AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Inżynierii Metali i Informatyki Przemysłowej

PROJEKT MAGISTERSKI

pt.

„Implementacja solwera frontalnego zrównoleglonego w wielowęzłowym heterogenicznym środowisku sprzętowym”

Imię i nazwisko dyplomanta:	**Paweł Wal**
Kierunek studiów:	**Informatyka Stosowana**
Nr albumu:	**240202**
Opiekun:	dr inż. Łukasz Rauch

Podpis dyplomanta: Podpis opiekuna:

Kraków 2015

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt magisterski wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Kraków, dnia . . . . . Podpis dyplomanta . . . . . .

Spis treści

1 WPROWADZENIE

Ilość dostępnych zasobów - takich jak moc obliczeniowa, pamięć dyskowa i operacyjna oraz przepustowość łączy - stale się zwiększa. Dzięki temu ciągłemu wzrostowi możliwości problemy które wydawały się zbyt duże by dało się je rozwiązać w rozsądnym czasie stają się możliwe do rozwiązania. Naturalnym jest jednak, że kiedy nierozwiązalne problemy stają się banalne, pojawiają się nowe problemy - jeszcze bardziej złożone, wynikające z próby odpowiedzi na jeszcze bardziej ambitne pytania.

Rozsądek nakazuje sądzić, iż wzrost wydajności pojedynczych urządzeń nie może trwać w nieskończoność. Prędzej czy później wzrost złożoności urządzeń zostanie zahamowany, choćby przez wzgląd na niemożność pomieszczenia większej ilości tranzystorów na pojedynczej płytce. W przypadku CPU zahamowanie tego procesu nie daje się jeszcze zauważyć. W roku 2014 firma Intel wypuściła procesory Broadwell oparte o proces technologiczny 14nm [19]. Na początku roku 2015 firma Taiwan Semiconductors zapowiedziała ambitne plany wprowadzenia na rynek procesorów w technologii 10nm w roku 2016 i 7nm w 2017 [26].

W przypadku GPU producenci mają jednak większy problem z miniaturyzacją elementów składowych swoich urządzeń. Wystarczy spojrzeć na to, jaką miniaturyzację planowała przeprowadzić NVIDIA. W 2009 roku na temat planów firmy w tej kwestii mówił na 46th Design Automation Conference William J Dally, wówczas szef działu naukowego w NVIDIA [23]. Jego ambitne projekcje przewidywały, iż do 2015 roku NVIDIA wprowadzi na rynek urządzenia wykonane w technologii 11nm, zawierające 5,000 mikrordzeni obliczeniowych i taktowane zegarami o częstotliwości 3 GHz. Specyfikacja urządzenia NVIDIA GeForce GTX TITAN X [22], jednej z najnowszych i najwydajniejszych kart graficznych wyprodukowanych przez firmę, dowodzi iż projekcje Dally'ego były mocno przesadzone: urządzenie wykonane jest w technologii 28nm, ma 3072 mikrordzenie i jest taktowane zegarem o częstotliwości 1000 MHz (1089 MHz w trybie Boost).

Alternatywą dla stworzenia urządzeń o większej wydajności jest wykorzystanie większej ilości urządzeń obliczeniowych w jednej maszynie. Jednakże części pozwalające na zamontowanie więcej niż jednego GPU w danej obudowie, takie jak płyty główne, wciąż jeszcze są urządzeniami specjalnego zastosowania a nie standardem. Im bardziej specyficzne jest zastosowanie konkretnego rodzaju sprzętu, tym jego cena staje się wyższa od typowych części. Dobrym przykładem jest tutaj płyta główna ASUS Rampage IV Black Edition, pozwalająca na zamontowanie aż 4 GPU; w trakcie pisania niniejszej pracy jej cena wynosi ponad 2000 złotych.

Oczywistym rozwiązaniem w tej sytuacji wydaje się przetwarzanie rozproszone. Dzięki jego zastosowaniu można uniknąć problemu coraz wolniej zwiększającej się wydajności pojedynczych urządzeń, gdyż do rozwiązania problemu można wykorzystać więcej urządzeń o mniejszej jednostkowej wydajności. Dzięki połączeniu odrębnych fizycznie maszyn w abstrakcyjny „klaster" nie ma konieczności umieszczania wszystkich posiadanych urządzeń obliczeniowych w jednej maszynie. Ta abstrakcja jest bardzo istotną cechą nowoczesnych platform do przetwarzania rozproszonego [7]. Z punktu widzenia użytkownika końcowego - tj. tego, kto wykonuje obliczenia na platformie przetwarzania rozproszonego - całość powinna funkcjono-

wać jak jeden spójny superkomputer o bardzo wysokiej mocy obliczeniowej. Abstrakcja musi też utrzymywać się niezależnie od tego że maszyny będą dołączać i opuszczać platformę, mając efektywnie stosunkowo niski średni czas do awarii. Przyjmuje się też że maszyny obliczeniowe nie muszą nawet być w jednej lokalizacji, dzięki połączeniu ich za pośrednictwem sieci Internet. Warto zauważyć, iż przetwarzanie rozproszone towarzyszy idei Internetu od samego jej zarania: możliwość stworzenia ogólnokrajowej sieci przetwarzania danych była jednym z argumentów za budową sieci ARPANET [8]. Rodzi to jednak oczywiście cały szereg nowych problemów, związanych z efektywnym wykorzystaniem łącz maszyny z dostępem do internetu oraz z utrzymaniem funkcjonowania całej platformy kiedy poszczególne elementy ten dostęp stracą.

Należy zdać sobie sprawę, iż podstawowym plusem przetwarzania rozproszonego jest możliwość zwiększania ilości urządzeń pracujących nad danym problemem (bądź zestawem problemów). W konsekwencji - zamiast wykorzystywać nieliczne urządzenia o bardzo wysokiej wydajności, i co za tym zazwyczaj idzie, cenie - można wykorzystać znacznie więcej urządzeń klasy konsumenckiej, tańszych i o mniejszej jednostkowej wydajności. Oczywistym kierunkiem jest też skupienie się na architekturach heterogenicznych. Kompozycje składające się z urządzeń różnego typu, takich jak CPU i GPU, nie tylko osiągają coraz wyższą wydajność (drugie miejsce na liście TOP500 z listopada 2014 [21] zajmuje heterogeniczny superkomputer Titan), ale też stają się coraz popularniejsze wśród konsumentów. W istocie większość typowych komputerów osobistych i biurowych to maszyny heterogeniczne; trudno patrzyć na to inaczej w sytuacji gdy niemal każda karta graficzna nadaje się również do celów obliczeniowych.

W przypadku komputerów osobistych szczególnie często występuje zjawisko „marnowania cykli". Oprogramowanie wykorzystujące cykle procesora niespożytkowane przez użytkownika i system operacyjny zostało opisane już w 1982 roku przez Schocha i Huppa [11]. Autorzy ci opisywali oprogramowanie które rozprzestrzenia się autonomicznie w sieci komputerowej i wykorzystuje maszyny - szczególnie te specjalnego przeznaczenia, jak serwery systemów plików czy sekwencji rozruchowych - jako dostawców mocy obliczeniowej. Naturalnie, w czasach nieustannego zagrożenia różnego rodzaju złośliwym oprogramowaniem, system rozproszony który sam rozprzestrzenia się w sieciach komputerowych brzmi co najmniej źle. Ponadto trzydzieści lat temu Schoch i Hupp pisali tylko o wykorzystywaniu mocy obliczeniowej procesorów. Obecnie jednak - z punktu widzenia architektur heterogenicznych - to nie CPU jest najbardziej interesującym urządzeniem. W typowym komputerze biurowym karta graficzna jest zawsze zasilana. Nie dotyczy to jedynie architektur hybrydowych, takich jak NVidia Optimus [24], w których proste funkcje spełnia zintegrowana karta graficzna (często firmy Intel), a akcelerator graficzny NVidia pozostaje w uśpieniu dopóki nie zostanie uruchomiona wymagająca aplikacja. Jeżeli jednak akcelerator jest zasilany a użytkownik korzysta jedynie z przeglądarki internetowej lub pakietu biurowego, nie jest on w pełni wykorzystywany - i pojawiają się zmarnowane cykle. W sytuacji, w której efektywność energetyczna urządzeń zaczyna być coraz istotniejsza (czego dowodem może być powstawanie takich list jak GREEN500 [17], klasy-

fikujących urządzenia według wskaźnika FLOP/wat), takie marnowanie cykli jest trudne do przyjęcia.

