FIIT grid
(Produktová dokumentácia)

Autori:	Bc. Ján Kalmár, Bc. Juraj Petrík, Bc. Juraj Vincúr,
	Bc. Martin Tibenský Bc. Pavol Pidanič, Bc. Radoslav Zápach
Kontakt:	tp-12@googlegroups.com
Akademický rok:	2013/2014
Vedúci práce:	Ing. Peter Lacko, PhD.

1. História zmien dokumentu

Dátum	Verzia	Zhrnutie zmien	Autor
25.10.2013	1.0	Vytvorenie dokumentu, formátovanie a štýly, vytvorenie kapitol úvod a prvý šprint	Ján Kalmár
1.11.2013	2.0	Vytvorenie kapitol druhý šprint	Ján Kalmár
14.11.2013	3.0	Vytvorenie kapitol tretí šprint	Ján Kalmár
18.11.2013	4.0	Vytvorenie kapitol Reversi klientská časť a Asimilátor	Martin Tibenský
21.10.2013	4.1	Dopnenie chybajucich sekcii	Ján Kalmár
11.12.2013	5.0	Vytvorenie kapitol pre šprinty 4 a 5	Ján Kalmár

Tabuľka 1: História dokumentu

Obsah

0. Úvod

0.1. Účel dokumentu

Tento dokument vznikol pre potreby predmetu Tímový projekt ako dokumentácia k riešeniu problému distribuovaného počítania na FIIT (FIIT Grid). V projekte sa vytvára infraštruktúra pre distribuované počítanie na platforme Boinc, na otestovanie funkčnosti a ako ukážkoví rámec pre budúce tými sa v rámci projektu vytvárajú aj rôzne úlohy na počítanie. Na projekte pracujeme šiesti, zadávateľom projektu je Ing. Peter Lacko, PhD.

0.2. Forma dokumentu

Tento dokument dodržiava metodiku na tvorbu technickej dokumentácie.

0.3. Použité technológie

V rámci projektu sa používajú následovné technológie:

- Apache2 web server
- PHP
- MySQL
- Boinc
- Java
- C++
- Redmine – systém pre správu projektov
- Eclipse – IDE
- Git – systém pre správu verzii
- Perkonik – nástroj na reviev kódu

0.4. Slovník pojmov

0.5. Zoznam skratiek

1. Prvý šprint

V prvom šprinte sme si dali následovné úlohy:

- Konfigurácia a inštalácia boinc servera
- Spustenie testovacej úlohy na serveri (Hello World)
- Preštudovanie možností boinc
- Analýza a návrh riešenia pre hru Reversi (Othello)

Výsledkom šprintu je funkčný boinc server spolu so spustenou testovacou úlohou, analýza hry reversi a analýza boinc API. S funkčným boinc serverom sa môžeme ďalej venovať návrhu a implementácii riešenia hry reversi, ktorú v konečnom dôsledku chceme riešiť distribuovane práve pomocou platformy boinc.

1.1. Platforma boinc (analýza)

1.1.1. Úvod

Boinc je skratka pre Berkeley Open Infrastructure for Network Computing a bol vyvinutý na Kalifornskej univerzite pre podporu projektu Seti@home, ktorý sa zaoberá hľadaním mimozemského života pomocou analýzy rádiových vĺn z kozmu. Neskôr sa Boinc stal open-source systémom pre kohokoľvek, kto potrebuje na riešenie svojich paralelne počítateľných problémov využívať vysoký výpočtový výkon a má viac kamarátov, ako peňazí na zaobstaranie superpočítača. Boinc umožňuje vytvárať projekty, na ktorých riešení sa môžu pomocou svojich počítačov podieľať dobrovoľníci (volunteer computing), alebo distribuované počítacie siete, vytvorené z počítačov univerzít a iných inštitúcií (grid computing).

1.1.2. Architektúra a pojmy

Systém pracuje na klasickej klient-server architektúre.

Projekt – každý projekt má svoj identifikátor (masterURL) a samostatnú serverovú časť.

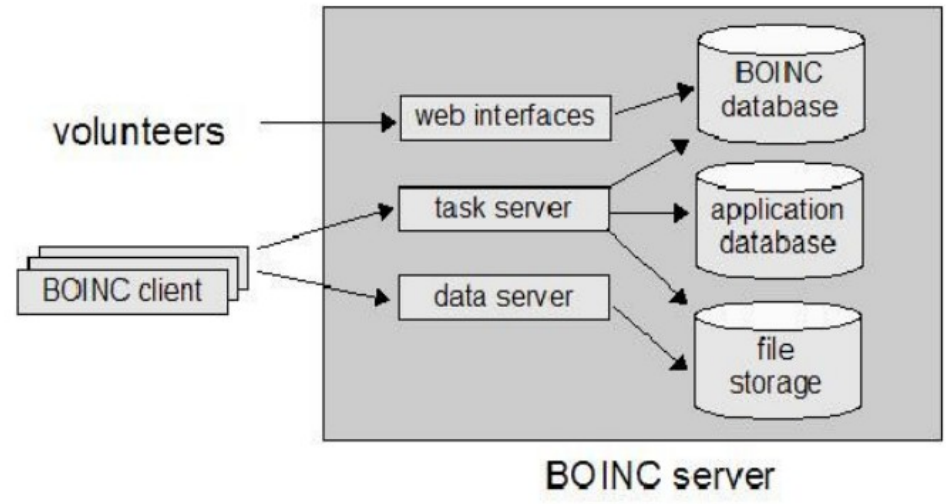
Aplikácia – je spravovaná pod projektom, môže obsahovať niekoľko programov v závislosti od toho, pre koľko platforiem ich chceme vytvárať a množinu **úloh (jobs)**.

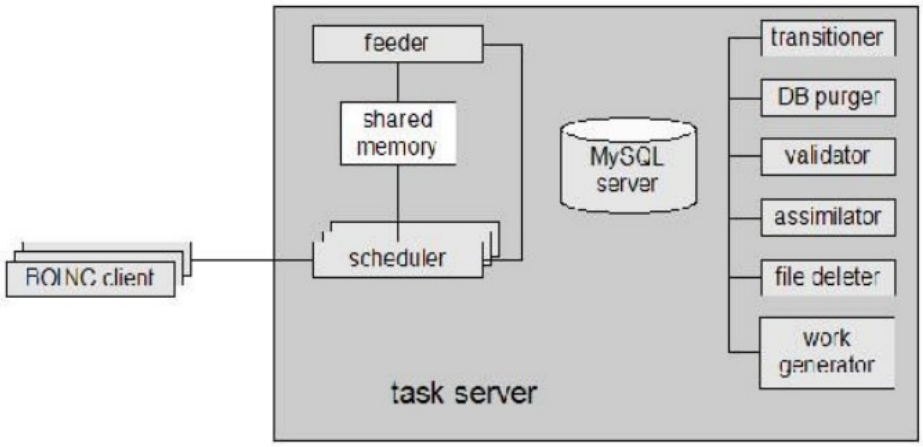
Úloha(job) – pozostáva z dvoch častí, **workunit** a **result**.

Workunit (pracovná jednotka) – v skratke práca, ktorú je potrebné poslať klientovi a vykonať.

Result – obsahuje zoznam súborov s výsledkami výpočtu a ďalšie atribúty ako čas, ktorý CPU spotreboval atď.

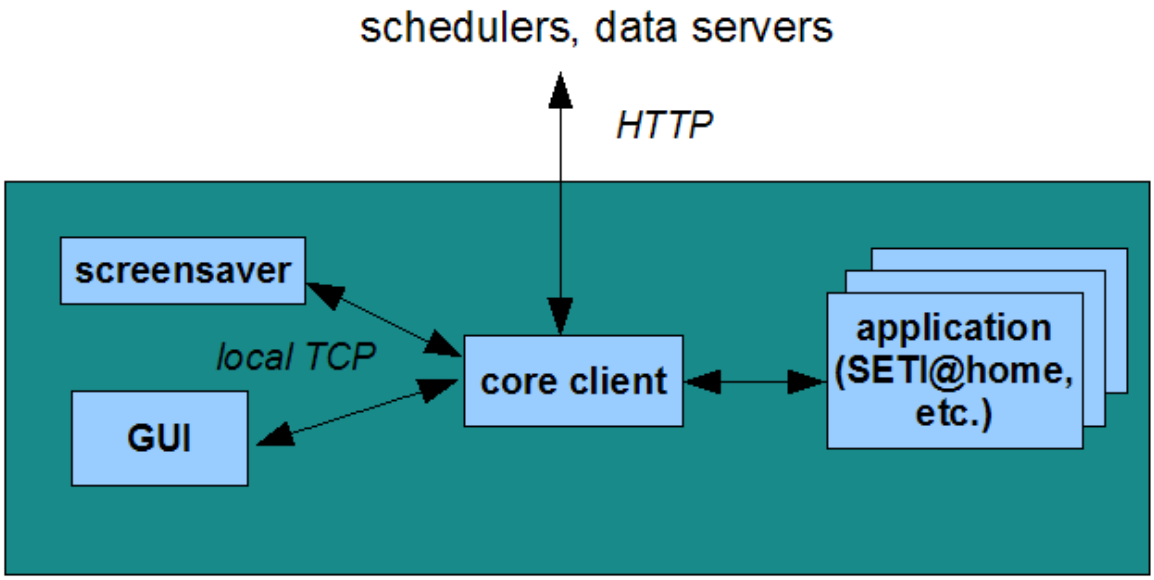
Serverová časť – slúži na správu projektov, pridelovanie úloh a správu dát súvisiacich s projektom.





Task server sa stará o generovanie workunits (work generator), ich pridelovanie vhodným klientom (scheduler), overovanie (validator) a spracovánie (assimilator) výsledkov. Overovanie výsledkov je potrebné hlavne v prípade použitia Boincu na dobrovoľnícke projekty, pri ktorých sa niektorí účastníci z rôznych pokútnych dôvodov snažia odosielať zámerne nesprávne výsledky. Ochrana proti takémuto správaniu spočíva vo vytvorení N klonov každého workunitu a následným porovnaním výsledkov. V prípade, že sa nedosiahne z počtu vrátených výsledkov nejaké vopred určené, aplikačne špecifické kvórum (minimálny počet súhlasiacich výsledkov), Boinc pridelí rovnakú úlohu ďalšiemu klientovi. Klienti navzájom o sebe nevedia, že riešia rovnakú úlohu.

