Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Jakub Jurkiewicz

Modelowanie rozproszonych systemów typu desktop grid

rozprawa doktorska

Promotor:

dr hab. Piotr Bała

Zakład Obliczeń Równoległych i Rozproszonych
Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika w Toruniu

Interdyscyplinarne Centrum
Modelowania Matematycznego i Komputerowego
Uniwersytet Warszawski

Kwiecień 2012

Oświadczenie autora rozprawy:
oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

10 kwietnia 2012	. . . . .
Data	*Jakub Jurkiewicz*

Oświadczenie promotora rozprawy:
niniejsza rozprawa jest gotowa do oceny przez recenzentów.

10 kwietnia 2012	. . . . .
Data	*dr hab. Piotr Bała*

Streszczenie

Przedmiotem badań opisanych w rozprawie doktorskiej są symulacje systemów typu desktop grid. Systemy te charakteryzują się częstymi włączeniami i wyłączeniami węzłów, a także wykorzystaniem sieci o zmiennej przepustowości. Ze względu na dużą losowość nie jest możliwe analityczne modelowanie takich systemów i pojawia się konieczność wykorzystania symulacji.

W pracy przedstawiono opracowany model systemu typu desktop grid, mający takie właściwości jak rzeczywisty system. W modelu uwzględniono awarie i przypadkowe wyłączenia węzłów. Do modelowania zmiennej przepustowości sieci zostały użyte szeregi czasowe. Zaprezentowano przykłady zastosowań opracowanego modelu, którymi są: optymalizacja polityki przydzielania zasobów oraz poszukiwanie elementów najbardziej ograniczających wydajność systemu. Przykłady te wskazały na przepustowość sieci jako element ograniczający wydajność systemu typu desktop grid.

W wyniku przeglądu istniejących systemów typu desktop grid stwierdzono brak łatwej integracji istniejących systemów typu desktop grid z gridami usługowymi. Systemy bezpieczeństwa istniejących systemów typu desktop grid są niewystarczające. W pracy przedstawiono system typu desktop grid opracowany na bazie oprogramowania warstwy pośredniej gridu usługowego UNICORE. Testy wydajnościowe potwierdziły dostateczną wydajność tego systemu. Przedstawiony system charakteryzuje się wysokim poziomem bezpieczeństwa wynikającym z wykorzystania oprogramowania UNICORE i jest łatwo integrowalny z gridem usługowym.

Porównanie wyników badań wydajnościowych opracowanego systemu z wynikami symulacji potwierdza poprawność modelu. Własności systemów typu desktop grid, identyfikowane przez symulator, odnoszą się również do rzeczywistego systemu.

Słowa kluczowe: systemy typu desktop grid, symulacje, UNICORE, integracja gridów, symulacje przepustowości sieci, szeregi czasowe

Klasyfikacja według ACM: C.2.4, C.2.m, C.4, G.3

Simulation of distributed systems - desktop grids

Abstract

The research described in the dissertation deals with the simulations of desktop grids. These grids are characterized by frequent startups and shutdowns of the nodes and they involve a variable bandwidth network. Due to the large randomness it is not possible to develop an analytical model of such systems. Therefore, it is necessary to use simulation.

The paper presents a developed model of a desktop grid system. The model show features of the actual system. The model takes into account failures and shutdowns of nodes. Time series have been used to model the variable bandwidth of the network. The dissertation includes examples of the model applications, namely: optimization of resource allocation and identification of bottleneck in the desktop grid. The examples pointed out that the network bandwidth could be one of reasons of limiting the desktop grid system performance.

A survey of the existing desktop grids proves that there is no easy integration between the desktop grid and the service grids. The existing desktop grids' security systems are inadequate. The paper presents a desktop grid system developed with involvement of the service grid middleware – UNICORE. Performance tests have confirmed sufficient capacity of the system concerned. The system shows a high level of security originated from the UNICORE software. The system is easily integratable with service grids.

Comparison of the performance test findings versus the simulation results has confirmed the correctness of the model in question. Properties of the desktop grid systems, identified by the simulator, apply to the actual system just as well.

Keywords: desktop grids, simulations, UNICORE, integration of grids, network bandwitdth simulation, time series

Classification according to ACM: C.2.4, C.2.m, C.4, G.3

Pragnę podziękować:

promotorowi doktorowi habilitowanemu Piotrowi Bale za cierpliwość, poświęcony czas i wskazówki,

doktorowi Krzysztofowi Nowińskiemu za cenne uwagi dotyczące prowadzonych badań,

mojej teściowej Ewie Barańskiej i siostrze Edycie Jurkiewicz za pomoc w redakcji tekstu pracy,

Rodzicom i Babci za wiarę we mnie,

oraz:

mojej żonie Agnieszce za miłość i wsparcie.

Badania opisane w pracy realizowano w trakcie wspólnego projektu ICM UW i Telekomunikacji Polskiej S.A 22/06/6727/K/2006/YCZ268 Grid System Monitoring and Control Tools finansowanym z grantu FSO 023/2006.

Spis treści

Rozdział I

Przedmiot, zakres i cel badań

Przedmiotem badań opisanych w niniejszej rozprawie doktorskiej są symulacje systemów typu desktop grid i gridów społecznościowych.

Problem wykonywania dużych obliczeń, przekraczających możliwości jednego komputera, od dawna rozwiązuje się poprzez stosowanie obliczeń równoległych i rozproszonych. Początkowo wykorzystywano w tym celu wyłącznie maszyny z pojedynczych centrów superkomputerowych. Następnie zaczęto łączyć komputery z różnych centrów i prowadzić obliczenia na maszynach rozproszonych geograficznie. Dalsze łączenie ośrodków przy użyciu sieci Internet spowodowało konieczność unifikacji zdalnego dostępu do zasobów – zaczęły powstawać systemy gridowe. W ciągu ostatnich dziesięciu lat upowszechniła się również idea wykorzystywania komputerów osobistych stojących na biurkach pracowników lub ochotników do wykonywania obliczeń w czasie, kiedy nie korzystają z nich użytkownicy. Takie podejście do obliczeń ma dwie podstawowe zalety: znaczącą oszczędność kosztów (nie ma konieczności zakupu dodatkowych maszyn i serwerowni) oraz mniejsze zużycie energii (wykorzystuje się i tak włączone komputery).

Systemy oparte na komputerach osobistych można podzielić na dwa podstawowe typy, w zależności od sposobu organizacji właścicieli węzłów obliczeniowych:

- **grid społecznościowe** (*community grid*) są systemami wspierającymi obliczenia społecznościowe. Wykorzystują one komputery ochotników, którzy przeznaczają czas swoich komputerów na wspomaganie ważnych projektów naukowych. Wielkość takich systemów sięga milionów węzłów;
- **desktop grid** są systemami wspierającymi obliczenia w ramach zorganizowanej grupy użytkowników. Najczęściej wykorzystują one komputery znajdujące się w jednej lub kilku instytucjach, dlatego ich skala rzadko przekracza 1000 węzłów.

W przypadku dowolnych, długotrwałych obliczeń istotnym zagadnienie jest prognoza czasu ich trwania. Dla dobrze znanych programów można oszacować

czas działania, jednak dla wielu zadań nie jest możliwa odpowiedź na pytanie, jak długo będzie trwało obliczenie. Do oszacowania czasu wykonania programu konieczna jest znajomość rozmiaru danych oraz parametrów komputerów, na których program będzie wykonywany.

Istotnym problemem jest więc odpowiedź na pytanie: jak długo trzeba czekać, aby wykonane zostały obliczenia o zadanej wielkości? Czas ten zależy od wielu czynników. Dla pewnej klasy zadań szacuje się oddzielnie części, których czas wykonania zależy wyłącznie od jednego czynnika np. dysku lub procesora.

W przypadku systemów typu desktop grid, w których użytkownicy chcieliby otrzymać wyniki obliczeń w ciągu kilkudziesięciu minut od zlecenia (a na pewno w czasie krótszym od dnia roboczego) zasadnicze znaczenie ma ogólnie pojęta zawodność węzłów obliczeniowych. Zarówno awarie jak i wyłączenia węzłów spowodowane koniecznością użycia komputera do innych celów można uważać za awarie. Powodują one zrywanie obliczeń, tj. zakończenie inne niż zakończenie programu sukcesem lub błędem obliczeń. Problem ten nie był dotychczas brany pod uwagę w eksperymentach i symulacjach.

Modelowanie matematyczne długotrwałych obliczeń nie jest możliwe analitycznie i konieczne jest stosowanie symulatorów. Dla systemów gridowych skonstruowano różne symulatory. Najważniejsze z nich zostały opisane w treści rozprawy. W chwili rozpoczęcia badań opisanych w rozprawie, żaden ze znanych symulatorów nie uwzględniał problemu przerywania obliczeń. Nie było to potrzebne – gridy społecznościowe wykorzystywały do długotrwałych obliczeń tzw. checkpointing, czyli zapisywanie wyników pośrednich. Natomiast w przypadku gridów usługowych komputery z centrów obliczeniowych ulegały awariom na tyle rzadko, że problem ten można było zaniedbać.

W przypadku, gdy nie można zaniedbać awarii komputerów, ani nie można łatwo aproksymować niedostępności węzłów obliczeniowych poprzez modyfikację ich wydajności (jak postępuje się w przypadku symulacji systemów typu community grid), jedynym wyjściem jest symulacja działania takiego systemu. Uznałem zatem, że obszarem moich prac badawczych będzie symulowanie systemów typu desktop grid.

W trakcie badań nie znaleziono istniejących rozwiązań zapewniających odpowiedni poziom bezpieczeństwa w systemach typu desktop grid. Rozwiązanie takie powinno zapewniać dostatecznie wysoki poziom bezpieczeństwa danych i zarazem umożliwiać interakcje z gridami usługowymi. Przyczyną tego, może być fakt, że większość systemów typu desktop grid wywodzi się z oprogramowania gridów społecznościowych, gdzie nie wolno było udostępniać węzłom danych, które nie mogły być upublicznione. System bezpieczeństwa był na ogół znacznie upraszczany. Nie było możliwości eleganckiego zarządzania informacją łączącą dane z węzłami, na których mogą być przetwarzane. Innym problemem jest stworzenie systemu bezpieczeństwa umożliwiającego łączenie systemów typu desktop grid

z gridami usługowymi. Oba te czynniki stanowiły motywację stworzenia systemu typu desktop grid, którego integralną częścią jest system bezpieczeństwa. Aby to osiągnąć postanowiłem zbudować system typu desktop grid na bazie istniejącego oprogramowania dla gridów usługowych, charakteryzującego się wysokim stopniem bezpieczeństwa. Dodatkowo starałem się tak zaprojektować ten system, by wymagania dotyczące dostępu do zasobów były minimalne, np. poprzez znaczące ograniczanie dostępu do dysku.

Niniejsza praca obejmuje zatem swoim zakresem kilka zagadnień, które dotyczą przede wszystkim odpowiedzi na pytania:

Czy istnieje możliwość stworzenia modelu systemu typu desktop grid o takich samych właściwościach jak rzeczywisty system, w szczególności modelu systemu, w którym węzły ulegają awariom i przypadkowym wyłączeniom?

Czy system stworzony na bazie istniejącego oprogramowania warstwy pośredniej, używanego w gridach usługowych, może być wystarczająco wydajny do pracy w systemie typu desktop grid?

Czy istnieje możliwość stworzenia wydajnego systemu typu desktop grid z akceptowalnym poziomem bezpieczeństwa?

Aby odpowiedzieć na pytanie pierwsze, został stworzony model i symulator systemu typu desktop grid oraz przeprowadzono szereg symulacji pokazujących zastosowania symulatora. Stworzony symulator rozbudowano o prosty, ale bardzo skuteczny model sieci, oparty na szeregach czasowych.

Dotychczas panowało przekonanie, że oprogramowanie warstwy pośredniej gridów usługowych nie ma dostatecznej wydajności na potrzeby systemów typu desktop grid. Opisany w niniejszej rozprawie system typu desktop grid oparty na oprogramowaniu UNICORE okazał się dostatecznie wydajny.

Porównanie wyników symulacji z wynikami pracy stworzonego systemu pokazuje, że stworzony symulator ma właściwości bardzo zbliżone do stworzonego systemu typu desktop grid. Oznacza to, że co najmniej pewne własności systemów klasy desktop grid można modelować, a pewne zachowania są przewidywalne.

Z przeprowadzonych badań wynika możliwość i zasadność symulacji systemów typu desktop grid, czego udowodnienie było celem niniejszej rozprawy doktorskiej. W pracy pokazano jak wykorzystać symulacje do optymalizacji systemu typu desktop grid. Przykłady pokazują jak optymalizować zarządzanie zasobami i przydziałem zadań. Innym zastosowaniem wskazującym na użyteczność symulacji jest znalezienie wąskich gardeł. Osiągnięciem pracy jest też stworzenie efektywnego systemu typu desktop grid opartego o oprogramowanie warstwy pośredniej bezpiecznego gridu usługowego – UNICORE.

Metodologię badań oparto na symulacjach komputerowych rzeczywistego sytemu. Wyniki symulacji porównano z rzeczywistością. Modele tworzono w sposób maksymalnie obiektowy. Do symulacji sieci użyto szeregów czasowych. Tworzony system typu desktop grid oparto w maksymalnym stopniu na istniejącym już oprogramowaniu.

Kompozycja pracy jest następująca: w rozdziale II przedstawiono historię rozwoju systemów gridowych, zdefiniowano pojęcie gridu, a także przedstawiono charakterystykę systemów gridowych, ze szczególnym uwzględnieniem systemów typu desktop grid. W rozdziale III przedstawione zostały standardy dotyczące systemów gridowych, a także opisane zostały najpopularniejsze platformy gridów, gridów społecznościowych i systemów typu desktop grid. Następnie w rozdziale IV opisane zostały wcześniejsze prace dotyczące modeli systemów gridowych. W rozdziale V przedstawiony został stworzony na potrzeby pracy model systemu gridowego. W rozdziale VI opisano parametryzację modelu sieci. Następnie w rozdziale VII zostały przedstawione przykłady wykorzystania symulacji: optymalizacja zarządzania zlecaniem zadań oraz poszukiwanie wąskich gardeł w systemie. Rozdział VIII zawiera opis stworzonego systemu typu desktop grid opartego o oprogramowanie Unicore. W rozdziale IX przedstawiono wyniki testów wydajnościowych stworzonego systemu, przy okazji potwierdzających prawdziwość praw odkrytych w trakcie symulacji. Praca jest zakończona rozdziałem X zawierającym podsumowanie wyników pracy i wnioski.

Wyniki opisane w pracy były publikowane w monografiach, czasopismach i na konferencjach:

- Jakub Jurkiewicz, Krzysztof Nowiński, Piotr Bała, Prediction of the Jobs Execution on the Community Grid, *CGW'07 Proceedings*, Kraków, 2008, str. 299-305,

- Jakub Jurkiewicz, Krzysztof Nowiński, Piotr Bała, Prediction of the Jobs Execution on the Community Grid with added network latency, *Distributed and Parallel Systems In Focus: Desktop Grid Computing*, Springer, 2008, str. 43-48,

- Jakub Jurkiewicz, Krzysztof Nowiński, Piotr Bała, Numerical simulations of the resource utilization in the community grids, *Polish Journal of Enviromental Studies*, 17(3B), 2008, str. 1230-1485,

- Jakub Jurkiewicz, Krzysztof Nowiński, Piotr Bała, UNICORE 6 as a platform for desktop grid, *Computer Science and Information Technology*, IMCSIT 2008, International Multiconference on Computer Science and Information Technology, 2008

- Jakub Jurkiewicz, Piotr Bała, Building UNICORE Based Desktop Grid, *UNICORE Summit 2010*, Schriften des Forschungszentrums Jülich IAS Series, vol. 5, 2010, str. 11-18.

Rozdział II

Systemy obliczeń rozproszonych

1. Retrospektywna analiza systemów rozproszonych

Komputery, czyli elektroniczne programowalne maszyny do wykonywania obliczeń, pojawiły się na początku lat czterdziestych dwudziestego wieku. Pierwsze maszyny były jednak drogie, duże i nie istniała możliwość zdalnego dostępu, ani łączenia ich mocy obliczeniowych [24]. Mogły one wykonywać jedno zlecenie na raz, zaś zmiana programu wymagała fizycznej ingerencji operatora – przełączania kabli. Na początku lat pięćdziesiątych stworzono możliwość programowania komputerów, bez konieczności fizycznej ingerencji w maszynę. Na początku lat sześćdziesiątych pojawiła się możliwość zdalnego wgrywania programów do pamięci komputera i uruchamiania zadań, co oznaczało początek łączenia komputerów. Prawdziwym przełomem było opublikowanie pracy [50] dotyczącej przełączenia pakietów. Na jej podstawie została stworzona sieć Arpanet, z której wyewoluował Internet, globalna sieć komputerowa. Jej powstanie stworzyło nowe możliwości przesyłania danych do komputerów, do których użytkownik nie ma bezpośredniego dostępu. Wcześniej przetwarzanie danych było zamknięte w obrębie jednego ośrodka.

W latach osiemdziesiątych powstało wiele rodzajów systemów rozproszonych. Większość wczesnych systemów, jakkolwiek rozległa terytorialnie, była ograniczona do jednej instytucji, która miała całkowitą kontrolę nad maszynami wykonującymi obliczenia. Przyznawanie uprawnień do korzystania z zasobów, w szczególności danych i mocy obliczeniowej, odbywało się w scentralizowany sposób lub na zasobach indywidualnych uzgodnień.

Sytuacja zmieniła się diametralnie w latach dziewięćdziesiątych dwudziestego wieku, kiedy upowszechnił się dostęp do sieci komputerowej Internet. Okazało się, że pojedynczy naukowiec ma dostęp do wielu ośrodków obliczeniowych, a dodatkowo może korzystać z wielu baz danych i systemów obliczeniowych.

Powszechny dostęp do sieci i możliwość korzystania z wielu komputerów na raz spowodowały powstanie mechanizmów zdalnego wywoływania procedur, np. Corba, oraz systemów ułatwiających rozpraszanie obliczeń, jak PVM [72] i MPI [43]. Mechanizmy te zapewniają biblioteki, dzięki którym odpowiednio napisane programy są automatycznie rozpraszane przez system. Poważną wadą tych systemów jest brak mechanizmów autoryzacyjnych i brak możliwości uruchamiania dowolnych programów.

Pod koniec lat dziewięćdziesiątych powstały pierwsze systemy gridowe [37], tj. takie systemy rozproszone, w których spełniony jest wymóg zunifikowanego dostępu do zasobów, lecz w których każdy zasób może być zarządzany przez różne osoby lub organizacje i może wykorzystywać inne oprogramowanie.

Obecnie systemy gridowe można podzielić na dwa podstawowe rodzaje – **gridy usługowe** (*service grid*) i **gridy oparte na komputerach osobistych**, które można podzielić na dwa typy: **gridy społecznościowe** (*community grid*) i **gridy oparte na stacjach roboczych** (*desktop grid*), nazywane w dalszej części pracy systemami typu desktop grid.

2. Gridy usługowe

Gridy usługowe są w pewien sposób następcami ośrodków obliczeniowych. Obecnie przeciętny naukowiec ma najczęściej dostęp do co najmniej kilku ośrodków obliczeniowych. Dodatkowo dane, z których korzysta, pochodzą z różnych źródeł. Ponieważ większość naukowców nie jest informatykami, wielu nawet bardzo dobrych specjalistów nie wykorzystuje efektywnie dostępnego potencjału komputerowego. Spowodowane jest to różnicami w interfejsach do poszczególnych zasobów, a także stopniem ich skomplikowania. Naukowcy powoli uczą się narzędzi dających dostęp do kilku określonych zasobów, jednak nie umożliwia to pełnego wykorzystania wszystkich zasobów. Te przyczyny leżały u podstaw powstania gridów usługowych, narzędzi służących do zunifikowanego dostępu do różnych zasobów. Jednym ze znaczeń słowa *grid* w języku angielskim jest sieć energetyczna. Użytkownicy sieci energetycznej, nie potrzebują wiedzy o energetyce, szczegółowych rozliczeniach z dostawcami prądu, ani rodzajach elektrowni, w których prąd powstał. Użytkownik podłącza urządzenie do kontaktu i otrzymuje pewien zasób – energię. Gridy usługowe działają na podobnej zasadzie: użytkownik uruchamia oprogramowanie klienckie i zleca wykonanie pewnego zadania. Najlepiej, żeby oprogramowanie klienckie było także narzędziem, z którego użytkownik korzysta do przygotowywania danych wejściowych i oglądania wyników. Dopuszczalne jest wymaganie od użytkownika wyboru maszyny, która powinna obsłużyć zlecenie (chociaż dobry system powinien automatycznie dobrać zasoby do zlecenia). Jednak użytkownik nie powinien myśleć, jaki system zainstalowany

jest na docelowej maszynie ani jak się go obsługuje. Użytkownik nie powinien też być zmuszany do wpisywania co chwilę hasła, lecz co najwyżej raz – na początku sesji. Dobre oprogramowanie gridowe powinno umożliwić sprawdzenie stanu obliczeń, kiedy nie ma dostępu do swojej stacji roboczej. Pożądane jest umożliwianie oglądania wyników pośrednich oraz przeprowadzenia korekty danych wejściowych i powtórzenia obliczenia.

Gridy usługowe tworzą maszyny pochodzące z różnych ośrodków obliczeniowych. Każda z tych maszyn świadczy jakąś usługę, np. daje dostęp do pewnych danych, na podstawie danych wykonuje obliczenie, etc. Administratorzy tych maszyn przyznają pewnym grupom użytkowników dostęp do określonych usług. Jednym z rodzajów usług jest *resource brokering*, czyli rozdzielanie zlecenia na konkretne podzadania i znajdowanie zasobów najodpowiedniejszych do wykonania zlecenia.

Przydział zasobów i szeregowanie w gridach usługowych

Przydział lub alokacja zasobów (*resource brokering*), czyli dobieranie zasobów do zlecenia, jest zagadnieniem ściśle związanym z systemami rozproszonymi. Szeregowanie (*scheduling*) jest to polityka mówiąca w jakiej kolejności dane czynności mają być wykonywane. Dobry mechanizm przydzielania zasobów powinien zawierać elementy szeregowania.

Zagadnienie szeregowania jest nieodłącznie związane z systemami wieloużytkownikowymi. Szeregowanie pojawia się w gridach w dwóch miejscach:

- jest elementem systemu przydziału zasobów;
- jest elementem systemu obsługującego zasób. Każdy administrator węzła obliczeniowego musi bowiem rozwiązywać problem zlecania większej ilości zadań, niż w danej chwili węzeł może wykonać.

Szeregowanie jako element systemu przydziału zasobów dotyczy tylko zleceń gridowych. Na węzłach natomiast dotyczy wszystkich obsługiwanych zleceń. także tych pochodzących spoza gridu.

Problem rozdziału zadań do maszyn można rozwiązać na wiele sposobów. Najprostszy polega na nie świadczeniu takiej usługi. Grid usługowy służy wtedy tylko jako warstwa pośrednia, która uniezależnia użytkownika od różnic pomiędzy systemami. Wymaga ona od niego wiedzy, gdzie jego obliczenia wykonają się szybciej, a gdzie wolniej. Rozwiązanie to znacząco upraszcza budowę gridu oraz autoryzację, gdyż nie jest konieczne zlecanie „w czyimś” imieniu. Użytkownik sam decyduje gdzie poszczególne fragmenty zadań będą wykonywane.

Inne podejście do alokacji zasobów polega na stworzeniu jak najprostszej polityki: wybór maszyny dla składowej części zadania jest dokonywany, gdy wszystkie

części ją poprzedzające zostały już wykonane. Wybierany jest wtedy węzeł losowy spełniający wymagania tej części lub węzeł, który najszybciej wykona tę część. Zaletą tego podejścia jest prostota oraz brak problemu z sytuacjami wyjątkowymi, takimi jak zrywanie obliczeń, czy duże fluktuacje sieci. Jego wadą jest zaniedbywanie informacji o powiązaniach miedzy zadaniami. Skutkiem zaniedbania takich powiązań może być zwiększony transfer sieciowy, gdy okazuje się że podzadanie wykorzystujące wyniki poprzedniego jest wykonywane zupełnie na zupełnie innej maszynie. Jest to szczególnie uciążliwe gdy istnieje konieczność transferu dużych plików.

Możliwe jest stosowanie bardziej zaawansowanych polityk przydziału zasobów. Uwzględniany są wtedy czasy wykonania podzadań i zależności pomiędzy podzadaniami. Optymalizacje te często polegają na wysyłaniu zadań do policzenia w miejscu gdzie znajdują się dane, co optymalizuje wykorzystanie sieci.

W sytuacji gdy wykonanie podzadania w zleceniu wymaga wcześniejszego wykonania grupy podzadań celowe jest wykonanie całej grupy na węzłach o podobnej wydajności. Często warto nie wykorzystać najszybszych dostępnych węzłów gdyż całe zlecenie i tak musi czekać na wolniejsze węzły. Użycie szybszych węzłów uniemożliwiłoby wykorzystanie ich do wykonania innych zadań i pogorszyło wydajność systemu.

W wielu systemach nie ma rozróżnienia między przerwaniem programu spowodowanym błędem programu, a błędem systemu zdalnego, na którym program się wykonuje. Z tego powodu jedyną metodą określenia, co spowodowało przerwanie programu, może być uruchomienie zadania jeszcze raz. Jest to jedną z przyczyn, dla których większość znanych systemów przydzielania zasobów i szeregowania w gridach usługowych restartuje zadania, gdy wystąpił błąd w trakcie wykonywania. Niektóre bardzo zaawansowane systemy szeregujące potrafią stwierdzić, że nastąpiła awaria węzła i restartują zadanie w razie jej wystąpienia.

3. Charakterystyka gridów społecznościowych

W latach dziewięćdziesiątych dwudziestego wieku, kiedy komputery osobiste stały się powszechne, a większość z nich była stale podłączona do sieci, narodziła się idea wykorzystania ich wolnej mocy obliczeniowej [6]. Pierwszym powszechnie znanym projektem w tym zakresie był projekt Seti@HOME [9]. Wykazano w nim, że wykorzystując moc komputerów należących do ochotników, można, nie ponosząc prawie żadnych nakładów na sprzęt, wykonać czasochłonne obliczenia. Osiągnięcia tego projektu należy rozpatrywać na dwóch płaszczyznach – informatycznej oraz socjologicznej. Wykazano techniczną możliwość zbudowania systemu do dużych obliczeń społecznościowych. Jednocześnie w socjologii pokazano, że wielu ludzi jest skłonnych poświęcić czas na zainstalowanie opro-

gramowania i udostępnić czas swoich komputerów na wykonanie zadań, które uważają za społecznie pożyteczne.

W roku 2002 na uniwersytecie Berkeley, w którym prowadzono projekt Seti, stworzono oprogramowanie BOINC [7], znacznie bardziej elastyczne niż oryginalne oprogramowanie Seti. Umożliwia ono między innymi uczestnictwo węzła obliczeniowego w kilku projektach.

Ciekawym przykładem gridu społecznościowego jest AlmereGrid [4], który powstał jako pierwszy na świecie grid miejski. Jest on oparty o sieć miejską miasta Almere. Sieć ta ma znacznie większą przepustowość nić większość sieci rozległych. Szybkie połączenia pomiędzy elementami tego systemu powodują specyficzną charakterystykę zadań i podejście do alokacji zasobów. Ze względu na te elementy AlmereGrid jest bardzo zbliżony systemów typu desktop grid. Jego podstawowe zastosowania obejmują projekty naukowe takie jak: symulacje dokowania białek, czy obróbka statystyczna danych medycznych. Równolegle toczą się prace nad zastosowaniem tego gridu w przemyśle. Jednym z potencjalnie możliwych zastosowań systemów typu desktop grid jest rendering, czyli tworzenie obrazów obiektów trójwymiarowych. Zadanie to jest konieczne do tworzenia filmów animowanych i wizualizacji projektów architektonicznych.

4. Charakterystyka systemów typu desktop grid

Mniej więcej w tym samym czasie co gridy społecznościowe pojawił się pomysł tworzenia systemów typu desktop grid, czyli systemów wykorzystujących czas stacji roboczych pracowników w trakcie gdy są one nieużywane [58, 61]. Na początku wykorzystywano stacje robocze, mocne maszyny, zbyt drogie do użytku domowego. Pionierem wykorzystania tego typu zasobów był projekt Condor [75]. W późniejszym okresie różnica pomiędzy stacjami roboczymi, a komputerami osobistymi się zatarła i zaczęła się także zacierać różnica pomiędzy systemami typu desktop grid a gridami społecznościowymi. Istnieje jednak zasadnicza różnica pomiędzy systemami typu desktop grid a gridami społecznościowymi. W gridzie społecznościowym dane trafiają na komputer ochotnika, który może zrobić z nimi, co zechce. Wprawdzie można się bronić przed próbą sfałszowania wyników przez ochotnika, jednak zawsze będzie on mógł uzyskać dostęp do przetwarzanych danych. Z tego powodu wielu rodzajów danych nie można przetwarzać w gridzie społecznościowym np. danych medycznych lub finansowych.

W odróżnieniu od gridu społecznościowego, w systemie typu desktop grid możliwe jest kontrolowanie przez administratorów, co się dzieje na węźle obliczeniowym, gdyż w wielu instytucjach użytkownik nie ma praw administratora na swoim komputerze osobistym.

Inną różnicą jest specyfika obciążenia komputerów i sieci. Pracownicy danej instytucji pracują na ogół w określonych godzinach, a poza tym czasem komputery stoją nieużywane (np. przez noc). Można wtedy przyjąć, że jeśli dany komputer osobisty jest nieobciążony o godzinie osiemnastej to będzie w takim stanie przez najbliższe kilkanaście godzin. W związku z powyższym można tak rozkładać zlecenia, żeby te o długim czasie obliczeń były zlecane w nocy.

Kolejnym elementem odróżniającym systemy typu desktop grid od gridów społecznościowych są szybkie połączenia sieciowe między komputerami. Umożliwiają one w systemie typu desktop grid rezygnację z zapisywania wyników pośrednich w trakcie obliczeń. Zamiast tego obliczenia są dzielone na podzadania, których wyniki pośrednie przesyła się do magazynu gridowego.

Opisane różnice pokazują, że mimo pozornego podobieństwa gridów typu desktop do gridów społecznościowych, stanowią one odrębną klasę i przedmiot osobnych badań.

4.1 Przydział zasobów i szeregowanie w gridach społecznościowych i systemach typu desktop grid

W gridach społecznościowych i systemach typu desktop grid przydzielanie zasobów i szeregowanie zadań wymaga rozwiązywania innych problemów niż w gridach usługowych Zasoby tych typów gridów są znacznie bardziej zawodne i mniej przewidywalne od zasobów gridów usługowych. W gridach społecznościowych dostawcy węzłów mogą zwracać celowo błędne rezultaty obliczeń. Z tych powodów stosuje się na ogół dublowanie zadań. W gridach społecznościowych jest to nawet potrajanie, żeby móc przeprowadzić głosowanie poprawności wyniku. Ponadto od pewnego czasu węzły gridów społecznościowych mogą jednocześnie pracować dla kilku gridów, co powoduje dodatkowe problemy z szeregowaniem. Większość z nich została opisana w pracy [10]. Kolejnym elementem komplikującym szeregowanie w gridach klasy desktop i społecznościowych jest dostępność maszyn i sieci w cyklu dobowym i tygodniowym.

Problemy z alokacją zasobów i szeregowaniem w gridach opartych na komputerach osobistych powodują, że niezwykle istotne staje się symulowanie takich systemów. W gridach opartych na komputerach osobistych symulacje służą sprawdzaniu przygotowanych polityk szeregowania i alokacji zasobów. Wsparcie tworzenia polityk szeregowania i systemów alokacji zasobów było jedną z motywacji dla badań opisanych w tej pracy – symulacji systemów typu desktop grid. Umożliwi to tworzenie lepszych polityk, które pozwolą na efektywniejsze obliczenia, poprzez lepsze wykorzystanie zasobów. Pozwala uniknąć zablokowania obliczeń spowodowanego przeciążeniem jednego z elementów systemu, np. sieci. Pomoże także w podawaniu przybliżonego czasu zakończenia zlecenia, co jest

bardzo istotne dla użytkownika, który chciałby wiedzieć czy wyniki dostanie za pięć minut, za godzinę, czy za kilka dni.