Takie zmarnowane cykle można wykorzystać - choćby do obliczeń naukowych czy inżynierskich. Na takiej zasadzie działają platformy przetwarzania rozproszonego na komputerach wolontariuszy. Platformy takie są obecnie wykorzystywane w badaniach dotyczących fizyki wysokich energii, biologii molekularnej, astrofizyki, badaniach klimatu i innych dziedzin [3]. Większość tego typu oprogramowania pracuje tylko kiedy maszyna-klient nie jest używana, a mimo to platformy te osiągają ogromne wydajności. Jeden z pierwszych i zarazem najlepiej znanych programów tego typu - SETI@home - w czerwcu 2015 miał średnią prędkość ponad 679 teraFLOP/s [25].

Platformy przetwarzania na podarowanych zasobach charakteryzuje nastawienie na heterogeniczny sprzęt oraz odporność na dużą rotację hostów. Zazwyczaj są stworzone tak, by być odporne na działania hostów kontrolowane przez złośliwych lub nieuczciwych użytkowników, które to hosty zwracają niepoprawne wyniki. Błędne wyniki mogą też zwracać komputery które uległy awarii. Zazwyczaj stosowanym rozwiązaniem tego problemu jest wielokrotne przetwarzanie tego samego zadania przez wiele maszyn niezależnie od siebie i konfrontowanie wyników [1, 3]. By wykorzystać większość z takich platform należy utrzymywać własną infrastrukturę i stworzyć komplet oprogramowania działającego wewnątrz platformy, co oczywiście angażuje zasoby - tyleż ludzkie, co finansowe. Trudno również wyobrazić sobie wykorzystanie takiej otwartej architektury przez firmę prowadzącą badania nad własną technologią.

Na przeciwnym biegunie w stosunku do przetwarzania na komputerach wolontariuszy stoją architektury gridowe. Zazwyczaj jest to własność instytucji, takich jak duże firmy czy uczelnie. Instytucje te zapewniają zasoby niezbędne, by grid mógł funkcjonować, oraz konieczną obsługę. Wykorzystanie zasobów gridowych należących do instytucji pozwala pozbyć się wielu problemów które dotyczą platform opartych na wolontariacie (nie do pomyślenia jest na przykład sytuacja zatruwania wyników badań). Nie ma też konieczności utrzymywania infrastruktury. Nadal jednak pozostaje problem stworzenia oprogramowania działającego w ramach architektury gridowej. Nie rozwiązuje to również sytuacji w której instytucja chcąca prowadzić badania nie posiada własnej infrastruktury, a umieszczenie danych dotyczących badań na komputerach będących pod kontrolą innego podmiotu nie wchodzi w grę.

W niniejszej pracy zostanie zaproponowane rozproszone oprogramowanie służące do rozwiązywania układów równań w postaci macierzy rzadkich. Opisane rozwiązanie stanowi punkt pośredni pomiędzy dwoma opisanymi wcześniej rodzajami oprogramowania rozproszonego. Z jednej strony jest stworzone w podobnym duchu co oprogramowanie do przetwarzania na zasobach wolontariuszy: odporne na rotację i awarie hostów, nie przyjmujące zbędnych założeń w stosunku do udostępnionych maszyn, mocy obliczeniowej urządzeń, fizycznej lokalizacji komputerów wykorzystywanych do obliczeń, czy czasu dostępności zasobów danej w ciągu doby. Z drugiej strony jest to rozwiązanie nakierowane bardziej na rozwiązanie problemu przetwarzania dużych zadań - dużych układów równań - w małych i średnich organizacjach, nie posiadających własnych architektur gridowych. Jego instalacja, utrzymanie i wykorzystanie mają być

możliwie uproszczone, tak by nie było konieczne tworzenie własnego oprogramowania, bądź korzystanie z usług specjalistów. Rozwiązanie ponadto ma się dobrze skalować, tak by wykorzystanie wszystkich dostępnych w danym momencie urządzeń obliczeniowych dawało realne przełożenie na czas uzyskania wyniku.

W dalszych sekcjach niniejszej pracy przeprowadzona zostanie analiza dostępnych rozwiązań, w tym ich mocnych oraz słabych stron z punktu widzenia rozpatrywanego przypadku. Zostaną również przedstawione założenia projektowe rozwiązania własnego, jego implementacja oraz wyniki przeprowadzonych na stworzonym oprogramowaniu badań wydajności.

2 PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ

W niniejszej sekcji rozważone zostaną istniejące oraz historyczne rozwiązania dotyczące przetwarzania na komputerach wolontariuszy oraz tworzenia niewielkich, wewnątrzorganizacyjnych sieci przetwarzania rozproszonego. Wskazane zostaną ich najistotniejsze, charakterystyczne cechy oraz przyjęte założenia projektowe. Zaznaczone zostaną też mocne i słabe strony wybranych przez ich autorów rozwiązań,

2.1 ENTROPIA

Projekt Entropia [4] był komercyjnym rozwiązaniem, tworzonym w Entropia, Inc. od 1997 roku. Głównym celem tego oprogramowania było zapewnienie możliwości tworzenia wewnętrznych, organizacyjnych architektur gridowych.

Entropia pozwalała na wykorzystywanie wyłącznie maszyn działających pod kontrolą systemu Windows. W momencie tworzenia tego oprogramowania przetwarzanie rozproszone było stosowane raczej w środowiskach akademickich, na maszynach działających pod kontrolą systemów z rodziny *NIX, na co Entropia miała być odpowiedzią.

Entropia, Inc. zakończyła działanie komercyjne w 2004 roku bez oficjalnego ogłoszenia tego faktu, ani powodu podjęcia takiej decyzji.

2.1.1 Architektura

Centralnym elementem sieci w Entropii jest serwer, pełniący kilka funkcji. Serwer służy zarówno do komunikacji z systemem użytkownika zlecającego zadanie, jak i do porozumiewania się z maszynami klienckimi.

Pierwszym etapem komunikacji użytkownika z systemem jest stworzenie zadania. Zadanie zawiera aplikację która ma być uruchamiana na komputerach klienckich, zbiór wejść dla aplikacji oraz innego rodzaju danych. Po otrzymaniu od użytkownika zadania, mechanizm zarządzania zadaniami w serwerze dzieli je na mniejsze podzadania oraz uruchamia narzędzia przetwarzania wstępnego jeżeli użytkownik takie zdefiniował.

Te podzadania są centralnie składowane i przekazywane do mechanizmu harmonogramowania zadań. Oprócz listy podzadań element ten utrzymuje też listę urządzeń na których może

uruchamiać zadania, wraz ze szczegółowymi informacjami o urządzeniu. W skład tych informacji wchodzi dostępność urządzenia oraz statyczne cechy urządzenia - ilość pamięci czy typ systemu operacyjnego. Dane te są wykorzystywane do efektywnego rozdzielania zadań pomiędzy aktualnie dostępnych klientów.

Mechanizm harmonogramowania odpowiada także za to, by zadania które długo nie zostały przekazane do przetwarzania otrzymały wyższy priorytet, co przesuwa je w górę kolejki zadań do wysłania. Dba również o ponowne uruchamianie zadań których wykonanie nie udało się i ponowne zlecenie zadań których wyniki zbyt długo nie wróciły z klienta. Jeżeli wykonanie danego podzadania konsekwentnie kończy się błędem, zostaje ono oznaczone jako błędne, o czym informowany jest użytkownik który zlecił zadanie.

Po zakończeniu wszystkich podzadań w danym zadaniu serwer wykonuje na nich etap przetwarzania końcowego, ponownie przy użyciu narzędzi zdefiniowanych przez użytkownika systemu. Po jego zakończeniu wyniki udostępniane są użytkownikowi. Może on pobrać albo zagregowane wyniki dla całego zadania, lub wyniki zakończenia poszczególnych podzadań.

2.1.2 Oprogramowanie klienckie

W ramach Entropii dostarczane było tylko oprogramowanie dla systemów operacyjnych z rodziny Windows - zarówno serwerowe, jak i klienckie.

W założeniu oprogramowanie klienckie Entropii miało być uruchamiane na komputerach osobistych klasy konsumenckiej, jednocześnie służących do innych zadań. Przykładem typowego zastosowania mogłaby więc być sytuacja, w której klient Entropii jest uruchomiony na wszystkich komputerach należących do jednej firmy - niezależnie od tego, czy używają ich pracownicy, czy nie.

Entropia wstrzymuje przetwarzanie zadań na pięć minut jeśli wykryje aktywność użytkownika na komputerze (obserwuje ją monitorując myszkę i klawiaturę). Autorzy oprogramowania nie zdecydowali się na wymaganie od aplikacji uruchamianych wewnątrz Entropii możliwości wstrzymania obliczeń. Po pierwsze, zawężałoby to możliwość zastosowania Entropii do oprogramowania w którym istnieje możliwość ingerencji w kod źródłowy. Po drugie, aplikacje które - celowo lub przypadkiem - nie wstrzymałyby działania po otrzymaniu stosownego sygnału zaburzałyby działanie maszyny klienckiej z punktu widzenia jej użytkownika.

