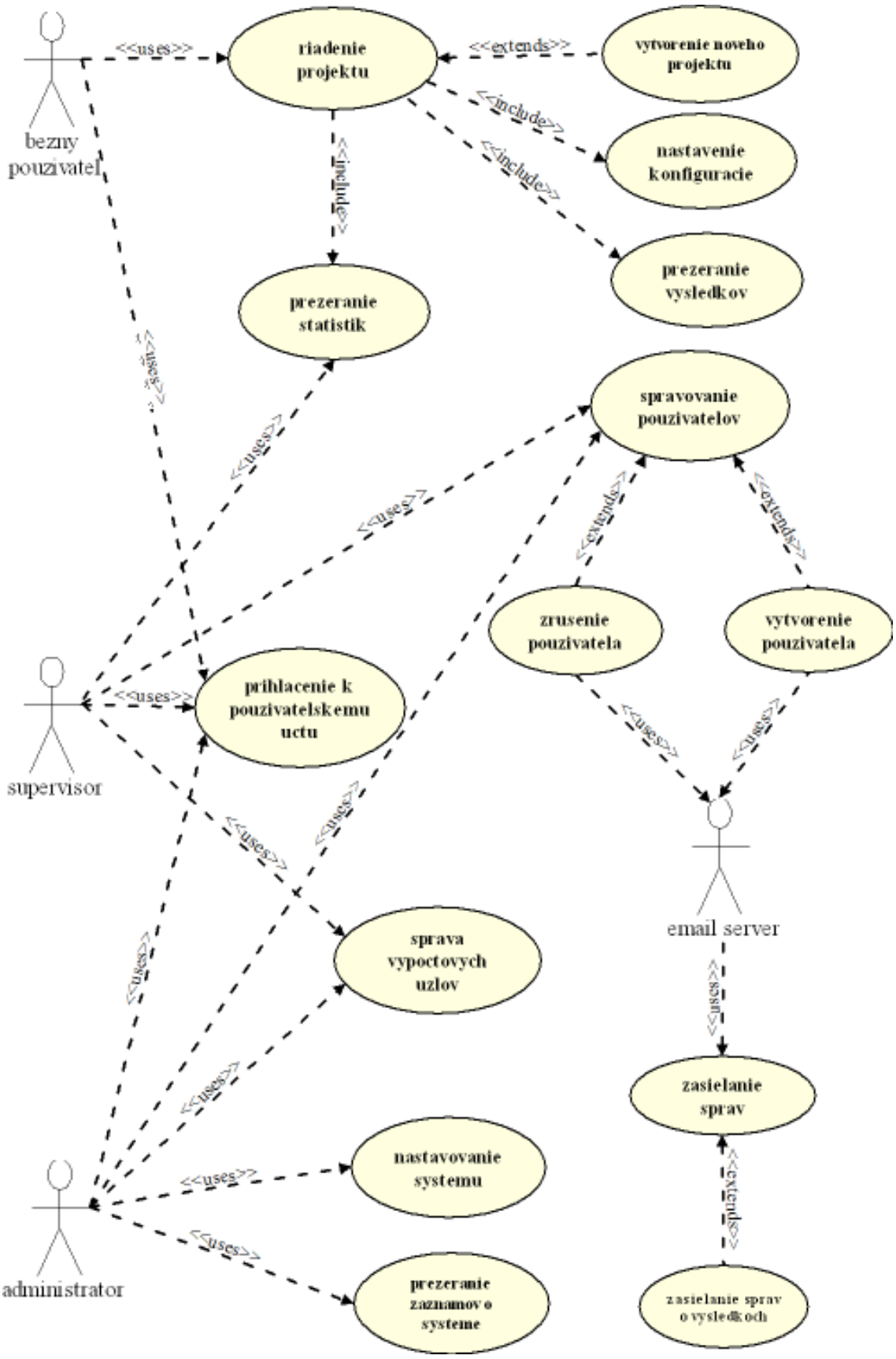
Táto podkapitola opisuje systém z pohľadu prípadov použitia. Snaží sa ukázať možnosti použitia systému z pozície tých, ktorý sa dostávajú do kontaktu so systémom.

4.2.1 Diagram prípadov použitia

Diagram prípadov použitia je znázornený na obrázku 2.

4.2.2 Hráči

Bežný používateľ: Je hráč, ktorý prichádza do kontaktu so systémom za účelom vykonania výpočtov a následného získania výsledkov.
Supervízor: Je hráč, ktorý vytvára a manažuje používateľov systému, a spravuje konfigurácie výpočtových uzlov, pre výpočty ktoré na nich vykonávajú bežný používatelia.
Administrátor: Je hráč, ktorý nastavuje parametre celého systému.
Emailový server: Je vonkajší systém, ktorý zasiela automatické správy používateľom systému.



Obr. 2: Diagram prípadov použitia

4.2.3 Prípady použitia

Prihlásenie k používateľskému účtu: Používateľ sa prihlási svojim menom a heslom, a systém mu následne pridelí práva, ktoré môže mať v systéme. V tomto prípade použitia je zahrnuté aj odhlásenie sa od systému.
Riadenie projektu: Bežný používateľ si môže nastavovať niektoré parametre svojho projektu, môže si vytvoriť nový projekt, môže si prezerať výsledky z výpočtov, môže si nahrávať súbory ktoré potrebuje v projekte, môže nastavovať konfiguráciu systému pre výpočet (napríklad výber uzlov na ktorých bude prebiehať výpočet), môže svoj výpočet spúšťať a zastavovať a získavať informácie o jeho priebehu.
Prezeranie výsledkov: Bežný používateľ si môže prezrieť výsledky na základe dokončeného výpočtu.
Vytvorenie nového projektu: Bežný používateľ si môže vytvoriť nový projekt. V systéme môže mať niekoľko nezávislých projektov.
Nastavenie konfigurácie: Bežný používateľ nastaví dostupnú konfiguráciu pre svoj výpočet. Napríklad počet výpočtových uzlov na ktorých sa bude výpočet vykonávať.
Prezeranie štatistík: Používateľ si môže prezerať pre neho dostupné informácie o systéme poprípade priebehu výpočtu.
Spravovanie používateľov: Administrátor alebo supervízor, môžu pridávať a rušiť používateľov, poprípade im dočasne zakázať prístup k používateľským účtom.
Vytvorenie používateľa: Administrátor alebo supervízor vytvorí nového používateľa.
Zrušenie používateľa: Administrátor alebo supervízor zruší existujúceho používateľa.
Správa výpočtových uzlov: Administrátor alebo supervízor môže nastavovať niektoré systémové konfigurácie, ktoré sa budú vzťahovať pre bežných používateľov. Napríklad disková kapacita, prístup k výpočtovým uzlom, volný čas pre spotrebovanie na výpočet, atď. Pričom nastavenia supervízora majú len dočasný charakter a musí ich potvrdzovať administrátor. Supervízor si môže vyberať prednastavené konfigurácie ktoré vytvoril administrátor.
Nastavovanie systému: Administrátor potvrdzuje a vytvára trvalé nastavenia celého systému.
Prezeranie záznamov o behu systému: Administrátor si môže prezerať informácie o celom systéme.
Zasielanie sprav: Emailový server, podla požiadaviek, rozosiela správy o udalostiach v systéme.
Zasielanie sprav o výsledkoch: Emailový server posiela bežnému používateľovi správu o stave jeho výpočtov.

4.3 Model údajov

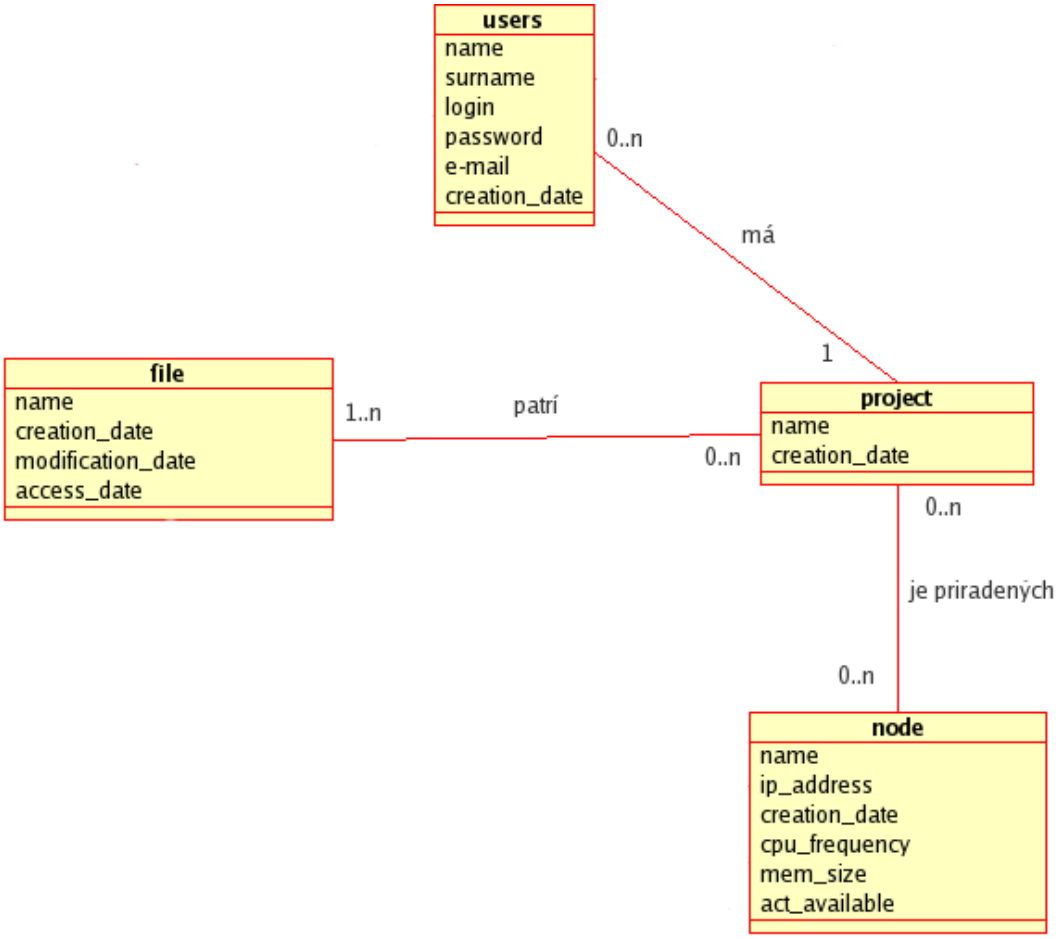
4.3.1 Logický model údajov

Pri vytváraní logického modelu údajov sme identifikovali nasledujúce entity:

- používatelia (`users`),
- projekty (`projects`),

- súbory (`files`),
- uzly (`nodes`).