Rozdział III

Opis technologii gridowych i systemów typu desktop grid

1. Wprowadzenie

Gridy komputerowe są to rozproszone systemy komputerowe, które mają umożliwić użytkownikom łatwe udostępnianie i korzystanie z zasobów komputerowych [37]. Łatwe korzystanie z zasobów wymaga by system komputerowy posiadał jednorodny system uwierzytelniania. System ten nie może wymagać ciągłego podawania hasła, lecz co najwyżej raz, na początku pracy. Użytkownik powinien korzystać ze zunifikowanego interfejsu do wszystkich zdalnych systemów, niezależnego od ich architektury. Możliwe jest różnicowanie interfejsu w zależności od rodzaju zasobu z którego użytkownik chce korzystać. Łatwe udostępnianie oznacza, że administratorzy nie muszą poświęcać dużo czasu na włączenia nowego zasobu, a nowy użytkownik nie musi się zgłaszać do wszystkich administratorów, żeby uzyskać dostęp do zasobów. Wadą gridów usługowych jest brak możliwości udostępniania zasobów komputerów osobistych przez ich właścicieli. Problem ten rozwiązują częściowo gridy społecznościowe i systemy typu desktop grid, pozwalają na wykorzystanie zasobów komputerów osobistych, np. mocy obliczeniowej czy dysków.

W początkowej fazie rozwoju powstało kilka systemów gridowych stworzonych na potrzeby obliczeń (UNICORE [34], Globus Toolkit [36]). Jednocześnie rozwijał się Condor – system mający służyć użyciu dostępnych zasobów stacji roboczych w ramach instytucji. W większości znanych gridów można wydzielić oprogramowanie warstwy pośredniej (ang. middleware), które umożliwia łatwe budowanie gridu. Oprogramowanie to rozwiązuje wiele problemów związanych z komunikacją i autoryzacją. Wyewoluowało ono z pierwszych gridów i zawiera bazowe elementy, umożliwiające budowę własnego gridu. Najpopularniejsze ro-

dzaje oprogramowania warstwy pośredniej w gridach usługowych to: UNICORE [34, 66], Globus Toolkit [36], gLite [53] i ARC [33, 69] oraz w gridach społecznościowych: BOINC [7]. Przykłady gridów usługowych to Nurdugrid [31], Egee [54], Chemomentum [67]. Seti@Home [9] i Folding@Home [52] to przykłady gridów społecznościowych.

Obecnie badana jest możliwość łączenia gridów usługowych i systemów typu desktop grid [55, 12, 49]. Większość połączeń polega na tworzeniu dodatkowego elementu, który przepisuje informacje z jednego systemu do drugiego. Poważną wadą tych rozwiązań jest brak możliwości pełnej wymiany informacji o bezpieczeństwie danych oraz uprawnieniach użytkownika. Rozwiązanie tego problemu jest jedną z motywacji tej pracy. Aby to osiągnąć postanowiono stworzyć system typu desktop grid oparty o jedno z najpopularniejszych oprogramowań warstwy pośredniej dla gridu usługowego – UNICORE. Dzięki temu w systemie typu dekstop grid będą stosowane dokładnie te same metody autoryzacji i uwierzytelniania co w gridzie usługowym. Oznacza to, że możliwe jest stworzenie systemu typu desktop grid z zaawansowanym mechanizmem bezpieczeństwa, zapewniającym dostatecznie wysoki poziom bezpieczeństwa danych i zarazem umożliwiającym interakcje z gridami usługowymi. Taki system bezpieczeństwa umożliwia udostępnianie zasobów komputera użytkownika gridu, w trakcie gdy ma on włączone oprogramowanie klienta gridu usługowego. Stworzenie systemu typu desktop grid w oparciu o oprogramowanie UNICORE potwierdziło wysoką wydajność i unikalność tego oprogramowania.

2. Wykorzystane technologie i standardy

Początkowo, kiedy sieć była bardzo wolna, tworzono wyspecjalizowane protokoły sieciowe dla systemów gridowych. Specjalizacja wymagała bardzo efektywnego zapisywania informacji. Takie podejście do komunikacji miało wiele zalet. Jego głównymi wadami były brak kompatybilności między różnymi systemami oraz trudne przeglądanie komunikatów, niezbędne przy poprawianiu oprogramowania. Przykładem może tu być problem oprogramowania UNICORE, które do komunikacji używał serializowanych obiektów Javy. Skutkiem był brak możliwości komunikacji z UNICORE poprzez programy pisane w innych językach. Poważny problem stanowiły różne wersje Javy odmiennie serializujące obiekty, przez co cały system musiał korzystać z tej samej wersji maszyny wirtualnej.

W ciągu ostatnich kilku lat, ze względu na brak kompatybilności pomiędzy różnymi systemami warstwy pośredniej (a często także ze względu na brak kompatybilności pomiędzy różnymi wersjami tego samego oprogramowania), podjęto próby standaryzacji oprogramowania gridowego. Stwierdzono, że najważniejsza

jest unifikacja oraz łatwość debugowania komunikatów przez człowieka. Spowodowało to powstanie nowych standardów opartych na XML i sieci WWW.

Specyfikacje dotyczące gridów obejmują wszystkie warstwy modelu ISO/OSI. Sprecyzowana jest również semantyka dotycząca elementów gridu. W tej części pracy zostaną omówione protokoły służące komunikacji w gridach według kolejności logicznej od najniższych warstw modelu OSI do najwyższych.

2.1 Protokoły sieciowe wykorzystywane w gridach

Na niższych warstwach sieci nie ma konieczności definiowania protokołów specyficznych dla gridu. Znacznie lepsze rezultaty daje zastosowanie protokołów ogólnego przeznaczenia. Protokoły te, są dobrze opracowane i istnieje dla nich wsparcie ze strony infrastruktury. Dostępne są także gotowe biblioteki umożliwiające korzystanie z nich, co znacząco obniża koszt stworzenia i utrzymania oprogramowania.

Dla warstw sieciowej i transportowej modelu ISO/OSI używa się w gridach protokołów TCP/IP. Protokoły te stanowią podstawę działania sieci Internet.

Warstwy sieciowa i transportowa stanowią jednak tylko podstawę, nad którą konieczne jest zdefiniowanie protokołów do zdalnego uruchamiania procedur, przesyłania komunikatów oraz zarządzania zasobami. Do tego celu służą protokoły z rodziny WebService, inaczej nazywanych serwisami webowymi.

Serwisy webowe

W latach dziewięćdziesiątych dwudziestego wieku powstała sieć WWW. Stanowi ona zbiór serwerów umożliwiających komunikację przy użyciu protokołu HTTP. W założeniach miała ona służyć udostępnianiu treści użytkownikom. Na początku stosowano wyłącznie komunikację człowiek – program (przeglądarka WWW) – sieć – serwer WWW. Rozwój sieci WWW był nie tylko ilościowy, ale także jakościowy. Serwery WWW zaczęły udostępniać treść generowaną dynamicznie. Ze względu na ilość informacji konieczne stało się umożliwienie komunikacji człowiek – program (przeglądarka WWW) – sieć - serwer WWW – sieć – serwer WWW. Pośredni serwer służy do gromadzenia i przetwarzania informacji generowanej przez inne serwery. Przykładem jest centralny serwer wyszukiwania połączeń lotniczych, który umożliwia jednoczesne przeszukiwanie serwerów różnych linii lotniczych. Serwisy WWW, umożliwiające dostęp innym serwisom przy użyciu protokołu HTTP jako warstwy sesji, nazywa się ogólnie serwisami webowymi. Specyfikacja tych serwisów opiera się na rekomendacjach W3C [2]. Specyfikacje zajmują się trzema następującymi dziedzinami [28]:

1. Komunikacją pomiędzy klientem a serwisem przy użyciu zdalnego wołania metod. Opis komunikacji zdefiniowany jest w protokole SOAP;

2. Metodą definiowania interfejsu serwisu – opisem typów usług udostępnianych przez serwis. Jest ona opisana w specyfikacji języka WSDL;

3. Odnajdowaniem serwisów – standard UDDI.

SOAP (Simple Object Access Protocol) jest protokołem umożliwiającym kodowanie komunikatów, takich jak wołanie funkcji oraz wyniku wykonania danej funkcji w języku XML. Umożliwia on zarówno komunikację synchroniczną wykorzystywaną w mechanizmach RPC (remote procedure call), jak i asynchroniczną – przekazywanie wiadomości. Protokół ten pozwala na wykorzystanie różnych protokołów transportowych – między innymi HTTP i SMTP. Wprowadzenie tego standardu umożliwia komunikację ze zdalnymi usługami niezależnie od języków, w jakich są one zaimplementowane. Dodatkowo oparcie standardu na formacie XML pozwala na debugowanie modułów poprzez podglądanie komunikacji. Protokół ten dotyczy głównie składni definiującej najprostsze typy i umożliwia używanie własnych typów. Protokół ten łączy elementy warstwy prezentacji i aplikacji.

WSDL (Web Service Description Language) to język formalny służący opisowi serwisów webowych. Umożliwia on zdefiniowanie serwisu oraz sposobu dostępu do niego. Wprawdzie nie zawiera informacji semantycznej, jednak zawarta w nim informacja o nazwach obiektów koniecznych do komunikacji zastępuje opis semantyczny definicją typów danych. Konieczna jest wiedza o tym, co oznaczają konkretne nazwy – opisana w standardach OGSA i WSRF. Protokół ten stanowi uzupełnienie protokołu SOAP i należy go przypisać tym samym warstwom.

UDDI (Universal Description, Discovery, and Integration) umożliwia użytkownikom (programom) odnajdowanie serwisów świadczących konkretne usługi, a także ustalanie, jak korzystać z danego serwisu. Pozwala także na uniezależnianie warstwy aplikacji od niższych warstw.

WSRF (Web Service Resource Framework) Ze względu na bezstanowość sieci WWW i faktyczną stanowość pewnych obiektów zaszła konieczność zdefiniowania protokołów umożliwiających definiowanie i odwoływanie się do trwałych obiektów – jak na przykład obliczenie czy magazyn w gridzie. WSRF jest to zbiór standardów organizacji Oasis. WS Resource, zdefiniowany w [3, 29], jest sposobem wyrażania relacji pomiędzy serwisami, które są bezstanowe, a zasobem – czyli obiektem, który posiada stan.

2.2 Protokoły specyficzne dla gridów

Na wyższych warstwach protokół musi już korzystać z pojęć dotyczących gridów. Konieczne jest wykorzystanie pojęć takich jak obliczenie, węzeł roboczy, zasoby etc... Poniżej opisane zostały standardy i protokoły specyficzne dla gridu.

OGSA Początkowo łączenie gridów opartych na różnych oprogramowaniach warstwy pośredniej było w zasadzie niemożliwe. Część pojęć była różnie rozumiana przez twórców różnych systemów gridowych. Powstały opracowania próbujące wytłumaczyć czym jest grid i jak należy nazywać jego składowe elementy [37, 40]. Na bazie dokumentu [40] zdefiniowano standard OGSA (Open Grid Service Architecture) [38]. Standard ten uznany został przez Global Grid Forum. Jest to zbiór wytycznych do budowy gridów. Zawiera on definicję różnych elementów gridów, takich jak zasoby obliczenia, czy akcje. Umożliwia to łączenie gridów, rozwiązując problemy pojęciowe.

JSDL (Job Submission Description Language) jest językiem służącym do opisu zlecenia obliczeniowego dla gridu. Jest on zdefiniowany w rekomendacji Global Grid Forum [11]. Język ten w dość pełny sposób oddaje różne aspekty zlecenia takie jak podział na zadania, wymagane zasoby etc. Powstał, by umożliwić łatwiejsze łączenie gridów opartych na różnym oprogramowaniu warstwy pośredniej, jednak nie jest jeszcze wszędzie stosowany.

W pełni obsługuje go UNICORE 6 oraz gLite. Dodatkowo istnieją mapowania dla najpopularniejszych rodzajów oprogramowania warstwy pośredniej takich jak Globus i Condor.

Język ten jest wdrażany stopniowo we wszystkie rodzaje oprogramowania warstwy pośredniej. Aby zapewnić w przyszłości kompatybilność tworzonego systemu typu desktop grid z innymi systemami warto oprzeć go na programowaniu które już korzysta z tego języka.

3. Bezpieczeństwo

Problemy związane z bezpieczeństwem można klasycznie podzielić na problemy z autoryzacją i uwierzytelnianiem. Uwierzytelnianie służy stwierdzeniu, że strona komunikacji jest rzeczywiście tym, za kogo się podaje. Autoryzacja natomiast to określenie, czy dana strona ma prawo do określonej czynności.

Początkowo, jak to opisano w pracy [39], w systemach gridowych problemem podstawowym było zapewnienie unifikacji, tj. możliwości pojedynczego logowania użytkownika oraz zunifikowanego interpretowania przez elementy gridu informacji o uprawnieniach użytkownika. Rozwiązania oparto na kluczach prywatnych

i certyfikacji, dzięki czemu w prosty sposób uzyskano bazową funkcjonalność - uwierzytelnianie. Rozwiązanie problemu autoryzacji było oparte na bazach autoryzacyjnych, mówiących wprost o prawach dostępu do zasobu i wirtualnych organizacjach, czyli uogólnionych grupach.

Kluczowym problemem związanym z bezpieczeństwem w gridach jest delegacja zaufania, czyli upoważnienie trzeciej strony do wykonania jakiejś operacji w imieniu upoważniającego. Ma to szczególne znaczenie np. dla problemów szeregowania i alokacji zasobów. Tworząc zlecenie użytkownik nie wie kto je będzie wykonywać, decydować o tym będzie bowiem alokator zasobów.

Aby umożliwić delegację zaufania początkowo używano certyfikatów proxy [77] – opisanych dużo wcześniej w pracy [60]. Certyfikat proxy to certyfikat podpisywany kluczem prywatnym wystawcy, który nie jest centrum autoryzacyjnym. Umożliwia on certyfikowanej stronie wykonywanie dowolnej akcji w imieniu podpisującego. Było to możliwe, bez dodatkowych czynności administracyjnych, dzięki zastosowaniu infrastruktury kluczy i certyfikatów.

Niestety certyfikaty proxy mają bardzo dużo wad, z których najważniejsze to [17]:

- problem z bezpieczeństwem klucza prywatnego proxy. Klucz jest zabezpieczany wyłącznie na poziomie systemu plików, a zdobycie go daje pełne prawa wystawcy. Aby zwiększyć bezpieczeństwo certyfikaty proxy mają bardzo krótką ważność: 24 lub 48 godzin, jednak jest to raczej obejście niż rozwiązanie problemu;

- problem z ustaleniem kto używał certyfikatu proxy i jaka była kolejność działań.

Mechanizm delegacji zaufania wprost jest wolny od tych wad [68]. Polega on na dołączeniu do zlecenia asercji (informacji podpisanej kluczem prywatnym) stwierdzającej, że właściciel odpowiedniego certyfikatu ma prawo wykonać daną akcję jako wystawca asercji. Osoba delegowana może przekazać delegację dalej, ale zawsze jest to delegacja dotycząca konkretnej czynności, co nie powoduje zmniejszenia bezpieczeństwa.

W gridach społecznościowych podejście do bezpieczeństwa jest zupełnie inne niż w gridach usługowych. Jeśli jakiś węzeł wykonuje zlecenie, jego właściciel i tak może dostać się do tych danych i nie można temu zapobiec. Z drugiej strony, właściciele węzłów chcieliby otrzymać gwarancję, że oprogramowanie, które uruchamiają nie zagrozi ich komputerom. Dane na serwerze są pobierane i umieszczane przez klientów, więc bezpieczeństwo można oprzeć na podobnych zasadach jak w serwerach ftps czy scp. W systemach typu desktop grid sytuacja się komplikuje. Mogą one bowiem przetwarzać dane, które nie powinny być udostępniane poza grupą roboczą. Powoduje to konieczność uwierzytelniania nie tylko

użytkownika, ale także węzła. Model bezpieczeństwa dla systemów typu desktop grid powinien być zatem oparty na zasadach dotyczących gridów usługowych.

4. Oprogramowanie warstwy pośredniej dla gridów usługowych

Systemy typu desktop grid ze względów wydajnościowych bazowały na oprogramowaniu przeznaczonym dla gridów społecznościowych. Obecnie wydajność serwerów nie jest już problemem, natomiast wyzwaniem staje się integracja z gridami usługowymi. Konieczne jest także zapewnianie bezpieczeństwa danych i maszyn. Budowa systemu typu desktop grid opartego na oprogramowaniu warstwy pośredniej gridów usługowych rozwiązuje oba problemy. Aby wybrać oprogramowanie najlepiej nadające się do tego celu należy porównać najpopularniejsze rodzaje tego oprogramowania.

Globus Toolkit

Globus Toolkit [36] był przez pewien czas najpopularniejszym oprogramowaniem warstwy pośredniej. Jest on bardzo zaawansowany i ma bardzo duże możliwości, co niekoniecznie jest jego zaletą. Właściwa konfiguracja i zestawienie ze sobą poszczególnych elementów przysparza wiele trudności i powoduje, że nie istnieje gotowa do użytku wersja „pudełkowa" tego oprogramowania. Konieczne jest tworzenie na bazie Gloobus toolkit własnego oprogramowania gridowego i dopiero ono może być łatwe w obsłudze. Obecna wersja Gloobus toolkit spełnia już wszystkie standardy opisane w części 2, jednak nadal proste i samodzielne użycie stanowi dość duży problem. Polityka bezpieczeństwa Glubus Toolkit bazuje na kluczach publicznych i certyfikatach proxy. Na Globus Toolkit oparto takie oprogramowania warstwy pośredniej jak ARC i gLite.

UNICORE

UNICORE jest oprogramowaniem, które powstało w celu umożliwienia łatwego i zunifikowanego dostępu do zasobów [34]. Nie zawiera on co prawda tylu dodatkowych funkcji co Globus Toolkit, jednak dzięki temu jego wersje są stabilne i wszystkie funkcje działają. W początkowych wersjach (do wersji 6) komunikacja w UNICORE odbywała się przy użyciu serializowanych obiektów [34, 66]. Rozwiązanie to miało tę wadę, że na wszystkich komputerach połączonych w grid trzeba było używać tej samej wersji Javy. W wersji 6 postanowiono zmienić komunikację w UNICORE na zgodną ze standardami opartami na serwisach webowych, co spowodowało, że oprogramowanie to jest jeszcze łatwiejsze w zarządzaniu

i utrzymaniu [64]. Obecna wersja systemu spełnia wszystkie standardy opisane w części 2. Polityka bezpieczeństwa UNICORE opiera się na kluczach prywatnych i bezpośrednich delegacjach zaufania. Dużą zaletą tego systemu jest łatwo rozszerzany interfejs graficzny.

ARC

ARC [33, 69] jest oprogramowaniem powstałym na bazie GlobusToolkit. Powstało ono na potrzeby systemu NorduGrid. Jest bardzo łatwe w użyciu – jednak wymaga homogenicznych systemów. Wszystkie węzły pracują pod kontrolą jednego systemu operacyjnego i bardzo trudno dołączyć komputery oparte na innych architekturach niż domyślna. ARC zapewnia wyłącznie podstawowe serwisy gridowe, a wszystkie dodatki dotyczące aplikacji i łatwości wykorzystania należy pisać tworząc docelowy grid. ARC nie dostarcza narzędzi służących do tworzenia interfejsu graficznego. Zapewnia on wyłącznie klienta tekstowego, co powoduje, że dla wielu użytkowników może być zbyt trudny w użytkowaniu. Do delegacji zaufania ARC korzysta z certyfikatów proxy.

GLite

GLite [54, 53] jest oprogramowaniem warstwy pośredniej stworzonym na potrzeby projektu EGEE, zapewniającego moc obliczeniową dla eksperymentów prowadzonych w LHC. GLite jest prostszym i „lżejszym" (lite) oprogramowaniem warstwy pośredniej niż GlobusToolkit i Condor, na których bazuje. Architektura oprogramowania GLite oparta jest na usługach (serwisach) zaimplementowanych zgodnie ze standardami WebSerwisów. GLite jest także zgodny ze standardami WSRF i OGSA. Ma on większość funkcji koniecznych do działania współczesnego gridu, takich jak: replikacja danych i zarządzanie złożonymi zadaniami. Wymaga on używania bardzo podobnych węzłów i jednocześnie trudne jest wykorzystanie do obliczeń wysoko specjalizowanych komputerów. Polityka bezpieczeństwa gLite bazuje na kluczach publicznych i wirtualnych organizacjach. Jego wadą jest delegacja uprawnień korzystająca z certyfikatów proxy.

5. Rozwiązania typu desktop grid i gridy społecznościowe

W roku 1984 w ramach oprogramowania Condor [75] rozpoczęto wykorzystywanie wolnej mocy stacji roboczych znajdujących się w jednej instytucji. System bezpieczeństwa w tym oprogramowaniu był bardzo uproszczony, gdyż stacje działały w zaufanej sieci zarządzanej przez jednego administratora. W latach

dziewięćdziesiątych kiedy pojawiła się sieć Internet pojawiły się pierwsze gridy społecznościowe. The Great Internet Mersenne Prime Search (GIMPS), służył do poszukiwania liczb pierwszych - rozpoczął on swoją działalność w roku 1996. W roku 1999 rozpoczął się projekt Seti@Home [9], który jest najbardziej znanym projektem wykorzystania wolnego czasu komputerów należących do niezależnych ochotników.

Szybki rozwój gridów społecznościowych i typu desktop rozpoczął się dopiero po roku 2002, kiedy powstało oprogramowanie warstwy pośredniej BOINC [7], a projekt Seti wykazał potencjalnie dużą moc gridów społecznościowych. Równolegle w systemie Condor wprowadzono zaawansowany system bezpieczeństw, dzięki czemu system ten stał się bardzo uniwersalny narzędziem. Około roku 2005 zaczęły powstawać systemy typu desktop grid w których komputery należały do więcej niż jednej instytucji.

Poniżej opisano najważniejsze implementacje oprogramowania warstwy pośredniej dla gridów społecznościowych i gridów typu desktop, z których korzysta się obecnie.

Condor

Condor [75] jest systemem służącym do zarządzania obliczeniami. Umożliwia on wykorzystanie do obliczeń zarówno dedykowanych maszyn jak i wolnego czasu komputerów osobistych. Takie podejście powoduje, że należy go zakwalifikować do gridów typu desktop. Istnieje rozszerzenie CondorG [41], które pozwala na wykorzystanie gridów opartych na systemach Globus Toolkit i UNICORE do obliczeń zlecanych systemowi Condor. Możliwe jest także wysyłanie zleceń z gridów usługowych do zasobów zarządzanych przez system Condor. System Condor jest rozwijany od roku 1985. Napisano go w języku C++. Obecna wersja posiada rozwinięty system kolejkowania i alokacji zasobów. Jest w stanie obsługiwać zależności pomiędzy zadaniami przedstawione w postaci acyklicznego grafu skierowanego. Przykładem jego zastosowań są symulacje Monte-Carlo [13] wykorzystywane w wielu projektach jak modelowanie układów planetarnych czy przetwarzanie dużych ilości danych naukowych [25].

Jedyną, ale bardzo poważną wadą CondorG są problemy z właściwym mapowaniem użytkowników gridu usługowego na użytkowników systemu Condor.

BOINC

BOINC (Berkeley Open Infrastructure Infrastructure for Network Computing) [7] jest oprogramowaniem warstwy pośredniej stworzonym na bazie doświadczeń projektu Seti@HOME. Jest to obecnie najbardziej popularna platforma do obliczeń społecznościowych. Całą komunikację w systemie BOINC inicjuje węzeł oblicze-

niowy, co daje bardzo wysoką wydajność i nieduże obciążenie sieci. Rozwiązanie to umożliwia także obejście takich problemów jak NAT i firewall, za którymi na ogół umieszczony jest węzeł. Pozwala ono także na odłączanie węzła obliczeniowego od Internetu. Dużą zaletą BOINCa jest możliwość używania jednego węzła obliczeniowego w kilku sieciach. Wiąże się to z korzystaniem z większej ilości centralnych serwerów – jeden na projekt i nie ma możliwości ustalania pomiędzy zlecającymi zadania, które z nich ma najwyższy priorytet. Możliwe jest ustalenie priorytetów dla projektów przez właściciela węzła obliczeniowego. Bardzo ważnym elementem systemu BOINC jest checkpointowanie, czyli zapisywanie wyników pośrednich. Jest to konieczne w przypadku długotrwałych obliczeń. Bezpieczeństwo w systemie BOINC ze względu na to, że jest on systemem gridów społecznościowych, jest skrajnie uproszczone. Bezpieczeństwo danych do pobrania jest zapewniane przez ukrywanie adresu URL, z którego dane mają być pobrane. Prosta autoryzacja następuje przy ładowaniu wyników na serwer. Ma ono jednak także wiele wad. Największą z nich jest brak wiedzy, o stanie w jakim znajdują się obliczenia i węzeł. Jej znaczenie nie jest bardzo duże w sytuacji gridu społecznościowego, w którym obliczenia ze względu na swój rozmiar trwają długo i zlecający nie potrzebuje wyników natychmiast. W przypadku systemu typu desktop grid, w którym czasem użytkownicy mogą zlecać mniejsze zadania w zamian oczekując wyników najszybciej jak to tylko możliwe, wada ta staje się znacząca.

Podsumowując – BOINC jest bardzo zaawansowanym systemem dobrze dopasowanym do obliczeń społecznościowych, jednak zastosowanie go w gridach typu desktop wymaga wielu przeróbek.

XtremWeb

XtremWeb, opisany w pracach [23, 20] wraz z rozwinięciem XtremWebCH [5], jest narzędziem, które miało umożliwić obliczenia w architekturze peer to peer. W odróżnieniu od architektury klient-serwer, która jest zaimplementowana w systemie BOINC, umożliwia one obliczenia, podczas których węzeł obliczeniowy komunikuje się bezpośrednio z innymi węzłami (lub składnicami). Dzięki temu możliwy jest bardzo zaawansowany przydział zadań do maszyn. Odciąża to znacząco sieć. Odciążenie jest możliwe także w desktop gridach opartych na systemie BOINC, jeśli się odpowiednio rozmieści serwery projektów i uniknie centralizacji. Dużą zaletą tego oprogramowania jest oparcie komunikacji na standardach bazujących na WebSerwices takich jak WSDL, SOAP czy UDDI, co ułatwia integrację z gridem usługowym. Wadą oprogramowania XtremWeb jako systemu typu desktop grid jest brak zaawansowanych mechanizmów dostępu do maszyn. Przykładem zastosowań gridu XTremWeb opisanym w pracy [23]

jest zrównoleglenie programu do analizy cząstek promieniowania kosmicznego – Aires [45].

AlmereGrid

AlmereGrid opisany w [4] jest przykładem systemu typu desktop grid, którego organizacja nie opiera się na grupach badawczych czy instytucjach, lecz jest ograniczana terytorialnie. To pierwszy na świecie grid miejski. Nieduże oddalenie komputerów powoduje ułatwia skorzystanie z wysokowydajnej sieci miejskiej. Umożliwia to większą ziarnistość obliczeń, gdyż narzut na przesyłanie danych jest znacznie mniejszy niż w ogólnoświatowych czy ogólnokrajowyh systemach typu dekstop grid. AlmereGrid bazuje na oprogramowaniu BOINC. System ten może być wykorzystywany zarówno do celów społecznych, takich jako projektowanie leków, jak również komercyjnie.

Entropia

Entropia opisana w [26, 19] jest przykładem komercyjnego systemu typu desktop grid mającego za zadanie wykorzystanie nieobciążonych komputerów. Daje on możliwość użycia większej ilości aplikacji dzięki uruchamianiu aplikacji obliczeniowych na węzłach w „piaskownicy", co zapewnia bezpieczeństwo węzłów. Dodatkowo, dzięki piaskownicy możliwe jest szyfrowanie danych przechowywanych na węzłach, co ma zapewniać ich bezpieczeństwo. System ten zawiera autoryzacje użytkowników – w przypadku gridu komercyjnego jest to konieczne. System posiada jednak pewne wady: jest zamknięty w ramach jednej firmy i nie jest przeznaczony do integracji z gridem usługowym. Przykłady zastosowania tego systemu to klasyczne aplikacjami rozproszonymi dla potrzeb nauki takie jak analiza sekwencji białek czy przewidywanie struktury cząstek. Możliwe są także zastosowania komercyjne, m.in. analiza ryzyka na potrzeby instytucji finansowych przy użyciu metod Monte Carlo.

SZTAKI Desktop grid

SZTAKI Desktop Grid [57] umożliwia wykorzystanie BOINCa do obliczeń desktop gridowych. Wprowadza on hierarchie zadań [56], co pozwala na łatwe wymuszenie kolejności wykonania obliczeń i jest konieczne w przypadku systemu, do którego zlecenia mają napływać od użytkowników nie będących administratorami zarządcy. System ten jest badzo rozbudowaną wersją systemu BOINC, w którym tylko właściciele serwera-zarządcy mogli zlecać zadania i nie było potrzeby tworzenia mechanizmu do ustalania konieczności zleceń – po prostu były one wykonywane zgodnie z pewnym porządkiem.

W Sztaki rozbudowano znacząco model bezpieczeństwa BOINCa, wprowadzając dwustronne relacje zaufania pomiędzy węzłem obliczeniowym i użytkownikami. Wykorzystano do tego celu infrastrukturę kluczy X.509 do podpisywania komunikacji.

Poważną wadą systemu SZTAKI jest brak łatwej integracji ze zwykłym gridem usługowym, jednak prowadzi się prace w tym kierunku [12].

Podsumowanie

Zarówno systemy wspierające pracę gridów usługowych jak i systemy wspierające pracę gridów społecznościowych są już bardzo zaawansowane. Duże wyzwanie stanowi natomiast łączenie tych systemów. Większość gridów usługowych nie daje bowiem możliwości wykorzystania wielu niezależnych węzłów liczących, natomiast gridy społecznościowe i gridy typu desktop mają jeszcze zbyt słabe mechanizmy bezpieczeństwa. Dodatkowe utrudnienie stanowi wymiana informacji dotyczących bezpieczeństwa. Podejmowane są próby tworzenia takich połączeń, na przykład w przypadku oprogramowania SZTAKI.

Uzasadnione wydaje się sprawdzenie, czy istnieje możliwość stworzenia wydajnego systemu klasy desktop grid, który dawałby się łatwo zintegrować z gridami usługowymi i opierałby się na oprogramowaniu warstwy pośredniej gridu usługowego, dysponującego dostatecznie zaawansowanymi mechanizmami bezpieczeństwa.

Rozdział IV

Modelowanie systemów gridowych

W literaturze można znaleźć informacje o wielu próbach symulacji systemów gridowych. Początkowo większość symulacji odnosiła się do gridów usługowych i nadal zdecydowana większość symulacji dotyczy gridów usługowych. Obecnie pojawiają się zestawienia różnych symulatorów gridowych [32, 62, 71, 59], jednak porównania te dotyczą symulacji gridów usługowych. Symulatory gridów usługowych z założenia nie uwzględniają awarii węzłów obliczeniowych, co jest kluczowe dla gridów społecznościowych i systemów typu desktop grid, gdzie wyłączenia węzłów często się zdarzają.

Większość symulatorów gridów usługowych nie uwzględniała także zmian ruchu w sieci lub ewentualnie w późniejszych wersjach próbowała je symulować, zakładając pełną znajomość topologii, co dla systemów typu desktop grid jest często niemożliwe. W pracy [10] zwrócono uwagę na brak symulatora gridów społecznościowych i konieczność stworzenia takiego narzędzia. Poniżej przedstawiono krótkie zestawienie istniejących symulatorów gridów. Oceniono w nim ich przydatność do symulacji gridów społecznościowych i systemów typu desktop grid. Ze względu na rosnącą popularność systemu BOINC powstało kilka symulatorów gridów społecznościowych. Natomiast ze względu na małą popularność systemów typu desktop grid, ich symulacje nie były praktycznie realizowane.

1. Symulacje i symulatory gridów usługowych

Bricks

Bricks [73] został stworzony do mierzenia wydajności szeregowania dla ogólnej klasy systemów – rozproszonych systemów obliczeniowych o zasięgu globalnym. Ponieważ systemy gridowe należą do tej klasy, można go także uznać za symulator gridów. W pierwszym okresie stosowano go głównie do badania

szeregowania pod kątem dostępnych zasobów i ich wydajności. Później zaczęto wykorzystywać go do analizy aplikacji przetwarzających duże ilości danych. Oznaczało to, że do zagadnienia szeregowania dołączył problem replikacji danych. Bardzo dużą zaletą tego symulatora jest uwzględnianie obciążenia i zmian w wydajności sieci. Symulator ten nie ma jednak mechanizmu umożliwiającego generację awarii węzłów.