Problem ten - oraz niektóre inne problemy związane ze źle napisanymi aplikacjami - został przez autorów Entropii rozwiązany przy pomocy wirtualizacji. Każde zadanie jest uruchamiane wewnątrz „piaskownicy”. Działanie tejże jest wstrzymywane kiedy wykryta zostanie aktywność użytkownika. W ten sposób aplikacja nie może też uzyskać dostępu do większej ilości zasobów niż pozwoli na to środowisko wirtualizacyjne. Przechwycone mogą też być odwołania do systemowych API służących do zapisu plików; aplikacja może uważać, że zapisuje pliki w folderze `C:\Program Files`, podczas gdy w istocie zapisuje je w folderze umieszczonym gdzieś wewnątrz struktury katalogów klienta Entropii.

Warto zwrócić uwagę, iż o ile jest to rozwiązanie podnoszące bezpieczeństwo całego układu - przy założeniu dużej dowolności uruchamianych wewnątrz Entropii aplikacji - jak

każda wirtualizacja dodaje ono własny narzut obliczeniowy i pamięciowy. Jest to efekt niepożądany z punktu widzenia minimalizacji czasu do rozwiązania. Trudno również wyobrazić sobie skuteczną wirtualizację oprogramowania korzystającego z kart graficznych.

2.1.3 Great Internet Mersenne Prime Search

Oryginalny projekt GIMPS, stworzony w 1996 przez George'a Woltmana, wykorzystywał proces manualny. Konieczne było wysłanie wiadomości e-mail pod wskazany adres by uzyskać pakiet wejść dla programu, pobranie programu, uruchomienie go i odesłanie - również e-mailem - wyników. Wraz ze wzrostem ilości autoignition złożoność i brak odporności na błędy tego procesu stały się niedostateczne. Mimo tego, jeszcze w 1996 roku odało się odkryć trzydziestą piątą liczbę Mersenne'a - $2^{1,398,269} - 1$.

Scott Kurowski - założyciel Entropia, Inc. - stworzył w oparciu o oprogramowanie tworzone w swojej firmie PrimeNet, co pozwoliło zautomatyzować wszystkie zadania dotyczące rozsyłania zadań i odbierania ich wyników. Umożliwiło to łatwiejsze zarządzanie siecią tysięcy wolontariuszy oraz bazą milionów zadań. Mimo zamknięcia Entropia, Inc., PrimeNet działa nadal; ostatnie odkrycie w ramach projektu nastąpiło 25 stycznia 2013, kiedy to zidentyfikowano 48 liczbę Mersenne'a [15].

2.2 SETI@HOME

SETI@home zostało oryginalnie stworzone w celu przetwarzania danych z projektu SERENDIP [13]. W ramach tego projektu zbierane były sygnały radiowe odbierane przez radioteleskop w Arecibo, w Puerto Rico. Następnie w tych danych wyszukiwane były charakterystyczne struktury, których pojawienie się mogło świadczyć o odebraniu sygnału radiowego powstałego sztucznie - potencjalnie pochodzącego od pozaziemskiej cywilizacji. Stąd pierwszy człon nazwy projektu - SETI, czyli Search For Extraterrestial Intelligence.

Z punktu widzenia przetwarzania rozproszonego bardziej interesujący jest drugi człon nazwy - @home. W ramach eksperymentu SETI@home po raz pierwszy zostało stworzone oprogramowanie dzięki któremu posiadacze zwykłych domowych komputerów i połączeń z internetem mogli uczestniczyć - a przynajmniej w pewnym sensie kontrybuować - w dużym projekcie naukowym. Oczywistą zachętą dla uczestników - wolontariuszy, pozwalających oprogramowaniu pracować kiedy nie korzystają z komputera - była minimalna szansa, że to właśnie ich komputery mogą znaleźć dowód istnienia technologicznie zaawansowanej cywilizacji pozaziemskiej.

Projekt SETI@home został stworzony do rozwiązania problemu którego nie dało się wówczas rozwiązać przy użyciu dostępnych urządzeń do obliczeń wysokiej wydajności; kłócić się można, że nadal nie istnieje możliwość przeprowadzenia tak dokładnych badań na takiej ilości danych w rozsądnym czasie nie korzystając z przetwarzania rozproszonego. Dane radiowe spływały bowiem w tempie 257 GB na tydzień.

2.2.1 Obróbka danych wejściowych

Architektura projektu była stosunkowo złożona, a jego utrzymanie wymagało serii skomplikowanych operacji. Przede wszystkim, kiedy projekt zaczynał pracę - w 1997 roku - nie istniała techniczna możliwość przesłania takiej ilości danych przez internet z Puerto Rico do USA. W związku z tym dane były przesyłane w formie taśm magnetyczych, przesyłką kurierską.

Na miejscu taśmy oczywiście należało zgrać przy użyciu odpowiednich urządzeń, oraz dodać do zapisanych na nich informacji metadane - takie jak punkt na niebie z którego zostały zebrane czy czas kiedy były nagrane. Był to czasochłonny, w dużej mierze ręczny proces wymagający odpowiedniej wiedzy technicznej od osób odpowiedzialnych za badania.

2.2.2 Tworzenie i wysyłka zadań

Dane z teleskopu opatrzone metadanymi były następnie dzielone na niewielkie paczki - o rozmiarze 250 kB. Tak mały rozmiar był konieczny po pierwsze dlatego, by czas pobierania na połączeniu modemowym był akceptowalny (według wyliczeń autorów projektu, na modemie o przepustowości 14400 baudów zajmowało to 2.3 minuty), a po drugie by czas przetwarzania zadania na komputerze klasy konsumenckiej był do zaakceptowania przez obie strony. Na typowych procesorach w 1997 roku było to zazwyczaj kilka dni.

Warto zauważyć, że ze względu na publiczną naturę projektu, dane do przetwarzania musiały być redundantne, to znaczy ten sam pakiet danych musiał być przetworzony przez dwie lub więcej niezależnych maszyn [2]. Pozwalało to uniknąć przypadkowych przekłamań w wynikach (powstałych na przykład w skutek niepoprawnego działania komputera wolontariusza) lub celowego zatruwania danych przez złośliwych użytkowników. Kopie danego zadania były więc przetwarzane aż do uzyskania kworum wyników zgadzających się ze sobą (z dopuszczalnymi pewnymi różnicami wynikającymi na przykład z różnych architektur procesorów w maszynach).

Wynikiem przetwarzania danego zadania była krótka wiadomość zawierająca położenie i częstotliwości sygnałów-kandydatów w danym wycinku danych radiowych. Informacja taka była prezentowana użytkownikowi, oraz wysyłana z powrotem do serwera, gdzie oczekiwała na kworum. Ostatecznie rezultat był zapisywany w bazie danych do późniejszych dokładniejszych badań.

2.2.3 Oprogramowanie klienckie

W przeciwieństwie do oprogramowania Entropia, nie został wykorzystany żaden rodzaj wirtualizacji ani zabezpieczania systemu operacyjnego klienta przed działaniami klienta SETI@home. Jest to zrozumiałe - ostatecznie odpowiedzialni za jego stworzenie byli też odpowiedzialni za jego dystrybucję. Można zatem uznać, że mieli kontrolę nad jego funkcjonowaniem i mogli zapewnić wolontariuszy o tym, iż jego uruchomienie jest bezpieczne.

Warto zauważyć, że dla wolontariusza wykorzystanie klienta było maksymalnie uproszczone. Wystarczyło, że pobierze on pakiet oprogramowania odpowiedni dla jego systemu ope-

racyjnego i go uruchomi. SETI@home automatycznie wykrywało momenty kiedy użytkownik nie korzystał z komputera i uruchamiało swoje przetwarzanie; wolontariusz więc nie miał ani potrzeby, ani możliwości ingerencji w jego działanie. Z drugiej strony, wytworzenie zadań oraz zarządzanie serwerową częścią projektu było trudnym zadaniem, wymagającym głębokiej technicznej wiedzy. Równie trudna była interpretacja wyników pochodzących ze zautomatyzowanego oprogramowania.

2.2.4 Podsumowanie

W krótkim czasie SETI@home zgromadziło wokół siebie wielu wolontariuszy; w 2002 roku były to już miliony komputerów [2]. Niewątpliwie miało to związek z łatwością zainstalowania oprogramowania klienckiego, a dodatkowym czynnikiem był coraz powszechniejszy dostęp do internetu. W pracy [3] Anderson - jeden z autorów SETI@home - i Fedak zauważają jednak również korelację dużych wzrostów ilości aktywnych hostów z wydarzeniami medialnymi.