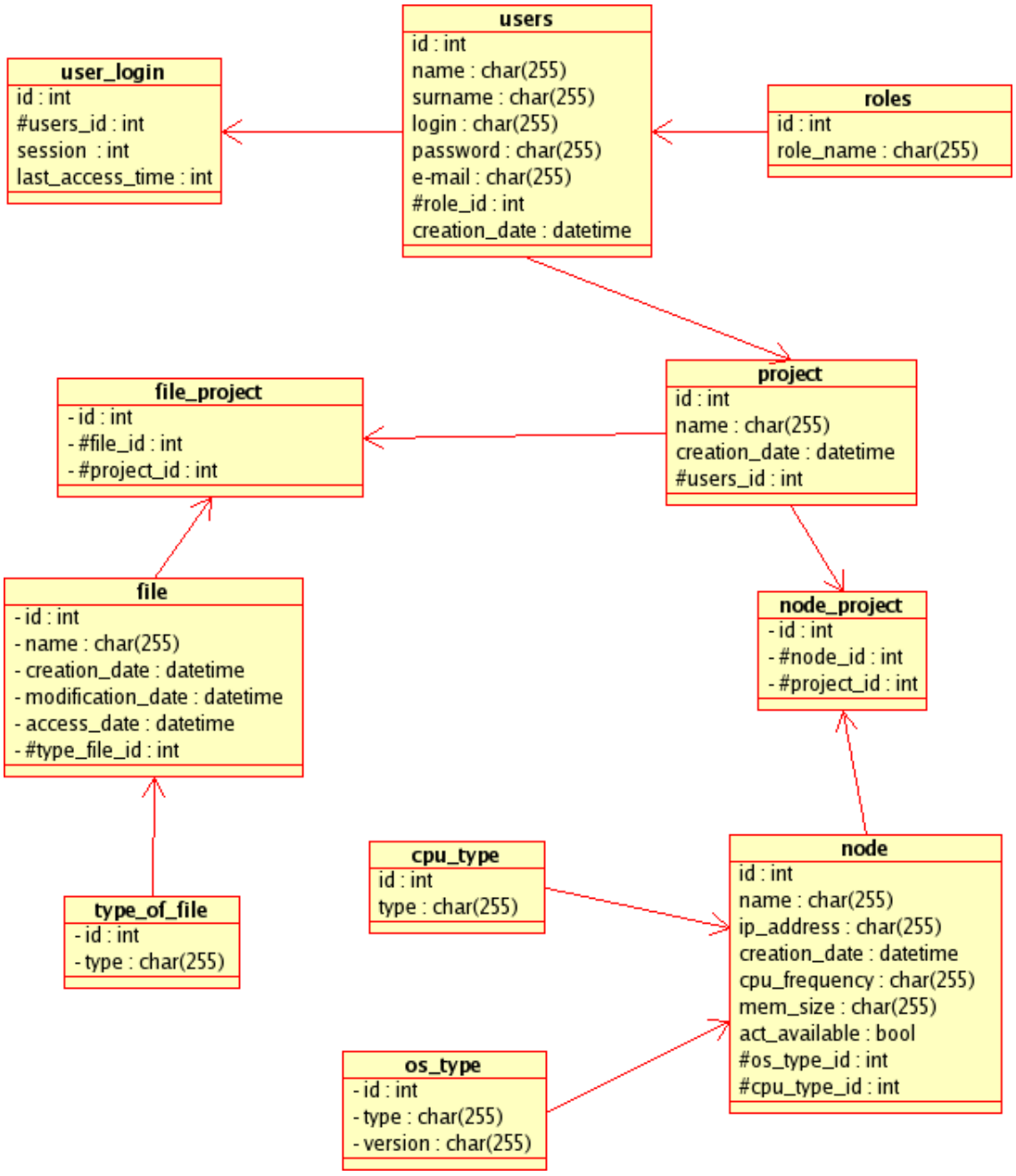
Ich vzťahy znázorňuje obrázok 3 – diagram vzťahov medzi entitami.



Obr. 3: Logický model

4.3.2 Fyzický model údajov

V logickom modeli údajov sme identifikovali základné entity systému. Kapitola návrhu fyzického modelu údajov sa zaoberá dekompozíciou základných entít, identifikáciou ich parametrov. V logickom modeli údajov existuje medzi entitami projekt - uzol a projekt - súbor vzťah s kardinalitou N ku N. Ten je potrebné eliminovať pomocou pomocných tabuliek. Na obrázku 4 je zobrazený fyzický model.



Obr. 4: Fyzický model

Pri dekompozícii entít a vytváraní zoznamu sme identifikovali ďalšie, vedľajšie entity.

Sú to entity:

- prihlásenie používateľa (`user_login`),
- typ súboru (`type_of_file`),
- typ operačného systému (`os_type`),
- typ procesora (`cpu_type`).

Elimináciou vzťahov s kardinalitou N ku N vznikli nasledujúce entity:

- súbor_projekt (`file_project`),
- uzol_projekt (`node_project`).

Parametre entít

1. používateľ (`user`),
   - meno (`name`),
   - priezvisko (`surname`),
   - prihlasovacie meno (`login`) - slúži na identifikáciu a autentifikáciu do systému,
   - heslo (`password`) - slúži na identifikáciu a autentifikáciu do systému,
   - adresa elektronickej pošty (`email`) - slúži ako kontakt na používateľa,
   - dátum a čas vytvorenia konta(`creation_date`) - údaj má informatívny charakter pre bežného používateľa aj pre administrátora.
2. projekt (`project`),
   - názov (`name`) - používateľ si zvolí názov projektu,
   - dátum a čas vytvorenia (`creation_date`) - údaj má informatívny charakter pre bežného používateľa (vlastníka projektu) aj pre administrátora.
3. súbor (`file`),
   - meno (`name`) - názov, meno súboru,
   - cesta k súboru (`path`) - cesta k súboru v súborovom systéme operačného systému,
   - dátum a čas vytvorenia (`creation_date`),
   - dátum a čas poslednej modifikácie (`modification_date`),
   - dátum a čas posledného prístupu (`access_date`).
4. uzol (`node`).

- názov (`name`) - jedinečný názov na ľahšiu identifikáciu uzla,
- dátum a čas pridania do systému(`creation_date`) - údaj má informatívny charakter pre administrátora,
- veľkosť operačnej pamäte (`mem_size`) - informuje používateľov o veľkosti operačnej pamäte,
- frekvencia procesora (`cpu_frequency`) - informuje používateľov o veľkosti operačnej pamäte,
- ip adresa (`ip_address`) - slúži na identifikáciu uzla v sieti,
- aktuálna dostupnosť (`act_available`) - informácia, či je uzol aktuálne dostupný.

5. typ používateľa (`role`),
   - názov skupiny (`role_name`) - slovný názov role - skupiny používateľov,
6. typ procesora (`cpu_type`),
   - typ procesora (`type`) - druh procesora (napríklad `i386`, `i486`, `ppc`, `alpha`, atď),
7. typ operačného systému (`os_type`),
   - typ operačného systému (`type`) - druh operačného systému (napríklad `FreeBSD`, `Linux`, atď),
   - verzia operačného systému (`version`) - verzia konkrétneho operačného systému (napríklad `FreeBSD 5.2Release`, atď),
8. typ súboru (`type_of_file`).
   - typ súboru (`type`) - (napríklad binárny, dátový - vstupný, dátový - výstupný, `makefile`, zdrojový, atď).

4.4 Vnútorný manažér

Vnútorný manažér (ďalej len manažér) predstavuje jadro celého navrhovaného systému. Ide vlastne o akúsi vnútornú logiku a činnosti, ktoré celý systém definujú, pretože má na starosti spravovanie a riadenie distribuovaných výpočtov.

Táto kapitola pojednáva o základných funkciách manažéra z používateľského hľadiska.

4.4.1 Riadenie kompilácie

Prvý krok, ktorý je potrebný na spustenie distribuovaného výpočtu je samotná kompilácia programu, ktorý predstavuje požadovanú výpočtovú úlohu. Úlohou manažéra je prijať zdrojové súbory spolu s pokynmi, ako ich skompilovať `makefile`. `Makefile` môže obsahovať prepínače či argumenty pre kompilátor (napr. prilinkovanie matematickej knižnice, knižnice na osbluhu vlákien a pod.).

Výsledkom operácie bude buď skompilovaný binárny program alebo výpis od kompilátora, ktorý sa ďalej predá webovému rozhraniu, aby sa používateľ dozvedel o príčinách zlyhania kompilácie.

4.4.2 Spravovanie uzlov

Úlohou manažéra je aj podrobné monitorovania každého uzla - ide o informácie o dostupnej pamäti, diskovej kapacite, vyťaženosti procesora a podobne. Tieto informácie sa použijú jednak ako štatistické podklady pre webové rozhranie, jednak aj ako podklady pre reštrikcie voči používateľom, ktorí príliš vyťažujú niektorý z uzlov.

Pod správu uzlov patrí aj zadefinovanie, ktoré uzly sa pre daný výpočet použijú. Táto informácia sa získa z webového rozhrania a manažér ju následne použije na vytvorenie (resp. overenie) prístupových práv k požadovaným uzlom (ako veľmi výhodný spôsob sa javí použitie technológie *secure shell*, kde sa do súboru uložia privátne kľúče pre uzly, kam chceme pristupovať a prihlásenie pomocou mena a hesla už nie je potrebné). Pri použití knižnice `PVM` sa v súbore zadefinujú uzly, ktoré sa majú pri výpočte použiť a používateľ, resp. manažér sa už o prideľovanie uzlov nemusí starať.

4.4.3 Spravovanie výpočtu

Ak už máme program skompilovaný a uzly pridelené, môžeme pristúpiť k samotnému výpočtu. Základ, okrem algoritmu pretransformovaného do programu, tvorí samozrejme jeho vstup. Bolo by dobré, keby manažér vedel pre každý uzol dodať iný vstup, aby to používateľ nemusel riešiť manuálne. Manažér teda pridelí dodané vstupy medzi uzly (samozrejme tu bude stále možnosť, že používateľ si otázku vstupov bude riešiť sám - napr. pri hľadaní prvočísiel je doslova žiadúce, aby toto koordinoval samotný program a nie manažér) buď tak, že bude spúšťat rôzne inštancie programov s rôznymi vstupmi cez `shell` alebo využitím funkcií knižnice `PVM`.

Program sa potom môže následne cez manažéra spustiť, zastaviť, spustiť s iným vstupom a pod.

4.4.4 Zbieranie výsledkov

Po skončení výpočtu je žiadúce, aby bol manažér schopný výsledky nejako zozbierať a poslať ďalej. Môže ísť o výpisy na obrazovku (ktoré si manažér presmeruje a zaznamená), či vygenerované súbory. Ďalej by nás mohli zaujímať štatististické údaje o behu programu, napr. koľko času strávil výpočtom, koľko času bežal dohromandy a pod.

4.4.5 Zaznamenávanie činnosti a ukladanie záznamov

Nemenej zaujímavou vlastnosťou manažéra je zaznamenávanie činnosti a možnosti replikácie vykonaného výpočtu na základe týchto záznamov.