GridSim

GridSim stworzony na Uniwersytecie w Melbourne jest bardzo zaawansowanym symulatorem. Początkowo służył on do modelowania gridu uwzględniającego czynnik ekonomiczny, czyli koszt obliczeń. W większości systemów gridowych przyjmuje się, że rozliczenia pomiędzy dostawcami zasobów i użytkownikami nie zależą do wykorzystanego faktycznie czasu. W GridSim przyjęto natomiast model gridu, w którym za każdorazowe skorzystanie z zasobu trzeba zapłacić. Każdy zasób ma inną cenę. Takie podejście nastręcza nowe problemy i wymusza konieczność optymalizacji w zakresie *czas obliczeń/cena*. Symulator powstał w wyniku pracy doktorskiej [63]. Posłużył on do stworzenia alokatora zasobów optymalizującego koszt wykonania obliczeń.

Później symulator ten rozbudowano o moduł umożliwiający symulacje sieci [70]. Model sieci użyty w symulatorze wymaga do symulacji znajomości topologii sieci, co bardzo często jest niewykonalne w przypadku desktop gridu. Opiera się ona na dokładnym symulowaniu ruchu pakietów, chociaż na ogół wystarczałaby prosta symulacja mówiąca o ilości dostępnego pasma.

Kolejną wadą symulatora GridSim, jest to, że nie jest on przeznaczony do symulowania systemów, w których awarie węzłów są częste i muszą być brane pod uwagę.

SimGrid

SimGrid [22, 21] jest nieco prostszym narzędziem niż GridSim. Zakłada on możliwość istnienia tylko jednego systemu kolejkującego. Przykłady jego wykorzystania to właśnie optymalizacja takiego systemu w kontekście istniejących aplikacji. Obecnie symulator ten wykorzystuje dość zaawansowane mechanizmy modelowania sieci, takie jak SSFNet [27] i GTNetS [65]. Wszystkie mechanizmy są oparte na modelowaniu sieci jako precyzyjnego mechanizmu i wymagają wyspecyfikowania topologii, a modelowanie ruchu niezależnego od gridu jest dość niedokładne. Symulator nie nadaje się także do symulowania systemów, w których węzły obliczeniowe są bardzo zawodne.

OptorSim

OptorSim zaprojektowano w celu optymalizacji rozmieszczenia danych w systemie DataGrid [15, 16]. Symulator ten był tworzony równolegle z projektem DataGrid. Optymalizacja składała się z dwóch części:

- ustalenia w jaki sposób rozmieszczać kopię danych w testowym systemie,
- przydziału zlecenia, tak by najlepiej wykorzystać ułożone wcześniej kopie danych.

Optymalizacja rozłożenia danych powoduje konieczność uwzględnienia przepustowości sieci. Symulator ten nie uwzględnia jednak wahań tej przepustowości. Interesującą cechą badań prowadzonych przy użyciu tego symulatora było wprowadzenie modeli ekonomicznych w celu optymalizacji pracy gridu. Badania prowadzone przy użyciu tego symulatora stanowią przykład, jak optymalizacja rozmieszczenia danych i właściwy dobór migracji obliczeń do danych może poprawić wydajność systemu rozproszonego. Jak większość symulatorów gridów usługowych, OptorSim nie uwzględniania zawodności węzłów obliczeniowych.

2. Symulacje, symulatory i emulatory gridów społecznościowych oraz gridów typu desktop

SimBOINC

SimBOINC powstał w wyniku pracy doktorskiej [51]. Jest dość prostym symulatorem, nie uwzględniającym awarii węzłów. Zamiast tego wykorzystuje parametr – średnią dostępność węzła. Nie dysponuje mechanizmami umożliwiającymi symulację sieci. Jest jednak interesujący, gdyż pokazuje, że systemy typu desktop grid mogą być wykorzystywane do obliczeń krótkotrwałych (rzędu godzin). Podstawowym zastosowaniem tego symulatora było wspomaganie optymalizacji i poszukiwanie takich rodzajów obliczeń, które w systemach typu desktop grid dawałyby się wykonywać efektywnie. Szczególny nacisk położono na optymalizację szeregowania zadań w obliczeniach, by umożliwić szybkie przetwarzanie zadań przez taki system.

SimBA

SimBA [74] to symulator zdarzeń dla systemu BOINC. Pozwala on na prześledzenie jak faktycznie zachowa się rzeczywisty system po optymalizacji działania elementów centralnych. Jego podstawowym zadaniem było przewidywanie czasu

trwania obliczenia w konkretnym systemie typu desktop grid przy zastosowaniu różnych parametrów dla elementów zarządzających. Symulator korzysta z zapamiętanych śladów – historii działania rzeczywistego systemu. Takie podejście skutkuje bardzo dobrą optymalizacją dla potrzeb obliczeń zbliżonych do wykonywanych na rzeczywistym gridzie. Nie pozwala ono jednak na przewidywanie, jak system się zachowa w sytuacji znacząco innej od takiej, której ślad jest już zapisany. Ponieważ w konkretnych gridach społecznościowych, jak wymieniony w pracy [74] Predictor@Home, zadania są standardowe, nie daje się go praktycznie zastosować do sytuacji gdy chcemy całkowicie zmienić typ zadania. Niemożliwe jest nie zapisywanie wyników pośrednich, skrócenie czasu trwania zadania i sprawdzenie, czy czas pomiędzy zgłoszeniem węzła do systemu i jego awarią (wyłączeniem) jest wystarczy na policzenie podzadań.

EmBOINC

EmBOINC opisany w [35] stanowi przykład emulatora systemu BOINC. Wykorzystuje on tzw. ślady – zapisy pracy systemu – do budowy charakterystyk jego elementów. Umożliwiają one optymalizację pracy niektórych elementów systemu w warunkach bardzo zbliżonych do rzeczywistych. Ponieważ celem stworzenia tego emulatora była jedynie optymalizacja serwera centralnego, w trakcie symulacji używano zwykłego oprogramowania węzła.

Emulatory są w jeszcze większym stopniu niż opisany wcześniej SimBA konstruowane pod kątem dokładnego dobrania parametrów dla wybranego systemu. Praca nad EmBOINC doprowadziła do wysokiej precyzji emulacji zachowania rzeczywistego systemu pod warunkiem, że nie nastąpiły zasadnicze zmiany w charakterystyce jego pracy. Elastyczność emulatorów jest zatem bardzo niska i niemożliwe jest prowadzenie przy ich użyciu badań modelowych systemu przy częstych i dużych zmianach warunków pracy.

3. Zestawienie symulatorów i podsumowanie

W stworzonej na bazie opracowania w pracy [71] tabeli IV.1, zestawiono opisane powyżej symulatory.

Przegląd literatury wskazuje wyraźnie, że jakkolwiek symulacje gridów usługowych są obecnie bardzo zaawansowane, w przypadku systemów typu desktop grid i gridów społecznościowych symulacje są dopiero w początkowej fazie badań. Z roku na roku zwiększa się popularność gridów opartych na komputerach osobistych oraz ich symulacji. W kilku przypadkach w trakcie badań stworzono wspólny symulator dla obu typów gridów opartych na komputerach osobistych, co ze względu na różną specyfikę zadań w gridach typu desktop i gridach spo-

Nazwa narzędzia	Rodzaj symulowanego systemu	Przeznaczenie	Cechy wyróżniające
Bricks	dużoskalowe systemy rozproszone	analiza algorytmów szeregowania	korzysta ze zmiennych obciążeń sieci
Gridsim	grid usługowy	optymalizacja szeregowania zadań	stosuje podejście ekonomiczne, bardzo rozbudowany, w nowych wersjach posiada już model sieci
Simgrid	grid usługowy	optymalizacja szeregowania zadań	uwzględnia obciążenia sieci
OptorSim	grid usługowy	optymalizacja replikacji danych pomiędzy węzłami gridu	uwzględnia topologie i maksymalną przepustowość sieci
SimBOINC	grid społecznościowy oraz system typu desktop grid	optymalizacja aplikacji i szeregowania zadań pod kątem osiągnięcia czasu obróbki zadania akceptowalnego w systemie typu desktop grid	duża elastyczność, pomimo braku zmiennej wydajności sieci bardzo dobry symulator
SimBA	grid społecznościowy – Boinc	przewidywanie czasu wykonania zleceń w systemie BOINC	sterowany śladami – bardzo dobra jakość wyników, jednak mała elastyczność
EmBoinc	grid społecznościowy – Boinc	przewidywanie wydajności systemu i czasu zakończenia obliczeń przy zmianach serwera centralnego	emulator – duża dokładność, mała elastyczność

Tabela IV.1: Zestawienie symulatorów systemów typu desktop grid

łecznościowych może prowadzić do błędnych wyników. Ponadto nie znaleziono w literaturze przykładów symulacji gridu, który miałby jednocześnie następujące cechy:

- umożliwienie symulacji zawodności węzłów, co jest konieczne w symulacjach systemów typu desktop grid i realizuje się w nich na różne sposoby,
- uwzględnienie obciążenia i wahania ruchu w sieci. Zadanie takie realizuje się w symulatorach gridów usługowych, które zajmują się problemem replikacji danych.

Słabości istniejących symulatorów stanowiły motywację do stworzenia modelu systemu gridowego, który spełniałby powyższe kryteria i zbadania jego zachowania w trakcie symulacji.

Rozdział V

Teoretyczny opis systemów typu desktop grid

1. Opis modelu

Gridy społecznościowe oraz systemy typu desktop grid są obecnie nastawione na długotrwałe obliczenia zajmujące czas rzędu co najmniej wielu dni. W przypadku wyłączenia węzła zapisywane są wyniki pośrednie obliczeń, a po ponownym włączeniu obliczenia są wznawiane. Rozwiązanie to jest bardzo efektywne w przypadku obliczeń długotrwałych, natomiast w przypadku obliczeń krótkotrwałych nie ma ono sensu. Użytkownik musi czekać na wznowienie pracy przez węzeł, który wykonywał pewną część obliczenia. Czas oczekiwania w przypadku systemów typu desktop grid i gridów społecznościowych może okazać się długi w porównaniu z czasem obliczenia, które zostało przerwane. W związku z tym możliwe jest, że efektywniej byłoby przeprowadzić to obliczenie na innym węźle. Przyspieszenie komputerów osobistych i dostępność wysokowydajnej sieci umożliwia wykonanie w systemach typu desktop grid obliczeń, dla których czas oczekiwania na wyniki jest rzędu godzin lub minut, a nie dni. W związku z tym warto rozważyć modele dla systemów, w których obliczenia będą krótkie, a wyniki pośrednie nie będą zapisywane.

Projekty takie jak Seti@HOME wykazały, że jest możliwe przeprowadzenie długotrwałych, bardzo dużych obliczeń na olbrzymim systemie, stosując zapisywanie wyników pośrednich. Ciągle jednak nie sprawdzano działania takiego systemu przy mniejszych obliczeniach, z udziałem mniejszej liczby węzłów, natomiast z pominięciem zapisywania – jako pozbawionego sensu – wyników pośrednich. Niezwykle istotna jest odpowiedź na pytanie, jak optymalizować taki system. Nie ma możliwości stworzenia systemu testowego, którego skala odpowiadałaby skali systemu produkcyjnego. Aby to umożliwić, opracowano symulator. Jego pod-

stawową cechą jest bardzo duża elastyczność w zakresie generowania kolejnych zdarzeń. Umożliwia on łatwą zmianę czasu zajścia zdarzenia ze względu na zajście innych zdarzeń. Zastosowanie technologii programowania obiektowego umożliwiło osiągnięcie tej elastyczności bez nakładania zbędnych ograniczeń, które pojawiłyby się przy definiowaniu wszystkich parametrów pracy modelu przy użyciu plików konfiguracyjnych. Symulator, o którym mowa, powinien wykazywać takie właściwości, jak docelowy system, tzn. wszelkie zależności i schematy działania powinny dawać podobne wyniki na symulatorze i na rzeczywistym gridzie.

1.1 Model gridu

Model gridu jest podzielony logicznie na następujące części:

1. Zasoby – węzły obliczeniowe (komputery osobiste), serwery zarządzające, a także elementy sieciowe huby, połączenia ...

2. Obliczenia - operacje wykonywane na węzłach obliczeniowych. Najmniejszym fragmentem obliczeń są zadania – pojedyncze elementy pracy do wykonania. Każde zadanie musi być wykonane w całości na jednym z węzłów obliczeniowych. Dowolny węzeł spełniający wymagania zadania może je wykonać, co oznacza, że w przypadku awarii węzła zadanie może zostać wykonane przez inny węzeł. Zadania są połączone w większe struktury – skierowane grafy acykliczne, które nazywane są zleceniami.

3. Zdarzenia i oś czasu – stworzony model gridu jest sterowany zdarzeniami. Istnieje pewna oś czasu, na której zapisywane są kolejne zdarzenia. Zdarzenia są wykonywane w kolejności od najwcześniejszego, przy czym wykonanie każdego zdarzenia może skutkować wygenerowaniem lub zmodyfikowaniem zdarzenia późniejszego od niego.

1.2 Zasoby

Węzły obliczeniowe – komputery osobiste Węzłami obliczeniowymi w systemie typu desktop grid są komputery osobiste. Jedną z cech komputerów osobistych może być mała wiarygodność. Ich specyfika to:

- częste wyłączenia mogące zrywać prowadzenie obliczeń,

- możliwość celowego zwracania niepoprawnych wyników.

Dla uproszczenia na potrzeby tej pracy przyjęto, że awarią węzła jest zarówno usterka sprzętu lub programu, jak i zażądanie zwolnienia węzła przez jego właściciela.

W większości wcześniej przeprowadzonych symulacji gridów nie brano pod uwagę zawodności węzłów obliczeniowych – zakładano, że ich wpływ na obliczenia w ośrodkach obliczeniowych jest znikomy.

Do modelowania awarii węzłów wykorzystuje się rozkład wykładniczy [79, 78]. Wynika to z założenia o niezależności awarii, ich przypadkowości oraz stwierdzenia, że czas pozostały do pojawienia się awarii nie zależy od czasu, który upłynął od startu komputera. Gęstość prawdopodobieństwa w rozkładzie wykładniczym jest opisywana wzorem:

$$f(x;\lambda) = \lambda e^{-\lambda x}, \tag{V.1}$$

gdzie $\lambda = \frac{1}{MTBF}$, a MTBF to średni czas pomiędzy awariami. MTBF jest czynnikiem charakteryzującym niezawodność systemów i będzie on używany w dalszej części pracy do określania niezawodności węzłów obliczeniowych. W dużych gridach obliczeniowych MTBF ma wartość rzędu miesięcy, natomiast w systemach typu desktop grid będą to jednostki rzędu godzin. W związku z tym, że rząd wielkości MTBF jest porównywalny z czasem wykonywania zadania, będzie on miał znaczący wpływ na przebieg symulacji.

W symulacjach założono, że po bardzo krótkim czasie po awarii komputer zaczyna znów działać, ale w międzyczasie „zapomina", że prowadził jakieś obliczenie. Uzyskane w ten sposób przybliżenie może być interpretowane jako system, w którym do węzła zarządzającego przypisany jest limit możliwych do podłączenia komputerów. W momencie wykrycia awarii któregoś z podłączonych komputerów pojawia się możliwość użycia jednego z komputerów oczekujących na połączenie z serwerem.

W systemach typu desktop grid zakłada się brak możliwości korzystania przez oprogramowanie węzła obliczeniowego z dysku. Ma to na celu zwiększenie bezpieczeństwa właścicieli węzłów. Dużo łatwiej jest wtedy ustalić właściwą politykę dostępu do dysku dla wszystkich elementów systemu typu desktop grid pracujących na węźle – po prostu nie ma możliwości zapisu na dysk. W takim przypadku nie jest możliwe zapisywanie wyników częściowych. Konieczna okazuje się także dużo częstsza komunikacja pomiędzy centrum a węzłami obliczeniowymi, co wpływa na obciążenie sieci.

Węzeł zarządzający

Najpopularniejszym modelem obliczeń, szczególnie w odniesieniu do systemów typu desktop grid, jest model, w którym istnieje węzeł centralny odpowiedzialny za przydzielanie i wysyłanie zadań na węzły obliczeniowe. W systemie może znajdować się więcej niż jeden węzeł tego typu, dlatego powinien on być nazywany węzłem zarządzającym. W trakcie prowadzonych badań rozważano jednak

wyłącznie system z jednym węzłem zarządzającym i w takiej sytuacji nazywanie go węzłem centralnym jest całkowicie uzasadnione. Węzeł ten odpowiada również za rozdzielenie zlecenia na zadania, a także łączenie wyników. Czas rozdzielenia i złączenia jest w używanym modelu pomijany. Możliwe jest jednak uwzględnienie go oraz branie pod uwagę faktu, że węzeł zarządzający nie powinien rozdzielać więcej niż określoną ilość zleceń naraz. Oprogramowanie węzła zarządzającego zawiera w sobie alokator zasobów oraz wykorzystuje politykę szeregowania.

Sieć

Ważną część modelu stanowi sieć – odpowiada jej graf, którego węzłami są komputery i elementy sieciowe. Komputery to węzły obliczeniowe oraz węzły zarządzające. Komputery znajdują się wyłącznie w liściach. Węzły nie będące komputerami są obciążone ruchem, który jest niezależny od gridu i ma priorytet nad ruchem w gridzie. Opis modelu ruchu w sieci, niezależnego od gridu, znajduje się w dalszej części tego rozdziału.

Transfer danych pomiędzy dwoma komputerami jest trójką składającą się z ilości danych do przesłania oraz pary komputerów. Każdemu transferowi odpowiada ścieżka w drzewie sieci. Pasmo, które jest dostępne dla systemu typu desktop grid jest dzielone równomiernie pomiędzy wszystkie połączenia. Obciążenie sieci można opisać następującymi równaniami:

$$s_i(t) = min(as_J(t) : J \in \{sciezka(i)\}) \tag{V.2}$$

gdzie: $s_i(t)$ to prędkość i-tego transferu w chwili czasu t, as_J to maksymalne dostępne pasmo na elemencie sieciowym J w chwili t dla pojedynczego transferu. Dostępne pasmo $as_J(t)$ zależy także od innych transferów, których ścieżki zawierają element J. Można to opisać wzorem:

$$avail_J(t) \geq \sum_{i:J \in \{sciezka(i)\}} min(as_J(t), s_i(t)), \tag{V.3}$$

gdzie $avail_J(t)$ to wielkość pasma dostępnego w elemencie J w chwili czasu t. W momencie, gdy w powyższym wzorze wystąpi równość, należy uznać, że element J ogranicza przepustowość sieci w gridzie. Musi bowiem istnieć wtedy co najmniej jeden transfer i_1, zawierający w ścieżce element J, taki że $as_J(t)$ jest mniejszy lub równy $s_i(t)$.

Korzystając z wzorów przedstawionych powyżej, bardzo trudno jest znaleźć wartości $s_i(t)$, choćby z powodu występowania cyklów w sieci. W symulowanych systemach typu desktop grid wykluczono komunikację bezpośrednią pomiędzy węzłami. Wszystkie transfery zawierają więc węzeł zarządzający jako węzeł końcowy. W rezultacie sieć z punktu widzenia modelu, w którym jest tylko jeden

węzeł zarządzający, jest drzewem z tym węzłem w korzeniu. Uproszczenie to znacząco ułatwia symulację.

1.3 Obliczenia

Komputery to inaczej maszyny liczące. Istotnym pytaniem, jakie należy sobie zadać przy modelowaniu systemu, to jaka jest charakterystyka obliczeń. W modelowanym systemie zajmujemy się wyłącznie obliczeniami, które mogą być prowadzone równolegle w bardzo dużym rozproszeniu. Przeprowadzone badania wskazują, że większość obliczeń przeznaczonych dla systemów typu desktop grid nie może, ze względu na zabezpieczenia sieci, wymagać komunikacji pomiędzy węzłami obliczeniowymi, a jedynie z węzłem centralnym. Obliczenie, czyli pewna praca zlecana przez użytkownika, będzie nazywana zleceniem, natomiast obliczenie, które jest już dobrze sprecyzowane i może być wykonane na węźle liczącym, będzie nazywane zadaniem lub podzadaniem. Charakterystyczne dla systemów typu desktop grid są zlecenia, które można podzielić na wiele równoległych zadań. W symulowanym systemie założono dodatkowo, że czas trwania zadań jest krótki, podobnie jak całych zleceń.

Zlecenia

Zlecenia przychodzą od użytkowników do węzła zarządzającego. Węzeł zarządzający dokonuje podziału zlecenia na podzadania. Najprostszy rodzaj obliczeń w systemie to obliczenia typu rozrzuć/zbierz, które polegają na:

1. rozdzieleniu zlecenia na podzadania,
2. wykonaniu wszystkich podzadań na węzłach obliczeniowych,
3. złączeniu wyników.

Zadania

Zadanie, lub dla jasności podzadanie, jest to atomowe obliczenie, które powinno zostać wykonane na pojedynczym węźle obliczeniowym. Zadania w symulowanym systemie składały się z następujących elementów:

1. przesłanie na węzeł danych obliczeniowych dla podzadania,
2. wykonanie obliczeń,
3. odesłanie wyników do węzła zarządzającego.

Dla uproszczenia założono brak możliwości zapisywania wyników częściowych, co oznacza, że jeśli węzeł zostanie wyłączony w trakcie trwania wykonywania zadania, zadanie to trzeba będzie powtórzyć. Aby przyspieszyć działanie systemu, a także zwiększyć wiarygodność, możliwe jest równoczesne zlecenie wykonania identycznego zadania kilku węzłom obliczeniowym.

Zależność zadań

W gridach usługowych bardzo często występuje zależność zadań, tj. istnieją obliczenia, w których pewna grupa zadań może być wykonana dopiero po innej grupie zadań. Przy założeniu o kontroli pośrednich wyników zleceń przez użytkownika, uproszczenie polegające na braku zależności w kolejności wykonania podzadań jest zasadne.

1.4 Zdarzenia

Zdarzenia

Działanie modelu opiera się na przetwarzaniu kolejnych zdarzeń. W systemie występują następujące grupy zdarzeń dotyczące węzła obliczeniowego:

- przyłączenie węzła do systemu,
- zlecenie węzłowi wykonania zadania,
- zakończenie przez węzeł obliczeniowy pobierania danych do obliczeń,
- zakończenie przez węzeł wykonywania obliczeń (po czym musi jeszcze nastąpić wysłanie rezultatów),
- zakończenie przez węzeł obliczeniowy przesyłania wyników zadania do węzła zarządzającego, co stanowi koniec obliczeń,
- wyłączenie węzła obliczeniowego.

Ponadto, okresowo realizowane jest zdarzenie polegające na zmianie przepustowości sieci. Każde zdarzenie może powodować powstanie innych zdarzeń w przyszłości, zmianę czasu innego przyszłego zdarzenia lub nawet usunięcie innego przyszłego zdarzenia. Przykładem zdarzenia, które generuje nowe zdarzenie, jest zakończenie przez węzeł pobierania danych do obliczeń, co powoduje rozpoczęcie obliczeń, a zatem powstanie zdarzenia na węźle. Z kolei zmiana przepustowości sieci powoduje zmianę czasu zdarzenia dotyczącego zakończenia przez węzeł pobierania lub wysyłania danych. Odłączenie węzła od systemu powoduje zerwanie obliczeń, które odbywają się na tym węźle, a co za tym idzie likwidację zdarzenia polegającego na zakończeniu przez węzeł wykonywania obliczeń.

2. Model przepływu danych w sieci

Symulatory gridów nie uwzględniają w większości zmian pasma sieciowego. Badania modelowe wykazują jednak, że ze względu na skorelowanie czasu pracy naukowców zlecających wykonanie zadań, a także obciążenie sieci, czynnik ten nie może być pomijany [46]. W pracach [42, 14] pokazano, że narzędziem bardzo dobrze nadającym się do modelowania obciążenia sieci w krótkich okresach czasu są szeregi czasowe. Prace te były ukierunkowane na przewidywanie obciążenia sieci w celu optymalizacji algorytmów trasowania (routing). Algorytmy te uwzględniają zmiany parametrów w sieci z rozdzielczością rzędu sekund, a ich działanie sprawdzano na przebiegach o czasie kilku godzin. Ponieważ dla potrzeb symulacji gridów interesujące czasy symulacji wyrażają się w dniach, a dostateczna rozdzielczość ma być rzędu minut, konieczne było przeprowadzenie dodatkowych badań.

Modele ARIMA i SARIMA

Do analizy szeregów czasowych niezwykle użytecznymi narzędziami są modele SARIMA stanowiące rozszerzenie modeli ARIMA na szeregi zawierające cykle sezonowe. Modele te zostały opisane w podręczniku [18] i od tego czasu były rozbudowywane, mieszane z innymi modelami, jak choćby w pracy [80]. W trakcie prowadzonych badań stwierdzono jednak bardzo dobrą jakość modeli najprostszych, które są opisane w tej części pracy.

W pierwszej kolejności zostaną opisane modele ARMA, czyli modele autoregresji ze średnią ruchomą. Dla szeregu czasowego $z = z_1, z_2, \ldots z_n$ przez $\tilde{z}_t = z_t - \mu$ oznaczane jest odchylenie z_t, od punktu μ, który dla procesów stacjonarnych jest średnią z_t. Proces autoregresji rzędu p ma wtedy postać:

$$\tilde{z}_t = \varphi_1 \tilde{z}_{t-1} + \varphi_2 \tilde{z}_{t-2} + \varphi_3 \tilde{z}_{t-3} + \ldots + \varphi_p \tilde{z}_{t-p} + a_t, \tag{V.4}$$

gdzie a_t oznacza biały szum. Jest on elementem losowości przebiegu czasowego, który nie może być przewidywany, lecz powinien mieć rozkład normalny. Klasę procesów autoregresji rzędu p w skrócie oznaczono jako AR(p).

Analogicznie do procesu autoregresji, proces średniej ruchomej rzędu q ma postać:

$$\tilde{z}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3} - \ldots - \theta_q a_{t-q} \tag{V.5}$$

Taką klasę procesów oznaczamy jako MA(q).

Proces MA rzędu q można przedstawiać jako proces AR z rzędem nieskończonym, jak również proces AR rzędu p można przedstawić jako proces MA z rzędem nieskończonym. Taka parametryzacja byłaby jednak skrajnie nieoszczędna, w związku z czym wprowadzono pojęcie procesu mieszanego obejmującego

średnią ruchomą i autoregresję (ARMA):

$$\tilde{z}_t = \varphi_1 \tilde{z}_{t-1} + \varphi_2 \tilde{z}_{t-2} + \ldots + \varphi_p \tilde{z}_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta q a_{t-q} \quad \text{(V.6)}$$

Klasę takich modeli oznaczono jako ARMA(p,q). Po zamianie stronami wzór przyjmuje postać:

$$\tilde{z}_t - \varphi_1 \tilde{z}_{t-1} - \varphi_2 \tilde{z}_{t-2} - \ldots - \varphi_p \tilde{z}_{t-p} = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta q a_{t-q} \quad \text{(V.7)}$$

W celu uproszczenia zapisu stosuje się często operator przesunięcia wstecz B, którego działanie jest wyrażone następująco: $Bz_t = z_{t-1}$. B^2 oznacza wtedy dwukrotne przesunięcie wstecz. Dla wygody używa się także zapisu: $\varphi(B) = 1 - \varphi_1 B - \varphi_2 B^2 - \ldots - \varphi_p B^p$. Wzór szeregu ARMA($p$,$q$) przyjmuje wtedy postać:

$$\varphi(B)\tilde{z}_t = \theta(B)a_t \quad \text{(V.8)}$$

Klasa ARMA nie nadaje się do modelowania procesów niestacjonarnych. W tej sytuacji dobrze jest czasem skorzystać z modeli scałkowanych – tj. takich, w których za pomocą modelu stacjonarnego opisujemy różnice między wyrazami ciągu oddalonymi od siebie o d. W tym celu niezbędny jest operator różnicowy przesunięcia wstecz wyrażony wzorem: $\nabla z_t = z_t - z_{t-1} = (1-B)z_t$.

Po skorzystaniu z tego operatora otrzymujemy modele ARIMA, czyli modele arma scałkowane. Szereg ARIMA (p,q,d) przyjmuje postać:

$$\varphi(B)\nabla^d \tilde{z}_t = \theta(B)a_t \quad \text{(V.9)}$$

Ponieważ $\nabla\tilde{z}_t = \nabla z_t$, właściwa postać modelu ARIMA jest następująca:

$$\varphi(B)\nabla^d z_t = \theta(B)a_t \quad \text{(V.10)}$$

Dla pewnej klasy szeregów czasowych, nazywanych szeregami sezonowymi, konieczne byłoby używanie wartości d znacząco większych niż 2. Specyfika szeregów sezonowych polega na cykliczności, czyli powtarzalności zachowania szeregu co k elementów. Wartość k nazywana jest okresem. Przykładowe okresy często występujące w świecie rzeczywistym to doba, tydzień, rok.

Ponieważ używanie wartości d większych niż 2 powoduje znaczącą komplikację modelu, dla sezonowych szeregów czasowych zostały opracowane oszczędne modele multiplikatywne.

Oszczędny operator różnicowy przesunięcia wstecz jest zdefiniowany następującym wzorem: $\nabla_S = 1 - B^S$. Operator ten okazuje się bardzo użyteczny, gdy badany szereg jest szeregiem ściśle sezonowym. Jeśli długość cyklu wynosi S, to dla szeregu t oczekuje się, że $\nabla_S t_k = 0$ dla każdego k. Właściwość ta pokazuje użyteczność operatora, ale jest prawdziwa tylko dla szeregów idealnych.

Przykładowy model uwzględniający jedynie wahania sezonowe ma postać:

$$\Phi(B^S)\nabla_S^D z_t = \Theta(B^S)a_t \tag{V.11}$$

O modelu takim mówimy, że jest to model SARIMA $(P,Q,D)_S$ (zakładając, że Φ jest stopnia P, a Θ stopnia Q). Uwzględniając jednocześnie powiązanie z_t z elementami $z_{t-1}, z_{t-2}\ldots$, należy jednocześnie zastosować model V.10. Przy odpowiednim podstawieniu otrzymujemy wtedy model multiplikatywny:

$$\varphi_p(B)\Phi_P(B^S)\nabla^d\nabla_S^D z_t = \theta(B)a_t \tag{V.12}$$

Taki model nazywany jest multiplikatywnym modelem SARIMA $(p,q,d)\times(P,Q,D)_S$ Tego rodzaju model można oczywiście rozszerzyć na więcej niż dwa człony.

Rozdział VI

Parametryzacja modelu sieci

Modele SARIMA opisane w poprzednim rozdziale dobrze nadają się do symulacji cykli w pracy sieci. Użycie takiego modelu w działającym symulatorze wymaga jego parametryzacji. Polega ona na wybraniu modelu charakteryzującego się optymalną wielkością cykli i minimalną liczbą członów, a przy tym dobrze oddającego rzeczywistość.

1. Dane doświadczalne

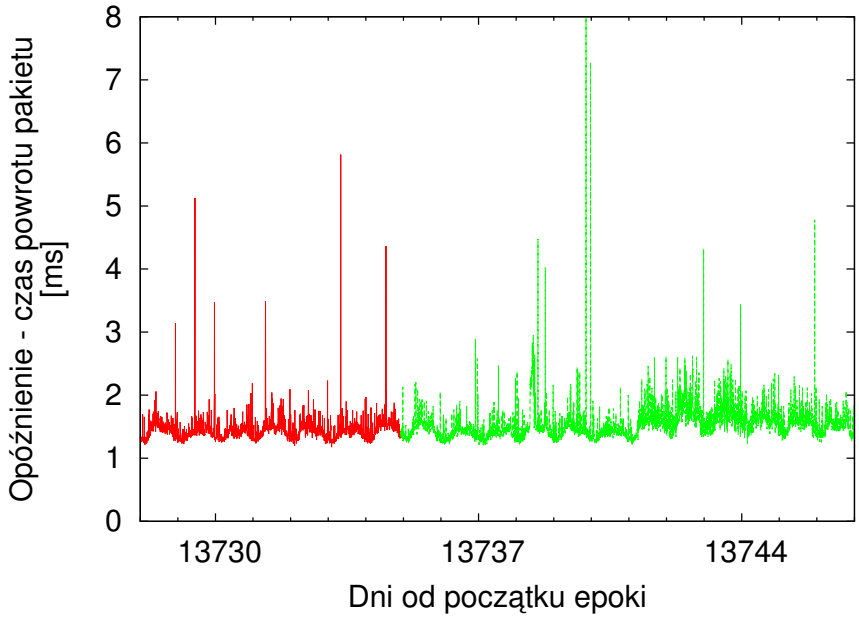
Cykle obciążenia w sieci dotyczą zarówno przepustowości, jak i opóźnienia. Obie te wartości są mocno ze sobą skorelowane. Dla uproszczenia badań użyto do wstępnej parametryzacji modelu tylko jednej z tych funkcji opóźnienia i założono, że model dla drugiej wartości będzie praktycznie identyczny z dokładnością do stałej.