Z punktu widzenia proponowanego rozwiązania aspekt medialny nie jest interesujący, gdyż sieci tworzone przy jego pomocy z założenia nie mają służyć do publicznych badań. Istotną kwestią jest jednak łatwość instalacji oprogramowania. Sprawia to, że użytkownik każdej indywidualnej maszyny może wyposażyć ją w odpowiednie oprogramowanie samodzielnie, bez dodatkowego przeszkolenia. W ten sposób we wdrożenie rozwiązania nie jest angażowana dodatkowa osoba (np. administrator systemu).

Projekt został przeniesiony na platformę BOINC.

2.3 BOINC

BOINC to spadkobierca SETI@home, stworzony przez jednego z jego autorów - Davida Andersona. SETI@home udowodniło iż w komputerach wolontariuszy drzemią ogromne zasoby. W pracy [1] Anderson odnotowuje iż mimo tego ogromu pojawiło się stosunkowo niewiele projektów które mogły by je wykorzystać.

Z tego spostrzeżenia narodził się BOINC. W założeniu miała to być otwarta, rozszerzalna platforma, stanowiąca swoiste środowisko pośrednie do tworzenia projektów rozproszonych. Oprogramowanie to jest skierowane głównie do środowisk naukowych, stanowiąc uogólnienie rozwiązania problemu który został rozwiązany przez zespół SETI@home.

Mianowicie BOINC skierowany jest do zespołów naukowych, które potrzebują dużej mocy obliczeniowej i są w stanie wyrazić swoją aplikację jako zautomatyzowane narzędzie, wykonujące wielokrotnie te same operacje na różnych zestawach danych. Zwalnia zespół naukowy z konieczności zagłębiania się w zagadnienia przetwarzania rozproszonego, choć nie ze zbudowania środowiska serwerowego w którym BOINC może pracować. Środowisko takie jest zresztą stosunkowo nieskomplikowane i składa się z typowych technologii sieciowych o otwartym źródle.

2.3.1 Narzędzia społecznościowe w BOINC

BOINC dostarcza też szereg narzędzi które mają za zadanie ułatwić przyciągnięcie wolontariuszy do projektu. Najistotniejszymi z nich jest system przypisywania użytkownikom punktów w zamian za wykorzystane przez aplikacje BOINC zasoby. Następnie na podstawie tych punktów tworzone są rankingi indywidualnych maszyn, użytkowników, czy też krajów. Wprowadza to do projektów element współzawodnictwa, który może napędzać przypływ nowych uczestników do projektu, oraz zwiększenie poświęcanego projektowi udziału w zasobach wśród już istniejących. Za pośrednictwem stron internetowych projektu użytkownicy mogą przeglądać swoje profile, formować zespoły czy wymieniać się wiadomościami na forach. Można więc powiedzieć, że BOINC jako standard wprowadza swoisty element społecznościowy do projektów o niego opartych.

2.3.2 Współdzielenie zasobów między projektami

W przypadku BOINC najważniejszym zadaniem było zapewnienie możliwości koegzystencji wielu projektów w obrębie jednej luźno zdefiniowanej platformy. W tym celu klient BOINC - następca klienta SETI@home - pozwala jednemu użytkownikowi na przyłączenie się do więcej niż jednego projektu i określenie jaki udział we wszystkich zasobach poświęconych na przetwarzanie przez BOINC ma otrzymać każdy z nich. Te preferencje uwzględniane są również przy zgłaszaniu się do serwerów projektów z informacją o tym, że klient jest aktywny i potencjalnie - żądaniem większej ilości zadań. W stosunku do klienta SETI@home użytkownicy BOINC otrzymują zresztą znacznie więcej możliwości wyboru kiedy BOINC ma korzystać z ich maszyny, w tym na przykład określenie w jakich godzinach może pracować, czy też jak może rozporządzać przydzieloną mu przestrzenią dyskową.

2.3.3 Wieloplatformowość

W przetwarzaniu na maszynach wolontariuszy naturalnym jest, iż zgłaszające się maszyny mogą pracować pod kontrolą różnych systemów operacyjnych. Mogą też korzystać z różnych procesorów w różnych architekturach. Już w SETI@home wprowadzono częściowe rozwiązanie tego problemu. Zestaw plików binarnych skompilowanych dla większości popularnych platform był umieszczony na serwerze. Kiedy klient zgłaszał zapotrzebowanie na pliki wykonywalne projektu, raportował rodzaj procesora, architekturę i system operacyjny. Według tego serwer projektu dobierał odpowiednie pliki binarne.

W BOINC wprowadzono ponadto mechanizm platformy anonimowej. Został on stworzony dla użytkowników wykorzystujących niepopularne procesory, takich którzy ze względów bezpieczeństwa nie mogą uruchomić oprogramowania które nie zostało skompilowane bezpośrednio na ich maszynach, oraz takich którzy chcą ręcznie zoptymalizować parametry kompilacji dla danej maszyny. Użytkownik który sam skompilował pliki wykonywalne projektu musi dodatkowo opisać je w konfiguracyjnym pliku XML. Jest to oczywiście dodatkowa czynność dla użytkownika i dodatkowe założenie co do jego wiedzy. Warto jednakże zwrócić uwagę iż

użytkownik który wie jak skompilować oprogramowanie na swoją platformę zazwyczaj daje sobie również radę z plikami XML.

2.3.4 Wykorzystanie GPU

Od pozostałych analizowanych projektów BOINC odróżnia się zdolnością do wykorzystania GPU. Od wersji 6 oprogramowania klient jest w stanie raportować obecność GPU wraz z jego typem (NVIDIA lub ATI/AMD) serwerom projektu [18]. Jeżeli jest dostępne w projekcie oprogramowanie wykorzystujące GPU, klient pobierze je i będzie używał go do przetwarzania zadań. Podobnie jak w przypadku ogólnym (CPU) wymaga to stworzenia i udostępnienia przez właścicieli projektu aplikacji wykorzystujących możliwości dostępnych urządzeń.

2.3.5 Podsumowanie

BOINC jest krokiem we właściwym kierunku. Przede wszystkim zdejmuje większość pracy w zakresie rozproszenia obliczeń z oryginalnego zespołu naukowego. Ułatwia też znalezienie wolontariuszy dla projektu oraz utrzymanie ich zainteresowania. Wzajemna promocja projektów opartych na BOINC i samej platformy leży w interesie wszystkich ośrodków ją wykorzystujących, a jak zauważyli Anderson i Fedak w [3], promocja zwiększa ilość dostępnych hostów w sposób niebagatelny, nawet jeśli chwilowy.

Jednakże nawet z użyciem BOINC odpowiedzialność za stworzenie oprogramowania do konkretnych zastosowań spoczywa na naukowcach tworzących projekt. Ze względu na jego otwartość dotyczą go też problemy z redundantnymi obliczeniami oraz z upublicznianiem dużych ilości danych dotyczących projektów.

Platforma nadal jest aktywna. Pojawiają się na niej nowe projekty.

2.4 XTREMWEB

XtremWeb był projektem który odznaczał się na tle tych wskazanych powyżej. Miał być bowiem platformą nie tylko do konkretnego zastosowania, ale również do generalnych eksperymentów w zakresie globalnego przetwarzania [6]. Projekt skupiał się na odporności na błędy, bezpieczeństwie uczestników oraz prostocie wdrożenia i użycia. Istotna była też heterogeniczność, rozumiana jako wiele różnych rodzajów sprzętu, systemów operacyjnych i podstawowego oprogramowania.

2.4.1 Tryby działania

Członkowie XtremWeb dzielili się na dwa rodzaje.

Pierwszym z tych rodzajów był wolontariusz. Wolontariusz łączył się z serwerem administracyjnym, pobierał oprogramowanie kliencie i instalował je na pojedynczej maszynie. Oprogramowanie wykorzystywało zasoby maszyny kiedy użytkownik jej nie używał.

Do XtremWeb można było też dołączyć jako współpracownik. Współpracownicy pobierali cały zestaw oprogramowania XtremWeb. W oparciu o niego mogli stworzyć swoją własną,

opartą o architekturę XtremWeb grupę obliczeniową i uruchamiać w jej obrębie swoją aplikację. XtremWeb prosiło jednak współpracowników o zgodę na podzielenie się mocą obliczeniową ich wolontariuszy: kiedy projekt współpracownika nie mógł dostarczyć więcej pracy zebranym przez niego wolontariuszom, otrzymywali oni zadania pochodzące bezpośrednio z głównej instancji XtremWeb.

Istniał też osobny tryb działania: intranetowy XtremWeb, dedykowany dla przedsiębiorstw. Przy pomocy tego oprogramowania można było stworzyć wewnątrzorganizacyjną sieć do przetwarzania rozproszonego, która nie miała żadnych połączeń ze światem zewnętrznym.