Aby bolo možné takúto operáciu realizovať, je potrebné zaznamenať tieto údaje:

- zdrojové kódy programu,`makefile`

- vstupné dáta
- rozpis, na ktorých uzloch má byť výpočet vykonaný
- aké vstupné dáta prislúchali akým uzlom

Tieto údaje musia byť uložené do databázy, aby sa dali znovupoužiť kedykoľvek to používateľ uzná za vhodné.

4.4.6 Realizácia

Na to, aby mohol manažér poskytovať spomínané funkcie, musí byť schopný komunikovať s webovým rozhraním (získavanie zdrojových súborov, vstupných dát, rozloženie uzlov pre výpočet, posielanie výsledkov), s knižnicou (resp. démonom) `PVM` (spúšťanie a zastavovanie výpočtu, pridelenie uzlov na výpočet) ako aj so samotným operačným systémom (spúšťanie kompilátora, tvorba súborov a adresárov, komunikácia s uzlami, prihlasovanie sa na uzly, zisťovanie záťaže, voľnej pamäti a diskovej kapacity, „zabíjanie“ nežiadúcich procesov, získavanie výstupov).

Aby bola realizácia modulárnejšia, rozhodli sme sa navrhnúť niekoľko častí manažéra:

- Démon,
  Démon bude na serveri s právami administrátora. Jeho úlohou bude komunikovať s webovou aplikáciou a získavať od nej vstupy pomocou soketovej komunikácie. Démon bude akési jadro systému, pretože bude riadiť ďalšie časti manažéra.

- `Statistix`,
  Úlohou časti `Statistix` bude zbierať v pravidelných intervaloch hodnoty parametrov jednotlivých uzlov. Bude zbierať údaje o vyťaženosti uzlov a ukladať ich do databázy. Na podnet používateľa získa aktuálne informácie o vyťaženosti uzlov. `Statistix` bude bežať na serveri.

- `Wrapper`.
  `Wrapper` bude predĺženou rukou démona na uzloch. Ak démon bude chcieť urobiť na uzle nejakú operáciu, bude nevyhnutné, aby na uzle prebehli bezpečnostné opatrenia - spustí na na danom uzle `wrapper`. Ten ako prvé vykoná volanie `chroot`, ktoré zmení koreňový adresár pre proces, ktorý chce démon spustiť (kompilácia, alebo výpočet). Ďalším krokom bude aplikovanie limitujúcich nastavení. Procesu sa nastavia príslušné limity, ktoré zadefinoval administrátor systému, alebo supervízor.

Forma implementácie jednotlivých častí môže a nemusí byť rovnaká. Pre lepšiu údržbu je vhodnejšia rovnaká forma implementácie. Projekt však obsahuje veľa rôznych technológií. Obsahuje databázu, webovú aplikáciu, dva rôzne operačné systémy, atď. Komunikácia a vzájomný prístup rôznych častí si niekedy vyžaduje použitie vyššieho programovacieho jazyka (webová aplikácia - databáza). Pri komunikácii iných častí (`wrapper` - operačný systém) si bude vyžadovať použitie nižšieho programovacieho jazyka.

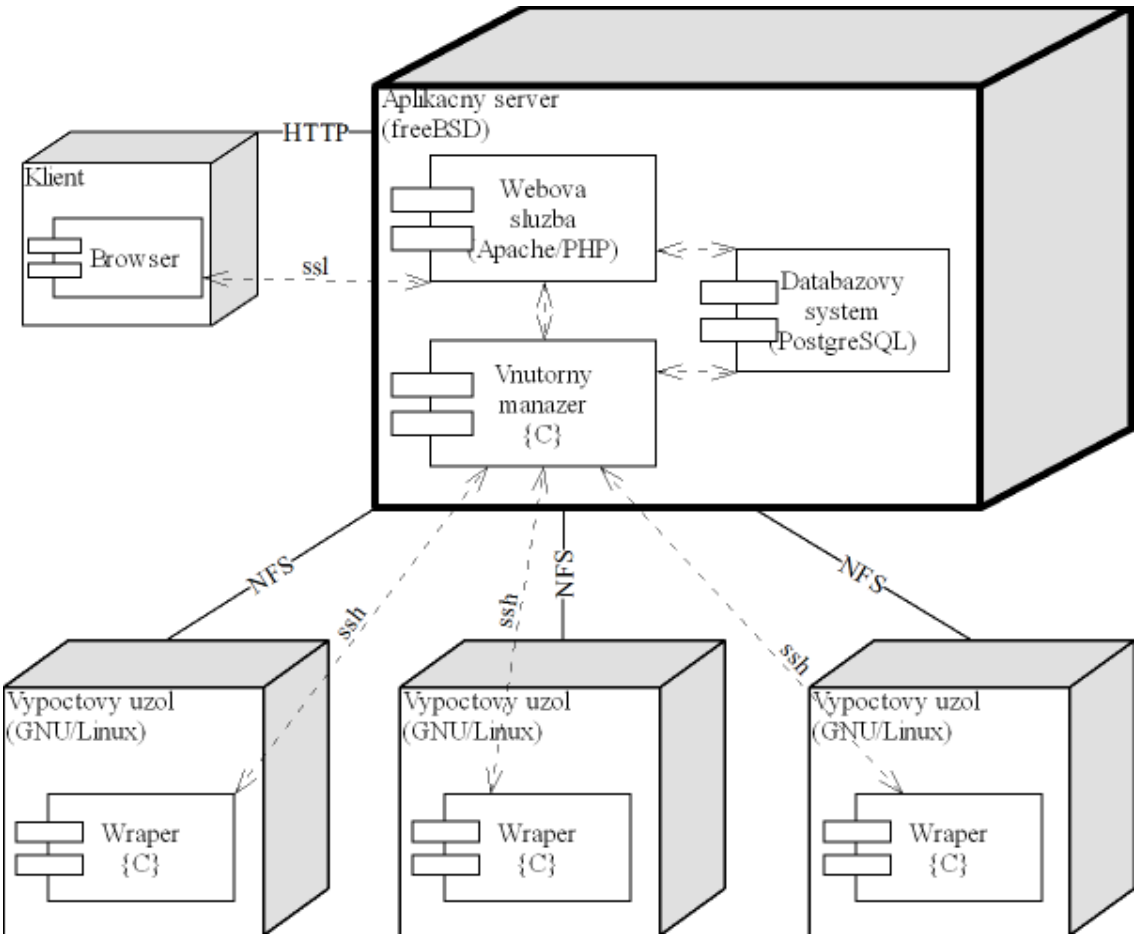
Previazanie častí s nástrojom `PVM` by nemal byť problém, pretože `PVM` sa dá efektívne ovládať pomocou príkazového riadku, takže to bude len vecou zavolania `PVM` démona so správnymi argumentami.

Na sledovanie stavu systému (a jeho vyťaženia) sa dajú s úspechom použiť existujúce programy, ktoré zisťujú všetky potrebné informácie o systéme aj jednotlivých procesoch (napr. `ps`, `time`, `top` a mnoho ďalších) a keďže ide opäť o programy pre príkazový riadok, spracovanie ich výstupu nepredstavuje väčší problém (pomocou nástrojov ako `sed` či `grep`). Ďalšou možnosťou je použitie stromovej štruktúry informácií o hodnotách parametrov operačného systému `MIB - Management Information Base`. Všetky dôležité hodnoty parametrov operačného systému sú uložené v hierarchii `MIB`. Pomocou programu `sysctl` je možné z príkazového riadku získať akúkoľvek hodnotu zo štruktúry `MIB`.

Zaznamenávanie činnosti sa dá spraviť tak, že okrem pracovných kópií vstupov od webového rozhrania si budeme vytvárať aj ďalšie, záložné kopie. Môže ísť napr. o adresárovú štruktúru pre každého používateľa s dátumom a časom prihlásenia, kde budeme mať zvlášť adresár na zdrojové kódy, vstupné dáta, rozpis pre uzly atď. Táto štruktúra sa pri odhlasovaní používateľa môže skomprimovať a uložiť do databázy.

4.5 Diagram rozmiestnenia

Táto podkapitola nám ukazuje aké je skutočné, fyzické rozmiestnenie hardvéru a jeho vzájomné prepojenie, ako aj jeho vzťahy so softvérom. Diagram rozmiestnenia je znázornený na obrázku 5.



Obr. 5: Diagram rozmiestnenia

5 Implementácia

V tejto kapitole je opísaná implementácia celého systému a rozdiely oproti pôvodnému návrhu.

Systém sa skladá z grafického webovského rozhrania, ktoré sme implementovali v jazyku PHP, z vnútorného manažéra (skriptovací jazyk bash) a wrappera (programovací jazyk C).

5.1 Grafické webovské rozhranie

V jazyku PHP sme využili objektovo orientované programovanie. Systém je rozdelený v ôsmich triedach. Všetky triedy sú podrobne zdokumentované na priloženom CD v adresári `php_dokumentacia/`.

5.1.1 Trieda Document

Základná trieda „Document“ slúži na vytváranie (generovanie) dokumentu. Sú tu definované štyri premenné: **Topmenu, Leftmenu, Maindata, Headrbox**, a do týchto štyroch premenných sa ukladajú časti HTML-kódu, podľa toho, kde chceme ktorý kód zobraziť. Na obrázku 6 je naznačené, ktorá premenná sa kde vypisuje. Zvyšný HTML-kód je definovaný priamo v metóde *PrintDocument()*. Kaskádové štýly sú definované v externom súbore. V triede sú ďalej definované metódy na nastavenie jednotlivých premenných. Napríklad na pridanie jednej položky do horného menu sa použije metóda *TopMenuAdd()*

5.1.2 Trieda Postgres

Táto trieda obsahuje dve metódy. Prvá metóda *Postgres()* nadviaže spojenie s databázou a otestuje či sa spojenie podarilo nadviazať. Ak nie, tak vypíše chybu a skončí. Druhá metóda je na testovanie chýb pri vykonávaní SQL dotazov. Volá sa TestError() a parametrami sa jej dá definovať či má pri chybe skončiť, alebo nie a akú správu má vypísať v prípade chyby.