Klientská časť – umožňuje dobrovolníkom vybrať si, na ktorých projektoch budú spolupracovať. Dobrovoľník si vždy vyberá len projekt, nie aplikáciu.



Navonok sa Boinc tvári ako jeden program, pod kožúškom sa však skladá z niekoľkých samostatných častí.

Core client – pomocou http protokolu komunikuje s projektovým serverom, pýta si prácu a odovzdáva výsledky. Slúži tiež na spúšťanie a manažovanie aplikácií.

Application - prekvapivo sú to projektové aplikácie, ktoré vykonávajú výpočty.

GUI – alebo Boinc manager, je grafické rozhranie, ktoré slúži na komunikáciu s core klientom. Používateľovi umožňuje napríklad spúšťať a prerušovať aplikácie. Tradične komunikuje s jadrom lokálne, ale ponúka aj možnosť vzdialené riadenia.

Screensaver – spúšťa sa pri nečinnosti a generuje ho aplikačná časť. Nie všetky aplikácie musia mať screensaver, ale môžeme ním na klientskej stanici zobrazovať napríklad ľúbivé obrázky týkajúce sa projektu, alebo aká časť z aktuálneho worunitu je už spracovaná.

1.2. Platforma boinc (inštalácia)

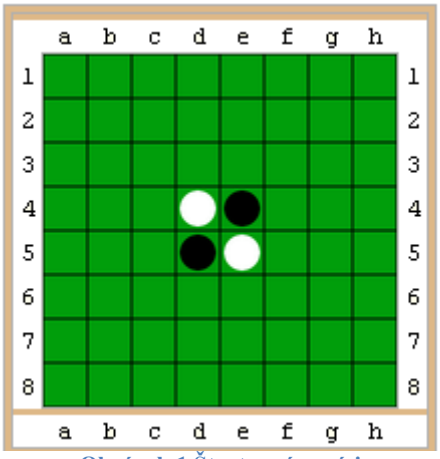
1.3. Reversi

1.3.1. Pravidla hry

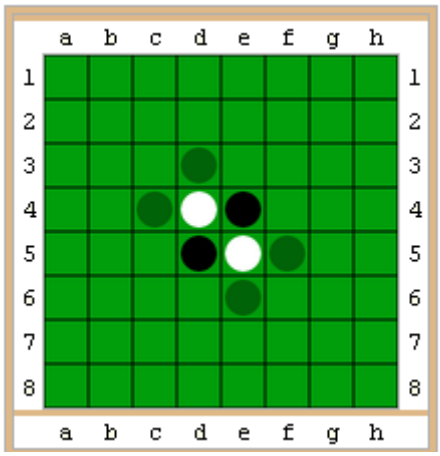
Hra Reversi (Othello) je stolová hra pre dvoch hráčov hraná (zväčša) na šachovnici 8 x 8 polí s kameňmi dvoch farieb.

Hráči na šachovnicu pokladajú kamene svojej farby tak, aby práve položený kameň a iný kameň svojej farby uzavreli súvislý rad súperových kameňov. Tieto uzavreté kamene

potom zmenia farbu a stanú sa kamene druhého hráča. Víťazom sa stáva hráč, ktorý má po zaplnení šachovnice viac kameňov ako protihráč. Prvý ťah má čierny hráč.[1]



Obrázok 1 Štartovná pozícia



Obrázok 2 Možné ťahy čierneho

1.3.2. Analýza riešení

4 x 4 - vyriešená za menej ako sekundu programami, ktoré používajú min-max metódu, ktorá generuje všetky možné pozície (takmer 10 miliónov). Výsledkom je, že biely zvíťazí.

6 x 6 - vyriešená za menej ako 100 hodín programami, ktoré používajú min-max metódu, ktorá generuje všetky možné pozície (takmer 3,6 trilióna). Výsledkom je, že biely zvíťazí.

8 x 8 - nebola zatiaľ matematicky vyriešená. Predpoklad je, že jej strom obsahuje 10^{54} vrcholov.

10 x 10 - nebola zatiaľ matematicky vyriešená. Predpoklad je, že jej strom obsahuje 10^{90} vrcholov[2].

1 ://reversi.hra-hry.sk/pravidla.html
2 http://en.wikipedia.org/wiki/Computer_Othello

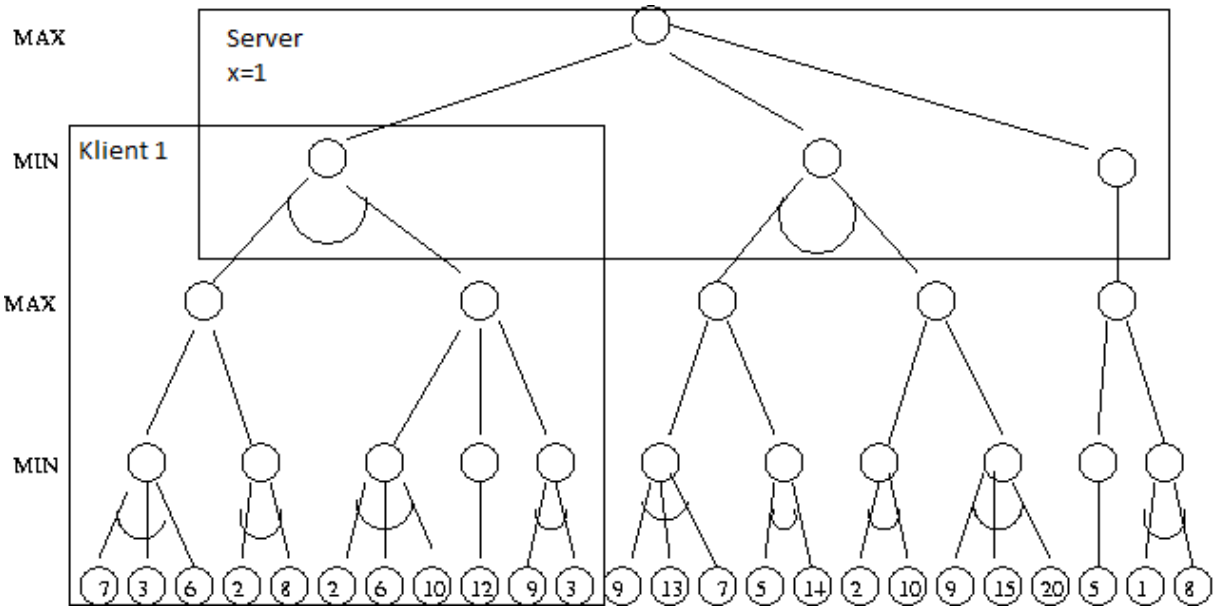
1.3.3. *Návrh riešenia*

Server

- Inicializácia úlohy - prehľadanie stavového priestoru hry od štartovnej pozície, do určitej hĺbky x.
- Vytvorenie workunitov pre klientov - každý obsahuje určitý nájdený stav hry v hĺbke x.
- Vyhodnotenie získaných dát od klientov - určenie víťazného hráča.

Klient

- vyžiadanie si workunitu.
- vyrátanie workunitu - prehľadávanie stavového priestoru hry od zadaného stavu v hĺbke x, až po finálne stavy (listy). Určenie víťaza v tomto strome.
- odoslanie výsledkov späť na server.



Obrázok 3 Rozloženie výpočtu

1.4. Zhodnotenie šprintu

Podarilo sa nám úspešne spustiť boinc server na našom tímovom serveri. Vytvorili sme jednoduchú „Hello World“ aplikáciu, ktorú tento server rozposlal klientom a následne prijal späť výstupy od klientov z tejto aplikácie.

Taktiež sme analyzovali hru reversi, ktorú sme sa rozhodli riešiť a navrhli najefektívnejší spôsob distribúcie čiastkových pod-úloh medzi klientov.

2. Druhý šprint

V druhom šprinte sme sa zamerali na vytvorenie prototypu aplikácie na riešenie hry reversi 6x6. Na toto riešenie sme použili rôzne algoritmy ako Alfa-Beta osekávanie, MTD-F, Negascout, Minimax a ich vylepšenia.

Určite sme si tieto ciele:

- Preskúmať možnosti rôznych algoritmov pre riešenie hry reversi
- Vytvoriť lokálnu aplikáciu pre riešenie hry reversi
- Vytvoriť generátor stavov a otestovať ho na serveri

2.1. Analýza algoritmov

2.1.1. Zadanie

Analyzovať možné algoritmy na riešenie hry reversi.

2.1.2. Analýza

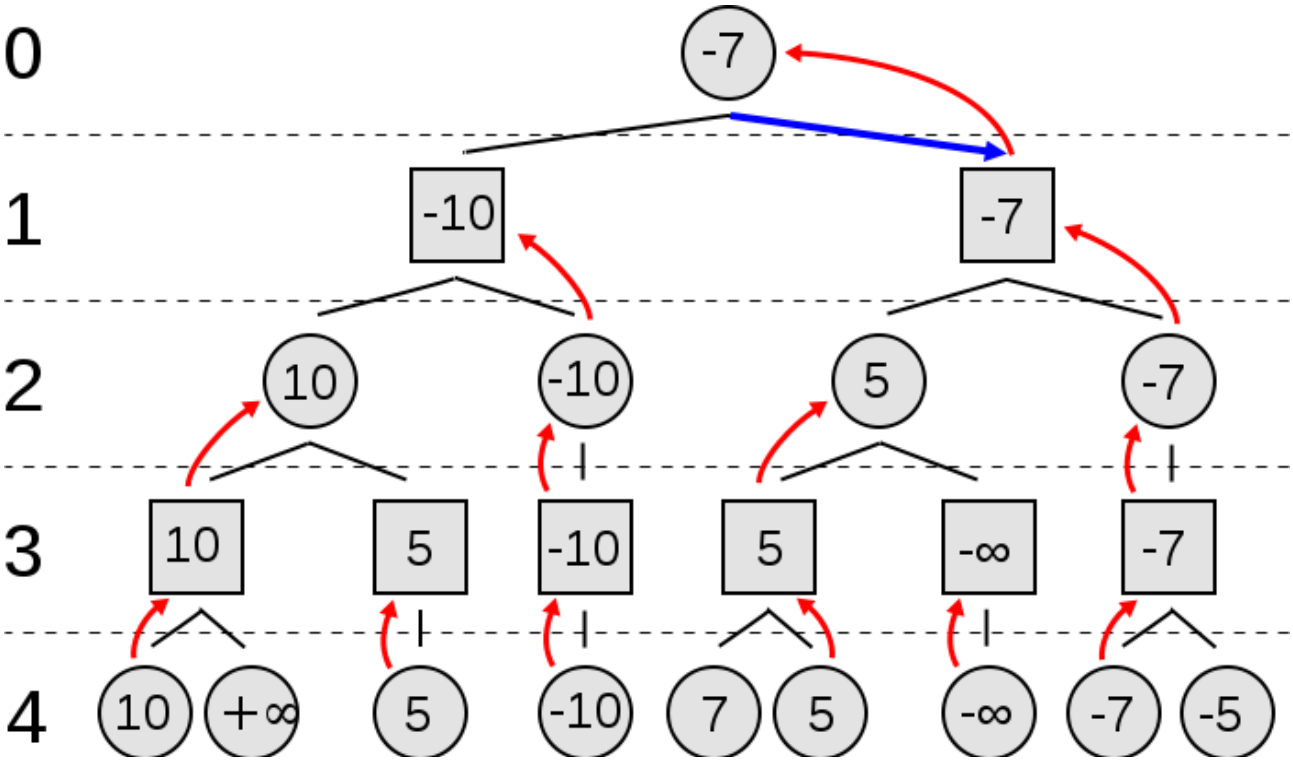
Minimax

Minimax je algoritmus rozhodovania používaný v teórií hier pre minimalizovanie možnej straty pre najhorší možný prípad (najväčšiu stratu). Je používaný pre hry hrané dvoma hráčmi, a uvádza:

- pre danú stratégiu 1. hráča, je najlepší možný výsledok 2. hráča V
- pre danú stratégiu 2. hráča, je najlepší možný výsledok 1. hráča -V

Inak povedané, minimax algoritmus garantuje 2. hráčovi výsledok V, bez ohľadu na stratégiu hráča 1.

Nasledujúci obrázok zobrazuje strom hry, na ktorý bol použitý minimax algoritmus. Kruhové nódy predstavujú hráča min a štvorcové hráča max.



Obrázok: minimax strom

Výhodou tohto algoritmu je nájdenie perfektnej hry, ale za cenu prehľadania všetkých uzlov.

Negascout

Negascout je nagamax algoritmus, ktorý je v niektorých prípadoch rýchlejší ako alfa-beta osekávanie. Funguje na podobnom princípe ako alfa-beta, ale spolieha sa na správne zoradenie listov v strome. Ak sú zoradené výhodne, môže byť rýchlejší o 10 a viac percent, ale ak je zoradenie náhodné je pomalší ako spomínané alfa-beta. To kvôli tomu, že hoci neprehľadá viac uzlov, prehľadá niektoré uzly viacnásobne.

Výhodou tohto algoritmu je zvýšenie efektívnosti oproti alfa-beta, ale za cenu potreby usporiadania uzlov.

MTD(f)

Mtd(f) je vylepšený minimax algoritmus, ktorý dosahuje najlepšie teoretické výsledky zo všetkých porovnávaných. Algoritmus vykonáva opakované alfa-beta prehľadávanie s nulovou veľkosťou hľadajúceho okna. Efektívnosť tohto postupu sa zabezpečuje transpozičnou tabuľkou, v ktorej sú uložené už prehľadané uzly.

Vysoká efektívnosť algoritmu je vyvážená jeho pamäťovou náročnosťou a zvýšenou réžiou na udržiavanie transpozičnej tabuľky.

2.2. Reversi klientská aplikácia (Prototyp)

2.2.1. Zadanie

Analyzujte možnosti riešenia stromu pre hru reversi pomcou algoritmu alfa beta osekávania a implementujte riešenie vo forme klientskej aplikácie pre systém Boinc.

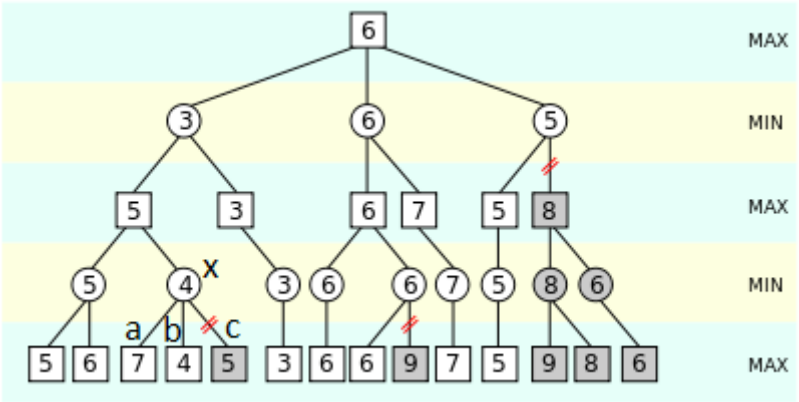
2.2.2. Analýza

Alogritmus alfa beta osekávania je vylepšením pre minimax algoritmus, pomocou ktorého môžeme odstrániť väčsie časti herného stromu a tým výrazne čas potrebný na prehľadávanie.

Osekávajú sa práve tie vetvy, ktoré ani potencionáne nemôžu ovlpyvniť výsledok algoritmu, preto vždy vracia výsledok, ktorý je rovnaký ako minimax.

Príklad: Na ťahu je biely hráč a strom prehľadávma do hĺbky 2. Vyberieme jeden z potencionálnych ťahov bieleho hráča (nazvime ho ťah1) a všetky možné odpovede čierneho hráča. Po analýze odpovedí zistíme, že po ťahu čierneho hráča môžeme dosiahnuť remízu. Potom vybiereme ďalší z potencionálnych ťahov bieleho hráča (ťah2). Po preskumání prvej odpovede čierneho hráča na ťah2 zistíme, že čierny hráč získa viac kameňov. Po tomto zistení už môžeme všetky ďalšie odpovede na ťah2 ignorovať, pretože s istotou vieme, že ťah1 je lepší, pretože odpoveď čierneho hráča mu umožní dosiahnuť najviac remízu. Ťah1 nám určil spodnú hranicu, teda minimálny najlepší výsledok, ktorý môžeme dosiahnuť a všetko pod ním môžeme ignorovať a odseknúť.

V prípade prehľadávania do hĺbky 3 a viac sa situácia komplikuje pridaním hornej hranice. Hornú hranicu musíme uvažovať kvôli tomu, ak sa v hĺbke tri vyskytuje príliš dobrá možnosť, ktorú súper v predošlom ťahu elimunuje vybraním vhodného ťahu. Dolná hranica jedného hráča je hornou hranicou druhého hráča.



Pri sekvenčom prehľadávaní možností zľava doprava pri uzle s hodnotou 4 môžme vylúčiť poslednú môžnosť a jej podstrom, pretože už máme lepšie riešenie. Inak povedané uzol C neovlivní hodnotu uzla X.

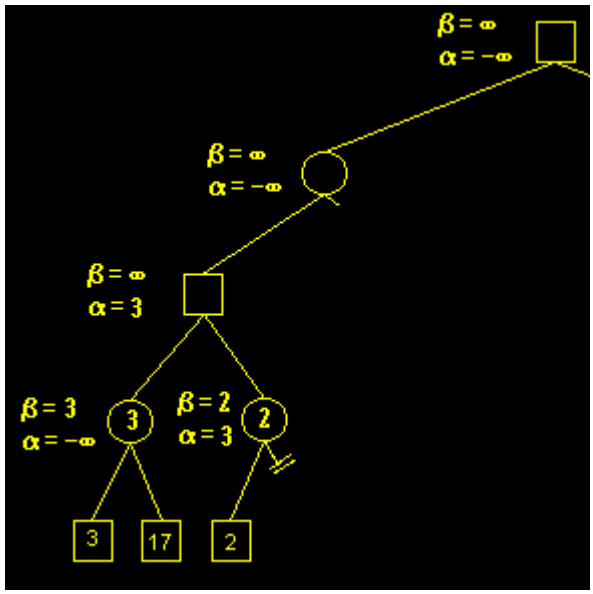
Alfa – aktuálna maximálna dolná hranica, v ktorej sa môže nachádzať výsledná hodnota. Na začiatku je nastavená na mínus nekonečno.

Beta – aktuálna minimálna horná hranica, v ktorej sa môže nachádzať výsledná hodnota. Na začiatku je nastavená na plus nekonečno.

Alfa a beta spolu tvoria okno, v ktorom sa môže nachádzať vrátená hodnota, teda alfa<=N<=beta.

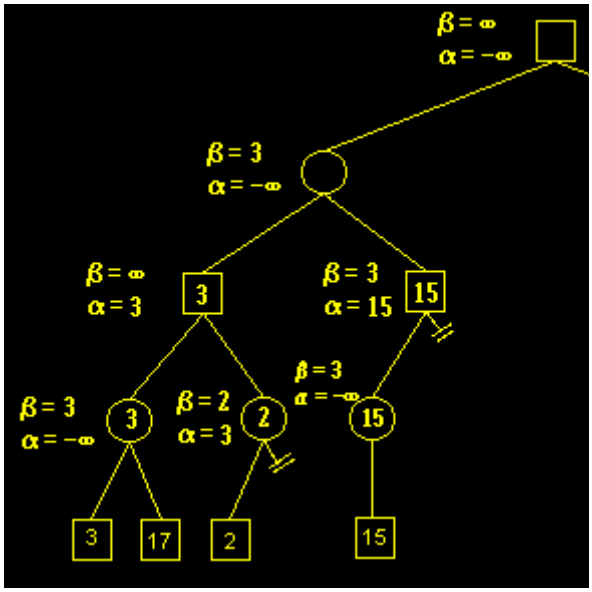
V minimaxe si MIN hráč vyberá vždy najmenšiu z hodnôt potomkov a preto aktualizuje betu. MAX hráč si naopak vyberá najväčsiu z hodnôt potomkov, preto aktualizuje alfu.

Príklad beta osekávanie v MIN vrchole s hodnotou 2 v hĺbke 3:



Obr. Beta osekávanie

Príklad alfa osekávanie v MAX vrchole s hodnotou 15 v hĺbke 2:



Obr. Alfa osekávanie

2.2.3. Návrh

Funkčné požiadavky na klientskú aplikáciu sú nasledovné:

1. Aplikácia musí vedieť spracovať vstupné súbory vygenerované generátorom.
2. Aplikácia musí vedieť vypočítať pridelený podstrom hry reversi pomocou alfa beta osekávania.
3. Aplikáci musí vedieť poslať výsledok na Boinc server vo formáte:

   stavCiernehoHraca_stavBielehoHraca_farbaHracaNaTahu

2.2.4. Implemetácia

Aplikáciu sme implementovali v programovacom jazyku Java.