2.4.2 Bezpieczeństwo aplikacji obliczeniowych

W XtremWeb aplikacje uruchamiane na komputerach klienckich - mimo tego że mogły pochodzić ze źródeł trzecich - nie przechodziły żadnego formalnego procesu weryfikacji. Była na nich przeprowadzana jedynie pobieżna, ręczna analiza. Oczywiście nie mogła ona wyczerpywać wszystkich przypadków użycia oprogramowania na wszystkich możliwych platformach. Niewiele więc mówiła właścicielom projektu o stabilności wykorzystywanego oprogramowania.

2.4.3 Komunikacja między klientem a serwerem

W XtremWeb, podobnie jak w pozostałych rozważanych aplikacjach, wszelką komunikację rozpoczyna klient. Dzięki temu oprogramowanie może funkcjonować w sieciach za NAT lub na filtrowanych połączeniach. Co ciekawe, autorzy zrezygnowali z wykorzystania protokołu HTTP, uznając go za powolny. Z drugiej strony nie narzucili żadnego konkretnego protokołu komunikacji między klientem a serwerem. W oryginalnej pracy dotyczącej XtremWeb wspomniane są za równo połączenia ogólne - po gniazdach sieciowych z wykorzystaniem TCP lub UDP - jak i protokoły wyższego poziomu, jak CORBA czy RMI.

Protokół komunikacji wyspecyfikowany w pracy Fedaka i współpracowników opisuje kilka podstawowych żądań które klient może wystosować w stosunku do serwera. Są to:

- żądanie *hostRegister* które rejestruje klienta na serwerze. Klient otrzymuje w odpowiedzi tzw. wektor komunikacji, czyli zbiór serwerów które może odpytywać o zadania.

- żądanie *workRequest* które stanowi prośbę o przydzielenie zadań. Klient identyfikuje swój system operacyjny i architekturę; na podstawie tych danych serwer wybiera zadanie i wysyła je do klienta.

- żądanie *workAlive* które jest swoistym sygnałem bicia serca; informuje ono serwer że klient pracuje.

- żądanie *workResult* które sygnalizuje zakończenie pracy. Towarzyszy mu wysłanie wyników na serwer.

2.4.4 Implementacja

Oprogramowanie klienckie we wcześniej opisanych projektach było napisane w językach kompilowanych. Dostarczane było tylko dla jednej platformy (Entropia) lub jako prekompilowane pakiety dla typowych platform (SETI@home i BOINC). Autorzy XtremWeb zdecydowali się na wykorzystanie do napisania klienta języka Java. Ponadto odwołania do niektórych konkretnych funkcji systemu operacyjnego realizował kod w C, zintegrowany z Javą za pośrednictwem Java Native Interface. W aplikacji która ma osiągać wysoką wydajność, wybór języka który uruchamiany jest za pośrednictwem wirtualnej maszyny może zastanawiać. Autorzy XtremWeb racjonalizują swój wybór kwestiami przenośności pomiędzy systemami operacyjnymi.

Aplikacja w języku Java spełnia jedynie funkcję zarządczą. Sprawdza czy użytkownik jest obecny przy maszynie oraz odpowiada za komunikację z serwerem. Kiedy maszyna staje się dostępna (czyli dostatecznie długo nie wykryto aktywności użytkownika), uruchamiane są nowe wątki. Pierwszym z nich jest wątek monitorujący; jeżeli wykryje on obciążenie maszyny pochodzące z innych źródeł niż obliczenia XtremWeb, zakończy działanie tychże. Drugi to wątek obliczeniowy, który uruchamia natywny kod obliczeniowy. Ostatni to wątek obserwacji czasu życia, wykonujący żądania *workAlive* do serwera.

Serwer składa się z kilku baz danych i modułów. Przede wszystkim przechowuje informacje o wszystkich aplikacjach. Oprócz tego przechowuje dane dotyczące zadań, w tym na jakich maszynach zostały przetworzone i w jakim czasie. Ponadto odpowiada za wybieranie zadań - dbając o to by każda z aplikacji otrzymała określony udział w dostępie do hostów w projekcie - oraz ich rozmieszczanie na komunikujących się z nim klientach.

Wdrożenie w sieci komputerowej oprogramowania XtremWeb wspierane jest przez zestaw skryptów dostarczanych wraz z oprogramowaniem. W szczególności jest to narzędzie xwLan, które pozwala administratorom sieci oprogramowanie klienckie w całej sieci jednocześnie.

2.4.5 Podsumowanie

W architekturze XtremWeb istniała możliwość tworzenia wewnątrzorganizacyjnych sieci służących do obliczeń rozproszonych, co z punktu widzenia niniejszej pracy było jedną z jego najważniejszych cech. Nie rozwiązywał on jednak konieczności tworzenia własnych aplikacji obliczeniowych. Ponadto wymuszał na autorach oprogramowania dostosowanie się do jego standardu komunikacyjnego.

Na moment pisania niniejszej pracy projekt wydaje się być zarzucony. Mimo tego że strona internetowa dotycząca projektu nadal jest aktywna, najnowszą opublikowaną wersją oprogramowania klienckiego jest 2.0.0, wydana w maju 2008 roku [27].

2.5 PORÓWNANIE KLUCZOWYCH ELEMENTÓW WYBRANYCH ROZWIĄZAŃ

Między zaprezentowanymi projektami można wskazać szereg wspólnych cech i założeń, istnieją też jednak fundamentalne różnice.

2.5.1 Cechy wspólne

Architektura. Jedną z najbardziej rzucających się w oczy jest architektura klient-serwer, wspólna dla wszystkich bez wyjątku projektów. Autorzy systemów BOINC i XtremWeb sugerują [1, 6], iż jednym z przyszłych kierunków ich badań może być wprowadzenie elementów sieci peer-to-peer (klient-klient) do swoich architektur, lecz w żadnym z projektów to rozwiązanie nie zostało jeszcze wdrożone.

Dystrybucja oprogramowania obliczeniowego. Wspólna dla prezentowanych projektów jest również metoda dystrybucji aplikacji obliczeniowych. Skompilowane dla różnych platform wersje aplikacji składowane są na serwerze i wysyłane klientowi który zgłasza na nie zapotrzebowanie. Jedynie BOINC dopuszcza wykorzystanie tzw. platformy anonimowej, gdzie w ramach projektu uruchamiany jest plik binarny dostarczony przez klienta.

Ogólny zarys klienta. Wyznaczyć można szereg cech oprogramowania klienckiego wspólnych dla rozważanych projektów. Klient musi być przede wszystkim automatyczny: konieczność ingerencji użytkownika powinna być ograniczona do minimum. Wszystkie prezentowane klienty są też skonstruowane tak, by respektować użytkownika - zawieszają swoje działanie kiedy wykryją aktywność użytkownika.

„Niestabilność" zasobów. Kondo i współpracownicy wskazują [9] dużą rotację dostępnych maszyn jako cechę charakterystyczną architektur rozproszonych opartych o komputery osobiste. W istocie, w [3] Anderson i Fedak odnotowują iż średni czas życia danego hosta (od pierwszego do ostatniego kontaktu) wynosi około 91 dni. Ponadto na uruchomionych maszynach klienckich, według ich danych, klient BOINC był uruchomiony tylko przez około 81% czasu, z czego aktywny (tj. wykonujący obliczenia) przez średnio 84%. Nietrudno wyliczyć z tego, iż przeciętny host biorący udział w projektach opartych o BOINC jest obliczeniowo aktywny przez nieco ponad 68% doby. Co interesujące, odporność na rotację hostów to cecha charakteryzująca nie tylko projekty przetwarzania na zasobach wolontariuszy, takie jak BOINC, ale też Entropię czy XtremWeb, których można używać jako platformy wewnątrzorganizacyjne.

2.5.2 Najważniejsze różnice

Zaufanie oprogramowaniu. W przypadku projektu SETI@home oraz platformy BOINC, oprogramowanie obliczeniowe jest rozprowadzane jako zaufane. Źródłem zaufania jest autorytet autorów i popularność danego projektu. W XtremWeb istnieje tylko ręczny, nieformalny proces weryfikacji zgłoszonych aplikacji klienckich; autorzy przyznają [6] iż może on być niewystarczający. W projekcie Entropia z kolei zaufanie do oprogramowania jest ograniczone. Jednakże dzięki wirtualizacji klienta zakres szkód który może ono wyrządzić w systemie jest ograniczony.

Zaufanie użytkownikowi. SETI@home i BOINC proponują model redundantnych obliczeń, w którym każde zadanie jest przetwarzane przez co najmniej dwa hosty, do uzyskania zbieżności wyników. Jest to model ograniczonego zaufania użytkownikowi. Z kolei XtremWeb oraz Entropia w wersjach zaprezentowanych w pracach [4] oraz [6] nie wprowadzają dodatkowej weryfikacji wyników uzyskanych od klientów, ufając im w ten sposób całkowicie.

Wsparcie GPU. Tylko BOINC potrafi wykorzystywać GPU na maszynach klienckich.