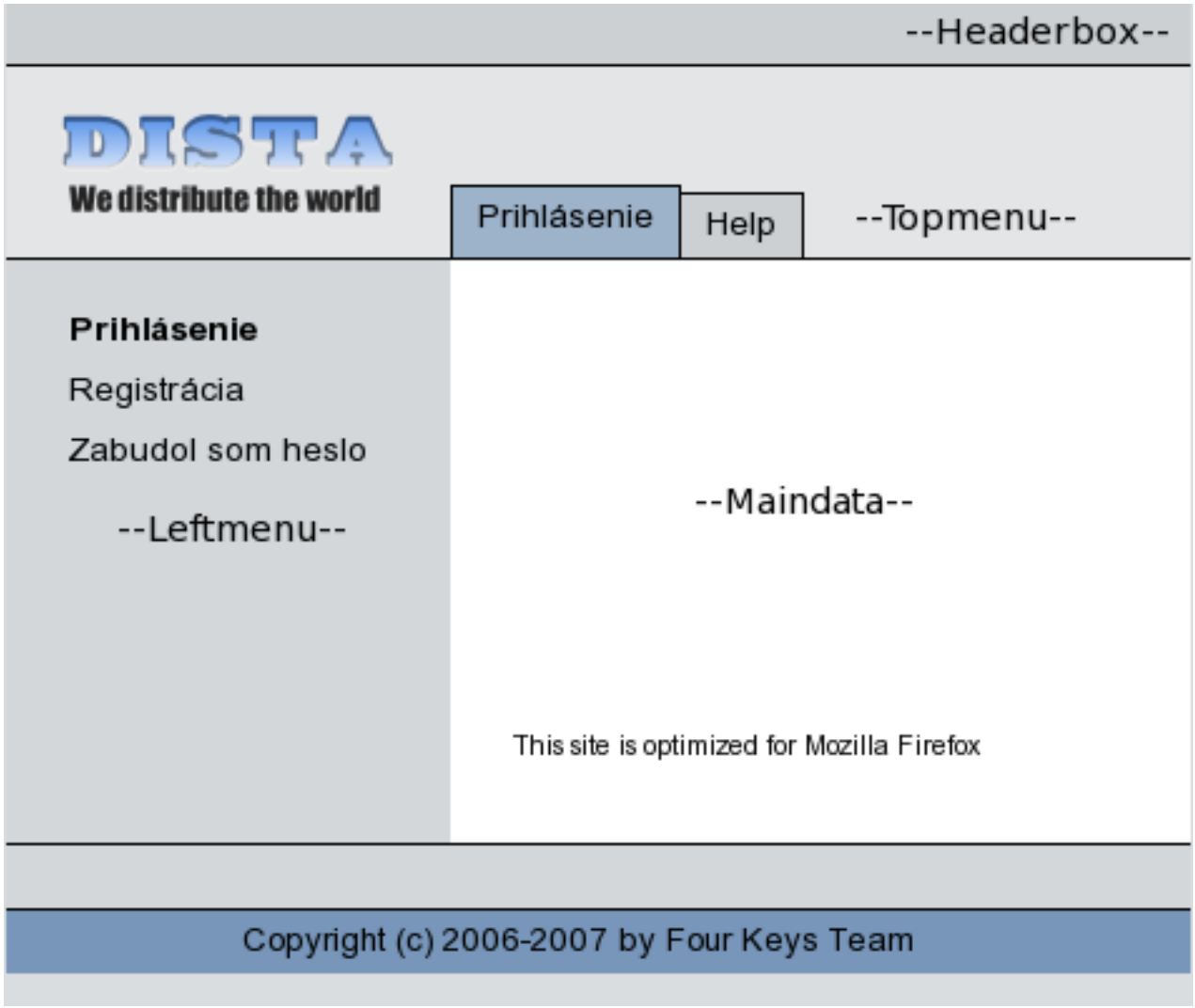
5.1.3 Trieda Misc

V triede Misc sú definované pomocné metódy (funkcie), ktoré sa „nehodili“ inde. Je tu opísaná metóda *GenPass()*, na vygenerovanie náhodného hesla a metóda na vypisovanie chyby pri vykonávaní externých príkazov.

Tieto tri triedy (*Document, Postgres a Misc*) sú implementované v súbore classes.php.

5.1.4 Trieda UserBasic

Ďalšou triedou je UserBasic. Táto trieda slúži na registráciu používateľov (metóda *Registration()*), na prihlásenie do systému (metóda *Login()*) a v prípade zabudnu-



Obr. 6: Generovanie dokumentu

tia hesla, je tu možnosť poslať si nové heslo na svoju e-mailovú adresu (metóda *ForgotPassword()*).

Metóda *Login()* urobí MD5 hash zo zadaného hesla a porovná ho s MD5 hashom konkrétneho používateľa v databáze. Pokiaľ sa hash-e nezhodujú, tak je v metóde nastavené časové zdržanie 5 sekúnd. Pri úspešnom prihlásení metóda nastaví **session**, pomocou ktorého sa neskôr kontroluje či je používateľ prihlásený. V prípade neúspešného prihlásenia sa z bezpečnostných dôvodov neudáva dôvod chyby.

Metóda *Registration()* zaregistruje používateľa do systému, t.j. vytvorí o ňom záznam v databáze, avšak používateľ je blokovaný do chvíle, keď ho správca systému povolí. Vtedy sa používateľ vytvorí aj v operačnom systéme. Táto časť bude podrobnejšie rozobratá v triede **User()**;

Metóda *ForgotPassword()* porovná či sa zhoduje e-mailová adresa používateľa, ktorú zadal pri registrácii, s tou ktorú zadal teraz (pri žiadosti o nové heslo) a v prípade zhody pošle používateľoví na jeho e-mail nové heslo. Pokiaľ si používateľ nepamätá svoju e-mailovú adresu, musí kontaktovať správcu, ktorý má oprávnenie

zmeniť mu heslo.

Táto trieda je implementovaná v súbore userbasic.php a používa sa v súbore login.php.

5.1.5 Trieda Form

Trieda **Form** slúži na vytváranie formulárov. Aby sme nemuseli pri každom formulári písať priamo HTML-kód, tak stačí zavolať iba konkrétnu metódu tejto triedy s parametrami a HTML-kód nám vygeneruje.

Začiatočný kód formulára sa vytvorí priamo v konštruktore triedy.

Metódy ktoré sa nachádzajú v tejto triede sú: *addText()* (pridá textové políčko - TextBox), *addPassword()* (pridá políčko na heslo) *addRadio()* (pridá RadioBox), *addCheckBox()* (pridá CheckBox), *addSelect()* (pridá selektovací formulár), *addFile()* (pridá formulár na vloženie súboru), *addHidden()* (pridá skrytú časť formuláru), *addSubmit()* (pridá tlačítko na odoslanie), *addCode()* (pridá ľubovolný text, alebo kód), *addNewLine()* (pridá nový riadok), *getCode()* (vráti celý vygenerovaný kód a ukončí formulár), *GetPartCode()* (vráti časť kódu od posledného zavolania tejto metódy, alebo od začiatku).

5.1.6 Súbor main.php

Po úspešnom prihlásení používateľa nasleduje presmerovanie na súbor main.php. V tomto súbore je definované používateľské menu. Triedy a ich metódy, ktoré sú tu volané, sú implementované v iných súboroch.

Na začiatku súboru sa načíta **session** a potom sa zavolá metóda *TestLogin()*[14].

Vytvoria sa inštancie tried a potom sa vypíše menu. Či sa má nachádzať v menu konkrétna položka sa testuje pomocou metódy *TestRights()*[15].

Po vygenerovaní používateľského menu, sa podľa parametrov poslaných metódou **GET** rozhodne ktorá metóda sa vykoná.

5.1.7 Trieda User

V tejto triede sú definované metódy na prácu s používateľskými kontami. Používateľské konto sa vytvára do databázy pomocou metódy *Registration* v triede **UserBasic**. V triede **User** existuje metóda na povolenie, respektíve znovu zablokovanie používateľského konta. Metóda sa nazýva *ChangeUserBlockedState()*. Pri prvom povolení používateľského konta sa zavolá vnútorný manažér, ktorý vytvorí dané konto aj v operačnom systéme.

Na vypísanie zoznamu používateľov slúži metóda *ListOfUsers()*. Používatelia sú vypísaný vo forme tabuľky. Pomocou tejto tabuľky sa dá meniť používateľom nastavenia ako aj aktivovať, zablokovať, alebo vymazať používateľa.

Na vymazanie používateľa slúži metóda *DeleteUser*. Táto metóda vymaže používateľa aj všetky jeho projekty, ktoré mal vytvorene. Nakoniec zavolá vnútorného manažéra.

[14]Metóda *TestLogin()* patrí do triedy Dista. Testuje či je používateľ prihlásený. Podrobnejšie vysvetlenie je v časti Trieda Dista.

[15]Metóda *TestRights()* patrí do triedy User, testuje či skupina do ktorej patrí konkrétny používateľ má oprávnenie na používanie konkrétnej funkcie systému.

Pomocou metódy *ChangeSettings()* si môže každý používateľ meniť svoje vlastné heslo a svoje osobné údaje. Správca systému môže pomocou tejto metódy meniť heslo a osobné údaje všetkým používateľom.

Metódy `TestRights() a SetRights()` slúžia na kontrolu a nastavovanie oprávnení systému. Pomocou metódy *SetRights()* je možné povoliť respektíve zakázať konkrétnej skupine určitú činnosť. Napríklad je možné povoliť skupine supervisors (privilegovaný používatelia) prácu s používateľskými kontami. Metóda *TestRights()* slúži ta testovanie oprávnení. Táto metóda sa spúšťa v každej inej metóde a otestuje či konkrétny používateľ má právo danú metódu zavolať. Metóda *TestRights()* hľadá oprávnenia v databáze, konkrétne v tabuľka *rights*.

Trieda **User** je implementovaná v súbore user.php.

5.1.8 Trieda Projekty

Táto trieda slúži na prácu s projektami. Na ich vytváranie, prezeranie, spúšťanie, mazanie, atd. Metóda *ListOfProjects()* slúži na vypísanie zoznamu projektov. Pomocou metódy *Projects()* sa zisťujú informácie o jednotlivých projektoch.

Pomocou metódy *ProjectsAdd* je možné vytvárať nové projekty. Metóda vytvorí adresár v adresári používateľa s názvom projektu a súčasne pridá meno projektu do databázy.

Metóda *ProjectDelete* Vymaže projekt z databázy a súčasne adresár projektu z disku a s jeho obsahom, t.j. so všetkými súbormi.

Metóda *Files()* je na prezeranie súborov, ktoré sa nachádzajú v projekte a súčasne je možné touto metódou súbory pridávať.

Metóda *ProjectConfigure* slúži na nastavenie spustenia projektu. Tu sa definuje či používateľ chce aby rozdelenie na uzli urobil automaticky systém, alebo si chce rozdeliť úlohy sám. Metóda potom zavolá ďalšiu metódu *RunWraper*, ktorá následne na to spustí pomocou secure shellu na každom definovanom uzle wrapper s definovanými parametrami.

V tejto triede sa ešte nachádzajú pomocné metódy na zistenie veľkosti projektu, na kompresiu súborov, a ďalšie.

5.1.9 Trieda Dista

Toto bola pôvodne hlavná trieda celého systému. Neskôr sa z nej presunuli metódy do tried **User a Projekty**, takže už tu ostalo iba niekoľko metód.

Základná metóda je *TestLogin()*. Táto metóda sa volá v súbore main.php a testuje či je používateľ prihlásený. Testuje hodnoty nastavene v premennej *SESSION* a súčasne testuje nečinnosť používateľa. Pokiaľ používateľ viac ako 30 minúť nepracoval v systéme dista, tak ho metóda automaticky odhlási.