Aplikácia využíva nasledovné triedy:

- Bitboard – trieda reprezentujúca hráčsku plochu. Poskytuje základné operácie ako generovanie dostupných ťahov, ich vykonanie a rotácie plochy.
- Search – trieda obsahuje metódy, ktoré reprezentujú rôzne algoritmy na prehľadávanie stromu riešení pre hru reversi. Obsahuje algoritmy Negascout, minimax, alphabeta.
- FileForDummies – trieda, ktorá zjednodušuje vytváranie vstupných a výstupných súborov
- Flipper – trieda, ktorá slúži na detekciu a vyfarbovanie políčok uzavretých protihráčom

2.2.5. Testovanie

Aplikáciu sme nasadili na riešenie problému reversi 4x4. Výsledky sa zhodovali s už zistenými hodnotami.

2.3. Vytvorenie projektu

1. Presuň sa do priečinka *HOME*/boinc/tools/
2. Spusti ./make_project [*MOZNOSTI*] project [*MENOPROJEKTU*]

   Hlavné možnosti:

   --*db_name* – názov databázy

   --*db_user* – uživateľské meno pre prihlásenie do databázy

   --*db_passwd* – heslo

   --*delete_prev_inst* – zmaže existujúci projekt s rovnakým názvom

   --*drop_db_first* – zmaže databázu existujúceho projektu s rovnakým názvom
3. Otvor *HOME*/projects/*MENOPROJEKTU*/*MENOPROJEKTU*.readme
4. Pokračuj podľa krokov v otvorenom súbore

2.3.1. Konfigurácia projektu

V priečinku projektu treba zmeniť nastavenia v súbore config.xml. Najprv treba nastaviť maximálny počet spustených aplikácií na jedno jadro.

```
<max_wus_in_progress>1</max_wus_in_progress>
```

Následne podľa potreby nastaviť validátor a asimilátor výsledkov. Oba programy musia byť umiestnené v priečinku *HOME*/projects/*MENOPROJEKTU*/bin.

```
<daemons>
  <daemon>
    <cmd>validator</cmd>
  </daemon>
  <daemon>
    <cmd>assimilator</cmd>
  </daemon>
...
```

2.3.2. Pridanie aplikácie

1. Na konci súbora project.xml nastavte meno aplikácie a užívateľsky prívetivé meno.

   …

```
</platfrom>
<app>
        <name>App1</name>
        <user_friendly_name>First Application</user_friendly_name>
  </app>
<app>
        <name>App2</name>
        <user_friendly_name>Second Application</user_friendly_name>
  </app>
</boinc>
```

2. Následne skopírujte spustiteľný súbor do priečinka *HOME*/projects/*MENOPROJEKTU*/apps/*MENOAPLIKACIE*/*CISLOVERZIE*/*MENOPLATFORMY*/

   CISLOVERZIE – ľubovolné číslo, musí však byť unikátne

   MENOPLATFORMY – zadefinované v project.xml
3. Presuňte sa do priečinka *HOME*/projects/*MENOPROJEKTU*
4. Spustite bin/update_versions
5. Spustite bin/stop && bin/start

Vytvorenie šablón pre aplikáciu

1. V priečinku *HOME*/projects/*MENOPROJEKTU*/templates vytvorte súbory *MENOAPLIKACIE*_in (vstupná šablóna) a *MENOAPLIKACIE*_out (výstupná šablóna).
2. Obsah vstupnej šablóny

```
<file_info>
  <number>0</number>
    //index vstupneho suboru zacina od 1
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
        //logicka adresa
        <copy_file/>
        //potrebne pre wrapper
```

```
    </file_ref>
  <rsc_fpops_est>36000000000000</rsc_fpops_est>
    //priblizny pocet potrebnych floating-point operacii
    <rsc_fpops_bound>150000000000000</rsc_fpops_bound>
    //maximalny pocet floating-point operacii
    <min_quorum>1</min_quorum>
    //pocet uloh pre jeden work unit
    <target_nresults>1</target_nresults>
    //pocet vysledkov ktore chceme dosiahnut
</workunit>
```

3. Obsah výstupnej šablóny

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>5000000</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>out</open_name>
    <copy_file/>
  </file_ref>
</result>
```

2.3.3. Vytvorenie práce

1. Spusti príkaz cp *VSTUPNYSUBOR* `bin/dir_hier_path *MENOSUBORU*`
2. Spusti bin/create_work –appname *MENOAPLIKACIE MENOSUBORU*
   VSTUPNYSUBOR – cesta k vstupnému súboru
   MENOSUBORU – názov pripraveného vstupného súboru

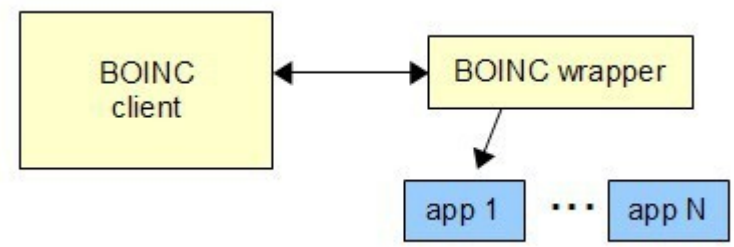
2.3.4. Verziovanie aplikácie

Aplikáciu je nutné verziovať pri každej zmene akéhokoľvek súboru alebo po pridaní aplikácie pre inú platformu. Postup pri verziovaní je nasledovný:

1. Skopírujte nové súbory do priečinka *HOME*/projects/*MENOPROJEKTU*/apps/*MENOAPLIKACIE*/*CISLOVERZIE*+1/*MENOPLATFORMY*/
2. Presuňte sa do priečinka *HOME*/projects/*MENOPROJEKTU*
3. Spustite bin/update_version

2.3.5. Použitie wrapper aplikácie

Pri použití iných programovacích jazykov ako C/C++ alebo FORTRAN nie je možné priamo využívať BOINC client api, je nutné použitie wrapper aplikácie, ktorá nepriamo sprostredkúvava komunikáciu medzi BOINC klientom a aplikáciou, ako aj aplikáciu spúšťa.



Nasadenie takejto aplikácie na BOINC server je podobné ako nasadenie klasickej, natívnej aplikácie, avšak je tu niekoľko zmien:

- Je potrebné do priečinku k samotnej aplikácie pridať wrapper aplikáciu.
- Je potrebné pridať logický súbor job.xml, ktorý slúži ako vstupný súbor pre wrapper aplikáciu.
- Je potrebnú upraviť súbor version.xml, ktorý opisuje súbory, ktoré sú súčasťou aplikácie.

Štruktúra súboru **job.xml**:

```
<job_desc>
    <task>
        <application>worker</application>
        [ <stdin_filename>stdin_file</stdin_filename> ]
        [ <stdout_filename>stdout_file</stdout_filename> ]
        [ <stderr_filename>stderr_file</stderr_filename> ]
```

```
        [ <command_line>--foo bar</command_line> ]
        [ <weight>X</weight> ]
        [ <checkpoint_filename>filename</checkpoint_filename> ]
        [ <fraction_done_filename>filename</fraction_done_filename> ]
        [ <exec_dir>dirname</exec_dir> ]
        [ <multi_process/> ]
        [ <setenv>VARNAME=VAR_VALUE</setenv> ]
        [ <daemon/> ]
        [ <append_cmdline_args/> ]
        [ <time_limit>X</time_limit> ]
    </task>
    [ other <task>s ]
    [
    <unzip_input>
       <zipfilename>foo.zip</zipfilename>
       ...
    </unzip_input>
    ]
    [
    <zip_output>
       <zipfilename>foo.zip</zipfilename>
       <filename>regexp</filename>
       ...
    </zip_output>
    ]
</job_desc>
```

Štruktúra súboru **version.xml**:

```
<version>
  <file>
    <physical_name>PNAME</physical_name>
    [ <main_program/> ]
    [ <copy_file/> ]
    [ <logical_name>LNAME</logical_name> ]
    [ <gzip/> ]
    [ <url>URL0</url> ]
    [ <url>URLn</url> ]
  </file>
  ... more <file>s