3 PROJEKT ROZWIĄZANIA WŁASNEGO

3.1 ZAŁOŻENIA

W niniejszej sekcji zostaną omówione założenia podjęte w stosunku do tworzonego oprogramowania oraz fundamentalne decyzje projektowe podjęte w trakcie jego tworzenia.

3.1.1 Cechy sprzętu

Najistotniejszą cechą prezentowanego oprogramowania jest współpraca klienta z jak najszerszą gamą urządzeń obliczeniowych oraz niewielkie wymagania w stosunku do sprzętu na którym uruchomiony będzie serwer. Konieczna była zatem identyfikacja istotnych cech charakterystycznych sprzętu oraz ich ewentualnych dopuszczalnych zakresów. Oprogramowanie następnie było tworzone tak, by funkcjonować poprawnie przy spełnieniu minimalnych wymagań. Poniżej przedstawiono wybrane do budowy założeń projektowych cechy charakterystyczne maszyn i urządzeń oraz ich konsekwencje.

Lokalizacja. Maszyny mogą znajdować się w różnych lokalizacjach geograficznych. W konsekwencji należy spodziewać się, iż będą znajdowały się w rozłącznych (w stosunku do serwera oraz względem siebie) sieciach lokalnych. Ich połączenie z Internetem może też mieć małą przepustowość lub być filtrowane. Ponadto pakiety przesyłane do i z danej maszyny mogą podlegać NAT (Network Address Translation, tłumaczeniu adresów sieciowych) [20], który wiąże się z modyfikacją źródłowych bądź docelowych adresów IP połączeń. Z tego założenia wysnuć można dwa wnioski. Po pierwsze, niedopuszczalna jest konieczność stworzenia połączenia z serwera do klienta (gdyż uniemożliwia to NAT). Wszelka komunikacja musi być inicjowana przez klienta, serwer zaś musi pozostawać pasywny. Ponadto rozmiar danych przesyłanych między serwerem a klientem musi być możliwie jak najmniejszy. Konieczne jest zatem zastosowanie algorytmów kompresji.

Parametry urządzeń obliczeniowych. Istotnymi parametrami dla urządzenia są czas aktywności, rozpatrywany jako ułamek czasu w którym urządzenie jest zasilane, oraz wydajność. Należy oczekiwać, iż urządzenie będzie dostępne przez krótkie okresy czasu w ciągu doby, oraz że urządzenie będzie miało niską wydajność. Urządzenie może też być wykorzystywane przez inne oprogramowanie podczas obliczeń, co jeszcze bardziej obniży jego wydajność postrzeganą przez oprogramowanie. Problem zatem należy dzielić na niewielkie zadania, których wykonanie na najsłabszym badanym sprzęcie nie zajmie więcej niż godzinę czasu rzeczywistego, a najlepiej krótszy czas. Rozmiar i charakter zadań powinien być dobrany do charakterystyki urządzenia, przekazanej serwerowi przez klienta. Konkretne informacje wchodzące w skład tej charakterystyki opisane są - jako detal implementacyjny - w podsekcji 4.4.2. Należy również wziąć pod uwagę sytuację, w której klient zostanie wyłączony - na przykład w wyniku wyłączenia maszyny - podczas przetwarzania zadania. Konieczny jest zatem mechanizm limitu czasu przetwarzania. Jeżeli po upływie określonego czasu od wysłania zadania nie zostanie otrzy-

many rezultat, zadanie powinno zostać przesłane ponownie jednemu z aktualnie dostępnych hostów.

Parametry maszyn. Parametry samej maszyny istotne z punktu rozważanego rozwiązania to ilość urządzeń obliczeniowych w jednej maszynie, rozmiar pamięci operacyjnej oraz dostępna przestrzeń dyskowa. Z jednej strony należy przyjąć założenie, że w maszynie jest jedno urządzenie obliczeniowe; jest to absolutne minimum konieczne by uruchomienie klienta miało jakikolwiek sens. Z drugiej strony aby optymalnie wykorzystać maszyny mające więcej urządzeń obliczeniowych, wartość ta powinna być raportowana serwerowi. Z punktu widzenia serwera ilość urządzeń obliczeniowych powinna wpływać na maksymalną liczbę zadań która może zostać przydzielona danemu klientowi. Co do pamięci operacyjnej - należy przyjąć iż dla samego klienta nie będzie jej wiele. Prowadzi to do wniosku, iż po pierwsze - nie należy bez potrzeby ładować danych dotyczących zadań - w tym przypadku części macierzy - do pamięci, a w miarę możliwości należy je przechowywać poza nią. Podczas przechowywania danych w pamięci absolutną koniecznością jest wykorzystanie formatów i narzędzi pozwalających na efektywne przechowywanie macierzy rzadkich. Ponieważ przestrzeń dyskowa jest ograniczona, dane powinny być przechowywane na dysku w postaci skompresowanej, oraz nie dłużej niż są istotnie potrzebne. Oznacza to, iż dane dotyczące danego zadania powinny zostać przez klienta skasowane kiedy tylko serwer potwierdzi poprawny odbiór wyniku.

3.1.2 Łatwość wykorzystania

We wprowadzeniu do niniejszej pracy oraz podczas analizy istniejących rozwiązań kilkukrotnie został podniesiony problem wymagań kadrowych wiążących się z wdrożeniem systemu przetwarzania rozproszonego. Do wdrożenia opisanych systemów zazwyczaj konieczny jest administrator systemów bazodanowych. Potrzebny jest też programista, lub realistycznie - zespół programistów, którzy stworzą aplikacje na odpowiednio szeroki zestaw platform. Co więcej, to dotyczy nawet najbardziej ogólnego, prostego przypadku i nie dotyczy specyficznych wymagań danego projektu - jak na przykład wymaganej w SETI@home umiejętności obsługi urządzeń do odczytu taśm magnetycznych.

Fundamentalnym założeniem dla systemu który z łatwością można wdrożyć w firmie są niewielkie wymagania co do systemu operacyjnego oraz monolityczność z punktu widzenia instalacji. Zarówno serwer jak i klient muszą być stworzone w technologii którą łatwo przenieść między systemami operacyjnymi oraz platformami sprzętowymi bez modyfikacji kodu źródłowego. Rozsądnym wydaje się przyjęcie, iż oprogramowanie musi zostać stworzone albo w języku skryptowym (interpretowanym), takim jak Python czy Ruby, albo w języku uruchamianym w maszynie wirtualnej i w ten sposób uniezależnionym od systemu operacyjnego, takim jak Java. Dobór języków programowania i innych narzędzi zostanie szerzej przedyskutowany w sekcji 4.1.

Dzięki wybraniu odpowiednich narzędzi wdrożenie systemu będzie możliwe na już istniejących maszynach, bez budowania maszyny-serwera dedykowanej dla niniejszego oprogramowania. Konieczne jest również uwzględnienie brzegowego przypadku w którym za serwer

służy jedna z maszyn klienckich. Serwer sam w sobie nie powinien więc używać urządzeń obliczeniowych. Oprogramowanie musi też cechować posunięty tak daleko jak to możliwe brak zależności zewnętrznych. Nie można na przykład przyjąć założenia, iż wraz z oprogramowaniem będacym przedmiotem niniejszej pracy na maszynie klienckiej zostanie zainstalowany serwer baz danych. W tej sytuacji jakikolwiek komponent bazodanowy który zostanie zastosowany, musi być dostarczany w postaci wyłącznie sterownika (a nie klienta i serwera). Ponadto powinien jako miejsce składowania danych wykorzystywać tylko lokalny system plików.

3.1.3 Łatwość modyfikacji

W swoistej opozycji do poprzedniego założenia - możliwości wdrożenia oprogramowania przez specjalistów w innej dziedzinie niż programowanie - stoi łatwość modyfikacji oprogramowania. Oprócz możliwości wdrożenia w proponowanej postaci, oprogramowanie powinno być łatwo dostosować do nowego problemu. Z jednej strony oprogramowanie musi więc być monolityczne. Zgodnie z zasadą „czarnej skrzynki" musi dawać spodziewane wyniki dla określonego zestawu danych wejściowych. Z drugiej strony jednak wewnętrznie musi być na tyle modularne, by elementy dotyczące na przykład komunikacji między serwerem a klientem można było z łatwością wykorzystać ponownie. Dzięki temu proponowane oprogramowanie będzie można dostosować do rozwiązywania innego problemu wymieniając logikę dotyczącą tworzenia zadań i ewentualnie definicję samego zadania. Można też będzie wykorzystać komponenty jako część zupełnie innego systemu.