Ďalšia metóda je *SyncUserDir()* a slúži na synchronizovanie údajov na disku s databázou. Táto metóda synchronizuje dáta konkrétneho používateľa. To znamená, že vyhľadá dáta na disku porovná či sú všetky v databáze. Ak nejaký projekt, alebo súbor je na disku, ale nie je v databáze, tak ho pridá do databázy. Následne na to skontroluje či všetky dáta v databáze sú uložené aj na disku. Pokiaľ nejaký súbor alebo adresár na disku chýba, tak vymaže informáciu o nom aj v databáze.

Posledná metóda v tejto triede je *SyncUsersGlobal()* a slúži na synchronizovanie používateľov v databáza s používateľmi v operačnom systéme. Táto metóda ešte nie je implementovaná.

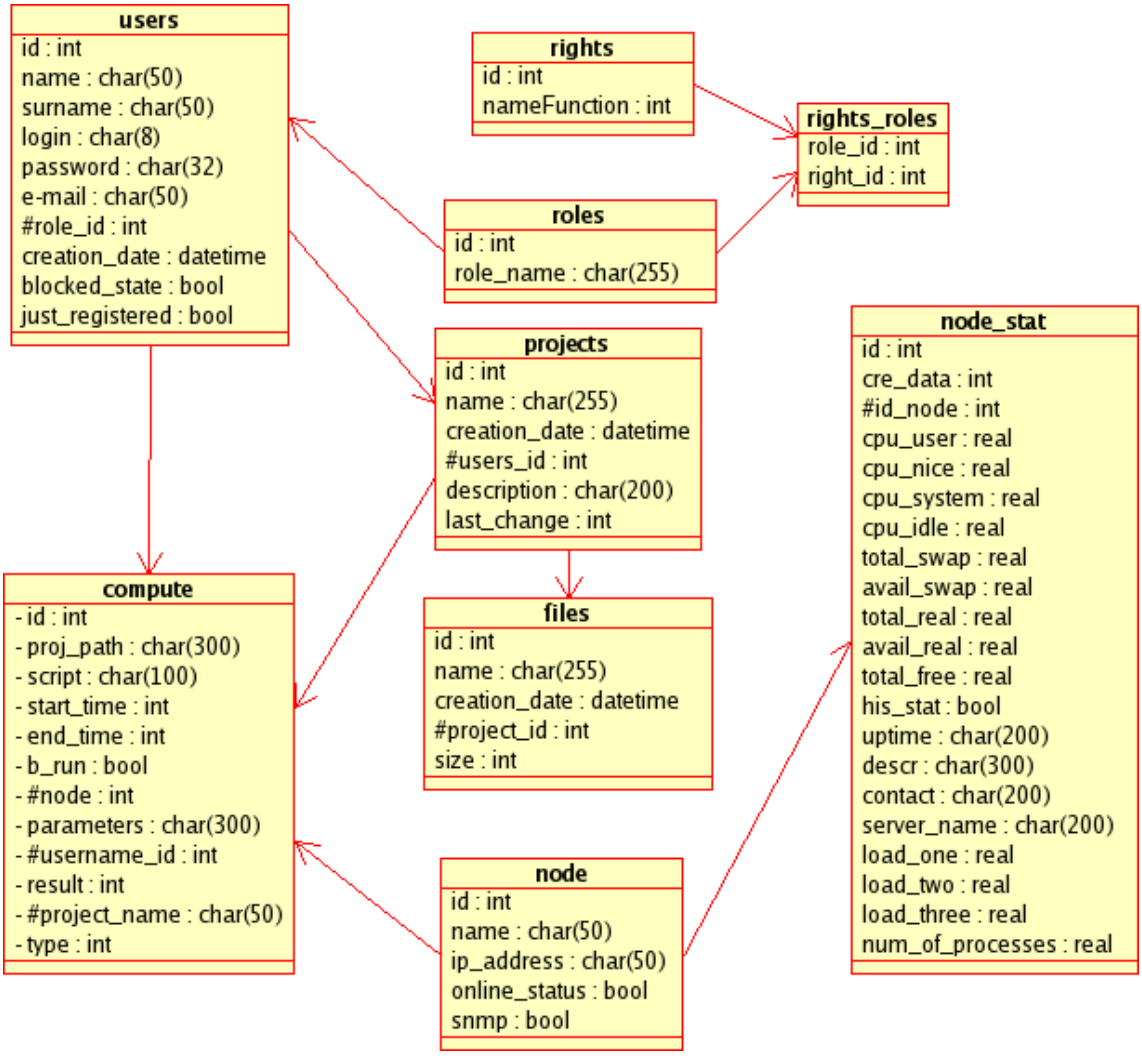
5.1.10 Súbory conf.php a lang.php

Konfiguračný súbor k webovskému rozhraniu je súbor conf.php. Nastavujú sa v ňom cesty pre domovské adresáre používateľov, prihlasovacie meno a heslo pre databázu.

Všetky texty celého webovského rozhrania sú v súbore lang.php. Takže je veľmi jednoduché zmeniť nejaký text, prípadne opraviť chybu, alebo preložiť celý systém do iného jazyka.

5.2 Databáza

Na obrázku 7 je finálny fyzický model databázy.



Obr. 7: Fyzicky model databázi

5.3 Monitorovanie uzlov

Na monitorovanie viacerých počítačov je ideálne použiť už existujúce riešenia. Jedným z osvedčených a v praxi veľmi často používaných riešení, je protokol **Simple**

Network Management Protocol - SNMP.

Náš systém monitoruje nasledujúce parametre:

- vyťaženie procesora,
- veľkosť využitia operačnej pamäti,
- počet procesov v operačnom systéme,
- veľkosť odkladacieho oddielu,
- vyťaženosť systému (**load**).

5.3.1 Uzly

Na každom uzle musí byť nainštalovaný démon **snmpd**. Na spustenie postačí štandardná konfigurácia. Dôležité je v konfiguračnom súbore povoliť sledovanie operačného systému z IP adresy servera. Právo na čítanie hodnôt postačuje.

5.3.2 Server

Na strane servera zabezpečuje sledovanie uzlov **PHP** skript. V prvej verzii bol tento skript implementovaný v interpretačnom jazyku **PERL**. Neskôr sme sa rozhodli pre jazyk php kvôli zachovaniu konzistencie vývoja celej aplikácie a kvôli prípadnej údržbe. Skript sa nachádza v adresári vebovej aplikácie a volá sa **snmp.php**. Na kontrolu, či skript monitoruje sme napísali malý program - *check_snmp.sh*. Nachádza sa v adresári implementácie interného manažéra (5.4).

5.3.3 snmp.php

Databáza systému obsahuje zoznam uzlov. Skript načíta tento zoznam z databázy a postupne ním prechádza. Po každom výbere uzla sa snaží od neho získať hodnoty parametrov, ktoré nás zaujímajú (vyťaženie cpu, atď.). V prípade, že uzol nevráti v požadovanom čase odpoveď, skript vyhodnotí, že daný uzol nemá podporu **SNMP**. Pre daný uzol nastaví v databáze príznak **snmp_status** na **false**. Vo vebovej aplikácii sa objaví pri danom uzle stav **offline**. Ak skript dostane požadované hodnoty, stav uzla bude **online** a do databázy sa zapíšu hodnoty parametrov, ktoré nás zaujímajú. Sledovanie vyťaženia procesora má dva režimy. V prvom sa sledujú hodnoty vyťaženia každú minútu. V druhom režime sa sleduje vyťaženie po dobu pol hodiny. Skript udržuje hodnoty z prvého režimu v databáze po dobu piatich minút. Potom staršie záznamy vymaže. Skript **snmp.php** sa spúšťa z príkazového riadku.

5.4 Interný manažér

Manažér je stupeň medzi operačným systémom (ďalej OS) servera, klientov a vebovým rozhraním systému na spravovanie výpočtov. Manažér sa nachádza na serveri a vykonáva činnosti vo vzťahu k OS, ktoré nemôže realizovať používateľ cez grafické rozhranie. Dalo by sa povedať, že je predĺženou rukou vebovej aplikácie, ktorá potrebuje robiť zásahy do OS servera a klientov.

Príkladom môže byť vytvorenie používateľa. Po registrácii nového používateľa a po jeho schválení správcom vebovej aplikácie, je potrebné vytvoriť používateľský účet na serveri, kde bude mať používateľ svoje súbory. Okrem toho je potrebné vytvoriť používateľa na všetkých uzloch. Čiže na všetkých OS klientských staníc. Tieto úlohy by vebová aplikácia zvládala iba veľmi ťažko. Vebová aplikácia nemôže tieto operácie vykonávať vzhľadom k právam, ktoré by sa jej museli priradiť.

Manažér realizuje operácie v operačnom systéme, ktoré si vyžadujú práva superpoužívateľa (root). Implementácia je v podobe skriptu v interpretačnom jazyku **BASH**.

Úlohy manažéra sú:

1. vracať identifikačné číslo, ktoré má používateľ v OS servera,
2. vracať identifikačné číslo, ktoré má používateľ v OS klienta,
3. pridávať používateľov,
   - vytvorenie účtu v OS servera,
   - pridanie záznamu o používateľovi do súboru **linux_passwd**,
   - pridanie záznamu o používateľovi do súboru **linux_group**,
   - vygenerovanie súkromného a verejného ssh kľúča,
   - nastavenie príslušných práv na čítanie, zápis a vykonávanie s ohľadom na prístup vebovej aplikácie k súborom používateľa,
   - synchronizácia systémových súborov OS klientov so súborom **linux_passwd** a **linux_group**,
4. odoberať používateľov,
   - vymazanie účtu v OS servera,
   - odobratie záznamu o používateľovi zo súboru **linux_passwd**,
   - odobratie záznamu o používateľovi zo súboru **linux_group**,
   - vymazanie súborov používateľa,
   - synchronizácia systémových súborov OS klientov so súborom **linux_passwd** a **linux_group**,
5. spravovať úlohy,
   - spúšťať,
   - kompilovať zdrojové kódy.

Každý používateľ systému na spravovanie distr. výpočtov má účet na serveri a na každom uzle. V OS Linux existujú systémové súbory **/etc/passwd**, **/etc/group**, **/etc/shadow**, ktoré zapezpečujú identifikáciu používateľov. Systém na distribuované výpočty zdieľa prostredníctvom NFS (network file system) časť svojho diskového priestoru.

Pôvodný plán nášho systému predpokladal server s dvoma uzlami v jednej lokálnej sieti. Uzly sa mali načítavať (boot-ovať) zo servera a ich diskový priestor mal tvoriť sieťový diskový oddiel zo servera. To znamená, že diskový priestor každého uzla mal byť rovnaký ako oddiel, ktorý nechal zdielať server. Interný manažér by v tomto prípade zapísal nového používateľa do súboru passwd. A tento súbor by bol systémovým súborom /etc/passwd pre každý uzol.

Keďže sme nemali k dispozícii takto naplánovaný systém, ale samostatne inštalované OS Linux na rôznych PC v rôznych sieťach, museli sme plán upraviť.