Do mierzenia opóźnienia w sieci wykorzystano pomiar czasu powrotu (round trip time – w skrócie *rtt*). Pomiar ten jest dokonywany przez program ping, przy użyciu pakietów ICMP.

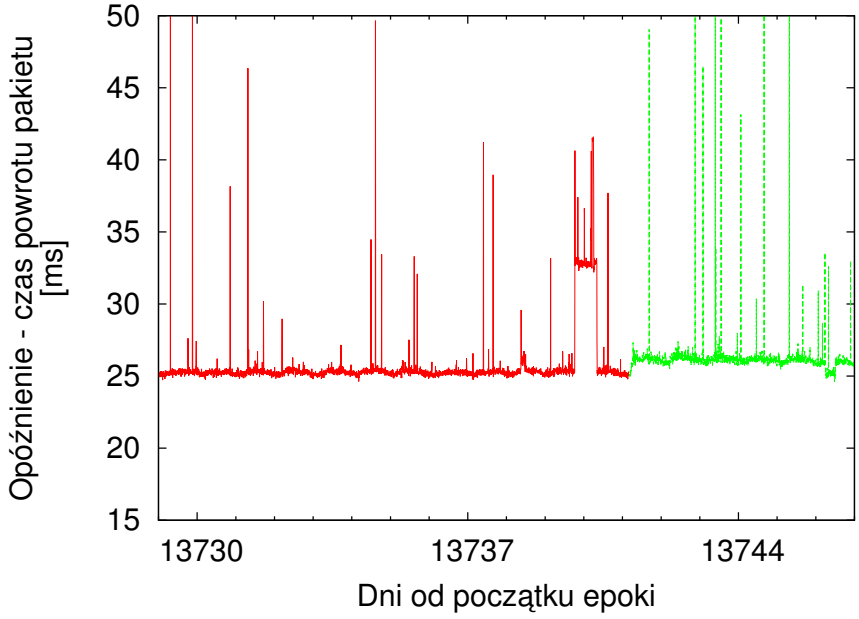
Przeprowadzono pomiar przebiegów czasowych dla połączeń: ICM – TP S.A. i ICM UW – UMK Toruń. Pomierzono też czas powrotu *rtt* dla pakietu o maksymalnej wielkości dostępnej w protokole ICMP. Dane z pomiarów wykorzystane do treningu modelu są przedstawione na wykresach 1 i 2. Na osi X zaznaczono czas w dniach, które upłynęły od początku epoki[1], a na osi Y – opóźnienie wyrażone w milisekundach.

Dane zostały podzielone na dwie grupy: dane treningowe i dane testowe. Dane treningowe posłużyły do znalezienia optymalnych możliwych parametrów modelu. Następnie wyniki działania gotowego modelu zostały porównane z danymi testowymi.

[1] 1 stycznia 1970.



Wykres 1: Przykładowy przebieg dla połączenia ICM – TP SA. Dane treningowe zostały oznaczone kolorem czerwonym, dane testowe – zielonym



Wykres 2: Przykładowy przebieg dla połączenia ICM – UMK. Dane treningowe zostały oznaczone kolorem czerwonym, dane testowe – zielonym

2. Wyniki parametryzacji modelu

Na wykresach 1 i 2, przedstawiających dane rzeczywiste, wyróżniają się cykle dobowe. Możliwe jest także występowanie cykli tygodniowych, ale są one znacznie mniej wyraźne. W celu przeprowadzenia dalszej dokładnej analizy szeregów, a następnie ich modelowania, wykorzystano model SARIMA. Ponieważ pomiary są dokonywane w odstępach sześciominutowych, cykl dobowy ma wielkość 240, a tygodniowy 1680. Analizowane będą modele SARIMA klasy:

$$(p_1,q_1,d_1)\times(p_2,q_2,d_2)_{240}\times(p_3,q_3,d_3)_{1680} \tag{VI.1}$$

Na wykresach wyróżniają się wysokie wartości losowe. Aby rozwiązać ten problem w modelu SARIMA, zastąpiono funkcję jej logarytmem $z_t = ln(zorig_t)$, gdzie $zorig_t$ jest wartością zmierzoną. Ponowne przejście z ciągu *zorig* do z następuje przy użyciu funkcji wykładniczej.

Istotna różnica pomiędzy wykresami polega na wyraźnym nielosowym zmniejszeniu opóźnienia dla jednego, co można zaobserwować na końcu danych treningowych na wykresie 2. Spadek ten nie występuje na wykresie 1. Dodatkowo tylko na wykresie 2, pomiędzy danymi treningowymi a testowymi, widoczny jest wzrost opóźnienia niemożliwy do przewidzenia w modelu.

Po przeprowadzeniu symulacji i minimalizacji parametrów w zakresie (-1,1), okazało się, że optymalne rezultaty osiągane są dla modelu:

$$(1,0,1)\times(0,1,1)_{240}\times(0,1,1)_{1680} \tag{VI.2}$$

Model ten, po zapisaniu w postaci równania różnicowego, przyjmuje postać:

$$\begin{aligned} z_t \;=\;& \tau z_{t-1} \\ &+z_{t-240}-\tau z_{t-241} \\ &+z_{t-1680}-\tau * z_{t-1681} \\ &-z_{t-1920}+\tau z_{t-1921} \\ &+a_t-\Theta a_{t-1} \\ &-\Phi a_{t-240}+\Theta\Phi a_{t-241} \\ &-\Omega a_{t-1680}+\Omega\Phi a_{t-1681} \\ &+\Omega\Theta a_{t-1920}-\Omega\Phi\Theta a_{t-1921} \end{aligned} \tag{VI.3}$$

Jest on na oszczędny, gdyż wielomiany występujące w równaniach modelu SARIMA – równanie V.12, z poprzedniego rozdziału, można zastąpić współczynnikami τ, Φ, Θ i Ω, które umożliwiają precyzyjne dopasowanie modelu do konkretnych danych doświadczalnych. Dla potrzeb przedstawiania parametrów zamiast Φ, Θ i Ω zostaną użyte odpowiednio symbole θ z odpowiednim indeksem:

1 – oznacza poprzednią próbkę (6 minut), 240 – oznacza dzień i 1680 – oznacza tydzień. Ostatecznie równanie przyjęło postać:

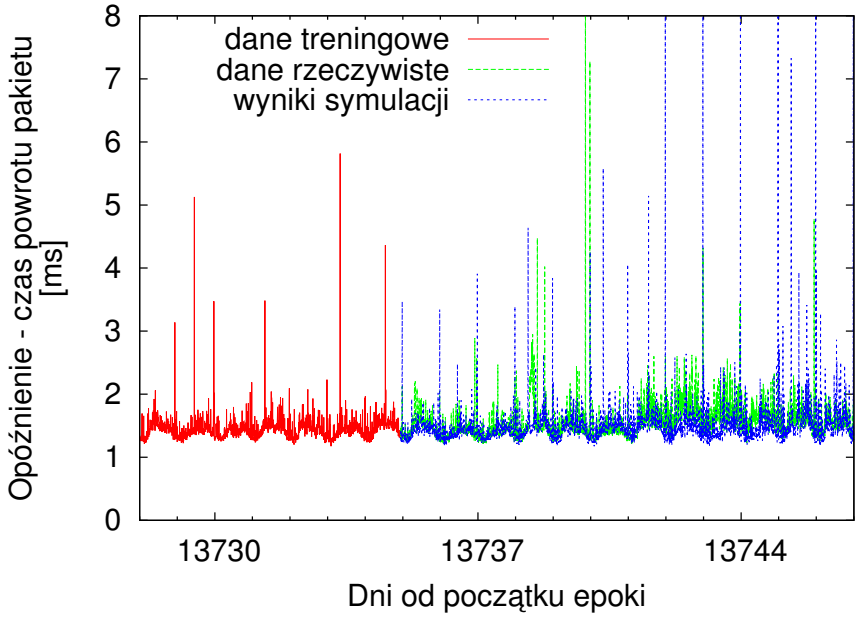
$$\begin{aligned} z_t &= \tau z_{t-1} \\ &\quad + z_{t-240} - \tau z_{t-241} \\ &\quad + z_{t-1680} - \tau * z_{t-1681} \\ &\quad - z_{t-1920} + \tau z_{t-1921} \\ &\quad + a_t - \theta_1 a_{t-1} \\ &\quad - \theta_{240} a_{t-240} + \theta_1 \theta_{240} a_{t-241} \\ &\quad - \theta_{1680} a_{t-1680} + \theta\theta_{1680} a_{t-1681} \\ &\quad + \theta_{240}\theta_{1680} a_{t-1920} - \theta\theta_{240}\theta_{1680} a_{t-1921} \end{aligned} \tag{VI.4}$$

W wyniku zastosowania tego modelu do szeregów przedstawionych na wykresach 1 i 2, po przeprowadzeniu optymalizacji współczynników, otrzymano prognozy przedstawione na wykresach 3 i 4. Wartości współczynników z równania VI.4 przedstawiono w poniższej tabeli:

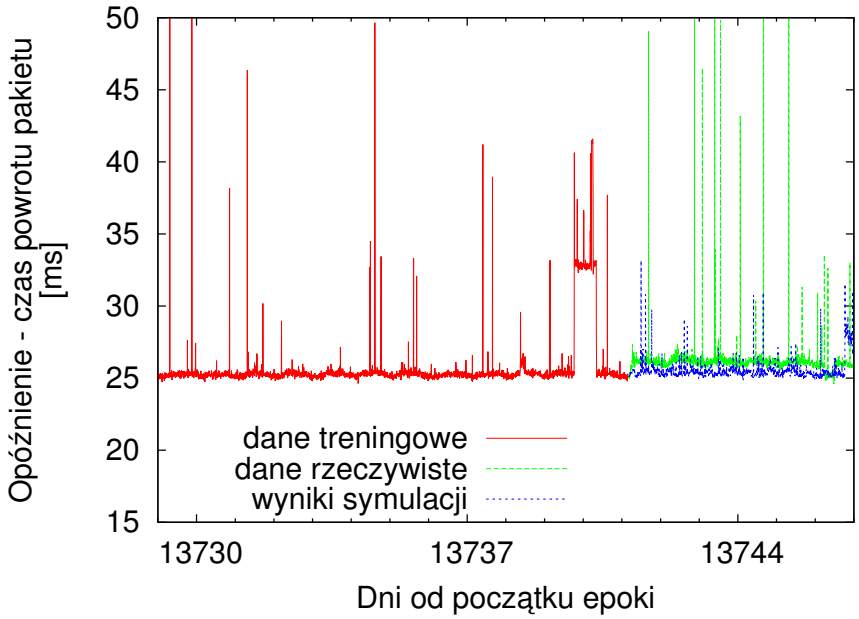
	τ	θ_1	θ_{240}	θ_{1680}
wykres 3	0.8327058	0.6181818	0.6765803	0.3305048
wykres 4	0.9569270	0.7135853	0.8096745	0.3865159

Na wykresie 3 widać wyraźnie, że w ciągu pierwszych kilku dni model zwraca wyniki pokrywające się z danym testowymi. Po około jednym tygodniu model przestaje się pokrywać z rzeczywistością, co jest spowodowane coraz większym wpływem generowanego losowo białego szumu. Zgodnie z oczekiwaniem dla danych z wykresu 2, wyniki prognoz przedstawionych na wykresie 4 są zdecydowanie różne od danych testowych. Widać jednak wyraźnie, że przez pierwszy tydzień model dobrze przybliża zachowanie standardowe sieci, tzn. przy założeniu, że nie ma wahnięć nielosowych (tzn. niewynikających z białego szumu – a_t). Na końcu pracy modelu widoczny jest również gwałtowny skok do góry spowodowany skokiem w danych treningowych i powtórzeniem go przez model tydzień później.

Ocena jakości prognoz wymaga analizy zastosowania modelu. Jedna z metod oceny to to minimalizacja błędu średniokwadratowego. Modele dążące do minimalizacji tego błędu są używane w sytuacji, gdy chcemy uzyskać dane w możliwie najmniejszym stopniu odchylone od danych rzeczywistych. Wykorzystywane są one przy opracowywaniu prognoz. W celu osiągnięcia tej minimalizacji, wykonuje się przybliżenie białego szumu (a_T) jego wartością średnią czyli zerem. Jest to spowodowane minimalizacją błędu średniokwadratowego i prowadzi do przybliżania funkcją stałą w sytuacji gdy jest ona stałą z dokładnością do białego szumu. Inne podejście zostało zastosowane przy opracowywaniu modelu na potrzeby symulacji



Wykres 3: Prognoza dla połączenia ICM – TP SA. Dane treningowe zostały oznaczone kolorem czerwonym, dane testowe zielonym, a prognoza niebieskim



Wykres 4: Prognoza przebiegu dla połączenia ICM – UMK. Dane treningowe zostały oznaczone kolorem czerwonym, dane testowe zielonym, a prognoza niebieskim

systemu typu desktop grid. Symulator powinien odtwarzać charakterystykę przebiegu, co oznacza, że jeśli w rzeczywistych danych szum ma rozkład normalny, to także w trakcie symulacji powinien mieć taki rozkład. W wielu przypadkach prowadzi to do zwiększenia rozbieżności (w sensie błędu średniokwadratowego) między danymi rzeczywistymi a tymi wykorzystywanymi do symulacji. Umożliwia jednak uruchomienie choćby kilku przebiegów symulacji z tymi samymi danymi początkowymi i sprawdzenie, na ile stabilnie w zmiennych warunkach zachowa się symulowany system. Dodatkowo służy to ograniczeniu przetrenowania optymalizowanego systemu, które mogłoby wystąpić przy wygładzeniu funkcji.

Do budowania prognozy użyto zatem zamiast wartości średniej wartości generowanej z rozkładu normalnego o tych samych parametrach co rozkład białego szumu. Spowodowało to, zgodnie z oczekiwaniami, zwiększenie, w sensie błędu średniokwadratowego, rozbieżności pomiędzy wygenerowaną prognozą a danymi testowymi.

Po okresie jednego tygodnia prognoza traci stabilność i przestaje poprawnie odzwierciedlać rzeczywiste zachowanie sieci, ale dla potrzeb symulacji tygodnia pracy systemu jest w zupełności wystarczająca.

Rozdział VII

Symulacje systemów typu desktop grid

1. Wprowadzenie

Na podstawie projektu BOINC, opisanego w części 5 rozdziału III, można przekonać się, że gridy społecznościowe, które wykorzystują czas wolny komputerów osobistych, mogą osiągać bardzo duże moce obliczeniowe [8]. Znaczną wydajność mogą osiągać także gridy typu desktop składające się z komputerów udostępnianych przez duże instytucje [76].

Podstawowe pytanie stawiane przez użytkowników systemów obliczeniowych brzmi: w jakim czasie nastąpi zakończenie danego obliczenia? W przypadku systemów składających się z niezawodnych węzłów wystarcza przyjęcie upraszczającego założenia, że znany jest czas konieczny do wykonania każdego podzadania na każdym z węzłów. W systemach typu desktop grid, założenie to nie jest wystarczające z powodu częstych wyłączeń węzłów. Może się bowiem okazać, że uruchomione obliczenie nie zdąży się zakończyć, w związku z czym trzeba je będzie powtórzyć od początku. Inne utrudnienie polega na ograniczonej i zmiennej przepustowości sieci. Jedno z powszechnych rozwiązań, stosowane gdy nie ma możliwości dokładnego wyliczenia parametrów systemu, polega na symulacji jego zachowania. Dzięki dużej ilości węzłów w systemach typu desktop grid wpływ pojedynczego zdarzenia losowego na wynik symulacji jest znikomy. Kilkakrotne powtórzenie symulacji powoduje, że błąd ten można potraktować jako całkowicie zaniedbywalny. Stanowi to dodatkową motywację do badania charakterystyki tych systemów przy użyciu symulacji.

W niniejszym rozdziale podano dwa przykłady wykorzystania symulacji w odniesieniu do systemów typu desktop grid. Pierwszy przykład to wykorzystanie symulacji do optymalizacji działania programów zarządzających systemami typu

desktop grid. Drugi z przykładów to poszukiwanie elementów systemu, których zmiana powoduje największe przyspieszenie obliczeń. To zagadnienie stanowi istotny element optymalizacji działania systemu.

2. Ustalenie właściwej polityki rozdziału zadań

Zastosowanie symulatora ma na celu, m.in., określenie możliwości różnicowania podejścia do kwestii rozdziału zadań w systemie typu desktop grid. Rezultaty przykładowych symulacji zostały przedstawione w pracach [48, 46].

2.1 Proste symulacje z nieskończenie szybką siecią

W pracy [48] opisano wyniki pierwszych badań, które zostały przeprowadzone na symulatorze bez uwzględnienia zmiennej przepustowości sieci. Założono, że transfery stanowią stałe obciążenie, a pasmo jest na tyle duże, że jedynym wynikiem ich oddziaływania jest stałe opóźnienie. Symulacje w tym przypadku pokazały, że symulator w prostych sytuacjach zachowuje się stabilnie i poprawnie.

Opis symulowanego systemu

Symulowany system składał się z komputerów o równych szybkościach i średnim czasie pomiędzy awariami (MTBF). Przeprowadzono symulacje dla różnych wartości tego czasu. W trakcie symulacji sprawdzano, jak długo będzie wykonywane jedno zlecenie typu rozrzuć – zbierz. Testowano zlecenia o różnych wielkościach, od 1 do 250 podzadań w jednym zleceniu. Wszystkie podzadania wymagały takiej samej ilości czasu procesora. System składał się z 200 węzłów obliczeniowych. Wzajemna relacja czasu danego podzadania i szybkości węzłów była taka, że czas jego wykonania na węźle wynosił 35 s.

Testowane były następujące rodzaje polityki przydziału zadań:

- **Losowy przydział** – polega ona na przydziale przychodzącego zadania do dowolnej losowej maszyny, która spełnia minimalne wymagania i jest w danym momencie dostępna. Zadania są przydzielane zgodnie z kolejnością czasu zlecenia. Żadna polityka gorsza od losowej nie powinna być brana pod uwagę.

- Polityka **optymalna** (idealna) – polega ona na przydzieleniu zadania do węzła o najkrótszym pozostałym czasie życia, ale umożliwiającym ukończenie wykonania zadania. Polityka ta jest w praktyce niewykonalna, gdyż nie jesteśmy w stanie przewidzieć, przez jaki jeszcze czas węzeł będzie włączony.

- **Wybór komputera o największym MTBF** – możliwe jest przybliżenie najlepszego przydziału zadań poprzez aproksymowanie pozostałego czasu życia przez MTBF. Ze względu na jednakowy MTBF węzłów polityka ta powinna dawać rezultaty takie jak polityka losowego przydziału. Została ona jednak sprawdzona w celu potwierdzenia poprawności działania systemu.

- **Zwielokrotnianie** – ze względu na awarie (wyłączenia) węzłów obliczeniowych, możliwe jest uzyskiwanie wzrostu wydajności poprzez inicjowanie danego obliczenia na więcej niż jednym węźle w tym samym momencie. W trakcie symulacji sprawdzone zostały różne maksymalne ilości kopii obliczenia, jakie mogą zostać uruchomione jednocześnie.

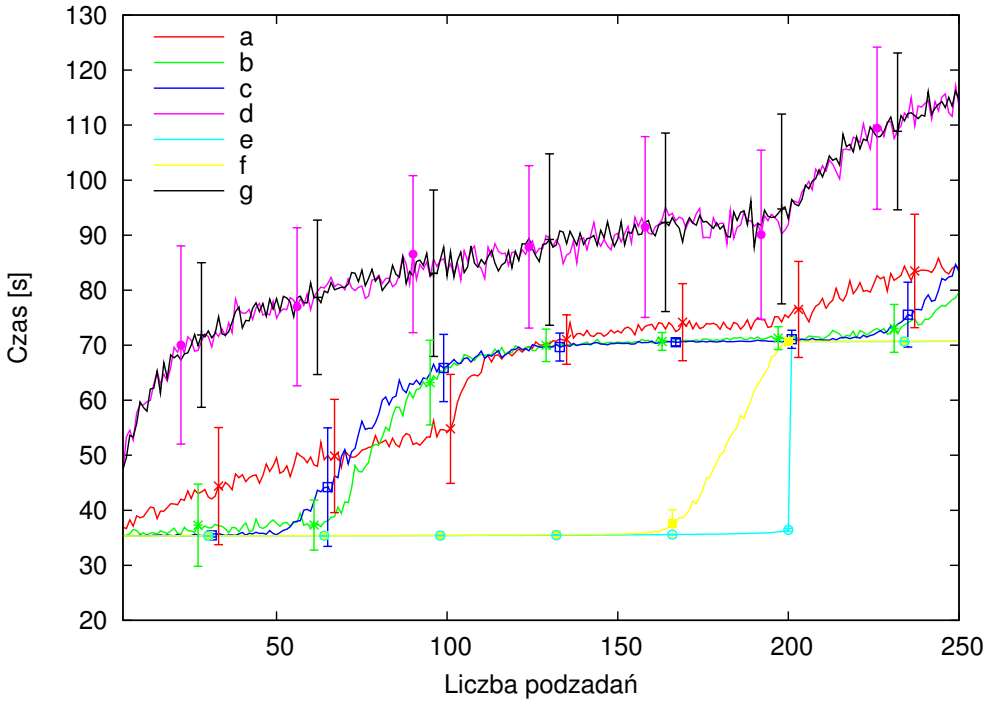
Oczekiwane wyniki

Dla dużych, względem czasu wykonania pojedynczego podzadania, wartości MTBF, czas wykonania zlecenia w zależności od liczby podzadań powinien być funkcją schodkową. Dla bardzo małych czasów pomiędzy awariami węzłów funkcja ta będzie zbliżona do linii prostej. W przypadku częstych awarii poszczególne wykonania podzadań są od siebie niezależne – bardzo rzadko zdarza się, aby w momencie uruchomienia zadania czasy pozostałe do awarii węzłów pozwoliły na wykonanie wszystkich podzadań. Interesujące są sytuacje pośrednie, czyli takie, w których ma miejsce powolne wygładzanie funkcji schodkowej do linii prostej wraz ze zmniejszanym czasem pomiędzy awariami węzłów. Obserwacja tego zjawiska była celem symulacji.

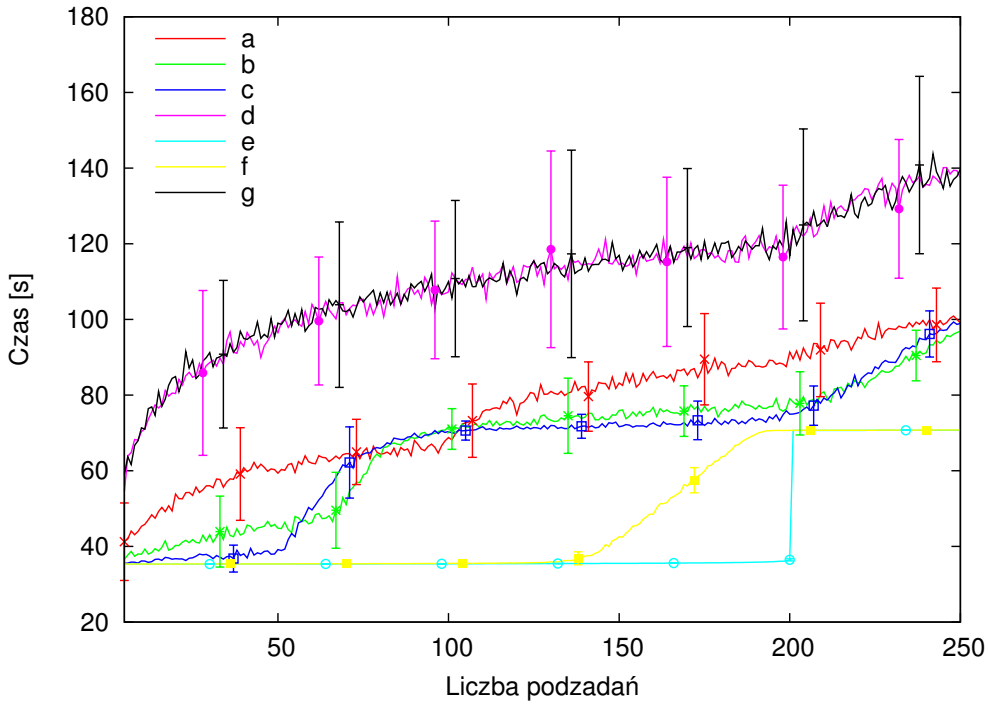
Wyniki

Przeprowadzone symulacje wykazały, że symulator pracuje zgodnie z oczekiwaniem. Na wykresach 1-8 przedstawiono wyniki symulacji dla różnych wartości MTBF. Dla porównania przedstawiono na nich także wyniki pracy systemu, w którym komputery nie podlegają awariom – modelowane jest to przez nieskończony MTBF, użyta strategia szeregowania zadań nie ma wtedy znaczenia ze względu na niezmienną szybkość pracy węzłów.

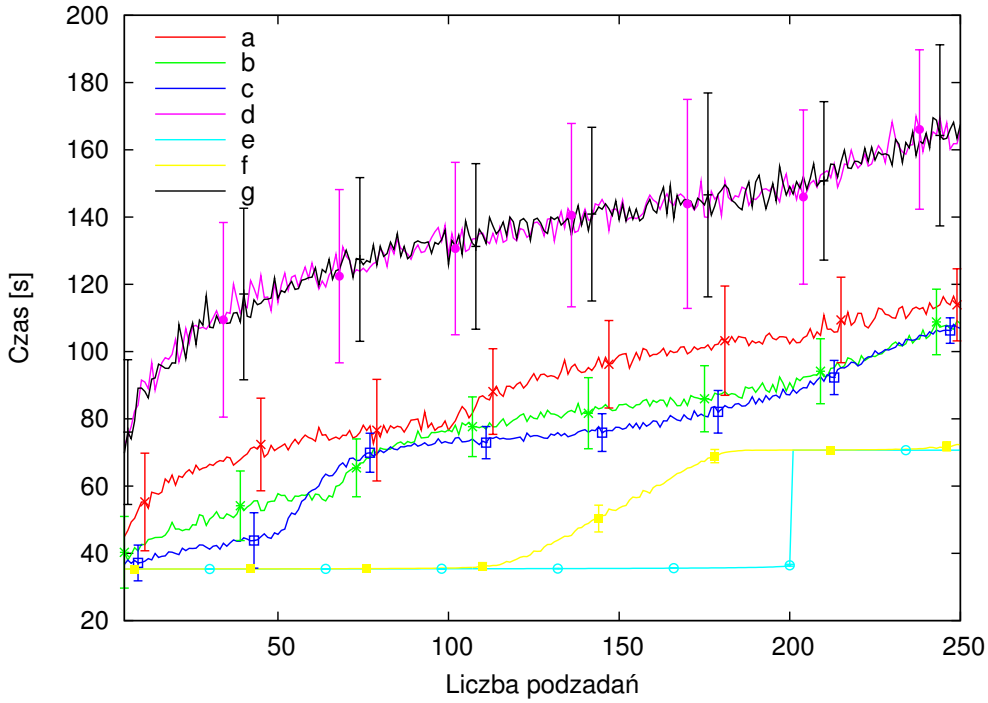
Proste symulacje potwierdziły, że przedstawiony model obliczeń oraz jego implementacja w najprostszych sytuacjach dają dobre wyniki. Awaryjność węzłów powoduje, że nawet przy optymalnym przydziale zadań nie jest możliwe uzyskanie tak dobrych rezultatów, jak w systemie, w którym węzły są niezawodne. Przy założeniach przyjętych dla pewnych wartości MTBF lepsze wyniki daje zwielokrotnienie co najwyżej dwukrotne, a dla innych wartości – większe. Wartość ta zależy także od stosunku liczby zadań w obliczeniu do liczby węzłów obliczeniowych w systemie.