3.1.4 Wydajność komponentów obliczeniowych

Pewien problem stanowi, w świetle wymienionych uprzednio założeń co do oprogramowania, wydajność komponentów obliczeniowych. Z jednej strony zasugerowany został wybór języków interpretowanych lub prekompilkowanych. Z drugiej strony narzut czy to interpretera, czy to maszyny wirtualnej jest problematyczny w przypadku oprogramowania realizującego obliczenia wysokiej wydajności. Wynika z tego, iż o ile możliwie jak najwięcej oprogramowania dotyczącego zarządzania systemem oraz komunikacji powinna być stworzona w języku wysokiego poziomu, lecz samo oprogramowanie obliczeniowe poiwinno być stworzone w języku niskiego poziomu i zoptymalizowane.

W przypadku prezentowanych uprzednio platform przetwarzania rozproszonego problemem jest to iż są to rozwiązania ogólne, a nie służące do rozwiązywania konkretnego problemu. W przypadku proponowanego rozwiązania zachowana jest pełna kontrola nad oprogramowaniem obliczeniowym. Nic nie stoi więc na przeszkodzie by razem z pakietem oprogramowania dla klienta i serwera rozprowadzać skompilowane dla najpopularniejszych platform, przy rozsądnych domyślnych ustawieniach kompilatora, pliki binarne. Oprócz tego rozprowadzany będzie oczywiście kod źródłowy. Dzięki temu dla zaawansowanego użytkownika możliwa będzie dodatkowa optymalizacja, czy to na drodze manualnej optymalizacji kodu źródłowego, czy przez dobranie parametrów kompilatora. Możliwe też będzie uruchomienie zarówno serwera jak i klienta na mniej popularnych platformach.

3.1.5 Wykorzystanie szerokiej gamy urządzeń

System rozproszony mający w założeniu nadawać się do wykorzystania z niemal dowolnymi urządzeniami musi być w stanie efektywnie wykonywać obliczenia na wielu różnych urządzeniach. Z jednej strony możliwe jest stworzenie oprogramowania pod każdy rodzaj urządzenia z osobna. W istocie nie ma wielu rodzajów urządzeń które należałoby rozważyć.

Jedną z klas urządzeń do wykorzystania są oczywiście CPU. Dla nich należałoby stworzyć odpowiedni moduł obliczeniowy, naturalnie odpowiednio zrównoleglony dla maszyn z więcej niż jednym procesorem. Ponadto konieczna jest obsługa kart graficznych. Jednym z najważniejszych, z punktu widzenia rozpowszechnienia urządzeń, dostawców takich kart jest NVIDIA. Firma ta stworzyła i wspiera swój własny zestaw bibliotek do tworzenia oprogramowania dla kart graficznych o nazwie NVIDIA CUDA [5]. Konkurentem NVIDIA jest AMD, która rozpoczęła produkcję kart graficznych po przejęciu ATI. Historycznie, AMD posiadało własny zestaw bibliotek do obsługi swoich procesorów i kart graficznych, nazwany AMD Stream SDK i podobny w założeniach i możliwościach do CUDA. Obecnie jako platforma rozwojowa dla urządzeń AMD proponowany jest AMD APP SDK [14], zbudowany w oparciu o standard OpenCL.

Tworzenie co najmniej trzech rozłącznych obszarów kodu dla trzech platform jest, nawet rozważając problem czysto intuicyjnie, nieoptymalne. Wprowadzenie jakiejkolwiek poprawki będzie wymagało wprowadzenia jej w co najmniej trzech miejscach. Ponadto każda z implementacji - by być optymalna - byłaby specyficzna dla platformy dla której została stworzona. Z tego powodu wprowadzona poprawka mogłaby funkcjonować inaczej w każdym z rodzajów oprogramowania, bądź nie być możliwa do wprowadzenia. Odpowiedni dobór narzędzi rozwiązujący ten problem jest szerzej omówiony w sekcji 4.1.

3.2 CYKL ŻYCIA ROZWIĄZANIA

Rozwiązanie jest w niniejszym oprogramowaniu uznawane za obiekt nadrzędny. Mimo tego przechowuje on jedynie informację o tym, jak rozwiązywany problem się nazywa.

Każde rozwiązanie ma co najmniej jedną falę. W większości przypadków fal może być więcej niż jedna. Falą nazywany jest zbiór zadań który opisuje dany krok w przetwarzaniu macierzy. Pierwsza fala rozwiązania to bezpośredni wynik dekompozycji macierzy, szerzej opisanej w sekcji 3.3. Nowa fala nie może zostać utworzona zanim wszystkie zadania w danej fali nie zostaną przetworzone. Każde z zadań delegowane jest do jednego klienta na raz. Szczegóły procesu żądania zadań oraz ich zwrotu opisane są w sekcji 3.6. Zadanie przetwarzane jest przez jedno urządzenie obliczeniowe (szczegóły tego procesu można znaleźć w sekcji 3.4), a następnie rezultat jest zwracany do serwera. Kiedy wszystkie zadania w danej fali zostaną przetworzone i zwrócone, uruchamiany jest na nich proces rozwiązywania zależności. Został on szczegółowo opisany w sekcji 3.5. W zależności od jego wyników może zostać wygenerowana kolejna fala rozwiązania jeżeli macierz nie jest jeszcze w stanie gotowym do rozwiązania. Może też zostać wykonane ostateczne rozwiązanie układu równań, opisane szerzej w sekcji 3.7.

3.3 DEKOMPOZYCJA MACIERZY

3.3.1 Założenia podstawowe

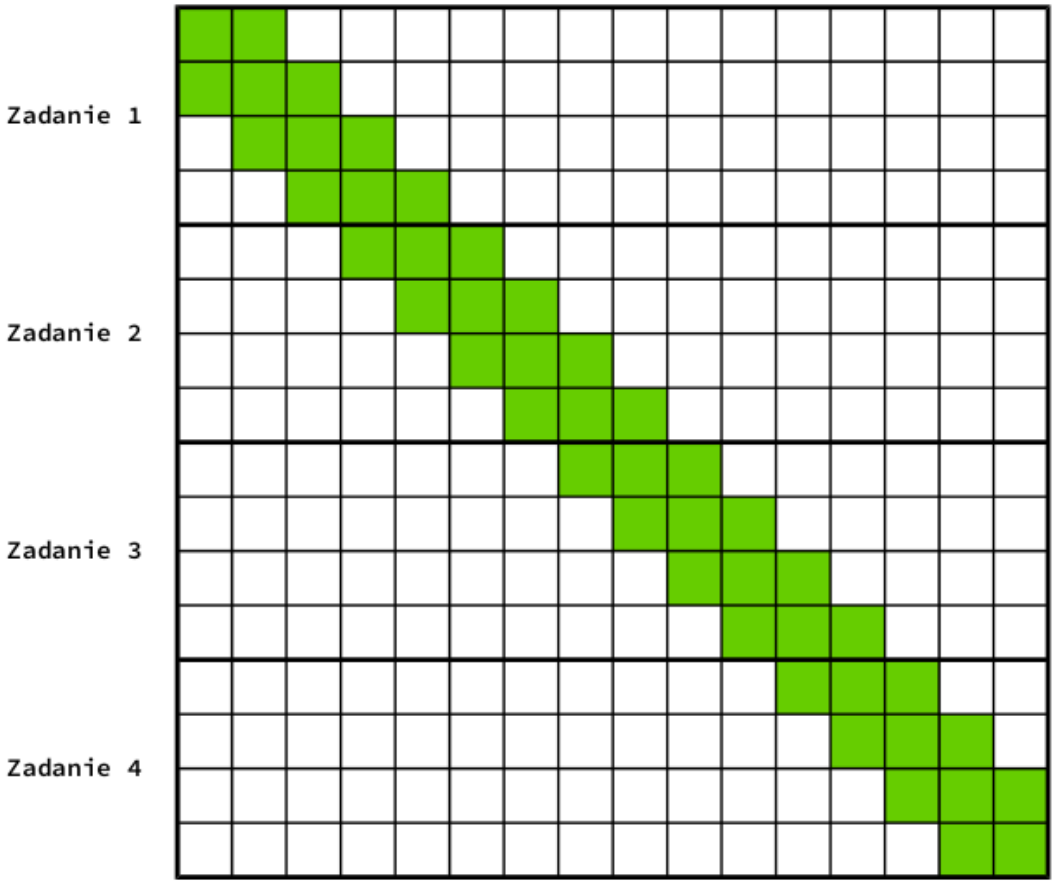
Celem rozwiązania problemu nierozwiązalnego (lub trudnego do rozwiązania) klasycznymi metodami, konieczne jest wprowadzenie metody podziału go na mniejsze problemy które można rozwiązać osobno od siebie. Jako rozwiązanie „osobno od siebie" rozumiane jest tu rozwiązanie nie wymagające komunikacji w trakcie przetwarzania części. Dopuszczalna jest komunikacja po przetworzeniu wszystkich części. Kolejnym z istotnych z punktu widzenia założeń jest unikanie próby umieszczenia całej macierzy w pamięci.