Interný manažér zapíše údaje o používateľovi do súborov **linux_passwd** a **linux_group**. Tie sa pripoja k existujúcim systémovým súborom **/etc/passwd**, **/etc/group**. Týmto zosynchronizovaním informácii o účtoch zabezpečujeme, že na každom uzle budú údaje o používateľoch systému na spravovanie distr. výpočtov.

Hlavná časť implementácie interného manažéra sa nachádza v jednom súbore. Je to skript **im.sh**. Súbor musí mať vlastníka superpoužívateľ a musí mať právo vykonania. Skript možno rozdeliť na dve časti. Na konfiguračnú časť a na aplikačnú. V prvej časti je potrebné správne nastaviť hodnoty premenných. Na nich totiž závisí správanie sa interného manažéra.

Premenné, ktoré treba vyplniť:

```
linux_group_init_id=5000
linux_group_name="dista"
linux_user_init_id=5000
```

Pre OS Linux je potrebné nastaviť id skupiny (**linux_group_init_id**) a prvé id používateľa (**linux_user_init_id**). V tejto skupine (**linux_group_name**) budú všetci používatelia systému DISTA.

```
linux_passwd_filename="/usr/data/internal_manager/linux_passwd"
linux_group_filename="/usr/data/internal_manager/linux_group"
linux_home_dirs="/usr/data/dista_data"
linux_passwd_home_dirs="/dista/data/dista_data"
linux_shell="/bin/false"
```

linux_passwd_filename je cesta k súboru, kde sa budú zapisovať informácie o používateľoch systému DISTA pre uzly s OS Linux. **linux_group_filename** je cesta k súboru, kde sa budú zapisovať informácie o používateľoch skupiny **linux_group_name** pre uzly s OS Linux. V adresári **linux_home_dirs** budú domovské adresáre používateľov systému DISTA. **linux_passwd_home_dirs** je cesta, ktorá sa zapíše do súboru **linux_passwd_filename** ako cesta k domovskému adresáru. Tá sa totižto pre OS Linux môže líšiť od cesty v OS FreeBSD (na serveri). **linux_shell** je cesta k interpretu príkazov (k **shell**u).

```
freebsd_group_apache="apache"
freebsd_group_apache_id=70
```

freebsd_group_apache je skupina, ktorá bude nastavená ako skupina pre domovské adresáre používateľov. Musí to byť skupina, ktorá prevádzkuje vebovú aplikáciu. V našom prípade je to skupina apache. **freebsd_group_apache_id** je id danej skupiny.

```
freebsd_home_dirs="/usr/data/dista_data"
freebsd_group_init_id=5000
freebsd_group_name="dista"
freebsd_group_filename="/etc/group"
freebsd_shell="/usr/sbin/nologin"
```

freebsd_home_dirs označuje cestu k domovským adresárom používateľov v OS FreeBSD. Musí byť totožná s hodnotou **linux_home_dirs**. **freebsd_group_init_id** je id skupiny **freebsd_group_name** pre OS FreeBSD. **freebsd_group_filename** udáva cestu k súboru, kde sa nachádzajú skupiny a **freebsd_shell** označuje cestu k interpretu príkazov, ktorý sa nastaví pre konto používateľa na serveri FreeBSD.

```
path_to_update_script="/usr/data/www/DISTA/update.php"
```

Path_to_update_script označuje cestu k pomocnému skriptu, ktorý zapisuje výsledok výpočtu do databázy.

Po rozšírení systému by mohol interný manažér nastavovať kvóty a limity pre používateľov.

5.5 Wrapper

Wrapper je časťou Interného manažéra. Jeho úloha bola popísaná v dokumente zaoberajúcom sa špecifikáciou a návrhom. V tejto časti je obsiahnutý konkrétny popis funkcionality, správania sa, vstupov a výstupov.

5.5.1 Prehľad funkcií

Ide o program v jazyku "C", ktorý sa ovláda argumentami na príkazovom riadku na dosiahnutie nasledovných funkcií:

- Štart programu pre výpočet
- Ukončenie programu pre výpočet
- Kompilácia / predpripravenie dát pre výpočet

Samotný *Wrapper* sa volá z *démona*, ktorý ho spúšťa nasledovným spôsobom (kurzívou sú uvedené názvy programov, ktoré sa nachádzajú v niektorom z adresárov, kde operačný systém hľadá spustiteľné súbory):

ssh -l "**používateľ**" "**uzol**" *cd* "**projekt**"; *chroot* "**nový koreňový adresár**"; *wrapper* "**príkaz**" "**projekt**" "**program**" "**args**"

Vysvetlenie:

- používateľ: prihlasovacie meno používateľa, ktorý ide vykonať príkaz

- uzol: IP adresa alebo meno uzla, na ktorom chce používateľ vykonať príkaz
- projekt: cesta k projektu, ktorý bude vykonávať výpočty. Do tohto adresára sa zapisuje všetok výstup – štandardný, chybový, ako aj súbory, ktoré produkuje samotný program na výpočet
- nový koreňový adresár: cesta na nový systémový adresár (prístupový bod "/")
- príkaz: príkaz pre *wrapper* – jedna z troch možností: "start", "stop", "compile"
- program: spustiteľný program pre projekt (pre príkaz "start" a "stop") alebo vstup pre program *make* (pre príkaz "compile")
- args: argumenty, ktoré sa odovzdajú na príkazový riadok

Najprv sa teda spustí program *secure shell*, ktorý prihlási používateľa na daný uzol. Zmení adresár na projektový, aby sa všetky výstupy ukladali do projektového a nie do domovského adresára. Potom sa vykoná príkaz *change root*, ktorý zmení systémový adresár "/" na nejaký iný (aby mal používateľ k dispozícii iba nástroje, ktoré má dovolené používať). Finálne sa spustí samotný *Wrapper* s príkazom a dodatočnými argumentami.

Pre korektné chovanie v rámci systému Dista je potrebné, aby mal používateľ na danom uzle / uzloch vygenerovaný účet , vygenerované kľúče pre *secure shell*, ktoré povoľujú prístup zo servera bez zadávania hesla a povolené spúšťanie príkazu *change root* v súbore /etc/sudoers pre daného používateľa. Všetky tieto akcie zabezpečuje *démon* pri vytváraní používateľa (registrácii a potvrdení administrátorom).

5.5.2 Príprava prostredia

Táto kapitola podrobne popisuje proces, ktorý prebehne tesne pred spustením niektorého z príkazov.

- volanie funkcie *fork()*: táto funkcia slúži na vytvorenie nového procesu, pričom v pamäti budú existovať obidva procesy – ako pôvodný (rodič), tak aj ten práve vytvorený (dieťa). Tento krok je potrebný na to, aby inštancia programu *wrapper* nezanikla po zavolaní funkcie *execl()*, ktorá spúšťa program na výpočet, resp. inú operáciu (kompiláciu, zastavenie programu).
- presmerovanie štandardného a chybového výstupu: táto operácia prebehne v rodičovskom procese. Výstupy presmerovávame z dôvodu, aby ich používateľ mal možnosť vidieť cez webové rozhranie – štandardný výstup sa presmeruje do súboru *stdout.log* a chybový výstup do súboru *stderr.log* pomocou funkcie *strdup()*. Tieto súbory potom číta webové rozhranie a zobrazuje v okne prehliadača, resp. ponúka na stiahnutie. Do obidvoch súborov sa zapisuje výstup zo všetkých uzlov, takže je potrebné ich vytvoriť tak, aby sa vždy iba pripojil nový obsah k pôvodnému – tzv. *append* mód. Na začiatku výpočtu sa tieto súboru zmažú, aby v nich neostal starý záznam z predošlých spustení.
- spustenie programu definovaného príkazom: táto operácia prebehne v dieťati pomocou funkcie *execl()*, ktorá sa už nevráti, pretože daný proces je prepísaný tým, ktorý sa spustí cez funkciu *execl()*. Rodič zatiaľ čaká cez funkciu *wait()* na návratový kód dieťaťa, ktorý potom odovzdá *démonovi*.

5.5.3 Spustenie programu

Ak je program v projektovom adresári k dispozícii (už skompilovaný alebo ide o skript), je pripravený na spustenie. *Wrapper* poskladá finálnu cestu z projektovej cesty a mena spustiteľného súboru a odovzdá mu parametre na príkazový riadok.

5.5.4 Zastavenie programu

Program sa môže zastaviť buď sám (dokončí požadovaný výpočet) alebo na príkaz používateľa. Tento príkaz je realizovaný spustením programu *killall*. Ako parametre mu dodáme názov bežiaceho programu, ktorý chceme ukončiť (musí byť identický s tým, ktorý sme spúšťali) a voliteľne ďalšie parametre pre program *killall*. Treba poznamenať, že momentálne projektová cesta nie je pri tomto príkaze používaná, ale v budúcnosti sa tento stav môže zmeniť, takže sa odporúča ju uvádzať korektne aj pri tomto príkaze.

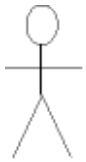
5.5.5 Kompilácia programu

Tento príkaz je určený primárne na kompiláciu programov v jazyku "C" použitím programu *make*. Vstupom pre tento program je súbor, tzv. *Makefile*, ktorý definuje akcie (postupnosť príkazov), ktoré sa majú vykonať pri požadovanej operácii. Pod požadovanou operáciou si môžeme predstaviť nielen kompiláciu, ale aj prípravu vstupov, vyčistenie adresára od nepotrebných súborov atď. Používateľ teda dodá *Makefile*, ktorý sa nahrá do projektovej cesty. Potom sa spustí program *make*, ktorému sa ešte dodajú dodatočné parametre definujúce požadovanú operáciu v rámci *Makefile*u.

A Použitá notácia

A.1 Model prípadov použitia

Hráč (obrázok 8) je osoba, organizácia alebo vonkajší systém, ktorý hrá úlohu v jednom alebo viacerých vzťahoch s našim systémom.



Obr. 8: Hráč

Prípad použitia (obrázok 9) je postupnosť činností, ktorá má merateľnú hodnotu pre hráča.

Obr. 9: Prípad použitia

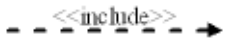
Asociácia (obrázok 10) je znázornená orientovanou úsečkou, ktorá vyjadruje smer inicializácie vzťahu medzi elementami modelu.

Obr. 10: Asociácia

Zovšeobecnenie (obrázok 11) je orientovaná úsečka a smeruje od konkrétnejšieho k všeobecnejšiemu.

Obr. 11: Zovšeobecnenie

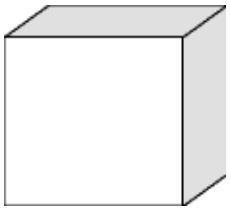
Závislosť (obrázok 12) umožňuje zahrnúť prípad použitia (ukazuje na neho orientovaná úsečka) do iného prípadu použitia.



Obr. 12: Závislosť

A.2 Diagram rozmiestnenia

Uzol (obrázok 13) predstavuje hardvér v systéme.