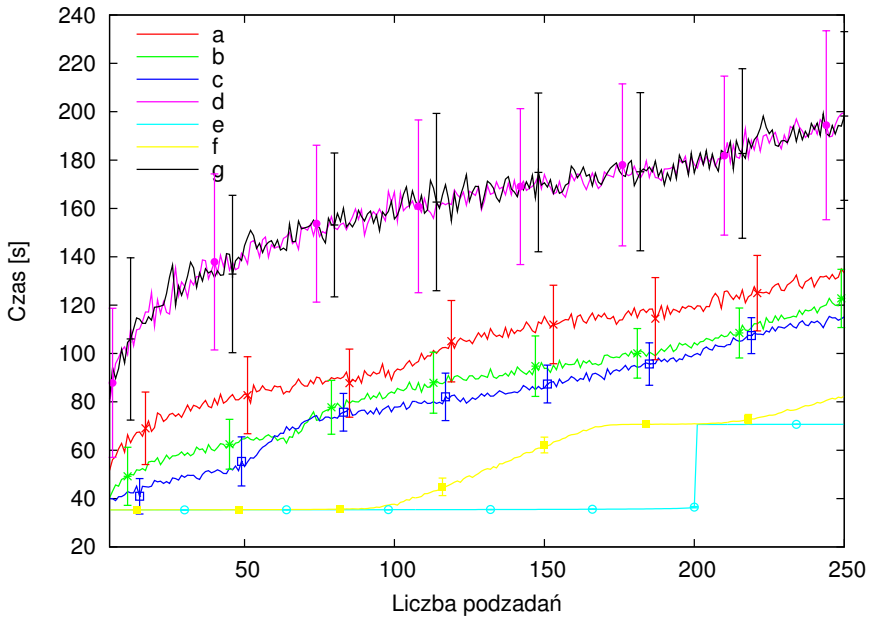
Wykres 1: Wykres średniego czasu wykonania zlecenia w zależności od liczby podzadań. Na wykresie oznaczono literami następujące strategie przydzielania zadań:
a) zwielokrotnianie – maksimum dwie kopie
b) zwielokrotnianie – maksimum trzy kopie
c) zwielokrotnianie – maksimum cztery kopie
d) najlepszy MTBF
e) węzły bez awarii
f) optymalna
g) losowa
Sytuacja gdy komputery nie ulegają awariom (krzywa e) została przedstawiona dla porównania. W pozostałych przypadkach wartość MTBF wynosi przybliżeniu 6-krotność czasu wykonania pojedynczego zadania



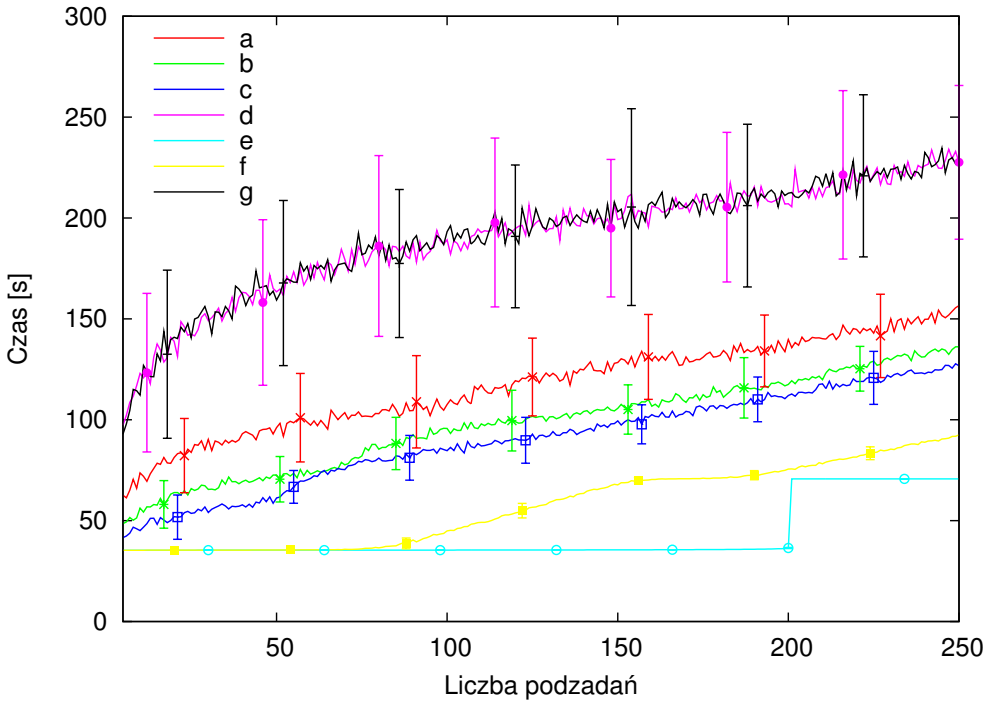
Wykres 2: Wykres średniego czasu wykonania zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 2.98 czasu wykonania pojedynczego zadania



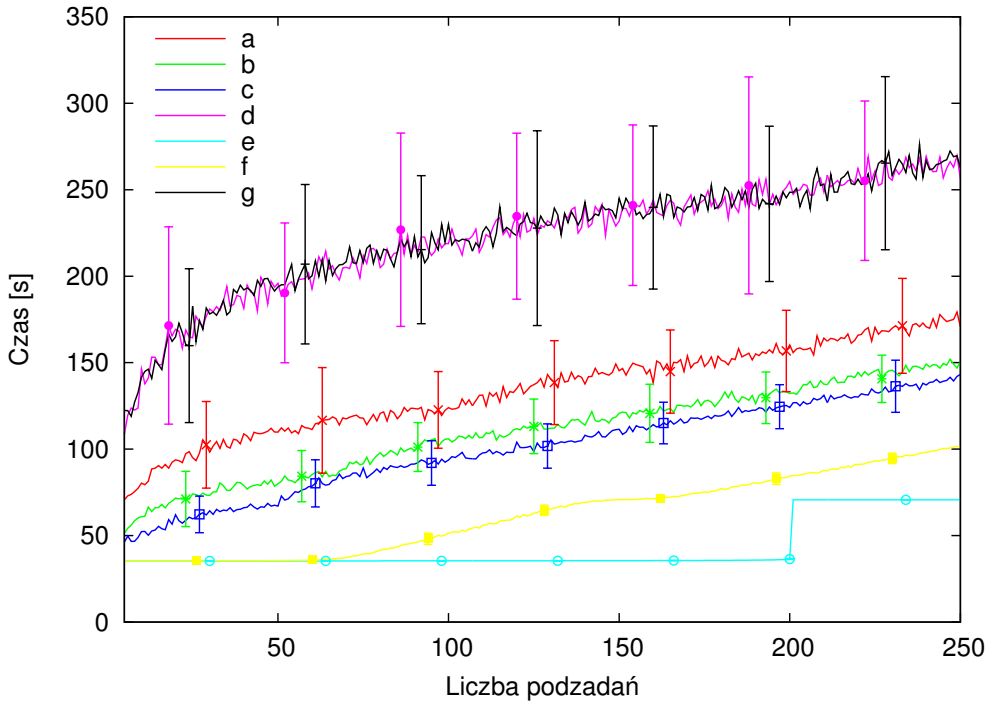
Wykres 3: Wykres średniego czasu wykonania zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 1.98 czasu realizacji pojedynczego zadania



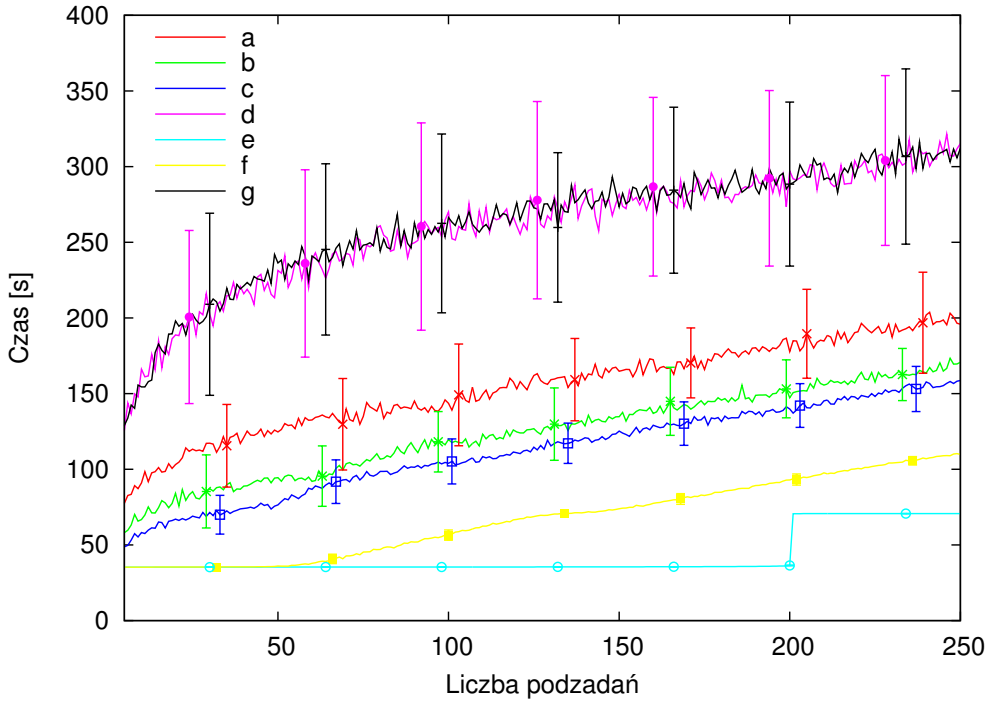
Wykres 4: Wykres średniego czasu realizacji zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 1.48 czasu realizacji pojedynczego zadania



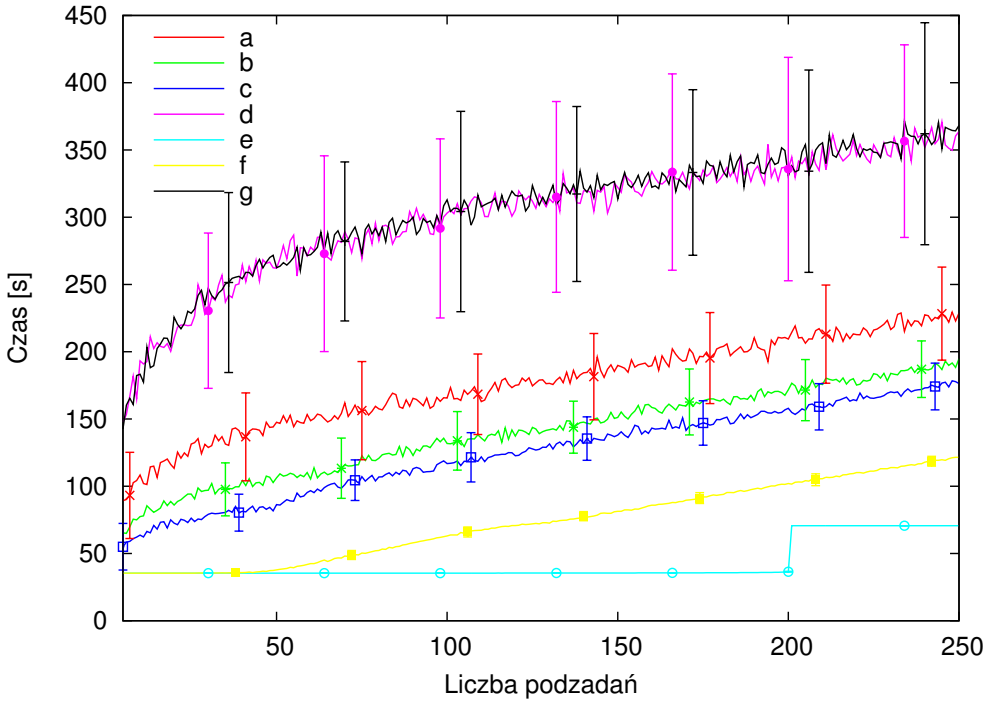
Wykres 5: Wykres średniego czasu realizacji zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 1.18 czasu realizacji pojedynczego zadania



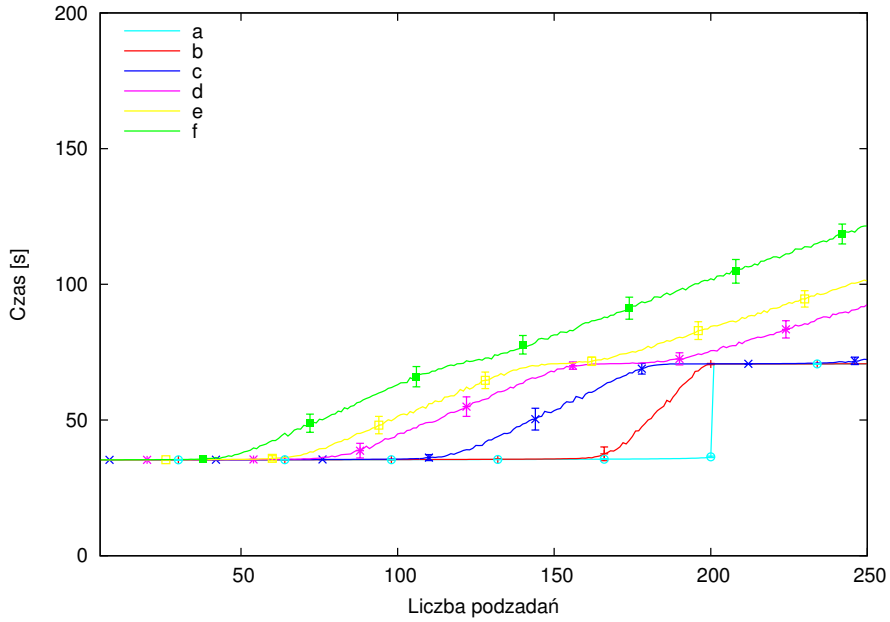
Wykres 6: Wykres średniego czasu realizacji zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 0.98 czasu realizacji pojedynczego zadania



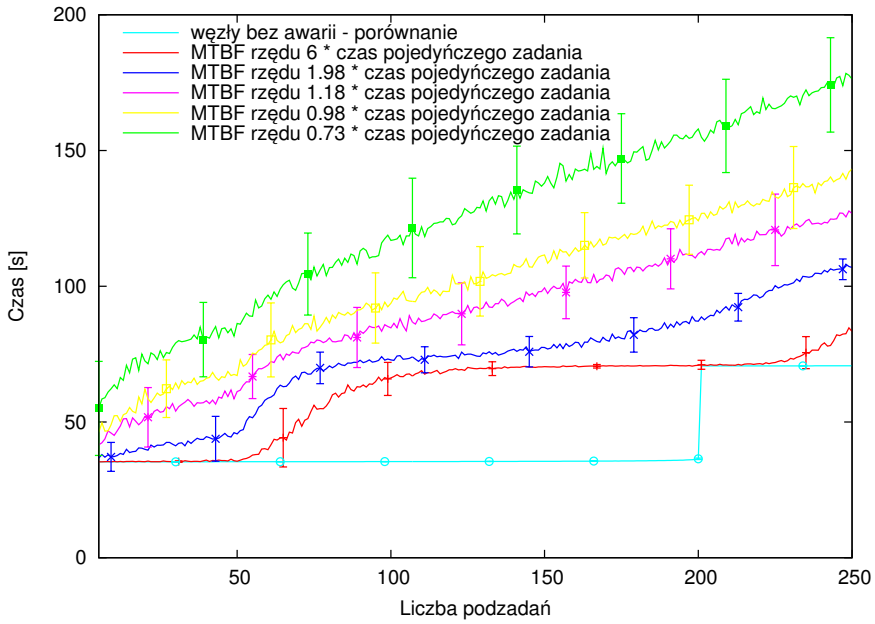
Wykres 7: Wykres średniego czasu realizacji zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 0.84 czasu realizacji pojedynczego zadania



Wykres 8: Wykres średniego czasu wykonania zlecenia w zależności od liczby podzadań. Opisy krzywych jak na wykresie 1. Sytuacja, gdy komputery nie ulegają awariom, została przedstawiona dla porównania. W pozostałych przypadkach MTBF wynosi w przybliżeniu 0.73 czasu realizacji pojedynczego zadania



Wykres 9: Porównanie czasu wykonania wszystkich obliczeń przy użyciu optymalnej polityki dla kilku wartości MTBF. Krzywa a, dla porównania, pokazuje czas wykonania obliczeń systemie bez awarii. Krzywe b-f, przedstawiają czasy wykonania obliczeń przy użyciu optymalnej polityki dla węzłów których wartości MTBF są następujące:
b) 6 krotność czasu pojedynczego zadania
c) 1,98 krotność czasu pojedynczego zadania
d) 1,18 krotność czasu pojedynczego zadania
e) 0,98 krotność czasu pojedynczego zadania
f) 0,73 krotność czasu pojedynczego zadania



Wykres 10: Porównanie czasu wykonania wszystkich obliczeń przy użyciu najlepszej, możliwej do implementacji polityki (zwielokrotnienie 4-krotne) dla kilku wartości MTBF. Krzywa a, dla porównania, pokazuje czas wykonania obliczeń systemie bez awarii. Krzywe b-f, przedstawiają czasy wykonania obliczeń przy użyciu optymalnej polityki dla węzłów których wartości MTBF są jak na wykresie 9

Wykres 1 przedstawia wyniki symulacji, gdy wartość MTBF jest dość duża w porównaniu z czasem obliczeń – tj, gdy stanowi 6-krotność średniego czasu policzenia pojedynczego zadania. W przypadku liczby zadań równej połowie liczby komputerów polityka polegająca na dublowaniu – krzywa a – przynosi najlepsze rezultaty – lepsze niż przy zastosowaniu polityk dopuszczających większe liczby duplikacji jednego obliczenia – krzywe b i c. Polityki dopuszczające większe liczby duplikacji obliczenia są bardziej odpowiednie w przypadku mniejszej i/lub większej liczby zadań.

Wykres dla sytuacji, gdy komputery nie ulegają awariom, jest prosty do przewidzenia – będzie to czas realizacji pojedynczego zadania pomnożony przez $\lceil m/n \rceil$, gdzie m to liczba zadań, a n to liczba węzłów. Na wykresach krzywa mu odpowiadająca jest oznaczona literą e.

W przypadku polityki optymalnej – krzywa f wykresu 1, a także w przypadku polityk zwielokrotniających obliczenia, wykresy są na pewnych odcinkach zbliżone to wykresów funkcji liniowych, stałych, z wyjątkiem miejsc, gdzie liczba węzłów jest zbliżona do liczby zadań pomnożonej przez maksymalną liczbę zwielokrotnień. Takie zachowanie jest bardzo zbliżone do zachowania wykresu w przypadku polityki optymalnej – jednak skoki następują dużo szybciej niż gdy liczba zadań jest zbliżona do liczby węzłów.

W przypadku polityki polegającej na dublowaniu liczby zadań, odcinki liniowe są już lekko pochyłe, a miejsce, w którym następuje skok, znajduje się dalej niż w przypadku polityk pozwalających na trzy- i czterokrotne zwielokrotnienie. Jest to zgodne z przewidywaniem okolic skoków wykresów – tj. zwielokrotnienie powinno odbywać się w okolicach punktów, gdzie $m = n/k$, przy czym n to liczba węzłów, m to liczba zadań, a k to maksymalna liczba kopii jednego zadania liczona w danym momencie w systemie.

Dla polityki losowej – krzywa g na wykresie 1, skok ma miejsce dla liczby podzadań przekraczającej nieznacznie wielokrotność liczby węzłów. Skoki te jednak są dużo słabsze i wolniejsze niż w przypadku polityki idealnej albo w przypadku zwielokrotnienia zadań. Jest to spowodowane wpływem losowych wyłączeń węzłów, który nie jest neutralizowany właściwą polityką rozdziału zadań. Polityka polegająca na wyborze węzła z największym MTBF – krzywa d, daje wyniki podobne do losowej – węzły mają ten sam MTBF.

Wykres 2 sporządzono dla MTBF odpowiadającego 2,98 średniego czasu realizacji zadania. Widać na nim, że polityki polegające na dopuszczeniu 3- (krzywa b), i 4-krotnych (krzywa c) replikacji, są zdecydowanie lepsze niż ta z replikacją 2-krotną. Nadal jednak występuje taka liczba węzłów, przy której 2-krotna replikacja jest lepsza od 3- i 4-krotnej. Podobnie jak w przypadku poprzedniego wykresu, dla polityki idealnej oraz polityk dopuszczających zwielokrotnienie nadal występują skoki. Są one jednak już bardziej spłaszczone i wolniejsze, co wynika z coraz większego wpływu losowych awarii węzłów.

Na wykresie 3, przy wartości MTBF około 1.98 średniego czasu wykonania pojedynczego zadania, okazuje się, że 2-krotna replikacja pojedynczego zadania nie daje wyników lepszych niż 3- i 4-krotna. Istnieje taka liczba węzłów, przy której 3-krotna replikacja jest lepsza od 4-krotnej. Wykresy wszystkich funkcji są jednak coraz bardziej pochyłe i coraz bardziej zbliżone do prostych, a skoki coraz mniejsze.

Na wykresach 4, 5 i 6 przedstawiono dalsze zwiększanie się awaryjności węzłów. Wykresom tym odpowiadają kolejne wartości MTBF – 1.48, 1.18, 0.98 czasu pojedynczego zadania. Opisane krzywymi b i c polityki polegające na 3- i 4-krotnym zwielokrotnieniu dają dla pewnej liczby podzadań jednakowy wynik, przy czym różnica ta jest mniejsza od odchylenia standardowego dla danej wartości. Dla pozostałej liczby podzadań polityka polegająca na 4-krotnym zwielokrotnianiu daje zdecydowanie lepsze rezultaty. Wykresy dla wszystkich polityk są także coraz bardziej zbliżone do linii prostych.

Dalsze pogarszanie się jakości węzłów przedstawiono na wykresach 7 i 8. Średnie wartości MTBF węzłów są mniejsze niż czas realizacji pojedynczego zlecenia. W przypadku takich parametrów polityka polegająca na 4-krotnym zwielokrotnieniu zadania staje się zdecydowanie optymalna. Od pewnego miejsca wykresy dla wszystkich właściwie polityk stają się liniami prostymi, co wynika z bardzo dużej liczby zerwanych obliczeń.

Aby dokładniej przedstawić efekty zmian średniego czasu pomiędzy awariami, sporządzono wykresy 9 i 10. Na wykresie 9 przedstawiono, jak wyglądają zmiany czasu realizacji zlecenia przy użyciu optymalnej – niemożliwej do implementacji – polityki szeregowania. Natomiast na wykresie 10 pokazano, jak zmiany te wyglądają dla najlepszej polityki z możliwych do zrealizowania – polegającej na zastosowaniu 4-krotnego zwielokrotnienia. Na wykresach wyraźnie widoczne są szybkie przejścia – dla w miarę niezawodnych węzłów, a także wypłaszczenia – dla węzłów coraz bardziej zawodnych.

Wniosek z przeprowadzonej symulacji wydaje się prosty: im większe zwielokrotnienie pojedynczego zadania, tym lepiej. Wniosek ten jest prawdziwy tylko dla systemu w którym panują warunki zgodne z założeniami przyjętymi dla tej symulacji.

Wyniki najprostszych symulacji potwierdzają zgodną z założeniami pracę symulatora. Jednocześnie pokazują, że ze względu na zawodność systemu nie zawsze można zapewnić matematyczny opis jego pracy. Symulacje dobrze ilustrują sytuacje szczególne, co uzasadnia ich stosowanie dla bardziej skomplikowanych przypadków oraz w odniesieniu do badania systemów typu desktop grid.

2.2 Symulacje uwzględniające parametry sieci

Z chwilą stwierdzenia poprawnej pracy symulatora w odniesieniu do prostych przypadków, wykorzystano go w celu ulepszenia polityki szeregowania i alokacji zasobów w odniesieniu do bardziej skomplikowanego systemu [46]. W systemie tym został uwzględniony model sieci opisanej w części 1.2 rozdziału V, a zdefiniowany w części 2 rozdziału V.

Parametry testowanego systemu i obliczeń w nim zlecanych

Modelowana sieć, do której podłączono węzły gridu, składała się z połączonych pierścieni. Zarządca został podłączony bezpośrednio do jednego z pierścieni. Węzły były podłączane do pierścieni dodatkowymi połączeniami w relacji jeden węzeł – jedno połączenie. Taka topologia została uznana za równoważną topologii drzewiastej w przypadku zastosowania polityki równego podziału pasma, opisanej w części 1.2.

Element sieciowy	Rodzaj i przepustowość
Pierścienie	ATM 155 Mb/s (19 MB/s)
Połączenie pierścieni	Światłowód 10 Gb/s
Podłączenie węzłów	ADSL 8 Mb/s (1 MB/s)

Tabela VII.1: Parametry elementów sieci

Modelowana sieć składała się z sześciu pierścieni, a jej parametry przedstawiono w tabeli VII.1.

Symulacje przeprowadzono dla trzech rodzajów sieci: nieskończenie szybkiej, limitowanej ze stałym obciążeniem pierścienia, do którego podłączony jest zarządca 1,56 MB/s, oraz z obciążeniem zewnętrznym o średniej 1,56 MBs, ale generowanym z szeregu czasowego. Obciążenia zewnętrzne miały wyższy priorytet dostępu do sieci niż obliczenia gridowe.

Symulacje przeprowadzono dla systemów zawierających od 100 do 700 węzłów. Dla urealnienia symulacji, rozkłady wartości MTBF komputerów były losowane z dwupunktowego rozkładu normalnego (unormowana suma rozkładów normalnych ze średnimi w $2/3$ i $4/3$ globalnej średniej). Globalny średni MTBF węzła wynosił 2700 s.

Dla potrzeb symulacji, na bazie historycznych danych z klastra ICM wygenerowano zbiór zleceń na okres 1 tygodnia. Każde zlecenie zawierało 64 jednakowe zadania wymagające od 900 do 2700 sekund obliczeń (bez transferu).

W historycznych danych pracy systemu wyróżnia się znikoma liczba zleceń wysyłanych przez użytkowników w poniedziałek nad ranem. Okres ten będzie

traktowany w symulacji jako tzw. catch up – liczba zleceń, jakie zostaną w systemie po jego upływie, będzie miarą jakości pracy tegoż systemu. Drugą z miar stanowi średni czas wykonywania zlecenia. Ponieważ we wszystkich symulacjach użyty został ten sam zestaw zleceń, średni czas, jaki upływa od wysłania zlecenia do systemu, do końca złączania wyników, stanowi bardzo dobrą miarę jakości pracy systemu.

Użyte polityki szeregowania i alokacji zasobów

W trakcie symulacji okazało się, że polityki szeregowania przedstawione w części 2.1, powodowały – powyżej pewnej liczby węzłów – pogorszenie jakości pracy systemu wraz ze wzrostem liczby węzłów. Scenariusz takiego zachowania był następujący: przy większej liczbie węzłów system próbował transferować w danej chwili więcej danych, transfery stawały się wolniejsze, w związku z czym trwały dłużej i w pewnym momencie przekraczały znacząco MTBF węzłów. W momencie awarii węzła transfer ulegał przerwaniu i realizacja całego podzadania była powtarzana od początku – w związku z tym również sam transfer musiał być ponowiony. Taki ponowny transfer zwiększał dodatkowo obciążenie sieci. Aby uniknąć tego rodzaju sytuacji, stworzyliśmy następującą politykę alokacji zasobów:

Polityka uwzględniająca obciążenie sieci polega na niedopuszczeniu do zbytniego wydłużenia czasu transferów danych. Zbytnie wydłużenie transferu to takie, w którym oczekiwany czas transferu zsumowany z czasem procesora jest większy niż średnia wartość MTBF węzłów wykonujących zlecenia. Wymaga on, by:

$$T < \overline{MTBF}, \tag{VII.1}$$

gdzie T to całkowity czas realizacji zadania, od momentu przypisania do węzła aż do momentu przesłania wyników. $\overline{MTBF}$ to średnia wartość $MTBF$. Czas T składa się z części stałej niezależnej od sieci – czasu obliczenia t_{je} i części zmiennej – czas transferu t_{tr}:

$$T = t_{je} + t_{tr}. \tag{VII.2}$$

Przy uproszczającym założeniu, że wszystkie transfery są tej samej wielkości, otrzymujemy:

$$\frac{n_{tr}}{n_{tot}} = \frac{t_{tr}}{t_{tr} + \overline{t_{je}}}, \tag{VII.3}$$

gdzie n_{tr} to liczba transferów, n_{tot} to liczba zadań przydzielonych w danej chwili do węzłów, $\overline{t_{je}}$ oznacza średni czas, w jakim zadanie jest liczone na węźle.

$$t_{tr} = D\frac{n_{tr}}{b} \tag{VII.4}$$

D to liczba danych koniecznych do przetransferowania (w obie strony) w trakcie realizacji zadania, a b to dzielone pasmo (pasmo pierścienia, do którego podłączony jest zarządca). Równania te są prawdziwe, gdy pierścień jest w 100% obciążony.

$$\frac{n_{tr}}{n_{tot}} = \frac{t_{tr}}{t_{tr}+\overline{t_{je}}} \tag{VII.5}$$

$$\frac{n_{tot}}{n_{tr}} = \frac{t_{tr}+\overline{t_{je}}}{t_{tr}} \tag{VII.6}$$

$$\frac{n_{tot}}{n_{tr}} = 1+\frac{\overline{t_{je}}}{t_{tr}} \tag{VII.7}$$

$$\frac{n_{tot}-n_{tr}}{n_{tr}} = \frac{\overline{t_{je}}}{t_{tr}} \tag{VII.8}$$

Po podstawieniu t_{tr} otrzymujemy:

$$\frac{n_{tot}-n_{tr}}{n_{tr}} = \frac{\overline{t_{je}}}{D\frac{n_{tr}}{band}} \tag{VII.9}$$

$$\frac{n_{tot}-n_{tr}}{n_{tr}} = \frac{\overline{t_{je}}b}{Dn_{tr}} \tag{VII.10}$$

$$n_{tot}-n_{tr} = \frac{\overline{t_{je}}b}{D} \tag{VII.11}$$

$$n_{tr} = n_{tot}-\frac{\overline{t_{je}}b}{D} \tag{VII.12}$$

Po podstawieniu n_{tr} co równania VII.4, równanie przyjmuje postać:

$$t_{tr} = D\frac{n_{tot}-\frac{\overline{t_{je}}b}{D}}{b} \tag{VII.13}$$

$$t_{tr} = \frac{D}{b}(n_{tot}-\frac{\overline{t_{je}}b}{D}) \tag{VII.14}$$

I ostatecznie warunek przydzielenia zadania węzłowi przyjmuje postać:

$$t_{je}+\frac{D}{b}(n_{tot}-\frac{\overline{t_{je}}b}{D}) < \overline{MTBF} \tag{VII.15}$$

Warunek ten należy zastosować tylko w sytuacji, gdy pierścień jest wypełniony, co wyraża się wzorem:

$$n_{tr} S_{ep} < b, \tag{VII.16}$$

gdzie S_{ep} to prędkość podłączenia węzła do najbliższego pierścienia. Opisane powyżej warunki są dodawane do dowolnej polityki alokacji zasobów i szeregowania, tworząc nową politykę uwzględniającą obciążenie sieci.

Wyniki

Na wykresach 11-14 przedstawione są rezultaty symulacji. Wykres 11 przedstawia zależność średniego czasu wykonania zlecenia od liczby węzłów w systemie. Przy liczbie pomiędzy 200 a 300 węzłów wykres, który do tego miejsca jest zbliżony do linii prostej (funkcja liniowa malejąca), nagle zaczyna się wypłaszczać, a dla niektórych polityk – wręcz rosnąć. Liczba węzłów, powyżej której dla polityki losowej średni czas wykonania zlecenia zaczyna rosnąć będzie nazywana wartością krytyczną. Powiększenie tego fragmentu znajduje się na wykresie 12. Duże podobieństwo do wykresu zależności średniego czasu realizacji zadania od liczby zadań wykazuje wykres liczby zadań niewykonanych, pozostałych w kolejce na koniec symulacji w zależności od liczby węzłów – wykres 13 i powiększenie obszaru zmian na wykresie 14.

Krzywe na wykresach oznaczają odpowiednio:

a – symulacje przeprowadzone z użyciem polityki przydzielania podzadania do węzła z największą wartością parametru MTBF, z uwzględnieniem obciążenia sieci. Symulacje przy stałej przepustowości sieci;

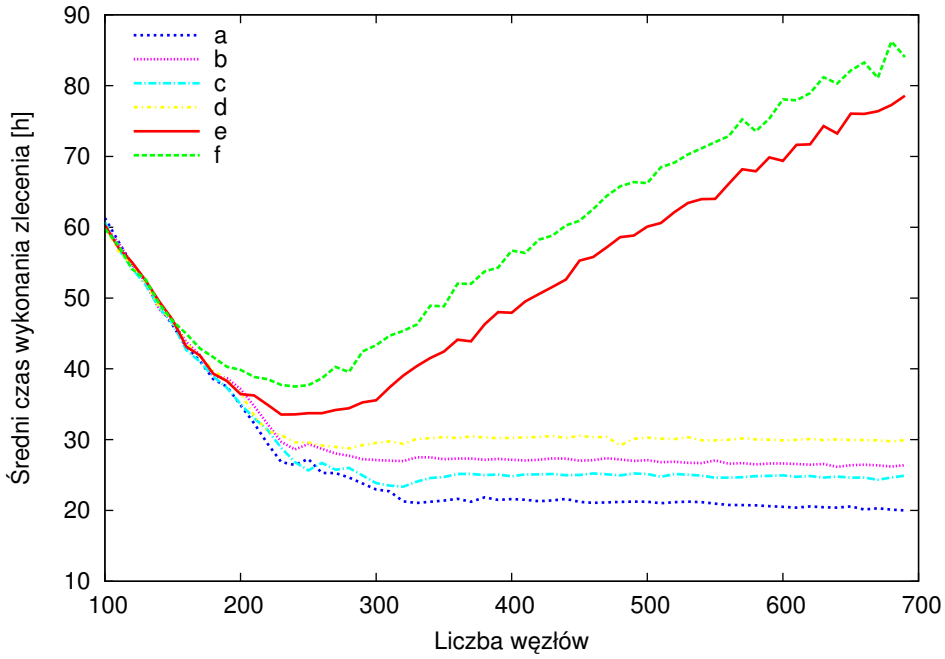
b – symulacje przeprowadzone jak dla krzywej a, ale ze zmienną przepustowością sieci;

c – symulacje przeprowadzone z użyciem losowego przydziału podzadania do węzła, ale z uwzględnieniem obciążenia sieci. Symulacje przy stałej przepustowości sieci;

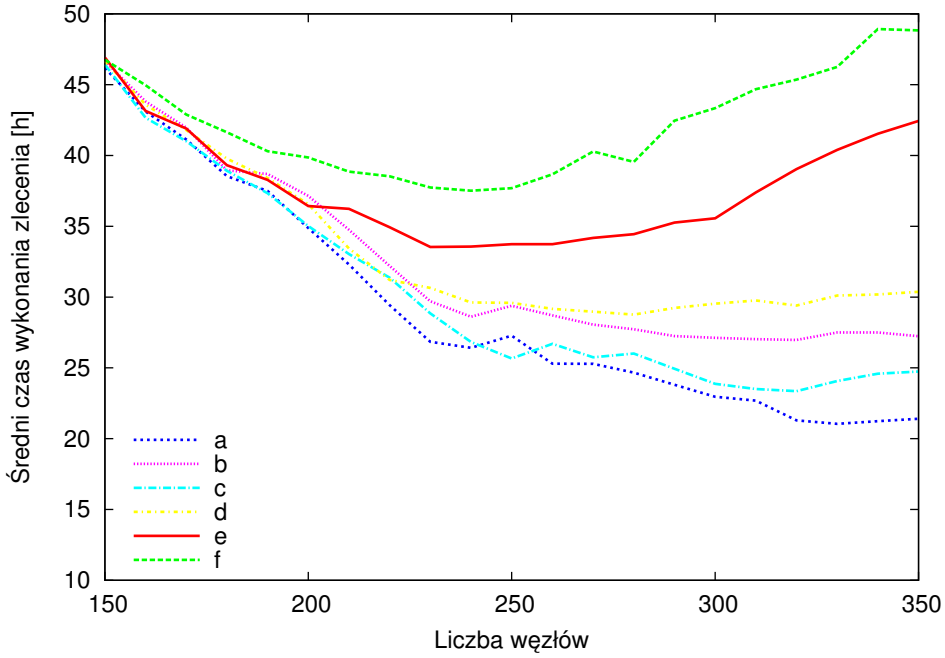
d – symulacje jak dla krzywej c, ale przy zmiennej przepustowości sieci;

e – symulacje przeprowadzone z użyciem losowego przydziału podzadania do węzła, bez uwzględnienia obciążenia sieci. Symulacje przy stałej przepustowości sieci;

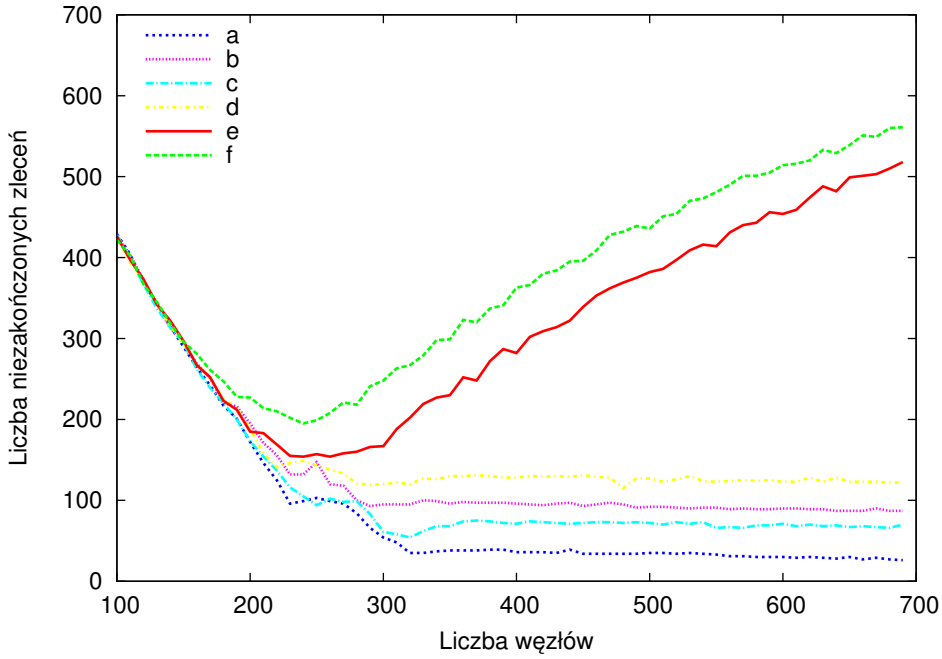
f – symulacje jak dla krzywej e, ale przy zmiennej przepustowości sieci.