Omawiane oprogramowanie dedykowane jest do rozwiązywania układów równań danych w postaci macierzy rzadkich, z pewną preferencją w kierunku macierzy pasmowych. Macierze te charakteryzuje niewielka ilość elementów niezerowych i grupowanie tych elementów wokół przekątnej macierzy. Macierze tego rodzaju często powstają podczas rozwiązywania problemów metodą elementów skończonych. Warto zauważyć iż w takich układach równań ilość zmiennych jest bezpośrednio powiązana z ilością elementów siatki oraz liczbą stopni swobody przypisanych elementom [10]. Podczas modelowania danego obiektu pokrycie go gęstszą siatką elementów skończonych powoduje dokładniejsze modelowanie, ale zwiększa też rozmiar macierzy. Łatwo można więc spostrzec iż w tym przypadku możliwość rozwiązywania większych układów równań przekłada się bezpośrednio na dokładność obliczeń inżynierskich.

3.3.2 Zasady podziału macierzy

Koncepcja podziału macierzy na części jest przedstawiona na rysunku 1. Z oryginalnej macierzy wycinane są części o predefiniowanej wysokości jeżeli to możliwe. Każda część odpowiada jednemu zadaniu, wszystkie zadania zaś przypisane są do jednej fali. Jeżeli jako wysokość części nie zostanie wybrana wartość która jest dzielnikiem rozmiaru macierzy, ostatnia część będzie krótsza (niższa). Nie powinno to mieć wpływu na funkcjonowanie pozostałych części systemu.

Na rysunku 1 pasmo zostało zaznaczone kolorem zielonym. Jak widać na każdy wycięty z macierzy obszar zawiera stosunkowo niewiele elementów. Ponieważ każdy taki obszar odpowiada jednemu zadaniu, można zauważyć że dla macierzy o preferowanej charakterystyce liczba danych do przesłania będzie stosunkowo niewielka. Nie wyklucza to oczywiście przetwarzania nawet macierzy gęstych. Wtedy jednakże zarówno czas przesyłu jak i przetwarzania części będzie dłuższy. Tylko jedna część macierzy powinna być ładowana na raz do pamięci dla każdego wykorzystywanego urządzenia obliczeniowego.



Rysunek 1: Koncepcja dekompozycji macierzy

3.4 PRZETWARZANIE ZADANIA

3.5 ROZWIĄZANIE ZALEŻNOŚCI MIĘDZY CZĘŚCIAMI MACIERZY

3.6 KOMUNIKACJA KLIENT-SERWER

3.7 ROZWIĄZANIE UKŁADU RÓWNAŃ

4 IMPLEMENTACJA

4.1 WYBÓR NARZĘDZI

Tu wstaw dlaczego Railsy. Dlaczego Ruby. Dlaczego C++. Dlaczego OpenCL.

4.2 PODZIAŁ MACIERZY

Tu wstaw boleśnie dokładny opis splittera.

4.3 SERWER ZADAŃ

4.3.1 Struktura bazy danych

4.3.2 Umieszczenie zadania w serwerze

4.3.3 Autoryzacja klienta

4.4 KLIENT

4.4.1 Instancje oprogramowania obliczeniowego

4.4.2 Negocjacja ilości przydzielonych zadań

Na pewno mechanizm z ilością urządzeń i max GWS; może benchmark

4.4.3 Rozwiązanie zadań

4.4.4 Odesłanie rozwiązanych zadań

4.5 OPROGRAMOWANIE OBLICZENIOWE

4.5.1 Solwer po stronie klienta

4.5.2 Negocjator zależności po stronie serwera

5 WYNIKI

5.1 CHARAKTERYSTYKA MASZYN

Skalowalność rozwiązania w aspekcie zmiennej ilości urządzeń obliczeniowych.

6 PODSUMOWANIE

7 BIBLIOGRAFIA

[1] Anderson, D.P.: BOINC: a system for public-resource computing and storage. *Grid Computing. Proceedings. Fifth IEEE/ACM International Workshop on*. 2004

[2] Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*. 2002 Vol. 45, No. 11, pp. 56-61

[3] Anderson, D.P., Fedak, G.: The Computational and Storage Potential of Volunteer Computing *Cluster Computing and the Grid. Sixth IEEE International Symposium on*. 2006 Vol. 1, pp. 73-80

[4] Chien, A., Calder, B., Elbert, S., Bhatia, K.: Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*. 2003 Vol. 63, pp. 597–610

[5] Cook, S.: CUDA Programming. A Developer's Guide to Parallel Computing with GPUs. *Waltham: Elsevier*. 2013.

[6] Fedak, G., Germain, C., Neri, V., Cappello, F.: *Cluster Computing and the Grid. Proceedings. First IEEE/ACM International Symposium on*. 2001 pp. 582-587

[7] Foster, I., Kesselman, C., Nick, J. M., Tuecke, S.:. Grid services for distributed system integration. *Computer*. 2002 Vol. 35, No. 6, pp. 37-46

[8] Heart, F., McKenzie, A., McQuillian, J., Walden, D.: ARPANET completion report, Technical Report 4799. *BBN*. 1978

[9] Kondo, D., Taufer, M., Brooks, C., Casanova, H., Chien, A.: Characterizing and evaluating desktop grids: an empirical study. *Parallel and Distributed Processing Symposium. Proceedings. 18th International*. 2004

[10] Milenin, A.: Podstawy MES. Zagadnienia termomechaniczne. *AGH*. 2010

[11] Shoch, J.F., Hupp, J.A.: The "Worm"Programs - Early Experience with a Distributed Computation. *Communications of the ACM*. 1982 Vol. 25, No. 3, pp. 172-180

[12] Stone, J.E., Gohara, D., Guochun S.: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering*. 2010 Vol. 12, No. 3, pp. 66-73

[13] Sullivan, III, W. T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., Anderson, D.: A new major SETI project based on Project Serendip data and 100,000 personal computers. *Astronomical and Biochemical Origins and the Search for Life in the Universe, IAU Colloquium 161*. 1997 p. 729

[14] APP SDK - A Complete Development Platform [online]. *AMD*. [dostęp: 2015-06-20], Dostępny w Internecie: <http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/>

[15] GIMPS Discovers 48th Mersenne Prime, $2^{57,885,161} - 1$ is now the Largest Known Prime [online]. *Great Internet Mersenne Prime Search*. [dostęp: 2015-06-01], Dostępny w Internecie: <http://www.mersenne.org/primes/?press=M57885161>

[16] GIMPS History - PrimeNet [online]. *Great Internet Mersenne Prime Search*. [dostęp: 2015-06-01], Dostępny w Internecie: <http://www.mersenne.org/various/history.php>

[17] The Green Lists [online]. *The Green500*. [dostęp: 2015-06-14], Dostępny w Internecie, <http://www.green500.org/greenlists>

[18] GPU Computing [online]. *BOINC*. [dostęp: 2015-06-18], Dostępny w Internecie: <http://boinc.berkeley.edu/wiki/GPU_computing>

[19] Intel launches first Broadwell processors [online]. *CPU World*. [dostęp: 2015-06-01], Dostępny w Internecie: <http://www.cpu-world.com/news_2014/2014090701_Intel_launches_first_Broadwell_processors.html>

[20] The IP Network Address Translator (NAT). *IETF*. [dostęp: 2015-06-18], Dostępny w Internecie: <https://www.ietf.org/rfc/rfc1631.txt>

[21] November 2014 [online]. *TOP500 Supercomputer Sites*. [dostęp: 2015-06-01], Dostępny w Internecie: <http://www.top500.org/lists/2014/11/>

[22] NVIDIA GeForce GTX TITAN X [online]. *TechPowerUp*. [dostep: 2015-06-01], Dostępny w Internecie: <http://www.techpowerup.com/gpudb/2632/geforce-gtx-titan-x.html>

[23] Nvidia SVP Predicts Rapid GPU Performance Growth [online]. *Nikkei Technology*. [dostep: 2015-06-01], Dostępny w Internecie: <http://techon.nikkeibp.co.jp/english/NEWS_EN/20090731/173702/>

[24] Optimus Technology [online]. *NVIDIA*. [dostęp: 2015-06-14], Dostępny w Internecie: <http://www.nvidia.com/object/optimus_technology.html>

[25] SETI@Home Project [online]. *BOINC Stats*. [dostęp: 2015-06-13], Dostępny w Internecie: <http://boincstats.com/en/stats/0/project/detail>

[26] TSMC Promises 10nm Production In 2016, 7nm in 2017 [online]. *WCCF Tech*. [dostęp: 2015-06-01], Dostępny w Internecie: <http://wccftech.com/tsmc-promises-10nm-production-2016-7nm-2017/>

[27] InriaForge: XtremWeb: Project Home [online]. *InriaForge*. [dostęp: 2015-06-13], Dostępny w Internecie: <http://gforge.inria.fr/projects/xtremweb/>