  [<dont_throttle/>]
  [<file_prefix>X</file_prefix>]
  [<needs_network/>]
  [<is_wrapper/>]
</version>
```

Na všetky súbory je nutné použiť tag <copy_file/>, inak aplikácia s použitím wrapperu nebude fungovať správne.

2.4. Generovanie stavov

2.4.1. Zadanie

Analyzujte možnosti reprezentácie stavov pre hru Reversi. Implementujte jednoduchý generátor ťahov pre plochu veľkosti 6x6 a 8x8. Nad týmto generátorom vytvorte genérator vstupných súborov pre klientskú časť.

2.4.2. Analýza

V súčasnosti existujú dve hlavné reprezentácie hracej plochy.

- String reprezentácia
- Bitboard reprezentácia

2.4.3. String reprezentácia

Hráčska plocha je reprezentovaná znakmi v reťazci, kde každý znak zodpovedá práve jednému políčku plochy. Každý hráč má v tomto prípade pridelený znak, ktorý ho reprezentuje. Prázdne políčka sú taktiež reprezentované práve jedným znakom. Celú hráčsku plochu je teda možné uložiť do jednej premennej.

2.4.4. Bitboard reprezentácia

Stav v hre je reprezentovaný pomocou bitboard-u. Bitboard sa skladá z troch 64-bitových čísel. Jedného pre čierneho hráča, jedného pre bieleho hráča a jedného pre prázdne políčka.

2.4.5. Zhodnotenie

Pri string reprezentácií dochádza k mrhaniu pamäťových aj výpočtových prostriedkov. Jedno políčko v tomto prípade zaberá 2B a jednou operáciou dokážeme vždy získať iba jeden dostupný ťah.

Bitovými operáciami vieme zistiť všetky dostupné ťahy paralelne. Celá plocha zaberá v tomto prípade 3 * 8B.

2.4.6. Návrh

Na základe analýzy sme sa rozhodli používať bitboard reprezentáciu. Identifikovali sme potrebu implementácie dvoch nástrojov:

- nástroj na generovanie vstupných súborov
- nástroj na vytvorenie work-unit-ov zo vstupných súborov

Požiadavky na nástroje:

- generovanie vstupných súborov s možnosťami

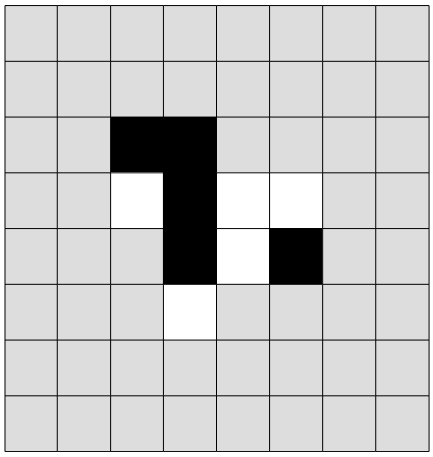
- rozdelenia generovania na dávky
- určenia cieľovej hĺbky
- určenia výstupného priečinka

- vytváranie work-unit-ov pre všetky súbory v danom priečinku

2.4.7. Implementácia

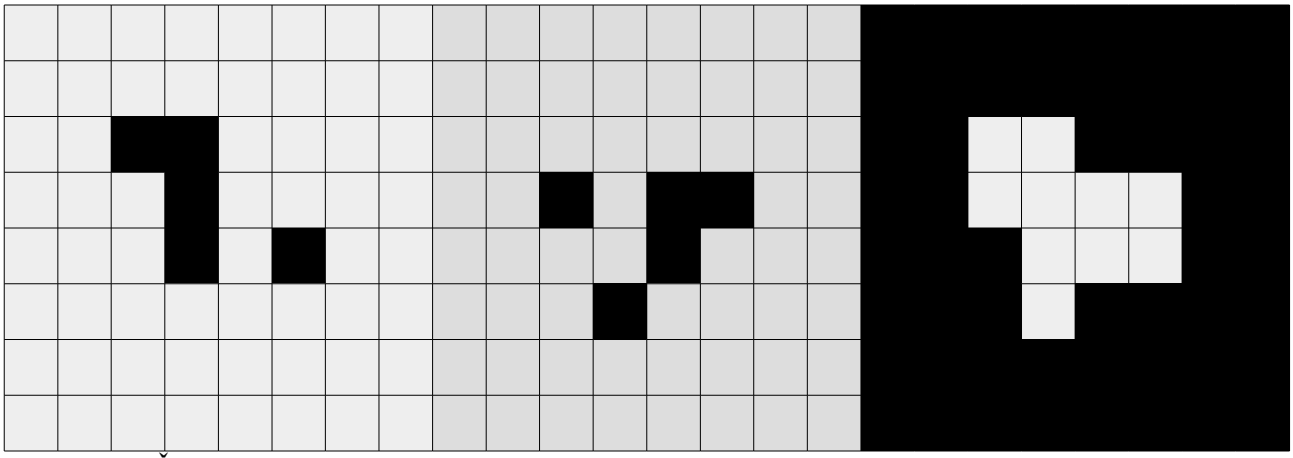
Postup generovania

Uvažujme nasledujúci stav hracej plochy. Čierne políčka v tabuľke reprezentujú čierneho hráča, biele bieleho hráča a šedé reprezentujú prázdne políčka.



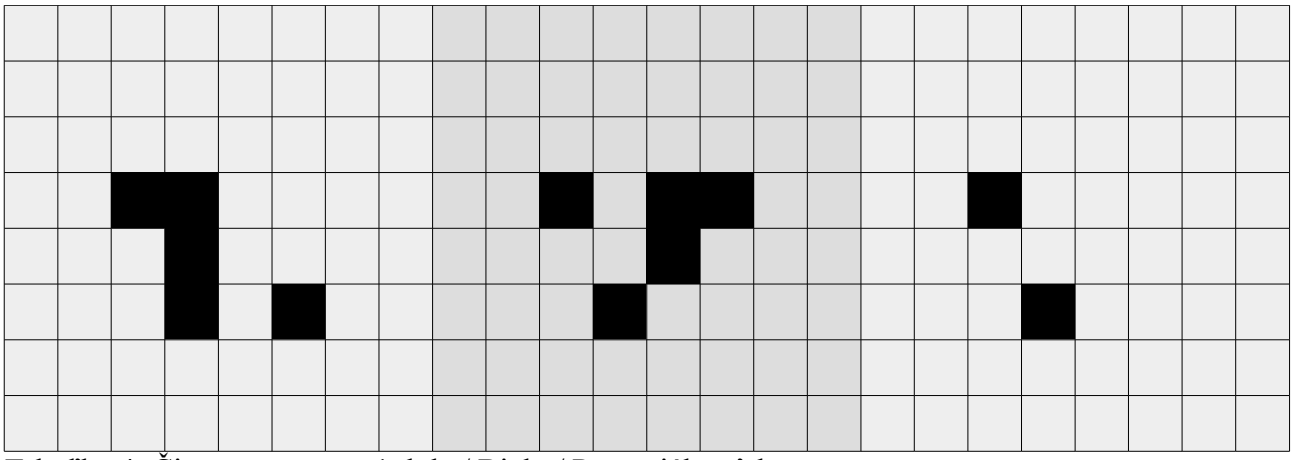
Tabuľka 2: Hracia plocha

Bitboard reprezentácia tohto stavu by vyzerala nasledovne (čierna farba = bit nastavený na 1, šedé = 0).



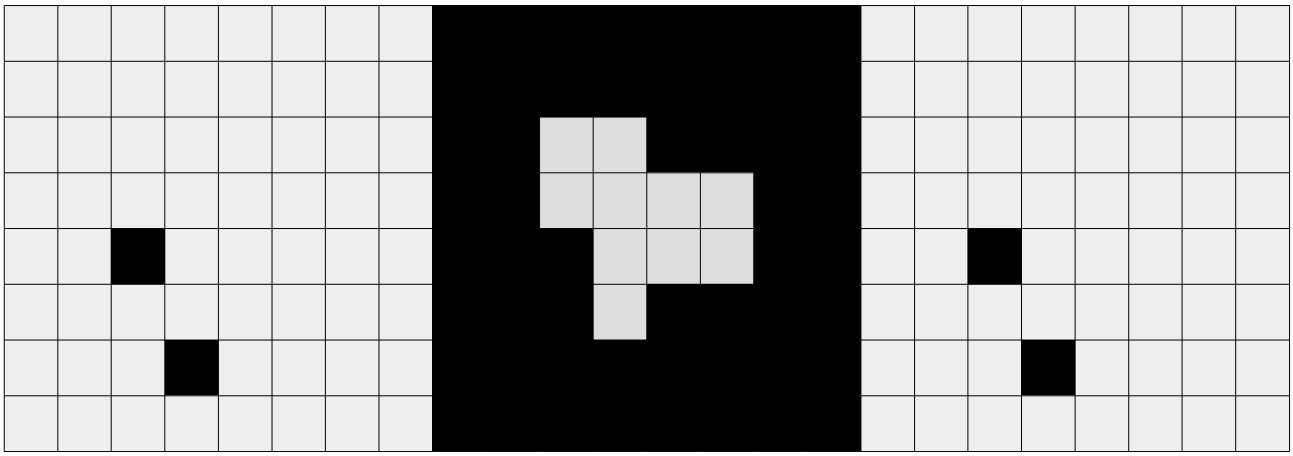
Tabuľka 3: Čierny / Biely / Prázdne

Množinu potenciálnych ťahov získame posunom plochy čierneho hráča o jeden riadok dole a následným vykonaním operácie AND s plochou bieleho hráča. Výsledkom je tretia plocha.



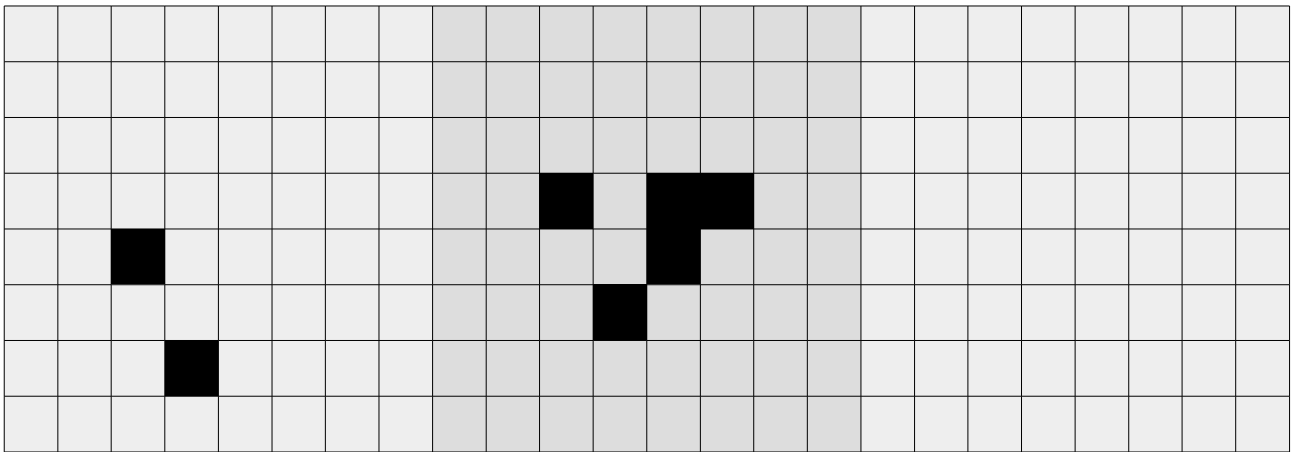
Tabuľka 4: Čierny - posun o 1 dole / Biely / Potenciálne ťahy

Následne posunieme plochu potenciálnych ťahov znovu o jeden riadok dole. Vykonáme bitový AND s plochou prázdnych políčok a tak získame legálne ťahy.



Tabuľka 5: Posun potenciálnych o 1 dole / Prázdne / Legálne ťahy

Vykonaním operácie AND medzi plochou potenciálnych ťahov a plochou bieleho hráča získame ďalšie potenciálne ťahy.



Tabuľka 6: Posun potenciálnych o 1 dole / Biely / Nové potenciálne ťahy

Takýmto spôsobom posunieme plochu celkom osemkrát. Celý tento postup opakujeme vo všetkých ôsmich smeroch.

Implementácia posunov bitboard-u:

- *hore* - bitový posun doľava o 8
- *dole* - bitový posun doprava o 8
- *vpravo* - bitový posun doprava o 1
- *vľavo* - bitový posun doľava o 1
- *diagonála 1* - bitový posun doprava o 9
- *diagonála 1 opačne* - bitový posun doľava o 9
- *diagonála 2* – bitovy posun doprava o 7
- *diagonála 2* - opačne bitový posun doľava o 7

2.4.8. Generátor vstupných súborov

java -jar generator.jar *POCETDAVKA MAXHLBKA VYSTUPP*

POCETDAVKA – long – počet vygenerovaných súborov v jednej dávke

MAXHLBKA – long – maximálna hĺbka, do ktorej sa vstupy majú generovať

VYSTUPP – string – cesta k priečinku, kam sa majú vstupné súbory generovať

Generovať sa dá postupne po *POCETDAVKA* súboroch. Program si pamätá, kde skončil. V pripade generovania odznovu je nutné zmazať vo výstupnom priečinku queue.ser súbor.

2.4.9. Formát vstupného súboru

Vstupný súbor pozostáva z týchto riadkov:

- string reprezentácia long-u zodpovedajúceho ploche čierneho hráča
- string reprezentácia int-u zodpovedajúceho počtu políčok čierneho hráča
- string reprezentácia long-u zodpovedajúceho ploche bieleho hráča
- string reprezentácia int-u zodpovedajúceho počtu políčok bieleho hráča
- string reprezentácia int-u, určujúca ktorý hráč je na ťahu
- string reprezentácia int-u zodpovedajúca hĺbke

2.4.10. Jednoduchý generátor práce

Generátor práce sme implmentovali vo forme jednoduchého bash skriptu, ktorý vytvára pre každý súbor v priečinku “generator/6x6INs“ práve jeden work-unit.