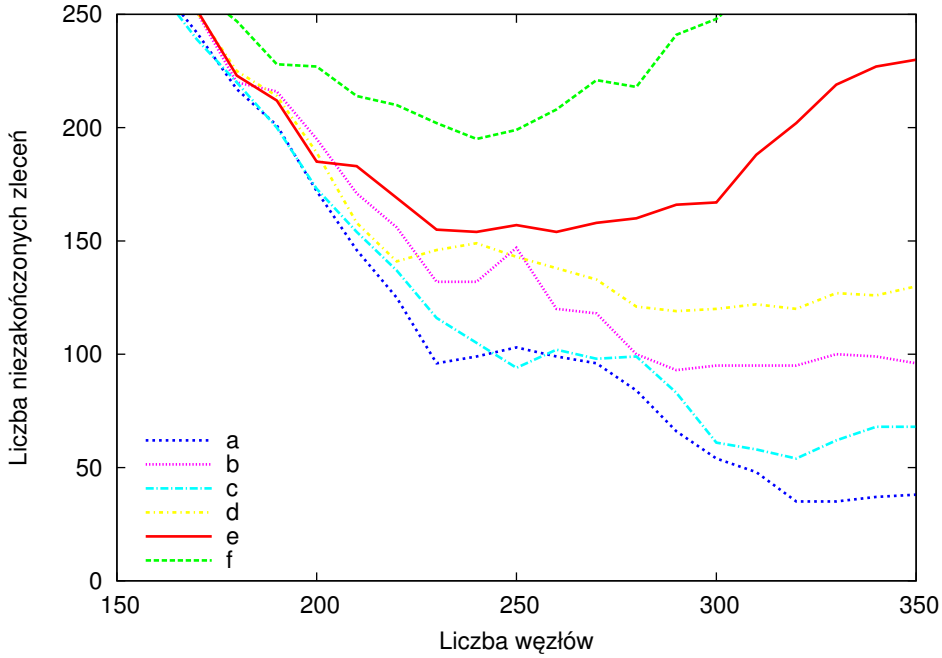
Wykres 11: Średni czas wykonania zlecenia (opis krzywych w tekście)



Wykres 12: Średni czas wykonania zlecenia – powiększenie fragmentu powyżej którego, w części polityk, następuje pogorszenie wyniku ze względu na nieuwzględnianie obciążenia sieci



Wykres 13: Liczba zleceń pozostałych w systemie na koniec symulacji (opis krzywych w tekście)



Wykres 14: Liczba zleceń pozostałych w systemie na koniec symulacji - powiększenie fragmentu powyżej którego, w części polityk następuje pogorszenie wyniku ze względu na nieuwzględnianie obciążenia sieci

Wykresy pokazują, że polityki uwzględniające obciążenie sieci przynoszą istotną poprawę wydajności systemu. Przedstawiono wykresy działania dla trzech różnych polityk: losowej oraz dwóch wersji polityki uwzględniającej obciążenie sieci poprzez opóźnianie obliczeń, tj. polityki dobierającej komputer losowo oraz polityki dobierającej komputer o największym parametrze MTBF. W przypadku polityki nieuwzględniającej obciążenia sieci – powyżej pewnego progu, pomimo dalszego zwiększania liczby komputerów, pogarsza się wydajność systemu, mimo że powinna się ona polepszać lub w skrajnym przypadku utrzymywać na stałym poziomie. Na wykresach 11 i 13 widać, że w przypadku przekroczenia granicznej liczby komputerów, polityka dobierająca komputery z lepszym parametrem MTBF daje lepsze rezultaty niż polityka losowa (w kontekście polityk uwzględniających obciążenie sieci). W trakcie testów stwierdzono nieskuteczność rozszerzania polityk o opcję zwielokrotniania zadań, jak miało to miejsce w przypadku nieskończenie szybkiej sieci (część 2.1). Wynika to z faktu, że w trakcie obliczeń zawsze jest dość zadań, aby obciążyć wszystkie dopuszczalne węzły. Właściwość ta jest jeszcze wyraźniejsza przy stosowaniu polityki uwzględniającej obciążenia sieci. Nieuwzględnienie obciążania sieci przy zwielokrotnianiu powoduje pogorszenie wyników. Zwielokrotnianie stanowi wartą zapamiętania politykę, gdyż można ją dodać jako rozszerzenie do innych polityk w sytuacji, gdy system wykazuje nadmiar wolnych wszystkich zasobów – z siecią włącznie.

Jednym z wyników symulacji jest określenie maksymalnej liczby węzłów, które mogą jednocześnie prowadzić obliczenia. W pierwszych próbach stworzenia symulatora uwzględniającego obciążenie sieci testowano warunek:

$$Dn_{tot}/band < \overline{MTBF} \tag{VII.17}$$

Oznacza on, że czas trwania transferów nie powinien przekroczyć czasu realizacji zadania w warunkach optymalnych. Ponieważ zadanie musi przesłać w sumie 160 MB, przepustowość pierścienia, do którego podłączony jest zarządca, wynosi 19 MB/s, z czego około 1,5 MB/s jest poświęcone na pracę poza gridem, w wyniku czego średnie dostępne pasmo wynosi 17 MB/s, średnie MTBF wynosi 2700 s. Po podstawieniu otrzymano:

$$\frac{160MBn_{tot}}{17,44\text{MB/s}} < 2700\text{s} \tag{VII.18}$$

$$n_{tot} < 295 \tag{VII.19}$$

Powyżej tego poziomu może występować wzrost wydajności spowodowany tym, że zarządca będzie mógł wybierać do obliczeń coraz lepsze komputery. Wzór ten, wskazujący optymalną liczbę węzłów w systemie, nie umożliwia jeszcze stworzenia dobrej polityki przydzielania zadań do węzłów, gdyż nie uwzględnia

chwilowych obciążeń sieci. Jednak jego zastosowanie powoduje bardzo wyraźny wzrost wydajności obliczeń. Wyraźne jest to na wykresach 11-14, gdzie powyżej liczby 295 węzłów (wartość krytyczna), następuje zdecydowane pogorszenie jakości wyników dla polityk nieuwzględniających obciążenia sieci.

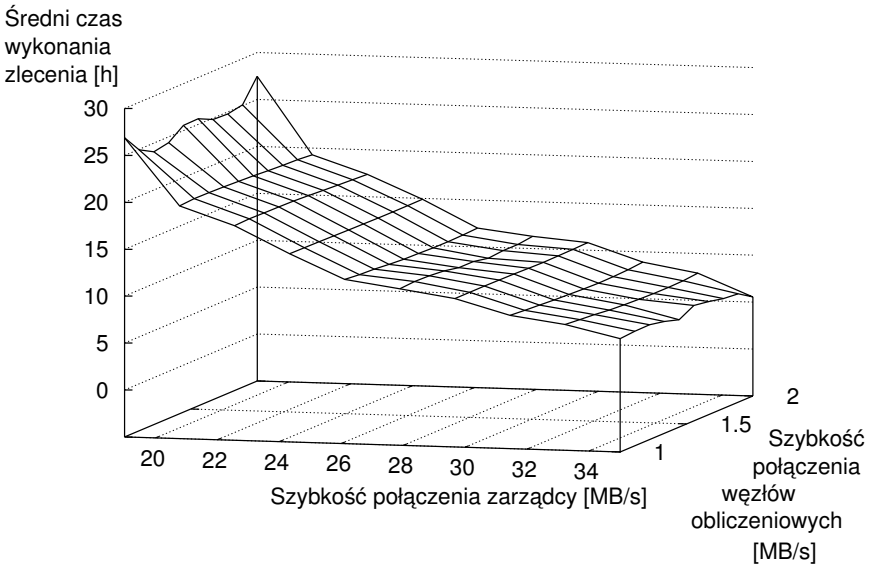
Na wykresach dla wszystkich testowanych polityk, sieć o zmiennym obciążeniu powoduje zmniejszenie wydajności systemu względem sieci, której obciążenie jest stałe. Wskazuje to na silną korelację pomiędzy obciążeniem sieci, a liczbą wysyłanych zleceń. Gdy pojawiają się zlecenia do wykonania, sieć jest najbardziej obciążona i konieczne staje się opóźnianie zadań. W sytuacji, gdy przy mniej obciążonej sieci nie pojawiają się nowe zlecenia, wykonywane są zadania wcześniej uruchomione.

Symulacje wykazały, że bardzo istotny i nie zawsze oczywisty jest dobór dobrej polityki przydziału zadań do węzłów. Techniki stosowane w normalnych systemach odbiegają od tych stosowanych w systemach typu desktop grid i w gridach społecznościowych. W systemach typu desktop grid często się okazuje, że opóźnienie wykonywania jakiegoś obliczenia, pomimo dostępności zasobów obliczeniowych, daje lepsze rezultaty niż wykonanie go natychmiast.

3. Przyczyny obniżenia wydajności w systemie typu desktop grid

Jedno z zastosowań symulatora polega na poszukiwaniu wąskich gardeł w systemie oraz sprawdzaniu, na ile dobrze wykorzystane są dostępne zasoby [47]. Wąskie gardło można zdefiniować jako element, którego nawet nieznaczna zmiana wpłynie w istotny sposób na pracę systemu. W przypadku badanych systemów typu desktop grid elementy, których parametry można zmienić, to: zarządca, połączenia sieciowe oraz liczba węzłów w systemie. Poprawienie wydajności obliczeniowej węzłów, choć teoretycznie możliwe, jest zbyt kosztowne. W przyjętym modelu obliczeń praca wykonywana przez zarządcę jest zaniedbywana, w związku z czym wydajność pracy zarządcy nie ma wpływu na szybkość obliczeń. Jedynymi badanymi elementami, które mogą stanowić wąskie gardła, są połączenia sieciowe oraz liczba węzłów w systemie. Połączenia sieciowe obejmują przepustowość pierścienia, do którego podłączony jest zarządca, oraz podłączenia węzłów obliczeniowych do systemu.

Parametry testowanego systemu: w trakcie symulacji wykorzystano model systemu i zestaw zadań jak w trakcie ustalania właściwej polityki przydziału zadań. Zarządca używał optymalnej strategii szeregowania: wyższy MTBF najpierw, połączonej z warunkiem uwzględniania obciążenia sieci. Strategia ta okazała się



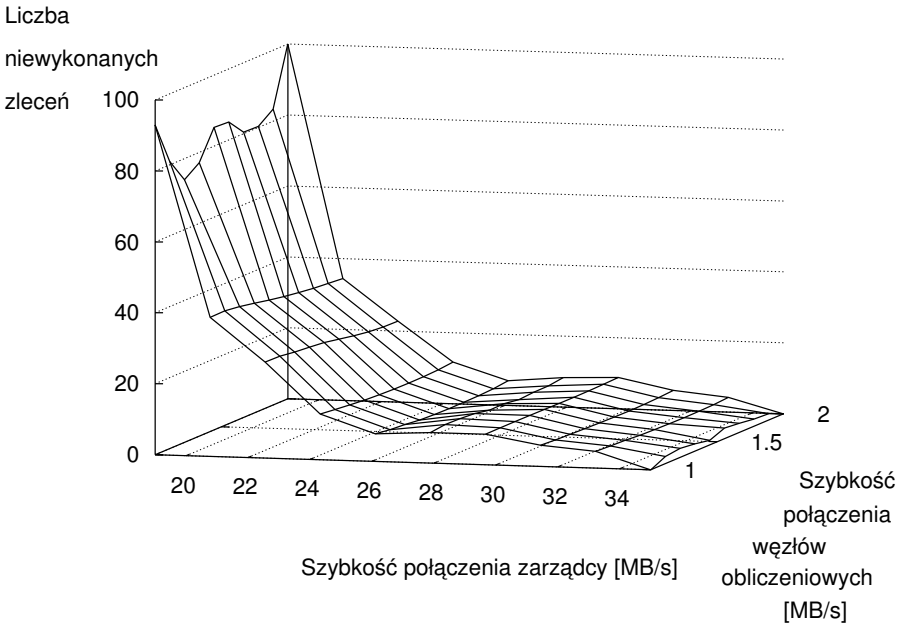
Wykres 15: Średni czas realizacji zlecenia jako funkcja przepustowości pierścienia i połączeń do węzłów dla 300 węzłów

najlepsza wśród wszystkich zidentyfikowanych. Symulowany system wykazywał następujące wartości parametrów:

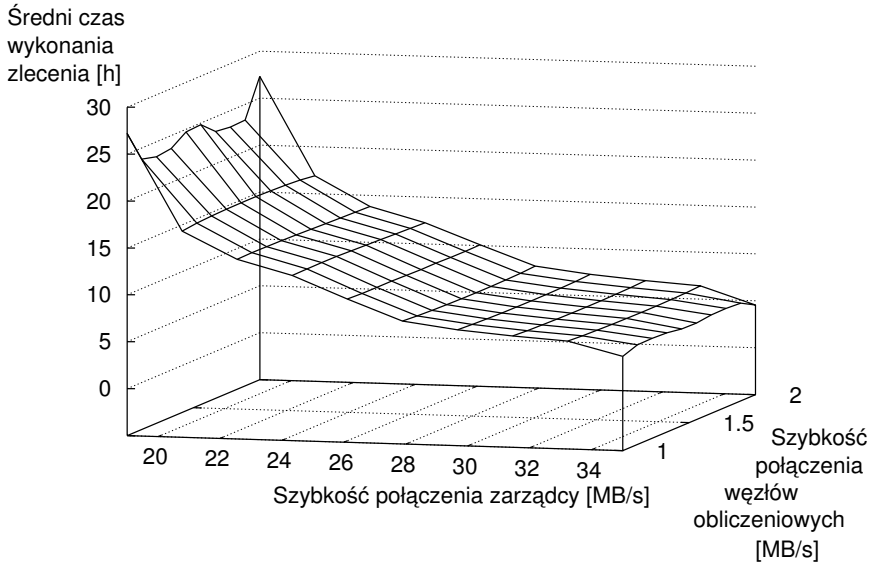
- liczba węzłów od 300 do 390 ze skokiem 10;
- przepustowość pierścienia zawierającego zarządcę od 19 do 38 MB/s ze skokiem 0,8 MB/s. Należy tu zaznaczyć, że średnio 1.56 MB/s było zużywane w tym pierścieniu na obciążenie zewnętrzne;
- przepustowość połączenia pomiędzy węzłami a pierścieniami – od 1 do 2 MB/s ze skokiem 0,05 MB/s

Na wykresach 15, 17, 19 i 21 została przedstawiona zależność średniego czasu realizacji zlecenia od przepustowości pierścienia, do którego podłączony jest zarządca, i prędkości podłączenia węzłów z pierścieniami, przy czym każdy wykres został sporządzony dla różnych ilości węzłów.

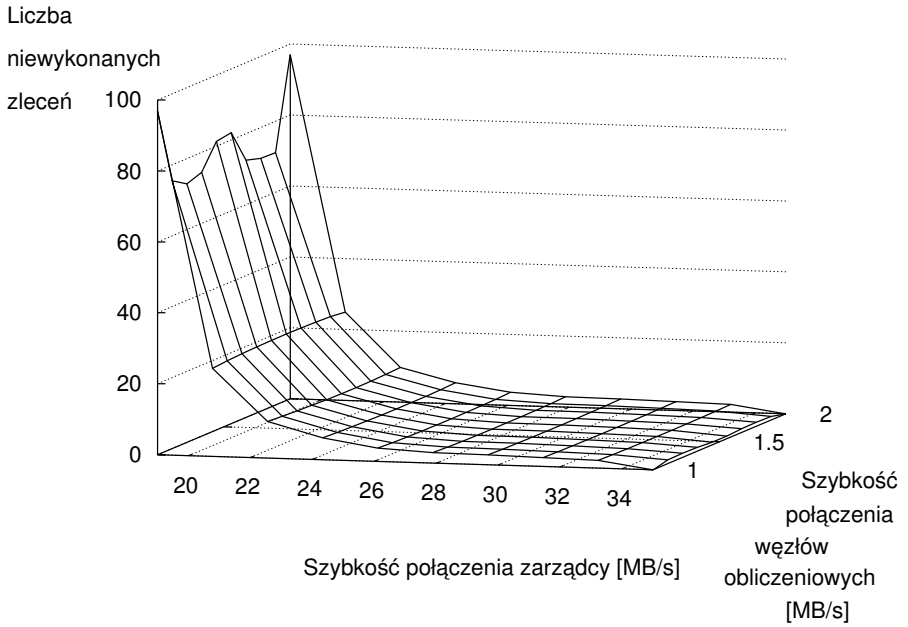
Na wykresach 16, 18, 20 i 22 przedstawiona jest zależność liczby zleceń pozostałych w kolejce na koniec symulacji od przepustowości pierścienia, do którego podłączony jest zarządca, i prędkości podłączenia węzłów z pierścieniami. Wykres 23 przedstawia średni czas realizacji zlecenia w zależności od przepustowości pierścienia i liczby węzłów, dla prędkości połączenia węzłów z pierścieniem wynoszącej 1 MB/s. Wykres 24 przedstawia liczbę zleceń pozostałych w kolejce



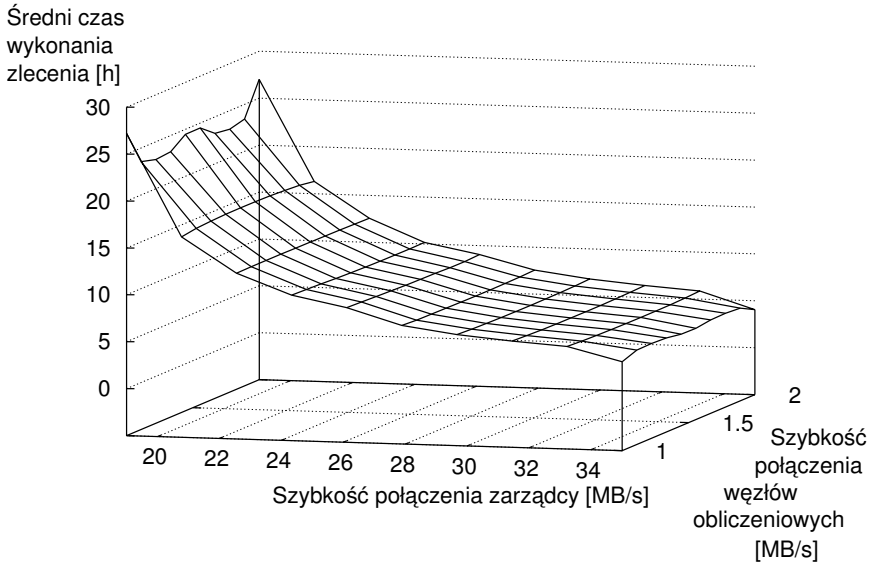
Wykres 16: Liczba zleceń pozostałych w kolejce na koniec symulacji jako funkcja przepustowości pierścienia i połączeń do węzłów dla 300 węzłów



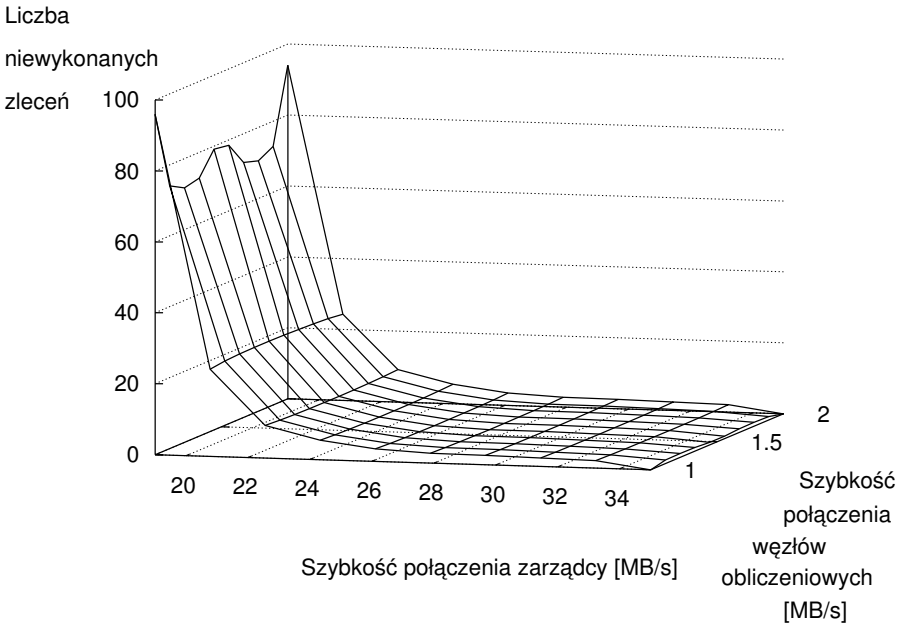
Wykres 17: Średni czas realizacji zlecenia jako funkcja przepustowości pierścienia i połączeń do węzłów dla 330 węzłów



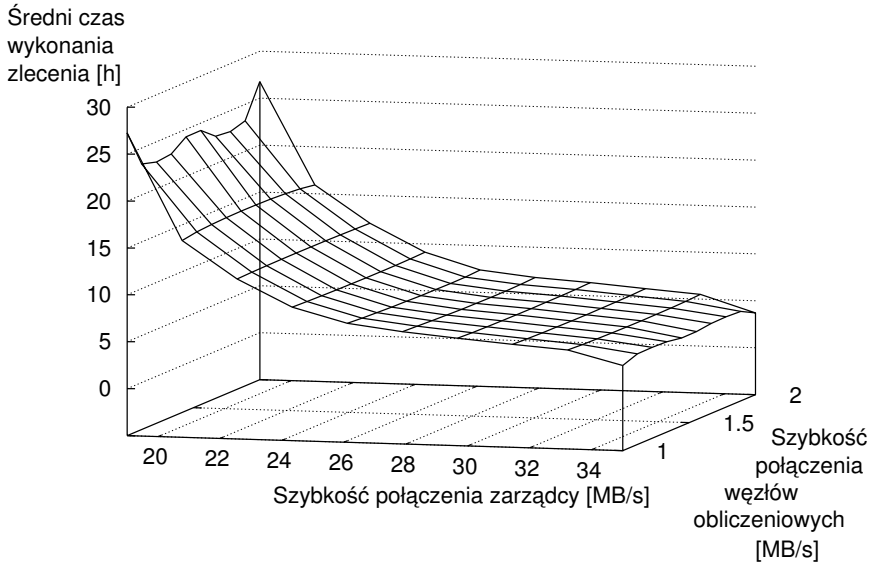
Wykres 18: Liczba zleceń pozostałych w kolejce na koniec symulacji jako funkcja przepustowości pierścienia i połączeń do węzłów dla 330 węzłów



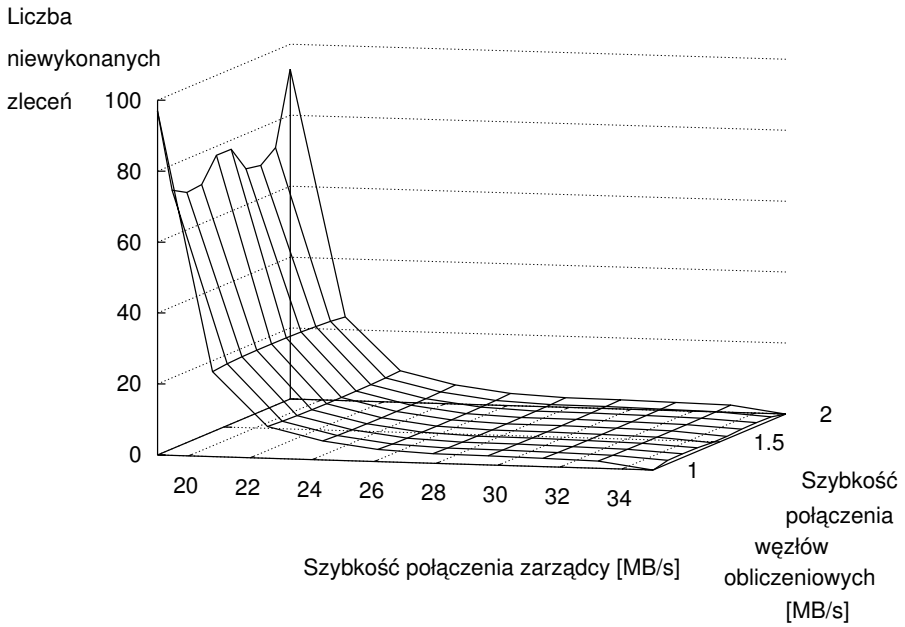
Wykres 19: Średni czas realizacji zlecenia jako funkcja przepustowości pierścienia i połączeń do węzłów dla 360 węzłów



Wykres 20: Liczba zleceń pozostałych w kolejce na koniec symulacji jako funkcja przepustowości pierścienia i połączeń do węzłów dla 360 węzłów



Wykres 21: Średni czas realizacji zlecenia jako funkcja przepustowości pierścienia i połączeń do węzłów dla 390 węzłów



Wykres 22: Liczba zleceń pozostałych w kolejce na koniec symulacji jako funkcja przepustowości pierścienia i połączeń do węzłów dla 390 węzłów

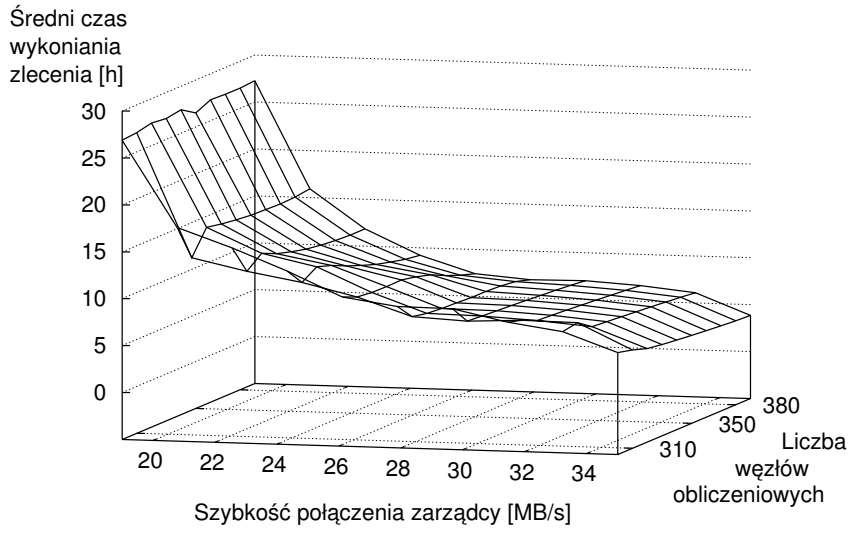
na koniec symulacji w zależności od przepustowości pierścienia i liczby węzłów, dla prędkości połączenia węzłów z pierścieniem wynoszącej 1 MB/s.

Przedstawione wyniki pokazują, że wpływ prędkości połączenia pomiędzy węzłami a pierścieniami na wydajność gridu jest znikomy. Jest to wyraźnie widoczne na wykresach 15-22. Zmiany wzdłuż osi oznaczającej prędkość podłączenia węzłów do pierścienia są minimalne. Rzut wykresu na płaszczyznę prostopadłą do niej stanowiłby krzywą, a nie obszar – jak miałoby to miejsce, gdyby szybkość podłączenia węzłów do pierścienia była czynnikiem istotnym.

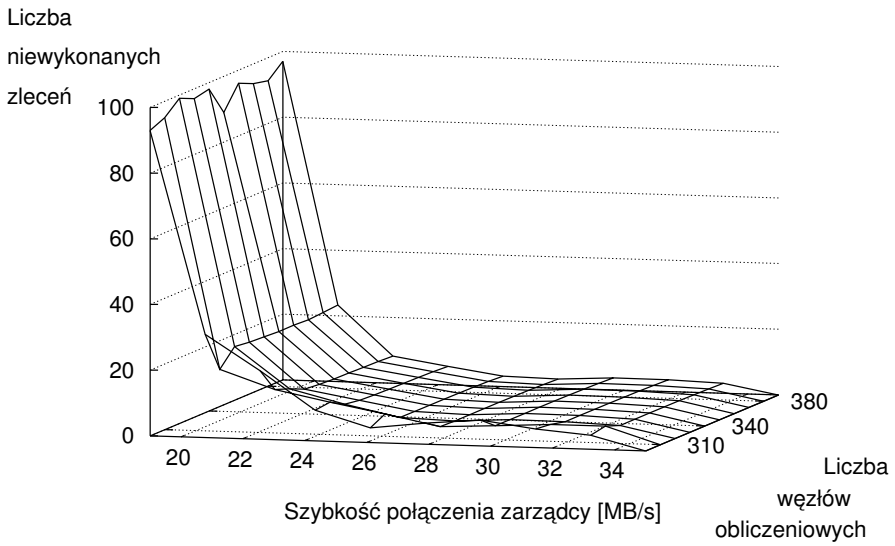
Ponadto, na wykresach 15-22, dla liczby komputerów 300, wzrost wydajności systemu w zależności od prędkości podłączenia zarządcy jest najmniejszy, co wynika z nieprzekroczenia optymalnej liczby węzłów dla podanych parametrów wydajności systemu – jest to przypadek skrajny. Przy większej liczbie węzłów, ze względu na poprawne działanie ochrony sieci przed przeciążeniem, wyniki symulacji nie różnią się znacząco. Poprawa w tym względzie wynika z faktu, że można częściej wybierać trochę lepsze węzły.

Zwiększenie przepustowości pierścienia, do którego jest podłączony zarządca o 1/5 (z 20 MB/s do 24 MB/s), spowoduje ukończenie prawie wszystkich zadań, a także skrócenie średniego czasu realizacji zlecenia o połowę.

W wyniku symulacji, jako najwęższe gardło została wskazana przepustowość pierścienia, do którego podłączony jest zarządca. Symulacje umożliwiły także



Wykres 23: Średni czas realizacji zlecenia jako funkcja przepustowości pierścienia i liczby węzłów



Wykres 24: Liczba zleceń pozostałych w kolejce na koniec symulacji jako funkcja przepustowości pierścienia i liczby węzłów

określenie, o ile należy zwiększyć wydajność elementu będącego wąskim gardłem tak, aby przestał on nim być. Można więc uznać, że stworzony symulator systemów typu desktop grid bardzo dobrze nadaje się do:

- analizy wykorzystania zasobów w tych systemach,
- poszukiwania elementów mających największy wpływ na wydajność przedmiotowych systemów.