Obr. 13: Uzol

Komponent (obrázok 14) predstavuje softvérovú entitu.

Obr. 14: Komponent

Komunikácia (obrázok 15) sa odohráva medzi komponentami.

Obr. 15: Komunikácia

Vzťah (obrázok 16) vzniká medzi hardvérovými uzlami.

Obr. 16: Vzťah

B Inštalačná príručka

Inštalácia systému je pre priemerne skúseného admistrátora pomerne nenáročná. Vyžaduje si však relatívne veľa času. Predtým, ako pristúpime ku konkrétnym nastaveniam a popisom inštalácie, stručne zhrnieme s akými technickými prostriedkami počítame.

Systém pozostáva z jedného PC, ktorý plní úlohu servera. Na tomto PC budeme inštalovať operačný systém FreeBSD verzia 6.2. Okrem servera budeme potrebovať aspoň dva osobné počítače s OS Linux. Na distribúcii a verzii veľmi nezáleží, my sme zvolili minimálnu inštaláciu OS Linux Slackware 10.0 bez modifikácie jadra systému.

Ako sme už spomenuli, pôvodný plán počítal s načítaním OS Linux zo servera. Keďže náš server nemal verejnú IP adresu a v lokálnej sieti sme nedostali ďalšie počítače, museli sme plán, a tým pádom aj inštaláciu modifikovať. Ak máte možnosť zostaviť systém v lokálnej sieti, je načítanie OS Linux zo servera ideálne riešenie.

Na tomto mieste však popíšeme technickú inštaláciu nášho riešenia.

B.1 Server

Na server sme nainštalovali OS FreeBSD 6.2 Release verziu bez modifikácie jadra. Zvolili sme vlastnú inštaláciu. Použili sme štandardné rozloženie diskových oddielov. Povolili sme post-inštalačný systém **ports**. Na overenie existencie chýb v aplikáciách sme použili program **portaudit**. Pri inštalácii je potrebné povoliť **NFS**. Štandardne je táto možnosť povolená, čiže ju stačí nezmeniť.

Súbor rc.conf

```
hostname="dista.ucebne"
ifconfig_re0="DHCP"
keymap="sk.iso2"

# SSHcko
sshd_enable="YES"

# Firewall
firewall_enable="YES"
firewall_logging="YES"

postgresql_enable="YES"
apache_enable="YES"

# NFS server
```

```
rpcbind_enable="YES"
nfs_server_enable="YES"
mountd_flags="-r"
```

Pravidlá firewallu:

```
00100 check-state
00200 allow ip from any to any via lo*
00300 deny ip from 127.0.0.0/8 to any in
00400 deny ip from any to 127.0.0.0/8 in
00500 deny ip from 224.0.0.0/3 to any in

00800 allow ip from any to any out via tun1
00900 allow ip from any to any in via tun1

00800 allow ip from any to any out via tun2
00900 allow ip from any to any in via tun2

01000 allow tcp from any to any out keep-state
01100 allow udp from any to any out keep-state
01200 allow tcp from any to any established keep-state
01300 allow tcp from any to any dst-port 22 in keep-state
01400 allow tcp from any to any dst-port 80 in keep-state
01500 allow tcp from any to any dst-port 443 in keep-state
01800 allow icmp from any to any out keep-state
01900 allow icmp from any to any in keep-state
02000 deny udp from any to any
02100 deny tcp from any to any
65535 deny ip from any to any
```

Nainštalovali sme **postgreSQL** verziu 8.2.3

```
# ./configure --prefix=/usr/local/pgsql
# make
# make install
```

Nainštalovali sme **apache** verziu 2.2.4

```
# ./configure --prefix=/usr/local/apache2 --enable-ssl
# make
# make install
```

Nainštalovali sme **php** verziu 5.2.1

```
# ./configure --prefix=/usr/local/php \
#        --with-config-file-path=/etc \
#        --with-apxs2=/usr/local/apache2/bin/apxs \
#        --with-pgsql=/usr/local/pgsql \
#        --with-snmp
# make
```

```
# make install
# strip /usr/local/php/bin/php
# ln -s /usr/local/php/bin/php /usr/local/bin
# cp php.ini-recommended /etc/php.ini
```

Cez systém ports sme nainštalovali **OpenSSL** verziu 0.9.7 a **SNMP** verziu 5.2.3.

Systém NFS

Nastavenie zdieľania oddielu /usr vrátane podadresárov pre počítače na IP adresách *10.4.0.1 10.4.0.2 10.5.0.1 10.5.0.2 localhost 127.0.0.1.*

```
# echo "/usr -alldirs -maproot=0 10.4.0.1 10.4.0.2 10.5.0.1 10.5.0.2 localhost \
127.0.0.1" > /etc/exports
```

Voľby:

- **alldirs** - umožňuje pripojiť ktorýkoľvek adresár v rámci zdieľanej partície
- **maproot=id/username** - nastavuje, ktorý používateľ zdedí práva superpoužívateľa
- **mapall=id/username** - umožňuje nastaviť používateľa ako jediného vlastníka všetkých zdieľaných súborov

Použité IP adresy zodpovedajú adresám, ktoré sme použili. Viac informácii o prepojení je k dispozícii v časti „prepojenie systému“.

Spustenie služby NFS

```
# rpcbind
# nfsd -u -t -n 4
# mountd -r
```

Zastavenie služby NFS

```
# killall -9 rpcbind
# killall -9 nfsd
# killall -9 mountd
```

Dôležité je, aby v súbore **/etc/hosts.allow** bola služba rpcbind pre všetky uzly povolená. Inak zdieľanie nebude fungovať. Taktiež odporúčame kontrolovať súbor /var/log/messages, kde sa nachádzajú záznamy o správaní a prípadných chybách práve spúšťaných programov. V prípade problémov je možné nájsť veľa odkazov na internete, ktoré môžu pomôcť problém bližšie špecifikovať a poskytnúť riešenia, napríklad [9].

B.2 Prepojenie počítačov

V prípade, že máte k dispozícii lokálnu sieť, kde môžete mať server prepojený s uzlami, je vaša situácia ideálna. Keďže to nebol náš prípad, opíšeme inštaláciu v iných podmienkach. Náš server sa nachádza v lokálnej sieti počítačov za firewallom, bez verejnej IP adresy. Iné počítače sme v danej lokálnej sieti k dispozícii nedostali. Uzly sú na serveri XENA, ktorý spravuje *Adam Hamšik*. Tento server sprístupňuje XEN virtuálne domény pre študentov Fakulty Informatiky a Informačných Technológii pre študijné účely. OS Linux, ktorý je umiestnený na serveri XENA má modifikované jadro, aby mohol na danom serveri bežať. Oba uzly majú verejnú IP adresu. Táto situácia je neštandardná. Ak by mal server verejnú IP adresu, nebol by žiaden problém počítače prepojiť pomocou VPN (virtual private network)-virtuálnou súkromnou sieťou.

B.2.1 VTUN

Situáciu sme vyriešili pomocou **VTUN** (**Virtual Tunnels over TCP/IP networks**) [10]. Tento nástroj umožňuje prepájať počítače pomocou tunelov a čiastočne vytvárať virtuálne siete. Princíp tejto aplikácie spočíva v komunikácii dvoch procesov. Klient - server. Okrem toho, aplikácia vtun vytvára virtuálne rozhrania (**virtual interfaces**). Tie sa v systéme javia ako sieťové zariadenia.

Aplikácia vtun má teda dve časti. Časť klient a časť server. Ak máme priamy prístup k serveru, vyzerá konfigurácia nasledovne: Na serveri a na uzle nainštalujeme vtun a vytvoríme na oboch konfiguračný súbor. Na serveri nastavíme, že vtun sa bude správať ako server, na uzle nastavíme, že sa bude správať ako klient.

Ak nemáme prístup na server, čo bol náš prípad, musíme konfiguráciu otočiť. Na serveri musíme nastaviť, že sa vtun bude správať ako klient a na uzle musíme nastaviť, že vtun sa bude správať ako server. Je to neštandardná situácia, do ktorej sme sa dostali vďaka tomu, že sme nemali priamy prístup k serveru. Tento postup však funguje. Je prirodzene pracnejší, ale predstavuje spôsob, ako sa s daným problémom vysporiadať.

Konfiguračný súbor uzla 1 vyzerá nasledovne:

```
options {
   port       9001;
   ifconfig   /sbin/ifconfig;
}
# Default session options
default {
   compress   no;
   encrypt    no;
   speed      0;
   keepalive  yes;
   stat       yes;
}
tunnel_uzol1 {
```

```
   password       foo;
   type       tun;
   proto      tcp;
   device       tun0;
   up {
      ifconfig "%% 10.4.0.1 pointopoint 10.4.0.2 mtu 1450";
   };
   down {
      ifconfig "%% down";
   };
}
```

Konfiguračný súbor pre uzol 1 na serveri vyzerá nasledovne:

```
options {
   port        9001;
   ifconfig    /sbin/ifconfig;
}
# Default session options
default {
   compress    no;
   encrypt     no;
   speed       0;
   keepalive   yes;
   stat        yes;
}
tunnel_uzol1 {
   pass        foo;
   type        tun;
   proto       tcp;
   device       tun1;
   up {
      # 10.4.0.2: tunnel interface (local)
      # 10.4.0.1: tunnel interface (remote)

      ifconfig "%% 10.4.0.2 10.4.0.1 mtu 1450";
   };
   down {
      ifconfig "%% down";
   };
}
```

Konfiguračný súbor uzla 2 vyzerá nasledovne:

```
options {
   port       9002;
   ifconfig   /sbin/ifconfig;
}
# Default session options
default {
```

```
   compress   no;
   encrypt    no;
   speed      0;
   keepalive  yes;
   stat       yes;
}
tunnel_uzol2 {
   password       foo;
   type       tun;
   proto      tcp;
   device       tun2;
   up {
      ifconfig "%% 10.5.0.1 pointopoint 10.5.0.2 mtu 1450";
   };
   down {
      ifconfig "%% down";
   };
}
```

Konfiguračný súbor pre uzol 2 na serveri vyzerá nasledovne:

```
options {
   port        9002;
   ifconfig    /sbin/ifconfig;
}
# Default session options
default {
   compress    no;
   encrypt     no;
   speed       0;
   keepalive   yes;
   stat        yes;
}
tunnel_uzol2 {
   pass        foo;
   type        tun;
   proto       tcp;
   device       tun2;
   up {
      # 10.5.0.2: tunnel interface (local)
      # 10.5.0.1: tunnel interface (remote)
      ifconfig "%% 10.5.0.2 10.5.0.1 mtu 1450";
   };
   down {
      ifconfig "%% down";
   };
}
```

Pred samotným spustením **vtun** je potrebné vytvoriť **ssh** tunel medzi server-uzol1 a server-uzol2. Aby sme nemuseli používať prihlasovanie na základe ručného

vypisovania hesla, použili sme prihlasovanie pomocou privátneho a verejného kľúča.