```
app_name="6x6"
for file in generator/6x6INs/*; do
        name=$(basename $file)
        cp $file `bin/dir_hier_path $name`
        bin/create_work -appname $app_name -wu_name $name $name
done
```

2.4.11. Testovanie

Správnosť vygenerovaných stavov sme určili porovnaním s výstupmi existujúcej implementácie hry Reversi s názvom EDAX, ktorá sa dodnes úspešne zúčastňuje mnohých súťaží a je vyvíjaná od roku 2004 dodnes.

2.5. Zhodnotenie šprintu

Vytvorili sme serverovú časť projektu, ktorá dokáže generovať stavy hry reversi do určitej hĺbky a následne vytvoriť workunity pre klientov. Každý workunit obsahuje unikátny stav hry v danej hĺbke, v ktorého prehľadávaní už pokračuje klient.

Analyzovali a implementovali sme taktiež možné algoritmy pre riešenie hry reversi na klientskej časti. Tieto sme porovnali medzi sebou z hľadiska časovej a pamäťovej náročnosti v kontexte distribuovaného počítania. Na základe získaných výsledkov sme sa rozhodli ďalej používať algoritmus Alfa-Beta osekávania.

3. Tretí šprint

V treťom šprinte sme sa zamerali na vytvorenie a nasadenie funkčného prototypu aplikácie na server a klient. Serverová časť vygeneruje strom riešenia hry reversi do určitej hĺbky a vytvorí workunity, ktoré sú následne distribuované klientom a ty prehľadávaním do hĺbky hľadajú riešenia.

Určili sme si nasledovné ciele:

- Dokončiť serverovú aplikáciu a nasadiť ju na server
- Upraviť nami vytvorené prototypy aby pracovali ako klientské aplikácie a nahrať ich na server za účelom ich sťahovania klientmi.
- Spustiť počítanie a nájsť riešenie hry reversi 6x6
- Otestovať a zmerať časy a úspešnosť vyššie spomenutých bodov

3.1. Asimilátor

3.1.1. Zadanie

Analyzujte možnosti a vytvorte asimilátor, ktorý bude vrátené výsledky ukladať do databázy.

3.1.2. Analýza

Výsledok vrátený klientom vo forme súboru uloženého v *upload* priečinku môžeme ďalej spracovávať pomocou programov nazývaných asimilátory. Asimilátory sa starajú o napríklad o premiestnenie súborov z Boinc upload priečinka, prípadne o ďalšie spracovávanie prijatých údajov a ich ukladanie do databázy.

Asimilátor môže byť v Boinc systéme vytvorený pomocou jednej štandardnej funkcie, ktorú systém volá po každom prijatí výsledku. Hlavička funkcie je v tvare:

int assimilate_handler(WORKUNIT& wu, vector<RESULT>& results, RESULT& canonical_result)

Asimilátor potom môžeme spúšťať ako Boinc daemon, jeho zapísaním do konfiguračného súboru *config.xml,* ktorý sa nachádza v projektovom priečinku. Formát zápisu je nasledovný:

<daemon>

<cmd> moj_asimilator -app meno_aplikacie </cmd>

</daemon>

Príkaz má nasledovné parametre:

`--app meno` – meno aplikácie, pre ktorú sa má spúšťať

`[--mod N R]` – asimiluje iba výsledky pre ktoré platí mod(id, N)=R. Takto môžeme spúšťať viac asimilátorov súčasne.

`[--dont_update_db]` – neoznačuje úlohy ako spracované, zanecháva ich v pôvodnom stave vhodné na testovacie účely.

Pre prvú časť nášho projektu v ktorej implementujeme aplikáciu pre vyriešenie stromu hry reversi, sme sa rozhodli zostrojiť vlastný asimilátor, ktorý bude vrátené vrcholy stromu zapisovať do projektovej databázy.

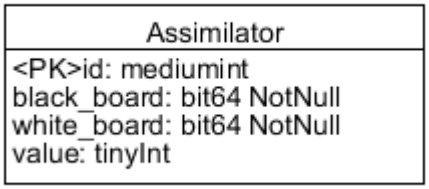
3.1.3. Návrh

Po analýze sme definovali tieto funkčné požiadavky na asimilátor:

1. Asimilátor musí vygenerovať riadky zodpovedajúce pre workunity – tak aby sme vedeli identifikovať, ktorý workunit zodpovedá ktorému riadku.
2. Asimilátor musí množinu vrátených výsledkov zo súborov od klientov vložiť do zodpovedajúcich riadkov.
3. Asimilátor musí spracovať súbor vo formáte:

   stavCiernehoHraca_stavBielehoHraca_farbaHracaNaTahu

Štruktúra tabuľky pre záznam výsledkov je nasledovná:



Obr. Štruktúra tabuľky pre asimilátor

Dátové typy zodpovedajú databáze MySQL.

Opis stĺpcov tabuľky:

- `id` – identifikátor, primárny kľúč tabuľky, veľkosť zvolená ako 3 B číslo, čo umožní vygenerovať až 16 777 216 workunitov.
- `black_board` – predstavuje stav hry z pohľadu čierneho hráča. Ako typ zvolený BIT(64), pretože pri veľkosti poľa 8×8 je možné obsadiť najviac 64 políčok.
- `white_board` – predstavuje stav hry z pohľadu bieleho hráča. Ako typ zvolený BIT(64), pretože pri veľkosti poľa 8×8 je možné obsadiť najviac 64 políčok.
- `value` – predstavuje výsledok riešenia pre počiatočné hracie pole vyjadrené pomocou stĺpcov `black_board` a `white_board`. Ako dátový typ je zvolené 1 B číslo, čo predstavuje najviac 256, pretože výsledný rozdiel medzi čiernym a bielym hráčom nemôže byť väčší ako 64.

Približná veľkosť jedného záznamu tabuľky[3]: 3 B + 2 × ((64+7) / 8) B + 1 B = 20 B.

3.1.4. Testovanie

Asimilátor sme testovali pomocou nami vygenerovaných testovacích súborov, ktoré formátom zodpovedajú výsledkom, ktoré budeme dostávať od klientov. Testovanie prebehlo úspešne, generovanie tabuľky aj ukladanie výsledkov do databázy prebehlo bez problémov.

3 http://dev.mysql.com/doc/refman/5.5/en/storage-requirements.html

3.2. Inštalácia wiki

3.2.1. Zadanie

Nainštalovať určitu wiki na naš tímovy server

3.2.2. Inštalácia

Prebiehala pomocou nástroja apt, vytvorenie symbolickej linky a nastavenie wiki

1. apt-get install mediawiki php5-gd php5-xcache php-pear
2. vytvoriť symbolicku linku na mediawiki do document root
3. ist na www.stránka.sk/wiki/config/index.php
4. vyplniť údaje na stránke

3.2.3. Spustenie

Po vyplneni údajov je wiki pripravená na používanie

3.2.4. Testovanie

Po nasmerovaní prehliadača na stránku projektu /wiki sa zobrazila stránka wiki

3.3. Príprava prototypu na nasadenie

3.3.1. Zadanie

Upraviť prototyp na riešenie hry Reversi 6x6, aby bol schopný prevádzky na platforme BOINC.

3.3.2. Analýza

Zadanie údajov pre program je na platforme BOINC riešené pomocou vstupných súborov. Vstupné súbory majú názvy logické a fyzické, t.j. BOINC server pošle BOINC klientskej aplikácií súbor s unikátnym názvom, ktorý ho identifikuje a následne ho premenuje na logický názov, takže náš program bude pracovať s logickým názvom, ktorý je vždy rovnaký.

Vytvorenie výstupného súboru, ktorý je následne odoslaný na server je riešené podobne, t.j. zapíšeme výsledok do súboru, s logickým názvom, ktorý je vždy rovnaký, následne ho klientská aplikácia premenuje na fyzický názov a odošle na server.

Keďže naša aplikácia je pustená prostredníctvom wrapper aplikácie, nemáme priamy prístup k BOINC api, takže ďalšie funkcie ako počet vyriešených percent projektu a checkpointing sú riešené pomocou pomocných súborov.

3.3.3. Návrh

Keďže prototyp neobsahuje funkcionalitu na načítavanie vstupných údajov a ani na zápis výstupných údajov zo súboru, je potrebné túto funkcionalitu pridať.

3.3.4. Implementácia

Boli pridané triedy, ktoré umožňujú jednoduchý zápis a čítanie súborov.

Ďalej boli pridané konštanty, ktoré reprezentujú názvy súborov.

3.3.5. Testovanie

Testovanie prebehlo formou kontroly načítania niekoľkých súborov a ich vypísaním na obrazovku.

Podobne prebehlo aj testovanie zápisu do súboru.

3.3.6. Zhodnotenie

Podarilo sa nám upraviť a otestovať prototyp. Výsledná aplikácia funguje spoľahlivo. Avšak v budúcnosti bude potrebné doimplementovať meranie dokončenej časti výpočtu, ako aj pridanie checkpointingu.

3.4. Zhodnotenie šprintu

V treťom šprinte sme sa zamerali na nasadenie aplikácii na server, táto úloha sa nám úspešne podarila a začali sme distribuovane počítať riešenie hry reversi 6x6. Generovanie práce prebiehalo pomocou shell skriptu spusteného manuálne, v projekte nám ešte chýba vytvoriť scheduler.

V čase končenia 3. šprintu sme distribuovane vypočítali 121 workunitov.

4. Big picture

BOINC platforma pre distribuované počítanie skrýva v sebe potenciál. V prvých 3 šprintoch sme sa snažili položiť základy nasadenia platformy pre potreby fakulty. Úspešne sme spustili BOINC server na tímovom serveri.

Hlavnou úlohou bolo overiť fungovanie na určitej úlohe. Vedúci tímu Ing. Peter Lacko, PhD. vyjadril na prvom stretnutí návrh vyriešiť symetrickú hru Reversi s veľkosťou hracej plochy 8×8. Riešenie perfektnej hry nebolo vypočítané. Vyriešenie takejto úlohy je aj v teoretickej rovine veľmi náročné vzhľadom na pamäťové nároky celkového návrhu a celkového počtu všetkých stavov hry. Predtým, ako sa pustíme do riešenia tohto problému, pokúsili sme sa overiť princíp riešenia na zjednodušenej verzii. Rozhodli sme sa, že pre prvý prototyp znížime rozmery hracej plochy na 6×6. Výhodou zjednodušenia úlohy je, že riešenie už bolo vypočítané, preto vieme jasne overiť, či výpočet prostredníctvom distribuovaného výpočtu bude správny.

Na tímovom serveri sa nám podarilo spustiť prvý funkčný prototyp a vygenerovať 52 127 výpočtových úloh. Klientské úlohy sa úspešne posielajú členom tímu, ktorí poskytli svoj procesorový výkon. Za prvý týždeň od spustenia prototypu sme získali 120 súborov s výsledkami. Do riešenia čiastkových úloh nezískavame nových dobrovoľníkov. Aplikáciám pre klientom momentálne chýba niekoľko vlastností, ktoré by ju činili používateľsky prístupnou. Nie je možné prerušiť výpočet ukončením aplikácie alebo vypnutím počítača a následné pokračovanie od posledne vypočítaného stavu.

Aplikáciu sme implementovali v programovacom jazyku Java, pre neexistujúce BOINC API pre tento jazyk sa nám nepodarilo využívať možnosti, ktoré BOINC platforma poskytuje . Avšak pre platformu BOINC existuje API pre C/C++ aplikácie, ktoré poskytujú všetky funkcie. V ďalších krokoch sa zameriame aj na doplnenie tejto funkcionality, pretože bez získania ďalších dobrovoľníkov nie je v kapacite tímu vyriešiť 52 127 čiastkových úloh v rozumnom čase. Pre komplikovanejší problém pri hracej ploche o rozmeroch 8×8 počet úloh bude niekoľkotisicnásobne väčší.

5. Štvrtý šprint

V štvrtom šprinte sme začali skúmať možnosti iných programovacích jazykov pre programovanie aplikácii. K tomuto nás viedol fakt, že boinc API je v C/C++ a pre Javu neexistuje binding. Ďalším dôvodom zmeny jazyka je, že klientská aplikácia v Jave sa javila ako pomalá a to, že v Jazyku C je napísaný kód solvra Edax, ktorý je vysoko optimalizovaný a rýchly.

Definovali sme si následovné ciele:

- Upraviť Edax aby ho bolo možné použiť pre reversi 6x6
- Porovnať časovú náročnosť aplikácii v jazyku C a jazyku Java
- Upraviť a zrýchliť súčasný prototyp v Jave

Dokumentácia:

Úprava edaxu

5.1. Porovnanie používania jazykov C a Java,

5.1.1. Zadanie

Zvážte možnosti klientských aplikácií implementovaných v jazyku C a Java.

5.1.2. Typ úlohy

Analytická

5.1.3. Analýza

Počas riešenia projektových úloh sme sa stretli s viacerými problémami pri programovaní v jazyku Java. Prvý hlavný problém spočíval v nemožnosti využitia Boinc API funkcií priamo v zdrojových kódoch jazyka Java. Následne sme odhalili viaceré problémy pri realizácii algoritmov pre hru Reversi. Kvôli garbage collectoru mala klientská aplikácia vysoké nároky na operačnú pamäť aj procesorový čas. Priemerne potreboval jeden proces 2 GB operačnej pamäte.

Tento problém sme sa rozhodli vyriešiť prechodom na programovací jazyk C. Boinc API je napísaný v jazyku C a poskytuje binding pre jazyky C, C++ a Fortran.

Vďaka tomuto môžeme priamo využívať výhody Boinc systému ako checkpointing. Checkpointing je pre náš projekt obvzlášť dôležitý, pretože je predpoklad, že pri riešení stromu hry Reversi o veľkosti pola 8x8 budú jednotlivé workunity bežať dlhšiu dobu. V prípade, že by systém spadol bez použitia checkpointov, klient by mohol stratiť hodiny

práce.

Taktiež sa nám na základe predbežných testov javí aplikácia napísaná v jazyku C niekoľkonásobne rýchlejšia, ako aplikácia napísaná v Jave. Na serveri trval výpočet jedneho workunitu v Jave približne ~4 hodiny, v jazyku C pomocou Edaxu trvalo vyriešiť celé reversi 6x6 ~25 minút.

5.1.4. Výsledok

Na základe vyššie uvedených argumentov sme sa rozhodli presunúť vývoj hlavnej klientskej aplikácie do programovacieho jazyka C. Pri jazyku Java preskúmame možnosti optimalizácie.

5.2. Wrapper aplikácie

5.2.1. Zadanie

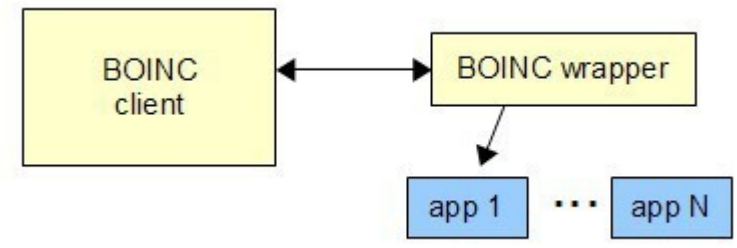
Analyzujte spúšťania aplikácií na platforme BOINC napísaných v inom programovacom jazyku. Pre tento projekt analyzujte programovací jazyk Java.

5.2.2. Typ úlohy

Analytická

5.2.3. Analýza

Pri použití iných programovacích jazykov ako C/C++ alebo FORTRAN nie je možné priamo využívať BOINC client api, je nutné použitie wrapper aplikácie, ktorá nepriamo sprostredkúva komunikáciu medzi BOINC klientom a aplikáciou, ako aj aplikáciu spúšťa.



Nasadenie takejto aplikácie na BOINC server je podobné ako nasadenie klasickej, natívnej aplikácie, avšak je tu niekoľko zmien:

- Je potrebné do priečinku k samotnej aplikácie pridať wrapper aplikáciu.

- Je potrebné pridať logický súbor job.xml, ktorý slúži ako vstupný súbor pre wrapper aplikáciu.
- Je potrebnú upraviť súbor version.xml, ktorý opisuje súbory, ktoré sú súčasťou aplikácie.

Štruktúra súboru **job.xml**:

```
<job_desc>
    <task>
        <application>worker</application>
        [ <stdin_filename>stdin_file</stdin_filename> ]
        [ <stdout_filename>stdout_file</stdout_filename> ]
        [ <stderr_filename>stderr_file</stderr_filename> ]
        [ <command_line>--foo bar</command_line> ]
        [ <weight>X</weight> ]
        [ <checkpoint_filename>filename</checkpoint_filename> ]
        [ <fraction_done_filename>filename</fraction_done_filename> ]
        [ <exec_dir>dirname</exec_dir> ]
        [ <multi_process/> ]
        [ <setenv>VARNAME=VAR_VALUE</setenv> ]
        [ <daemon/> ]
        [ <append_cmdline_args/> ]
        [ <time_limit>X</time_limit> ]
    </task>
    [ other <task>s ]
    [
    <unzip_input>
       <zipfilename>foo.zip</zipfilename>
       ...
    </unzip_input>
    ]
    [
    <zip_output>
       <zipfilename>foo.zip</zipfilename>
       <filename>regexp</filename>
       ...
    </zip_output>
    ]
</job_desc>
```

Štruktúra súboru **version.xml**:

```
<version>
  <file>
    <physical_name>PNAME</physical_name>
    [ <main_program/> ]
    [ <copy_file/> ]
    [ <logical_name>LNAME</logical_name> ]
    [ <gzip/> ]
    [ <url>URL0</url> ]
    [ <url>URLn</url> ]
  </file>
```

```
  ... more <file>s

  [<dont_throttle/>]
  [<file_prefix>X</file_prefix>]
  [<needs_network/>]
  [<is_wrapper/>]
</version>
```

Na všetky súbory je nutné použiť tag <copy_file/>, inak aplikácia s použitím wrapperu nebude fungovať správne.

5.2.4. Zhrnutie

BOINC wrapper poskytuje možnosti spúšťania binárnych súborov nepoužívajúcich BOINC API a vytvorených v inom programovacom jazyku. Podporované jazyky sú Java, Python.

5.3. Inštalácia BOINC servera

5.3.1. Zadanie

Nainštalovať BOINC server.

5.3.2. Typ úlohy

Inštalačná.

5.3.3. Inštalácia

Boinc server sa dá inštalovať viacerými spôsobmi:

- Nainštalovanie upravenej distribúcie Debianu, ktorá už má predinštalovaný BOINC server a je pripravená na používanie. Táto distribúcia sa dá stiahnuť na BOINC stránke[4].
- Použitie balíka, ktorý poskytuje zdroje softvéru danej distribúcie, ak v zdrojoch softvéru existuje balík boinc-server (alebo podobný), tak je tento spôsob doporučený.
- Vlastnou kompiláciou boinc servera.

V našom projekte sme sa rozhodli pre poslednú možnosť a to vlastnú kompiláciu.

Postup inštalácie

1. Nastavenie systému:

4 http://boinc.berkeley.edu/trac/wiki/VmServer

1.1. Inštalácia potrebných balíkov:

```
m4 make dh-autoreconf pkg-config git vim(alebo nano prípadne iný
editor) libapache2-mod-php5 mysql-server-5.1 libmysqlclient-dev
php5-mysql php5-cli php5-gd phpmyadmin python python-mysqldb libssl-dev
```

1.2. Vytvorenie nového používateľa, pod ktorým bude boinc spúšťať úlohy:

```
usermod -G -a boincadm www-data
```

1.3. Nastavenie MySQL servra:

```
$ mysql -h localhost -u root -p
> GRANT ALL ON *.* TO 'boincadm'@'localhost';
> SET PASSWORD FOR 'boincadm'@'localhost'='';
```

2. Stiahnutie zdrojových súborov boinc-u

```
git clone git://boinc.berkeley.edu/boinc-v2.git boinc-src
```

3. Kompilácia a inštalácia

```
$ cd ~/boinc-src
$ ./_autosetup
$ ./configure --disable-client –disable-manager
$ make
```

Po vykonaní týchto krokoch je boinc server nainštalovaný v priečinku boinc-src, v priečinku tools sú potrebné binárky a skripty na vytvorenie projektu.

5.3.4. Spustenie

BOINC server sa spúšťa pomocou skriptov z priečinka boinc-src/tools

5.3.5. Testovanie

Testovanie úspešnosti inštalácie servera spočíva v spustení testovacieho projektu.

Postup:

1. Vytvorenie projektu:

```
cd boinc-src/tools
./make_project --test_app NAZOV_PROJEKTU
```

2. Nastavenie projektu:

```
cd ~
cd NAZOV_PROJEKTU
cat NAZOV_PROJEKTU.readme
postupovať podľa inštrukcii v readme
```

3. Po vykonaní všetkých inštrukcii by mal byť projekt funkčný. Funkčnosť sa overí pripojením klienta na adresu projektu.
4. Inštalácia BOINC klienta na PC
5. Pridanie nového projektu v klientovi, adresa testovacieho projektu je v tvare:

   http://adresa_servera/NAZOV_PROJEKTU

6. Ak je všetko nainštalované a nastavené správne, tak boinc client by mal začať dostávať workunity a začať pracovať.

5.4. Zhodnotenie šprintu

V štvrtom šprinte sme analyzovali možnosti jazyka C. Dospeli sme k záveru, že vzhľadom na zistené výhody a nevýhody používaniu Javy v našom projekte bude použitie jazyka C vhodnejšie.

6. Piaty šprint

V piatom šprinte sme si naplánovali revíziu práce štvrtého šprintu a dokončenie úloh, ktoré sa nestihli v štvrtom šprinte.

Koncom piateho šprintu sme si dali za cieľ mať funkčnú aplikáciu pre riešenie hry reversi 6x6 pomocou programu Edax.

6.1. Generátor vstupných súborov pre Edax

6.1.1. Zadanie

Analyzovať a preskúmať kód Edax-u. Navrhnúť úpravy potrebné na úpravu Edax-u aby slúžil ako generátor vstupných súborov, z ktorých je možné vytvárať workunity.

6.1.2. Typ úlohy

Analytická

6.2. Analýza

Edax je naprogramovaný v jazyku C. Je to vysoko optimalizovaný kód s množstvom hardcodami funkcii, ktoré každá vykonáva jednú konkrétnu úlohu.

Edax vyžaduje vstupné súbory v FEN formate, tento formát má nasledovnú štruktúru:

V riadku informácie o hracej ploche, každá informácia oddelená ;

- Reprezentácia bitboardu
    - `-` - znazoňuje prázdne poličko
    - O – poličko obsadené jedným hráčom
    - X – políčko obsadené druhým hráčom
- X alebo O – určuje ktorý hráč je na ťahu
- Ťah : ohodnotenie; - zoznam ťahov s ohodnotením

Generovanie stavov pomocou Edaxu si vyžaduje značnú úpravu. Upravený Edax pre riešenie hry reversi 6x6 používa aspiration_serach definovaný v súbore root.c, tento search treba upraviť aby prehľadávanie skončil v určitej hĺbke a v tejto hĺbke ukladal uzly do súboru.

Ohodnotenie stavov sa vykonáva pomocou funkcie move_evaluate definovanej v súbore move.c, tato funkcia počíta ohodnotenie na základe použitého prehľadávania a závisi od mnohých nastavení Edaxu.

Stav v Edaxe je definovaný ako štruktúra, ktorá obsahuje reprezentáciu bitboardu hráča a

protihráča ako aj zoznam možných ťahov. Táto štruktúra sa vypĺňa pomocou rôznych funkcii zo súborov move.c, bitboard.c a bit.c.

6.2.1. Zhrnutie

Pre úpravu Edaxu na generátor bude treba väčšie pochopenie celého edaxu, avšak je teoretický možné upraviť aspiration_search aby pri prehľadávani ukladal stavy do súboru v určitej hĺbke.

6.3. Asimilátor

6.3.1. Zadanie

Rozšírte vytvorený asimilátor o ďalšiu funkcionalitu, ktorá zabezpečí spracovanie čiastočných výsledkov uložených v databáze a výpočet finálneho výsledku úlohy.

6.3.2. Typ úlohy

Implementačná

6.3.3. Analýza

Analýza sa nachádza v šprinte 3 – kapitola xyz

6.3.4. Návrh

Definovali sme tieto funkčné požiadavky na asimilátor:

1. Asimilátor musí generovať strom hry reversi do takej hĺbky x, ako generoval Generátor pri vytváraní workunitov.
2. Asimilátor musí každému uzlu v hĺbke x priradiť jeho zodpovedajúcu hodnotu z databázy. Túto hodnotu sme dostali ako výsledok výpočtu daného uzla (workunitu) u klienta. Ak databáza neobsahuje hodnotu daného uzla, tak mu priradí maximálnu, alebo minimálnu možnú hodnotu, v závislosti od hráča, ktorý je na ťahu.
3. Asimilátor, pomocou alpha-beta algoritmu prejde vytvorený strom hry a určí finálny výsledok.

Použitie alpha-beta algoritmu má niekoľko výhod:

- zrýchli prehľadávanie stromu hry
- dokáže zo získaných čiastočných výsledkov určiť, ktoré uzly (workunity) sú irelevantné pre finálny výsledok. Tieto workunity sa vymažú z databázy. Takto sa zmenší počet workunitov a teda celkový čas na výpočet úlohy.

Tento asimilátor sa bude periodicky spúšťať po pridaní určitého počtu čiastočných výsledkov do databázy, aby sa dosiahlo čo najefektívnejšie odsekávanie workunitov.

6.4. zhodnotenie šprintu

V čase písania dokumentácie sme ešte nestihli ukončiť a teda ani zhodnotiť šprint.

7. Big picture

Platforma BOINC pre distribuované počítanie skrýva v sebe potenciál. Úlohou projektu je vytvoriť infraštruktúru pre distribuované počítanie. Na tímovom serveri je nainštalovaný BOINC server.

Po návrhu vedúceho projektu sme sa rozhodli vyriešiť symetrickú hru Reversi 8×8, ktorej riešenie perfektnej hry nebolo vypočítané kvôli pamäťovým nárokom. Pre prvý prototyp sme veľkosť hracej plochy znížili na 6×6, pretože riešenie perfektnej hry je možné overiť už existujúcimi riešeniami, ktoré neboli vyriešené distribuovane.

Prototyp bežal už v treťom šprinte a bol vytvorený v programovacom jazyku Java. Vygenerovali sme 52 127 výpočtových úloh, ktoré boli distribuované členom tímu. Prvý prototyp odhalil niekoľko nedostatkov, ktoré vyplývajú požívaním jazyka Java a platformy BOINC. BOINC API neposkytuje funkcie pre prácu s týmto jazykom. Riešenie jednej čiastkovej úlohy trvá neprimerane dlho na počítačoch slabších konfigurácií. Spomalenie výpočtu je spôsobené volaním Garbage Collector-a virtuálneho stroja jazyka Java.

Kvôli tomu do konca piateho šprintu bolo vrátených len 183 súborov obsahujúcich riešenie čiastkových úloh, čo je porovnaní s celkovým počtom úloh veľmi malé číslo, ktoré pri podobnej rýchlosti riešenia nebude postačovať na vyriešenie hry s hracím poľom veľkosti 8×8.

Tento prototyp bude musieť byť nahradený novým. Rozhodli sme sa, že druhý prototyp bude vytvorený v C/C++. Porovnávacie testy rýchlosti riešenia jednej aplikácie v natívnom kóde ukázali, že rozhodnutie vymeniť jazyk Java bolo správne. Čas riešenia sa znížil z mesačných hodnôt na čas niekoľko desiatok minút. Tiež môžeme využiť poskytované API funkcie platformy BOINC pre C/C++ a tak spraviť našu aplikáciu užívateľský prístupnejšou.

Pre ďalšie používanie fakultného BOINC servera inými tímami a používateľmi bola vytvorená používateľská príručka a wiki stránka, ktorá je priebežne dopĺňaná.

8. Plán

8.1. Plán do februára

12. 12. 2013 – 7. 1. 2014
- prestávka a čerpanie nových síl
- študovanie Edax kódu
- dokončenie nového prototypu bez BOINC API

7. 1. 2014 – 21. 1 .2014
- práca na druhom prototype, využívanie BOINC API
- implementácia asimilátora
- nasadenie prototypu v jazyku C

21. 1. 2014 – 3. 2. 2014
- riešenie klientských úloh
- hľadanie ďalšej výpočtovej úlohy
- práca na používateľskej príručke a wiki

3. 2. 2014 – 17. 2. 2014
- vyhodnotenie riešenia Reversi 6×6
- práca na používateľskej príručke a wiki

8.2. Plán na letný semester

- Po vyhodnotení prototypu hry Reversi s veľkosťou hracieho poľa 6×6 riešiť hlavný problém hra Reversi s veľkosťou hracej plochy 8×8.
- Analyzovať možnosti distribuovaného riešenia druhej výpočtovej úlohy
- Vytvoriť používateľské príručky a napísanie návodov na wiki pre tímy a používateľov, ktorí nadviažu na našu prácu.
- Pokúsiť sa vytvoriť skripty a nástroje pre jednoduchšiu prácu s fakultným gridom.