4. Podsumowanie

Przedstawiony model gridu zachował się, w łatwych do opisania teoretycznie przypadkach, zgodnie z oczekiwaniami. Symulacje potwierdziły jego poprawność. Model ujawnił ciekawe własności systemów typu desktop grid:

- w systemie nieobciążonym, a także w systemach z bardzo szybką siecią zwielokrotnianie obliczenia jest bardzo dobrą polityką. Jeśli zadania zaczynają się pojawiać w sposób ciągły, możliwe jest skompensowanie zysku ze zwielokrotniania ostatnich podzadań zlecenia przez dłuższe oczekiwanie na rozpoczęcie obliczeń dla podzadań z następnego zlecenia.
- w przypadku przeciążania sieci – w szczególności pełnego obciążenia połączenia zarządcy – opóźnienie wykonania obliczeń i niewysyłanie podzadań do gotowych węzłów może poprawić wydajność systemu.

Przeprowadzone symulacje pokazują, że warto użyć symulatora do rzeczywistego gridu w celu poprawienia jego wydajności. Umożliwia on nie tylko wskazanie elementu mającego największy wpływ na wydajność, ale również pozwala stwierdzić, jaka jego poprawa będzie wystarczająca dla poprawienia jakości pracy systemu. Z symulacji sieci wynika, że modele oparte na szeregach czasowych mogą być bardzo efektywne, a wyniki ich pracy zgodne z rzeczywistością.

Symulacje pokazały, że bardzo istotny wpływ na szybkość wykonania zleceń może mieć przepustowość podłączenia zarządcy do sieci. Symulacje pokazują o ile należy zwiększyć tę wartość, aby przestała być ona elementem najbardziej spowalniającym wykonanie obliczeń. Omawiany w pracy model pozwala na symulację elementów stanowiących tak zwane wąskie gardła: połączenia sieciowe, liczba węzłów w systemie, wydajność węzłów. W przypadku modelu systemu uwzględniającego ograniczoną przepustowość sieci, okazało się, że istnieje krytyczna ilość węzłów. Po jej przekroczeniu w przypadku polityk alokacji zasobów uwzględniających wydajność sieci zwiększenie ilości węzłów nie powoduje istotnego przyspieszenia obliczeń. W przypadku polityk nie uwzględniających obciążenia sieci po przekroczeniu krytycznej liczby węzłów następuje spowolnienie obliczeń.

Opisana w tym rozdziale symulacja systemu typu desktop grid pozwala na odnalezienie innych rozwiązań zwiększenia jego wydajności niż bardzo kosztowne (a często niemożliwe) zwiększenie wydajności każdego z węzłów. Pozwala na ustalenie elementu którego nieduża (mało kosztowna) poprawa wydajności będzie mieć największy wpływ na zwiększenie wydajności systemu.

Rozdział VIII

System typu desktop grid oparty na oprogramowaniu UNICORE

1. Wstęp

Przegląd systemów gridowych prowadzi do wniosku, że brakuje łatwej integracji systemów typu desktop grid z gridami usługowymi. W ostatnich latach prowadzono badania mające na celu połączenie gridów usługowych z gridami typu desktop grid lub gridami społecznościowymi. Badania te opisano w pracach [55, 12, 49].

W większości przypadków, aby umożliwić połączenie pomiędzy systemami, tworzono oddzielny fragment oprogramowania – interfejs między systemami. W ten sposób zrealizowano połączenie systemu UNICORE z systemami typu desktop grid i gridami społecznościowymi, opisane w pracy [49]. Rozwiązanie tam opisane ma tę zaletę, że tworzy się wspólny interfejs między różnymi rodzajami gridów usługowych a różnymi systemami typu desktop grid i gridami społecznościowymi. Rozwiązanie to ma jednak tę wadę co wcześniej wspomniane rozwiązania: konieczność modyfikacji interfejsu, gdy tylko zmienia się jeden z systemów. Innym problemem jest trudność w ustaleniu, jak ma przebiegać autoryzacja w połączonych systemach.

Powyższych problemów można uniknąć, budując jeden z systemów w oparciu o oprogramowanie warstwy pośredniej przeznaczonej dla drugiego systemu. Należy jednak ustalić czy podstawą do budowy takiego systemu powinno być oprogramowanie gridu usługowego, czy systemu typu desktop grid. Architektura systemów typu desktop grid jest na ogół bardzo prosta i nie umożliwia ona wielopoziomowego dostępu i organizacji zasobów. Systemy typu desktop grid nie oferują także możliwości sporządzenia skomplikowanej specyfikacji zasobów, która jest niezbędna w gridach usługowych. W związku z tymi cechami oprogramowania

dla systemów typu desktop jako naturalna pojawia się konieczność użycia oprogramowania warstwy pośredniej gridu usługowego w celu utworzenia systemu łączącego oba typy systemów gridowych. W gridach usługowych komputery były łączone w klastry i dopiero cały klaster był uważany za węzeł z punktu widzenia gridu. W trakcie wstępnych badań stwierdzono, że UNICORE w wersji 6, zużywa na tyle mało zasobów oraz jest na tyle efektywny, że potencjalnie może zostać użyty jako oprogramowanie warstwy pośredniej dla systemu typu desktop grid. Konieczne jest potwierdzenie tego w praktyce.

Istotną przesłanką, którą należy brać pod uwagę przy wyborze warstwy pośredniej dla systemu typu desktop grid, jest zapewnienie bezpieczeństwa polegającego na uwierzytelnianie i kontroli dostępu. Niezbędny jest mechanizm zapewniający tworzenie podzadań przez zarządcę oraz wysyłanie ich z komputera zarządcy. Konieczne jest istnienie mechanizmu, za pomocą którego węzeł otrzyma wiarygodną informację, że dane podzadanie pochodzi od użytkownika, który tworzył całe zlecenie. Mechanizm, który to umożliwia, nazywa się delegacją zaufania.

Do delegacji zaufania stosowane są dwa mechanizmy: certyfikaty proxy i delegacja zaufania wprost. Certyfikaty proxy umożliwiają certyfikowanej stronie wykonywanie dowolnej akcji w imieniu podpisującego. Delegacja zaufania wprost polega na wystawieniu podpisanej asercji informującej, że posiadacz określonego certyfikatu może wykonać konkretną akcję w imieniu wystawcy asercji[1]. W systemach typu desktop grid może być konieczne upoważnianie węzłów do dostępu do danych użytkownika. Przekazanie w tym celu certyfikatu proxy użytkownika, choć zapewnia tę funkcjonalność, umożliwia właścicielowi węzła wykonywanie także innych czynności związanych z prawami wystawcy certyfikatu. Jest to niedopuszczalne, gdyż właściciel węzła w systemie typu desktop grid może być niezaufany. W przypadku delegacji zaufania jedynym uprawnieniem, które uzyska właściciel węzła, będzie dostęp do tych danych użytkownika, które w trakcie obliczenia są transferowane na węzeł, a zatem do danych, do których i tak ma dostęp.

Delegacja zaufania wprost jest znacznie lepszym rozwiązaniem dla systemów typu desktop grid. Zapewnia obu stronom komunikacji zdecydowanie wyższy poziom bezpieczeństwa.

Wszystkie opisane cechy są motywacją do sprawdzenia w praktyce, czy UNICORE w wersji 6 nadaje się na oprogramowanie warstwy pośredniej systemu typu desktop grid. Stworzono system UNICORE Desktop Grid (UDG) – oprogramowanie dla systemu typu desktop grid oparte o UNICORE 6. W rozdziale niniejszym została opisana budowa tego oprogramowania, natomiast w rozdziale IX opisane są proste testy wydajnościowe, przeprowadzone przy użyciu stworzonego systemu.

[1] W części 3 rozdziału III znajduje się dokładniejszy opis obu mechanizmów.

2. Architektura systemu UNICORE 6

Architektura systemu UNICORE oparta jest na modelu warstwowym. Wyróżniona została warstwa oprogramowania klienckiego, warstwa pośrednia inaczej nazywana warstwą serwisów UNICORE oraz warstwa systemów docelowych. Maszyny docelowe są na ogół systemami komputerowymi dużej mocy takimi, jak klastry czy superkomputery. Posiadają one własne systemy kolejkowe i specjalizowane wersje oprogramowania.

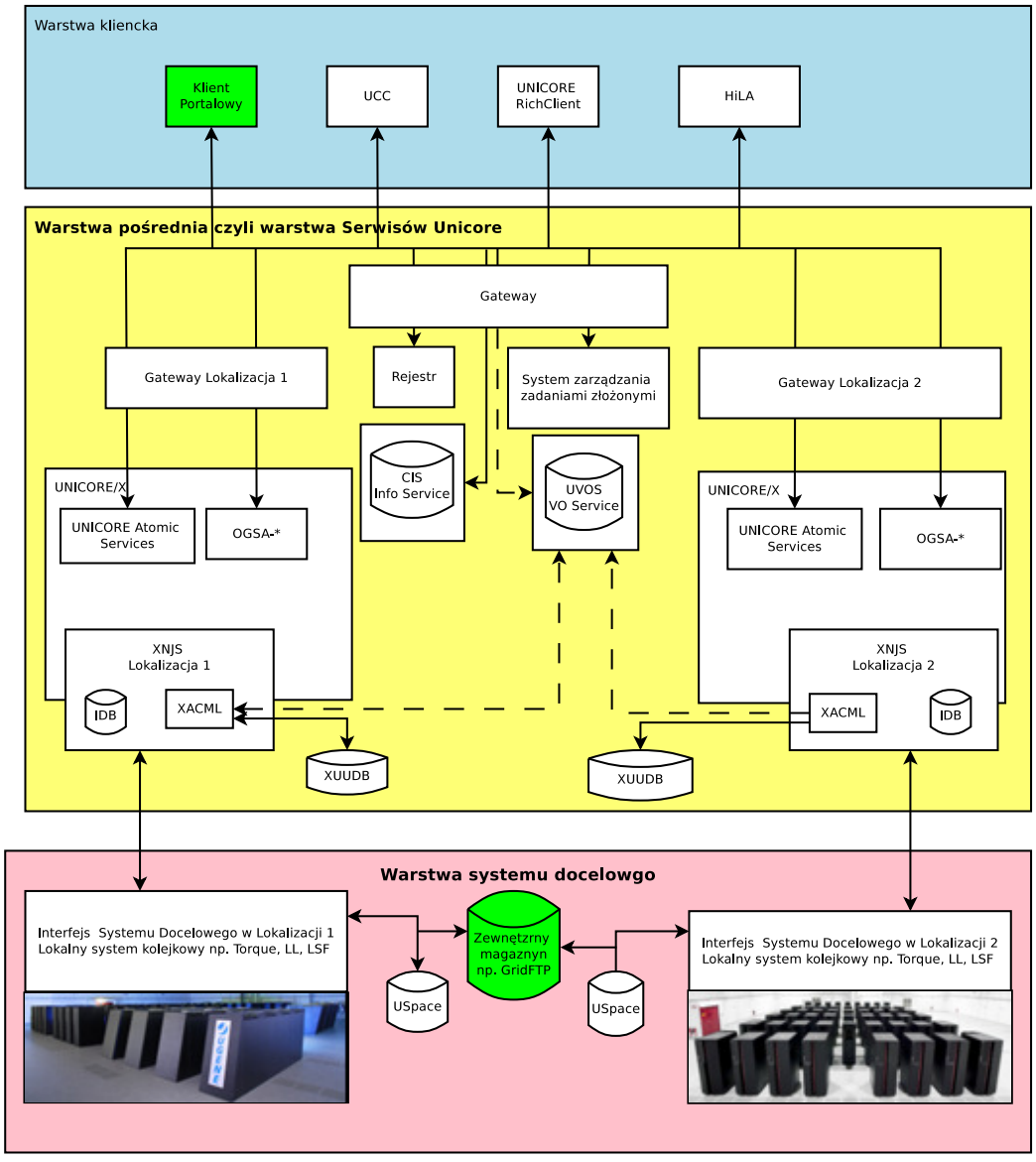
Obliczenia są tworzone przy użyciu oprogramowania klienckiego, następnie przekazywane do warstwy pośredniej, która umieszcza je na maszynie docelowej. Po wykonaniu obliczeń wyniki poprzez warstwę pośrednią są przekazywane z warstwy systemu docelowego do oprogramowania klienckiego. Architektura systemu UNICORE 6 jest przedstawiona na rysunku 1.

Warstwa kliencka umożliwia użytkownikom wygodne korzystania z systemu. Jest realizowana na różne sposoby:

- klienta portalowego,
- zaawansowanego klienta graficznego – RichClient,
- klienta tekstowego – UCC (UNICORE Comandline Client),
- biblioteki dostarczającej interfejs programistyczny – HiLA,

Warstwa pośrednia, czyli warstwa serwisów, zajmuje się komunikacją oraz bezpieczeństwem. Odpowiada ona za uniezależnienie opisu zadania od architektury, na której to zadania ma być wykonane. Warstwa serwisów umożliwia dzięki temu współdziałanie różnych systemów. Składa się ona z następujących modułów:

- Gateway,
- UNICORE/X,
- XUUDB – bazy użytkowników i kluczy,
- Rejestr,
- System zarządzania złożonymi zadaniami,
- IDB – baza tworzonych zadań,
- UVOS – system zarządzania wirtualnymi organizacjami,
- CIS – system informacji o stanie gridu.



Rysunek 1: Architektura systemu UNICORE (na podstawie [1])

Ostatnią warstwą jest warstwa systemu docelowego, czyli miejsce styku systemu UNICORE z systemami zarządzania zasobami takimi, jak systemy plików i systemy kolejkowe. W skład jej wchodzą:

- TSI – interfejs systemu docelowego,

Przykładowe uruchomienie zadania rozpoczyna się od stworzenia zadania przy użyciu zaawansowanego klienta. Następnie dzięki systemowi informacji o zasobach oraz informacji skladowanej w rejestrze, użytkownik wybiera, gdzie chciałby przeprowadzić obliczenie. Potem przy użyciu UNICORE/X, jest ono tłumaczone z postaci abstrakcyjnej na zlecenie dla systemu docelowego. W tym czasie sprawdzane jest, czy użytkownik ma prawo zlecić wykonanie tego zadania na konkretnym systemie docelowym. Tworzony jest także identyfikator zadania tak, aby możliwe było sprawdzenie jego stanu i odebranie wyników. W końcu zadanie zostaje zlecone do wykonania w systemie docelowym przy użyciu modułu TSI.

3. Opis poszczególnych modułów systemu UNICORE

3.1 Oprogramowanie klienckie

Obecnie UNICORE dostarcza kilka różnych typów oprogramowania klienckiego:

- UNICORE Rich Client, który jest oparty na platformie Eclipse. Jest to oprogramowanie wspierające wszystkie funkcje systemu UNICORE i umożliwiające graficzny dostęp do tych funkcji;
- Interfejs konsolowy – UCC (*UNICORE command line client*). Umożliwia korzystanie z UNICORE'a na konsoli tekstowej i w skryptach;
- HiLA (*High Level API for Grid Applications*) – jest to interfejs programisty, umożliwiający łatwe zlecanie zadań gridowych z własnych aplikacji. Jest to szczególnie użyteczne przy tworzeniu własnego oprogramowania, które umożliwia przygotowanie danych wejściowych dla zadań gridowych i późniejszą analizę danych;
- Klient portalowy – umożliwia on korzystanie z systemu UNICORE za pomocą portalu. Jest to bardzo wygodne do sprawdzenia w jakim stanie znajdują się zlecone obliczenia. Pozwala także na ich poprawienie w sytuacji, gdy użytkownik nie może w danej chwili zainstalować zaawansowanego klienta UNICORE.

3.2 Warstwa serwisów

W warstwie serwisów są następujące komponenty: gateway, XUUDB, UVOS, UNICORE/X, rejestr, system zarządzania zadaniami złożonymi i CIS.

Gateway jest elementem każdej lokalizacji (site) systemu gridowego, który musi być dostępny z sieci zewnętrznej. Sprawdza on tożsamość osoby łączącej się, zapewniając bezpieczeństwo pozostałym modułom. Sprawdzanie tożsamości odbywa się na podstawie certyfikatów X.509. Cały ruch sieciowy danej lokalizacji przechodzi przez gateway. Dzięki temu pozostałe moduły w danej lokalizacji mogą znajdować się na komputerach nie mających połączenia z siecią Internet, a tylko z komputerem na którym znajduje się gateway. Gateway umożliwia udostępnianie całej lokalizacji pod jednym adresem IP.

XUUDB jest to baza danych zawierająca informacje o użytkownikach i ich prawach dostępu. Dokonuje mapowania certyfikatów na identyfikatory użytkowników i role jakie mogą oni pełnić w systemie. Możliwe jest wykorzystanie jednej bazy dla więcej niż jednej lokalizacji.

UVOS jest systemem informacji o wirtualnych organizacjach. Stanowi drugie źródło uzyskiwania informacji o uprawnieniach użytkownika poza XUUDB. Używa on standardu SAML do przechowywania i przesyłania informacji o uprawnieniach.

UNICORE/X jest to serwer będący głównym elementem każdej instalacji UNICORE'a. Umożliwia on komunikację przy użyciu użyciu protokołów z rodziny serwisów webowych. Wyróżnić można dwa rodzaje dostarczanych usług. Jeden z nich to usługi zgodne ze standardami z rodziny OGSA. Drugi to atomowe serwisy UNICORE'a, które umożliwiają zlecanie, monitorowanie zadań, sprawdzanie stanu systemów docelowych i inne czynności administracyjne. Atomowe serwisy UNICORE są własnym protokołem systemu UNICORE. Są one konieczne do korzystania z zadań administracyjnych specyficznych dla systemu UNICORE. UNICORE/X zawiera moduł XNJS, który zajmuje się sprawdzeniem uprawnień przy użyciu XUUDB i systemu UVOS, a następnie uruchomieniem zadań na systemie docelowym. XNJS wykorzystuje bazę IDB zawierającą mapowanie zadań używanych w serwisach webowych, na konkretne polecenia służące do komunikacji z TSI. XNJS stanowi interfejs do warstwy systemu docelowego. Istnieje możliwość uruchomiania zadań lokalnie na serwerze, na którym działa UNICORE/X. Ta możliwość została wykorzystana w trakcie budowy systemu UDG.

UNICORE umożliwia zunifikowany dostęp do usług (*serwisów*) świadczonych przez różne miejsca. Usługi mogą być włączane i wyłączane przez niezależnych administratorów. Powoduje to konieczność istnienia usług przechowujących informację o innych usługach. Składnikiem oprogramowania UNICORE, który świadczy taką funkcjonalność, jest rejestr. Usługi rejestrują się w rejestrze, gdy stają się

aktywne. Co pewien czas rejestr sprawdza dostępność serwisów i aktualizuje swój stan usuwając informację o niedostępnych usługach.

UNICORE posiada możliwość tworzenia zleceń złożonych, tj. takich w których dane wyjściowe jednego podzadania są danymi wejściowymi (lub ich częścią) dla drugiego podzadania. W tym celu wykorzystywany jest dwuwarstwowy system zarządzania zadaniami złożonymi. Składa się on z wysokopoziomowej części łączącej się z użytkownikiem i części niskopoziomowej odpowiedzialnej za wykonywanie konkretnych podzadań oraz monitorowanie gridu. Część niskopoziomowa zawiera w sobie politykę alokacji zasobów, która może być łatwo zmieniana.

CIS, czyli wspólny system informacji, zbiera informacje o stanie systemów docelowych. Przechowuje on zarówno informacje statyczne jak i dynamiczne. Informacja statyczna to np. rodzaj systemu i dostępne oprogramowanie. Informacja dynamiczna to np. obciążenie węzła i długość kolejki. Systemu tego nie należy mylić z rejestrem, w którym są przechowywane wyłącznie adresy usług wraz z ich typami. W CIS przechowywane są znacznie bogatsze opisy dostępnych usług.

3.3 Warstwa systemu docelowego

TSI, czyli interfejs systemu docelowego, zajmuje się wykonaniem poleceń przychodzących z XNJS w konkretnym systemie docelowym. TSI jest modułem niezmienionym od UNICORE w wersji 5, dzięki czemu migracja do wersji 6 nie stwarza problemów. Umożliwia używanie UNICORE 6 na wielu systemach kolejkowych takich, jak Torque, LoadLeveler, LSF, SLURM i OpenCCS.

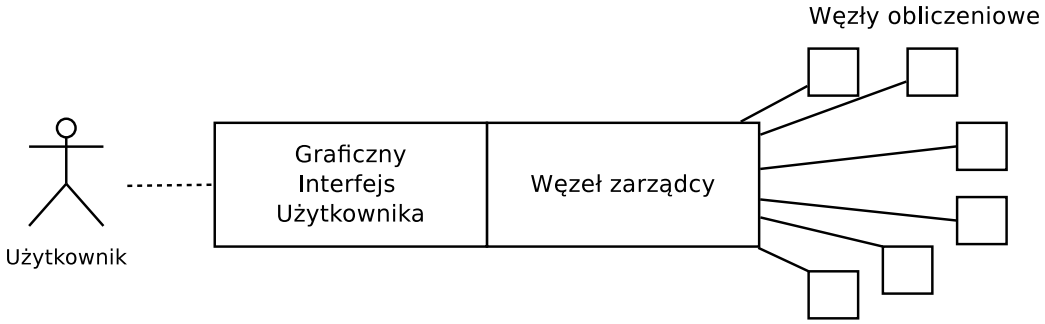
USpace jest to magazyn, w którym dla każdego zadania w systemie UNICORE jest tworzony odrębny katalog, w którym TSI i XNJS przechowują dane wejściowe dla zadania, dane wyjściowe, strumienie: wejścia, wyjścia i błędów. Zadania mogą w trakcie wykonania korzystać także z zewnętrznych magazynów poprzez różne protokoły np. GridFTP czy HTTP.

4. Architektura systemu UDG

4.1 Ogólny schemat systemu

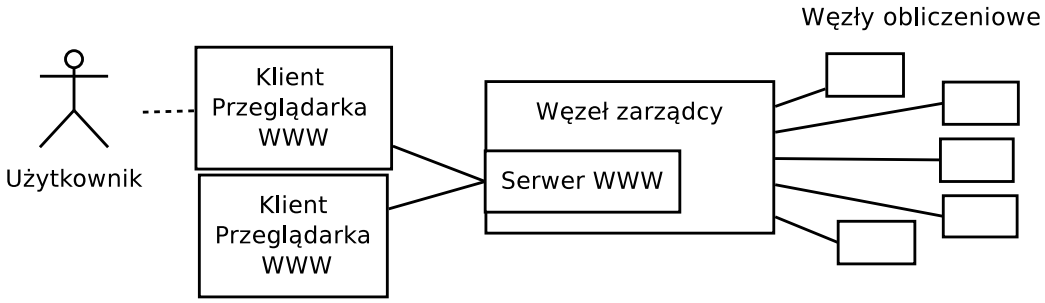
Komponenty systemu UNICORE zostały wykorzystane do zbudowania systemu UDG: *UNICORE Desktop Grid*. W stworzonym systemie typu desktop grid istnieją dwie klasy komputerów: węzły obliczeniowe i zarządcy. Węzły obliczeniowe to komputery osobiste wykonujące pracę w systemie typu desktop grid. Zadaniem zarządcy jest rozdzielanie pracy, złączanie wyników i monitorowanie

stanu węzłów. Dodatkowo zarządca zajmuje się przechowywaniem danych. Zarządca jest elementem, do którego zgłaszane są zlecenia do wykonania. W każdym systemie na jednego zarządcę przypada wiele węzłów obliczeniowych.



Rysunek 2: System w konfiguracji do testów. Graficzny interfejs użytkownika jest zintegrowany z oprogramowaniem zarządcy. Użytkownik za pomącą tego oprogramowania zleca wykonanie obliczeń, które są dzielone przez zarządcę i wysyłane do węzłów

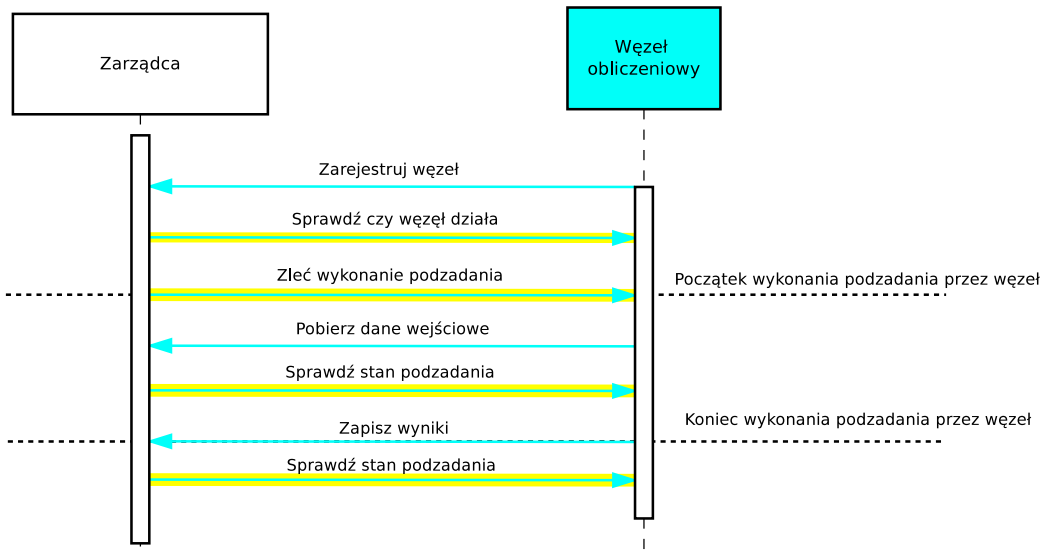
Pierwszy schemat, służący tylko do testów, przedstawiony na rysunku 2 opiera się na zintegrowaniu graficznego interfejsu użytkownika z zarządcą. Układ ten pozwala bardzo wygodnie mierzyć czas obliczeń. Łatwo jest zmieniać mierzone parametry i dodać funkcjonalność polegającą na wykonaniu i pomierzeniu czasu serii zleceń. Pozwala także uniknąć zakłóceń w pomiarach wydajności spowodowanych czasem działania klienta.



Rysunek 3: System w wersji WWW. Węzeł zarządcy zawiera serwer WWW. Za pośrednictwem interfejsu webowego klienci łączą się z serwerem i przygotowują zlecenia, które są przekazywane bezpośrednio zarządcy.

W drugim zaimplementowanym schemacie, przedstawionym na rysunku 3, klient za pomocą interfejsu WWW łączy się z zarządcą i wysyła zlecenie, które jest dzielone na podzadania i wykonywane na węzłach obliczeniowych.

4.2 Komunikacja pomiędzy zarządcą a węzłami



Rysunek 4: Diagram sekwencji dla komunikacji pomiędzy zarządcą a węzłem obliczeniowym.

Na rysunku 4 przedstawiono sekwencję wywołania usług koniecznych do prowadzenia obliczeń w systemie typu desktop grid. Kierunek strzałki oznacza stronę, która wywołuje usługę. Zarządca musi udostępniać następujące usługi: rejestracji węzła w systemie oraz możliwość korzystania z magazynu w celu pobierania danych wejściowych, a także zapisywania danych wyjściowych. Węzeł z kolei udostępnia zarządcy następujące funkcje: sprawdzenie stanu węzła i podzadań na nim wykonywanych oraz zlecanie wykonania podzadania. Nie ma konieczności pobierania wyników wykonanego podzadania, ponieważ to węzeł wysyła wyniki wykonanego podzadania do zarządcy natychmiast po jego zakończeniu.

Linie pogrubione żółtym kolorem oznaczają usługi, które powinien udostępniać węzeł. Węzły w systemie typu desktop grid są komputerami osobistymi. Bardzo często nie mają one możliwości przyjmowania połączeń przychodzących z zewnątrz, a wręcz często nie mają zewnętrznego adresu IP.

Komunikacja w systemie UNICORE jest realizowana poprzez protokoły opierające się na serwisach webowych. Aby wykorzystać możliwie dużo jego komponentów w systemie UDG, konieczne było przyjęcie takiej samej komunikacji. W celu uruchamiania funkcji węzła zarządca powinien mieć możliwość połączenia z węzłem. Ponieważ węzeł nie jest widoczny z sieci zewnętrznej, zarządca nie ma dostępu do węzła. Aby umożliwić połączenie od zarządcy do węzła konieczne było skorzystanie z proxy – pośrednika. Stworzono rozproszone proxy https. Składa się

ono z dwóch części: serwerowej oraz węzłowej. Część serwerowa musi znajdować się na komputerze, do którego dostęp mają wszystkie inne komputery w systemie. Z nią łączą się klienci https. Połączenia są przekazywane do docelowego serwisu przy użyciu odpowiedniej części węzłowej. Cześć węzłowa jest uruchamiana na węźle. Po uruchomieniu nawiązuje ona połączenie z częścią serwerową, podając informacje o adresie komputera, który obsługuje. Wewnątrz nawiązanego połączenia są tunelowane połączenia przychodzące do węzła. Dzięki zastosowaniu takiego rozwiązania węzły obliczeniowe nie muszą być dostępne z sieci zewnętrznej. Umożliwia to stworzenie gridu, w którym węzły korzystają z serwerowej części oprogramowania UNICORE (Gateway, UNICORE/X).

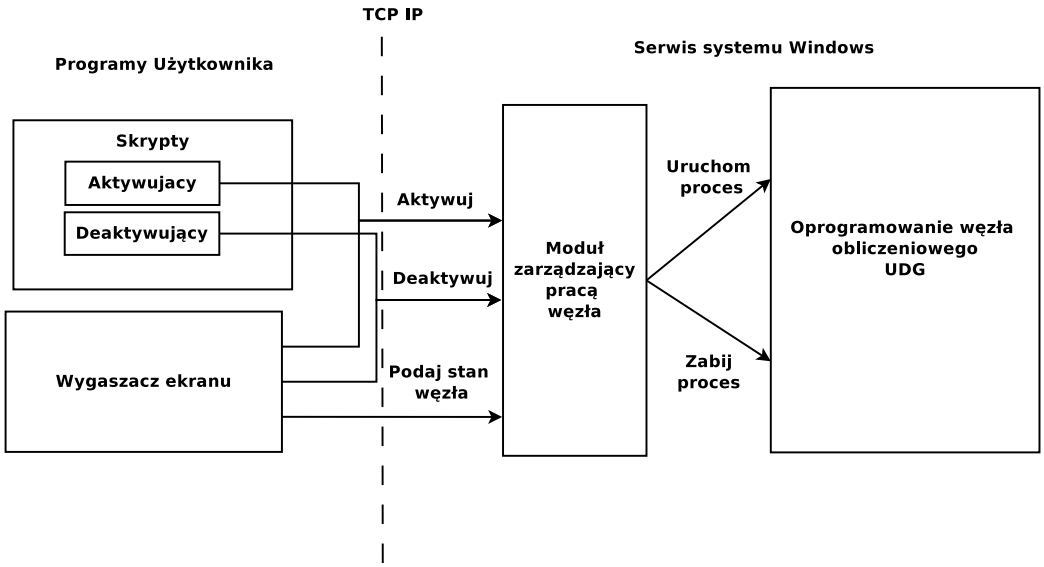
4.3 Bezpieczeństwo systemu UDG

Bezpieczeństwo w systemie UDG jest oparte na systemie bezpieczeństwa UNICORE i wykorzystuje certyfikaty X509. Certyfikatami identyfikują się użytkownik, zarządca i węzły. Zdarza się, że więcej niż jeden węzeł identyfikuje się tym samym certyfikatem. Ponieważ zadanie w trakcie wykonania nie przechowuje danych na węźle, musi ono także korzystać z certyfikatu do pobierania danych i zapisania wyników. Dla zwiększenia bezpieczeństwa każde zadanie otrzymuje oddzielny klucz, umożliwiający wykonanie wyłącznie tych czynności. Dzięki temu bardzo trudne jest celowe niszczenie lub zaburzanie wyników więcej niż jednego zadania. Zastosowanie bezpośredniej delegacji zaufania umożliwia przekazanie węzłom wiarygodnej informacji o zlecającym obliczenie. Umożliwia to węzłom przypisanie wyższych priorytetów zadaniom pochodzącym z ich grupy roboczej – wirtualnej organizacji.

Możliwość identyfikacji węzłów pozwala na zastrzeżenie miejsca, gdzie zlecenie ma być wykonane, np. tylko w ramach instytucji, do której należy zlecający. Wymaga to jednak zapewnienia, że klucze tych węzłów nie zostaną upublicznione.