B.2.2 Autentifikácia používateľa na základe kľúčov

Autentifikácia na základe kľúčov používa dvojicu kľúčov. Privátny a verejný kľúč. Pojmu kľúč sa netreba obávať, ide o textové súbory. Dôvod vytvárania kľúčov je jednoduchý. Nie je potrebné vypisovať heslo pri každom prihlásení. Privátny kľúč zostáva na počítači odkiaľ sa chceme prihlásiť na vzdialený počítač. Na vzdialenom počítači musí byť verejný kľúč zapísaný v súbore **$HOME/.ssh/authorized_keys** [16]. Pre zjednodušenie pomenujme počítač z ktorého sa chceme prihlasovať (**O**) a počítač na ktorý sa chceme prihlasovať (**K**).

Na **0** vygenerujeme kľúče pomocou príkazu:

```
# ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/usr/home/MENO_POUZIVATELA/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_dsa.
Your public key has been saved in id_dsa.pub.
The key fingerprint is:
45:7e:f6:74:29:b2:0d:98:76:6e:69:ca:9e:2d:0e:9a MENO_POUZIVATELA@DOMENA
```

Po volaní príkazu **ssh-keygen -t dsa** sa proces pýta na umiestnenie a názov privátneho kľúča. Následne sa pýta na **passphrase**. Je to reťazec, ktorý sa použije na generovanie privátneho kľúča. Predpokladajme najjednoduchšiu možnosť; nebudeme meniť názov privátneho kľúča, a zadáme prázdny **passphrase**. Privátny kľúč bude v našom domovskom adresári a bude sa volať id_dsa. Náš verejný kľúč sa bude volať id_dsa.pub. Verejný kľúč zapíšeme na počítači **K** do súboru **$HOME/.ssh/authorized_keys**. Máme spojenie **O**-**K**. Spojenie **K**-**O** sa nebude realizovať na základe kľúčov. Ak chceme mať viacero párov kľúčov, môžeme zmeniť názov privátneho kľúča z predvoleného id_dsa napr. na id_dsa_uzol1. Zodpovedajúci verejný kľúč je rovnako potrebné zapísať na **K** do súboru **authorized_keys**.

B.2.3 Vytvorenie ssh tunelu

Ak sa pozrieme na konfiguračné súbory **vtun** pre spojenie uzol1-server, v sekcii **options** nájdeme položku **port**. V oboch konfiguračných súboroch je hodnota 9001. To znamená, že na uzle1 bude démon **vtund** „počúvať“ na porte 9001 a klientská časť programu **vtun** sa bude snažiť pripojiť na port 9001. Keďže nebudeme na uzle povolovať port 9001, použijeme **ssh** tunel.

Na serveri spustíme:

```
# ssh -f -i ~/.ssh/id_dsa -L 9001:localhost:9001 username@uzol1 "sleep 43100"
```

Ak sa nepreruší spojenie, máme na dvanásť hodín presmerovaný port 9001 cez tunel na uzol1. Keď sa budeme dotazovať na lokálne zariadenie **lo0** (**localhost**) na port 9001, budeme sa v skutočnosti dotazovať na port 9001 uzla1.

[16] $HOME je označenie domovského adresára.

Oživenie virtuálneho spojenia sa robí nasledovne:
Na uzle 1 sa spustí:

```
# vtund -s -n -f /etc/vtun/vtund_server.conf &
```

Na serveri sa spustí:

```
# vtund -f ~/vtund_client_uzol1 tunnel_uzol1 localhost
```

Analogicky treba postupovať pri zriaďovaní virtuálnej siete medzi serverom a uzlom2.

Vzhľadom k tomu, že je potrebné kontrolovať tunely a virtuálne spojenia, vytvorili sme skripty, ktoré sa spúštajú každé dve minúty a kontrolujú stav spojení. Na pravidelné spúšťanie kontroly sme použili nástroj **cron** [**?**].

Kontrola tunelu na prvý uzol.

```
if [‘netstat -na | grep 9001 | wc -l‘ = 0 ]; then
  ssh -f -i ~/.ssh/id_dsa_uzol1 -L 9001:localhost:9001 admin@uzol1 "sleep 43100"
fi;
```

Kontrola tunelu na druhý uzol.

```
if [‘netstat -na | grep 9002 | wc -l‘ = 0 ]; then
  ssh -f -i ~/.ssh/id_dsa_uzol2 -L 9002:localhost:9002 admin@uzol2 "sleep 43100"
fi;
```

Kontrola virtuálnej siete na prvý uzol.

```
if [ ‘netstat -r | grep tun1 | wc -l‘ = 0 ]; then
  /usr/local/sbin/vtund -f vtund-client_uzol1.conf tunnel_uzol1 localhost
fi;
```

Kontrola virtuálnej siete na druhý uzol.

```
if [ ‘netstat -r | grep tun2 | wc -l‘ = 0 ]; then
  /usr/local/sbin/vtund -f vtund-client_uzol2.conf tunnel_uzol2 localhost
fi;
```

Po úšpešnom zriadení spojenia je čas na inštaláciu vebovej aplikácie, interného manažéra a wrappera. Na zdieľanom disku je vhodné si zvoliť adresár, v ktorom budú takmer všetky súčasti systému na spravovanie distribuovaných výpočtov. My sme si za taký adresár zvolili adresár **/usr/data**. Zároveň sme vytvorili linku **/dista** ktorá ukazuje na adresár **/usr**. Prirodzene si môžete zvoliť vlastné adresáre, dôležité je však vhodne editovať súbor interného manažéra.

B.3 Inštalácia častí systému

B.3.1 Inštalácia interného manažéra

(/usr/data/internal_manager)

Do adresára **/usr/data** sme umiestnili adresár **internal_manager**, kde sa nachádza skript vnútorného manažéra a pomocné súbory ako **linux_passwd** a **linux_group**. Po skopírovaní interného manažéra do adresára je potrebné editovať prvú časť skriptu **im.sh** podľa opisu z časti „Interný manažér“. Súčasťou celkového riešenia je aj skript *update.php*, ktorý je v adresári vebovej aplikácie.

B.3.2 Inštalácia wrappera

(/usr/data/wrapper)

Do adresára **/usr/data** sme umiestnili adresár **wrapper**. Tu sa nachádzajú zdrojové súbory wrappera, makefile a ostatné pomocné súbory. Dôležité je kompilovať wrappera pod operačným systémom Linux a nie pod FreeBSD. Tieto operačné systémy nie sú štandardne binárne kompatibilné.

B.3.3 Inštalácia vebovej aplikácie

(/usr/data/www/DISTA)

Vebovú aplikáciu sme umiestnili do adresára **/usr/data/www/DISTA**. Okrem súborov vebovej aplikácie sa tu nachádzajú aj súbory **snmp.php** (monitorovanie uzlov) a **update.php** (aktualizácia záznamov o výsledkoch výpočtov). Ak vebovú aplikáciu umiestnite kamkoľvek, pamätajte na to, že je potrebné prekonfigurovať aplikáciu apache, kvôli prístupu k zvolenému adresáru. Dôležité je aj pamätať na konfiguračnú časť interného manažéra. V nej je sa nachádzajú definície ciest ku php skriptom, ktoré sú súčasťou vebovej aplikácie.

Literatúra

[1] Stránka projektu PVM
http://www.csm.ornl.gov/pvm/

[2] PVM: Parallel Virtual Machine, Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam
http://www.netlib.org/pvm3/book/node2.html#SECTION00110000000000000000
PVM

[3] Parallel-Pvm-1.4.0
http://search.cpan.org/ dleconte/Parallel-Pvm-1.4.0/Pvm.pm

[4] Stránka projektu MPI
http://www-unix.mcs.anl.gov/mpi/

[5] MPI: A Message-Passing Interface Standard
http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html A Message-Passing Interface Standard

[6] PVM and MPI: A Comparison of Features - G.A Geist, J. A. Kohl, P. M. Papadopoulos
http://www.csm.ornl.gov/pvm/PVMvsMPI.ps

[7] Why are PVM and MPI So Different - William Gropp, Ewing Lusk
http://www.mcs.anl.gov/ gropp/bib/papers/1997/pvmpaper.ps

[8] PVM and MPI are completely different - William Gropp, Ewing Lusk
http://www.mcs.anl.gov/ gropp/bib/papers/1997/fgcspaper.ps

[9] *http://www.freebsddiary.org/nfs.php*

[10] *http://vtun.sourceforge.net*

[11] Kosek, Jiří: PHP: Tvorba interaktívních internetových aplikací. Praha: Grada, 1999. 490 s. yISBN 80-7169-373-1

[12] Zend PHP: Documentation http://www.php.net/docs.php
1. 11. 2006

[13] WikipediA PHP
http://en.wikipedia.org/wiki/Php
1. 11. 2006

[14] ROOT Databáze v Linuxu (Tutoriály na Rootu)
http://tutorialy.root.cz/linux-na-serveru/databaze-v-linuxu/
1. 11. 2006

[15] WIKIPEDIA MySQL
http://en.wikipedia.org/wiki/Mysql
1. 11. 2006

[16] POSTGRESQL GLOBAL DEVELOPMENT GROUP PostgreSQL: About
http://www.postgresql.org/about/ 1. 11. 2006

[17] Volunteer computing, 16 Aug 2006.
http://boinc.berkeley.edu/volunteer.php. [2006-11-06]

[18] Desktop grid computing with BOINC, 16 Aug 2006.
http://boinc.berkeley.edu/dg.php. [2006-11-06]

[19] Overview of BOINC, 23 Sep 2005.
http://boinc.berkeley.edu/intro.php. [2006-11-06]

[20] How did the Condor project start?.
http://www.cs.wisc.edu/condor/background.html. [2006-11-06]

[21] Alan Beck, High Throughput Computing: An Interview with Miron Livny, 27. June 1997.
http://www.cs.wisc.edu/condor/HPCwire.1. [2006-11-06]

[22] Scott Fields, Hunting for wasted computing power, 1993.
http://www.cs.wisc.edu/condor/doc/WiscIdea.html. [2006-11-06]

[23] Jeff Mausolf, An eagle-eye view of the Condor project, 15 Feb 2005.
http://www-128.ibm.com/developerworks/library/gr-condor/. [2006-11-06]

[24] Sun Grid. http://www.sun.com/service/sungrid/. [2006-11-06]

[25] Sun Grid Compute Utility - Frequently Asked Questions.
http://www.sun.com/service/sungrid/faq.xml. [2006-11-06]

[26] Sun Microsystems, Inc. Sun Grid Compute Utility Developer's Guide, September 2006.
http://www.sun.com/service/sungrid/SunGridDevelopersGuide.pdf. [2006-11-06]

[27] Grid MP Features and Benefits.
http://www.ud.com/products/gridmp_fabs.php. [2006-11-06]

[28] Ashok Adiga, Nina Wilner: Grid in action: Harvesting and reusing idle compute cyc-
les, 28 Jun 2005. http://www-128.ibm.com/developerworks/grid/library/gr-harvest/?ca=dgr-lnxw01HarvestingGrid. [2006-11-06]