4.4 Włączanie i wyłączenie węzła

Węzeł obliczeniowy systemu typu desktop grid powinien umieć wykorzystywać czas komputera niezajęty przez użytkownika. W tym celu konieczny jest mechanizm automatycznie włączający i wyłączający moduły węzła związane z prowadzeniem obliczeń. Aby to umożliwić wprowadzono mechanizm podobny do stosowanego w oprogramowaniu BOINC [7]. Mechanizm ten, przedstawiony na rysunku 5, wykorzystuje wygaszacz ekranu, aby uzyskać informację o braku aktywności użytkownika. Wygaszacz ekranu działa z uprawnieniami użytkownika komputera. W celu zapewniania bezpieczeństwa komputera i ograniczenia uprawnień węzła, węzeł powinien być uruchamiany z uprawnieniami dedykowanego użytkownika. Konieczne jest umożliwienie wygaszaczowi ekranu uruchomienia



Rysunek 5: Schemat przedstawiający architekturę węzła UDG w systemie Windows. Zaznaczone są mechanizmy służące do włączania i wyłączanie węzła

węzła z prawami innego użytkownika niż jego właściciel. Do wykonania tej czynności konieczne są uprawnienia super użytkownika, których wygaszacz ekranu nie powinien mieć. Przedstawiony mechanizm rozwiązuje ten problem inaczej. Moduł zarządzający pracą węzła obliczeniowego działa cały czas. Po uruchomieniu czeka na połączenie TCP, z którego może otrzymać komendy: aktywuj i deaktywuj węzeł, a także prośbę o podanie informacji o stanie węzła. Moduł ten działa z uprawnieniami dedykowanego użytkownika, dzięki czemu oprogramowanie węzła przez niego uruchomione też będzie działać z tymi uprawnieniami. Możliwa jest także komunikacja z tym modułem przy użyciu dedykowanych skryptów.

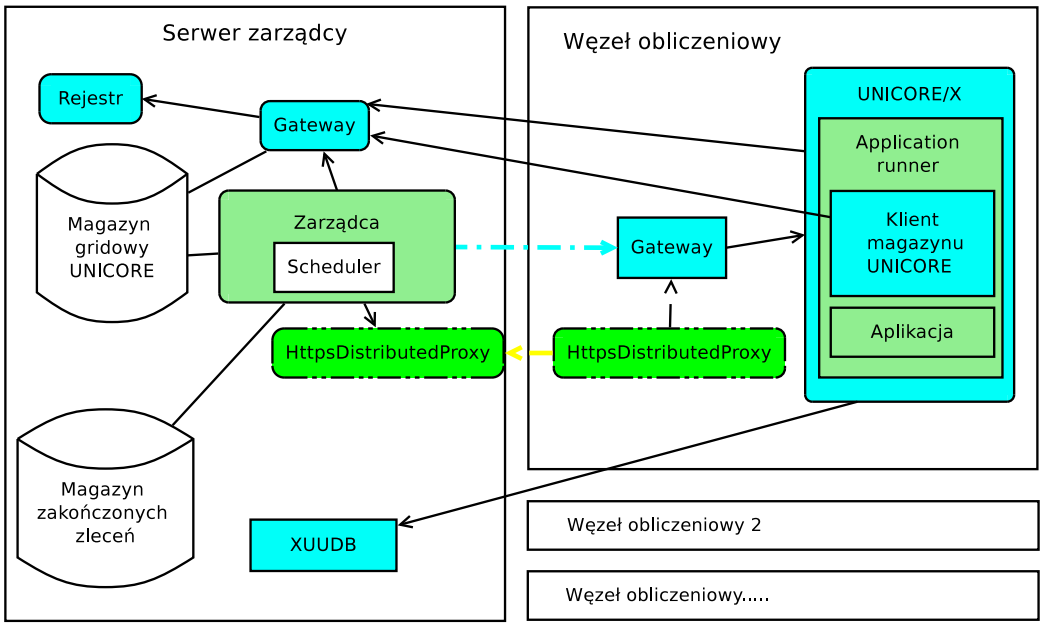
4.5 Szczegółowa architektura zarządcy i węzłów systemu UDG

Oprogramowanie gridowe na węźle obliczeniowym składa się z następujących części:

- Gateway – w niezmienionej postaci pochodzący z UNICORE;
- UNICORE/X jest to moduł odpowiedzialny za przyjęcie zlecenia i wykonanie zadania. Zadaniem jest ApplicationRunner z odpowiednimi parametrami. W tym celu wykorzystuje XNJS do uruchamiania lokalnych zadań. Moduł ten także odpowiada za obsługę zleceń monitorujących;

- ApplicationRunner – jest to zadanie uruchamiane przez UNICORE/X. Składa się ono z dwóch elementów: klienta oprogramowania UNICORE, umożliwiającego dostęp do magazynów danych oraz właściwą aplikację wykonującą obliczenie. Aplikacja zawiera klasę javy realizującą prosty interfejs przyjmujący na wejściu ciąg bajtów i zwracający również ciąg bajtów. Tego rodzaju interfejs umożliwia opracowanie bardzo restrykcyjnej polityki dla zarządcy bezpieczeństwa maszyny wirtualnej Javy. Polityka ta zabraniałaby aplikacji korzystania z dysku i połączeń sieciowych. Umożliwia to uruchamianie na węzłach obliczeniowych dowolnych aplikacji, zapewniając jednocześnie pełne bezpieczeństwo komputera. Aplikacja taka może być wysyłana wraz ze zleceniem lub pobierana z sieci, co rozwiązuje problem instalacji nowych aplikacji. ApplicationRunner przyjmuje cztery parametry wywołania: nazwę aplikacji[2], lokację pliku z danymi, lokację, pod którą powinny zostać zapisane wyniki, klucz prywatny umożliwiający dostęp do magazynów danych;

- rozproszonego proxy https – części węzłowej (HttpsDistributedProxy).



Rysunek 6: Schemat architektury systemu UDG, z zaznaczonym rozmieszczeniem modułów i połączeń pomiędzy nimi

[2]W przyszłości zamiast nazwy aplikacji będzie mógł być dostarczany adres, pod którym znajduje się kod wykonywalny (bytecode Javy).

Rozmieszczenie modułów w węźle obliczeniowym UDG jest przedstawione na rysunku 6. Głównym modułem w węźle jest UNICORE/X, wewnątrz którego umieszczony jest ApplicationRunner wykorzystujący klienta magazynu danych i uruchamiający aplikacje. Wszystkie zlecenia do UNICORE/X przechodzą przez Gateway. Aby umożliwić wykorzystanie węzłów niedostępnych z sieci zewnętrznej, połączenia są tunelowane przez proxy https.

Zarządca zawiera następujące elementy:

- magazyn zadań dla węzłów oraz wyników zwracanych przez węzły. Do tego celu został wykorzystany moduł USpace;
- magazyn zadań, które zostały zakończone – zadanie zakończone jest przenoszone z magazynu opisanego powyżej, aby utrudnić ewentualne próby zaburzenia pracy gridu;
- rejestr – wykorzystywany do składowania informacji o zarejestrowanych węzłach. Rejestr opisuje także magazyn danych dla węzłów;
- XUUDB – serwis autoryzacyjny realizowany przez bazę danych o certyfikatach i uprawnieniach z nimi związanych. Baza ta jest wspólna dla całego systemu UDG (korzystają z niej węzły);
- oprogramowanie zarządcy obejmujące system szeregowania i alokacji zasobów.

Rozmieszczenie modułów w zarządcy UDG jest przedstawione na rysunku 6. Polityka przydziału zadań jest wewnętrznym, wymiennym elementem oprogramowania zarządcy. Zarządca, by korzystać z rejestru, musi korzystać z pośrednictwa Gatewaya. Komunikacja zarządcy z węzłem odbywa się za pomocą rozproszonego proxy, którego serwerowa część również jest zainstalowana na serwerze zarządcy. W prostym systemie wszystkie węzły mogą korzystać ze wspólnej bazy XUUDB, także umieszczonej na serwerze zarządcy.

5. Porównanie systemu UDG z istniejącymi systemami typu desktop grid

System UNICORE Desktop Grid pokazuje, że możliwe jest opracowanie systemu typu dekstop grid na bazie oprogramowania UNICORE. System ten powstał na potrzeby sprawdzenia czy wydajność modułów systemu UNICORE jest wystarczająca do stworzenia systemu typu desktop grid oraz sprawdzenia wyników symulacji. Posiada on zatem uproszczony interfejs użytkownika. Można jednak wskazać na następujące zalety stworzonego systemu UDG:

- system ten jest łatwo integrowalny z gridem usługowym (gdyż posługuje się tym samym oprogramowaniem warstwy pośredniej),
- stworzony system posiada bardzo elastyczny i zaawansowany system bezpieczeństwa.

Możliwości rozbudowy przedstawionego systemu UDG są bardzo duże. Integralną część stanowi system bezpieczeństwa taki jak w systemie UNICORE. Umożliwia wykorzystanie go w środowisku składającym się z kilku instytucji, zapewniając bezpieczeństwo danych i komputerów, jednocześnie umożliwiając zlecanie wykonania części obliczeń do maszyn należących do ochotników.

Rozdział IX

Eksperymenty wydajnościowe

1. Cel eksperymentu

Podstawowym celem badań opisanych w tej pracy jest symulacja systemów typu desktop grid. Aby sprawdzić, w jakim stopniu przeprowadzone symulacje oddają rzeczywistość, przeprowadzono proste eksperymenty wydajnościowe przy użyciu opracowanego systemu typu desktop grid.

Ze względu na aspekt losowości nie jest możliwe dokładne przewidzenie czasu obliczeń systemu opartego na zawodnych komputerach połączonych siecią, której przepustowość podlega wahaniom. Z tego powodu badanie nie dotyczyło zgodności uzyskanych w trakcie symulacji czasów obliczeń z czasami obliczeń na rzeczywistym systemie, lecz zgodności trendów opisujących zachowanie systemu, zaobserwowanych przy użyciu symulacji. Jedna z zasad, opisana w części 3 rozdziału VII brzmi:

Połączenie węzła centralnego z siecią może stanowić wąskie gardło systemów typu desktop grid, których celem jest prowadzenie dobrze zrównoleglonych obliczeń i zwracanie wyników w czasie kilkunastu lub kilkudziesięciu minut.

Pojawia się pytanie, czy właściwość ta jest specyficzna jedynie dla symulacji, czy może jest również prawdziwa dla rzeczywistego systemu.

Przy okazji eksperymentów obejmujących badanie praw rządzących opracowanym systemem niezbędne okazało się przeprowadzenie eksperymentów wydajnościowych.

1.1 Parametry wydajnościowe

W trakcie testów wyznaczono podstawowe parametry wydajnościowe, tj. sumaryczny czas wykonania danych obliczeń oraz przyspieszenie, a także efektywność (będącą parametrem pochodnym).

Przyspieszenie i efektywność W pracach [44, 30] zdefiniowano miary jakości systemu równoległego: przyspieszenie (*speedup*) i efektywność (*efficiency*). Przyspieszenie jest parametrem informującym, ile razy szybciej działa system złożony z n komputerów od systemu składającego się z jednego komputera. Parametr ten wyraża się wzorem:

$$S(n) = \frac{T(1)}{T(n)} \tag{IX.1}$$

gdzie n to liczba użytych węzłów lub procesorów, $S(n)$ to przyspieszenie, a T to czas wykonania obliczeń przy zadanej liczbie n. Można wyróżnić przyspieszenie względne i bezwzględne. Względne to takie, w którym do ustalenia $T(1)$ – czasu dla jednego węzła – bez równoległości, używamy tego samego oprogramowania, co dla pozostałych wartości n. Bezwzględne przyspieszenie wymaga zastąpienia $T(1)$ czasem wykonania obliczeń przy użyciu najszybszego znanego algorytmu sekwencyjnego. W dalszej części pracy, jako przyspieszenie przyjęto przyspieszenie względne, przy czym $T(1)$ to czas wykonania obliczeń w systemie jednowęzłowym, ale przy użyciu najszybszej możliwej sieci (w której opóźnienie sieciowe jest zaniedbywalnie małe).

Efektywność jest to parametr informujący, w jakim stopniu wykorzystano moc n komputerów zawartych w systemie, w porównaniu z mocą wykorzystaną w systemie opartym na jednym komputerze. Parametr ten można wyrazić poprzez przyspieszenie i liczbę węzłów w systemie:

$$E(n) = \frac{S(n)}{n} \tag{IX.2}$$

Zaletą tej miary jest to, że niezależnie od liczby węzłów wielkość określająca efektywność powinna oscylować w przedziale od zera do jednego. Sporadycznie może ona być stała. W przypadku, gdy jej wartość jest równa jeden, mamy do czynienia z idealnym wykorzystaniem mocy obliczeniowej.

2. Sieć jako wąskie gardło systemu typu desktop grid

2.1 Planowanie eksperymentu

W celu przeprowadzenia eksperymentu wykorzystane zastało oprogramowanie opisane w rozdziale VIII. System został wdrożony z zarządcą umieszczo-

nym na jednym z komputerów w ICM UW, natomiast węzły znajdowały się na komputerach klastra PL GRID. W celu przetestowania innych wariantów sieci, przeprowadzone zostały eksperymenty polegające na umieszczeniu zarządcy na jednym z węzłów klastra PL GRID. Testowano dwa warianty: z użyciem sieci Gigabit Ethernet i InfiniBand. Opóźnienia sieci były następujące: pomiędzy zarządcą w ICM a węzłami średni czas transmisji w obie strony wynosi około 20 ms dla pakietu wielkości 65kB, natomiast pomiędzy węzłami: 1,4 ms dla sieci Gigabit Ethernet oraz 0,3 ms dla sieci InfiniBand. Odpowiada to następującym prędkościom sieci: 8 MB/s przy zarządcy w ICM (wolne połączenie), 128 MB/s – Gigabit Ethernet i 500 MB/s InfiniBand.

Jako zadania testowego użyto obliczanie zbioru Mandelbrota. Jest to obliczanie wartości elementów tablicy, gdzie każdy z punktów oblicza się niezależnie od pozostałych. Zadanie to jest bardzo dobrze skalowalne.[1] Jako jednostkę wielkości zadania przyjęto 14 sekund pracy jednego rdzenia wybranego komputera. Zadania były zbierane w większe zestawy – obliczenia.

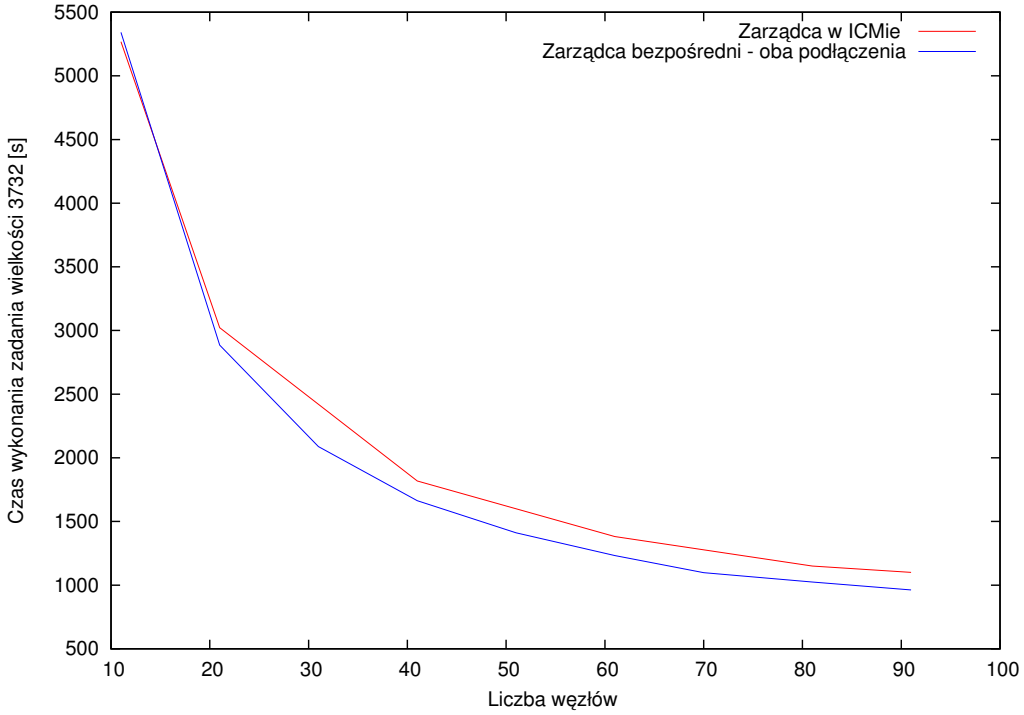
Test przeprowadzono dla dwóch obliczeń: jednego mającego sumaryczną wielkość 432 jednostek oraz drugiego – o wielkości 3732 jednostek. Każdy z nich składał się z rysowania 11 zbiorów Mandelbrota, z tym że w większym zadaniu pojedynczy obraz dzielono na 48, a w mniejszym – na 32 części. Większe obliczenie przy zarządcy znajdującym się na klastrze było testowane tylko przy użyciu sieci Gigabit Ethernet. Wydajność była wtedy na tyle bliska optymalnej, że zrezygnowano z testów przy użyciu sieci InfiniBand.

Wadą eksperymentu był brak symulacji wyłączeń komputerów. Pomimo to wyniki powinny być zgodne z przewidywaniami. Jest to możliwe ze względu na bardzo krótki czas testowanych zleceń (najdłuższe zadania trwały około 1200 s, czyli i tak znacznie krócej niż MTBF symulowanych węzłów).

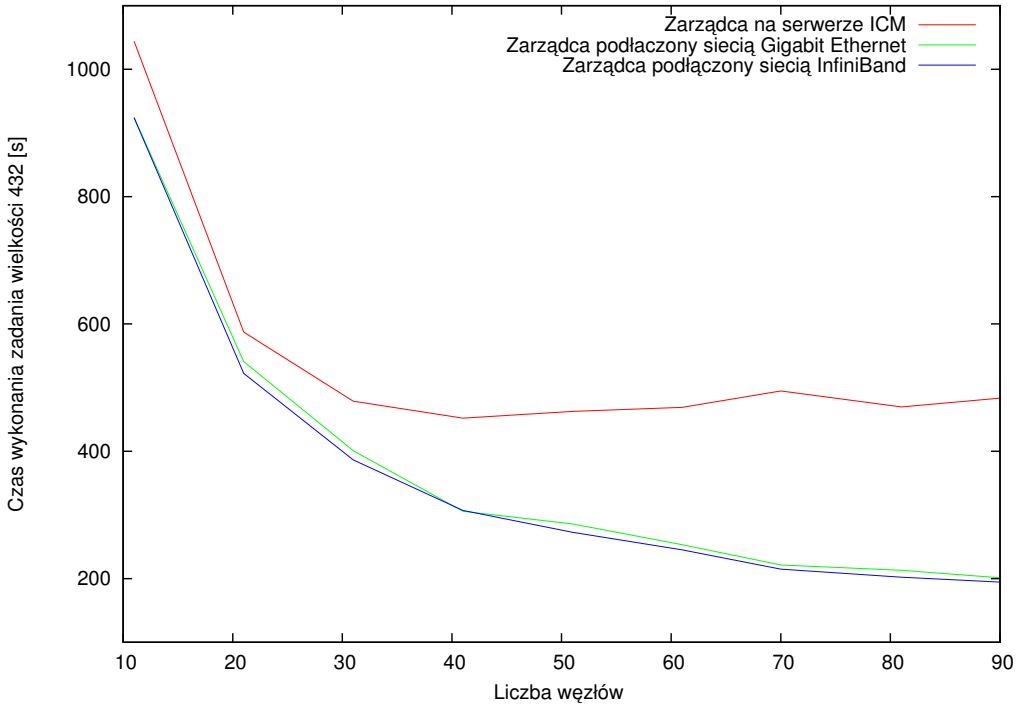
2.2 Wyniki

Czasy potrzebne do wykonania zadania o rozmiarze 3732 jednostek przedstawiono na wykresie 1. Dla tego rozmiaru zadania zależność czasu wykonania obliczenia od połączenia zarządcy jest praktycznie bez znaczenia. Wynika to z faktu, że czas poświęcany przez system na przesyłanie danych jest znikomy w porównaniu z czasem poświęcanym przez węzły na wykonanie obliczeń. Przyspieszenia zaczynają się różnić w zależności od połączenia zarządcy dopiero przy około 80 węzłach, co ilustruje wykres 3. Powyższe spowodowane jest wzrostem oddziaływania czasu przesyłania danych na całkowity czas wykonania obliczenia.

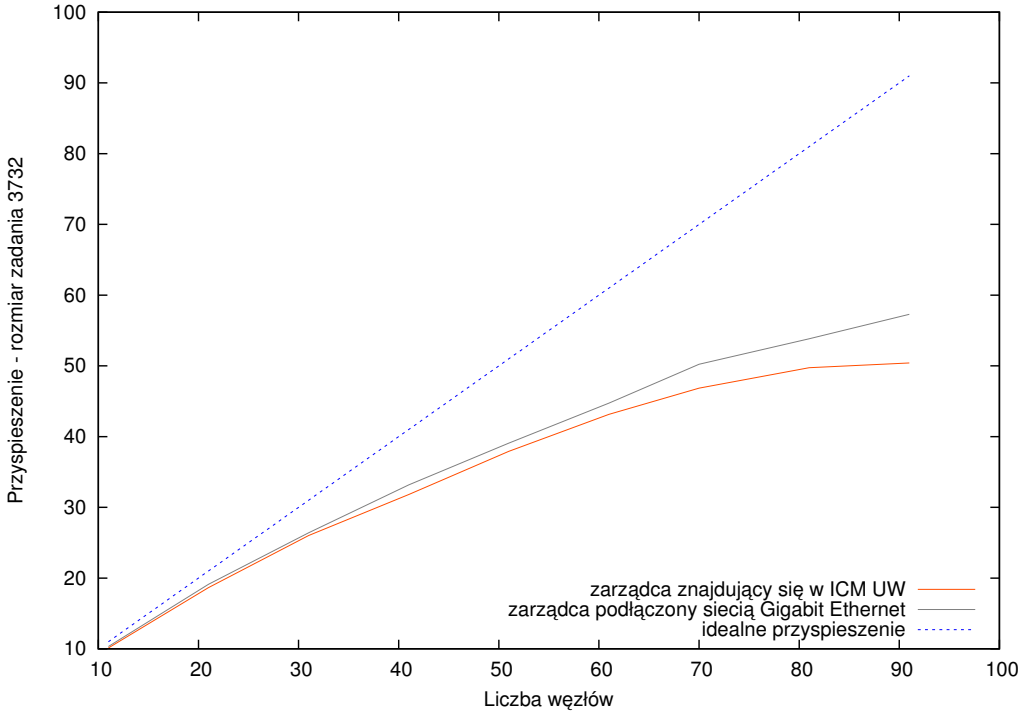
[1] Możliwe jest wystąpienie problemu z tzw. load balancingiem. Wartości funkcji dla niektórych elementów tablicy są liczone dłużej niż dla innych. Zdecydowano nie rozwiązywać tego problemu – w zadaniach rzeczywistych tego typu problemy również się pojawiają i najczęściej okazują się nierozwiązywalne.



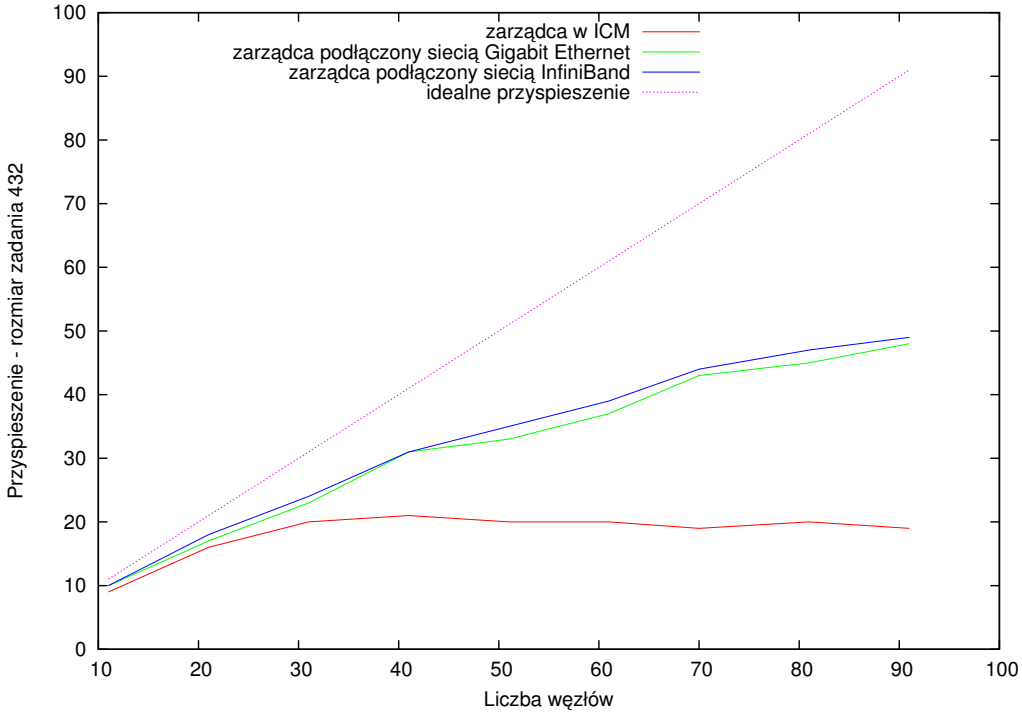
Wykres 1: Zależność pomiędzy czasem wykonania obliczeń dla wszystkich zadań a liczbą węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 3732 jednostek



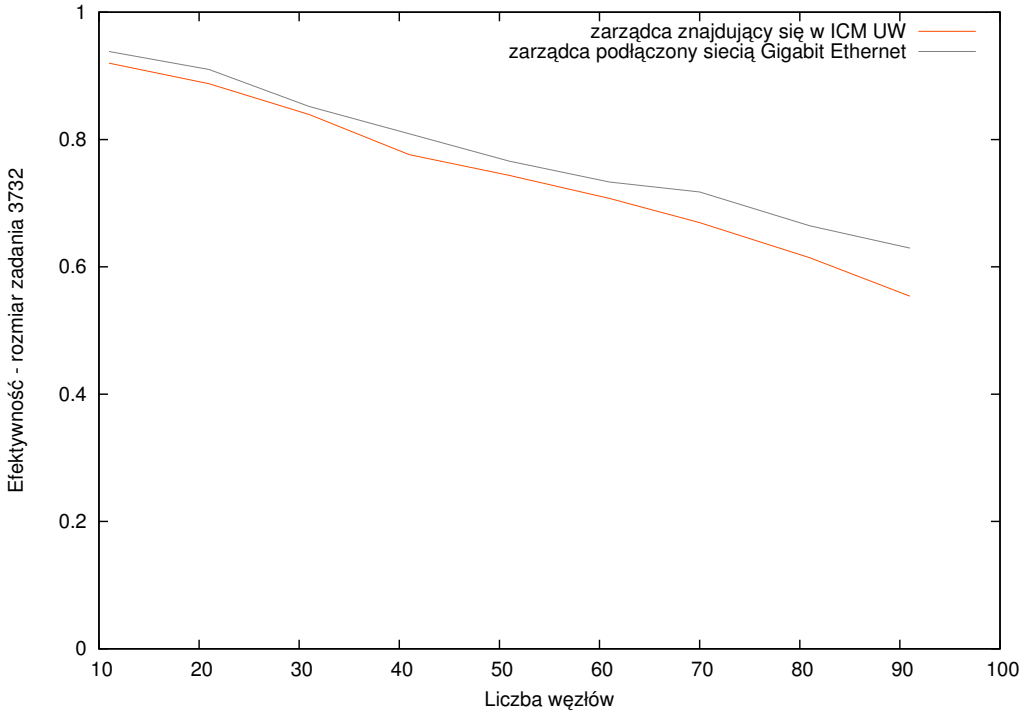
Wykres 2: Zależność pomiędzy czasem wykonania obliczeń dla wszystkich zadań a liczbą węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 432 jednostek



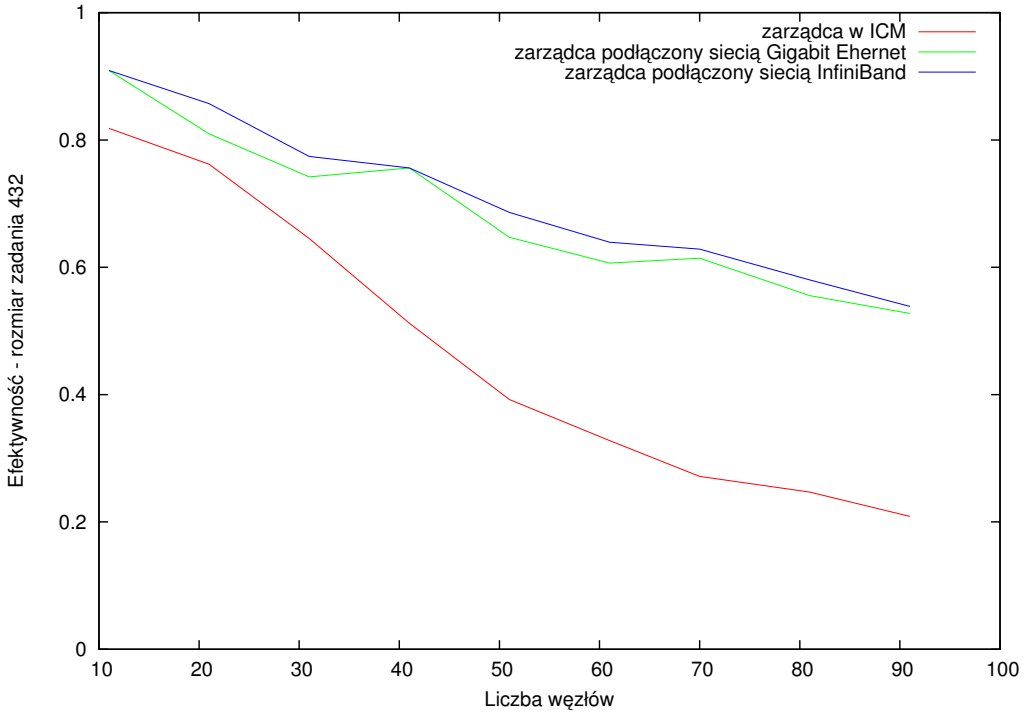
Wykres 3: Przyspieszenie obliczeń w zależności od liczby węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 3732 jednostek



Wykres 4: Przyspieszenie obliczeń w zależności od liczby węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 432 jednostek



Wykres 5: Efektywność systemu w zależności od liczby węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 3732 jednostek

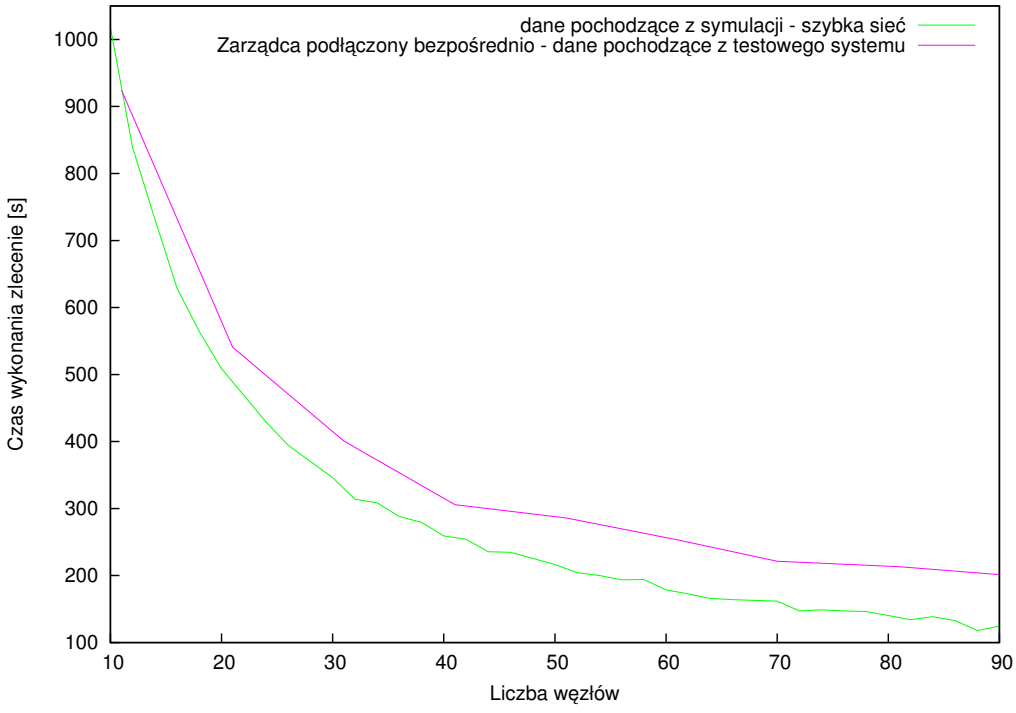


Wykres 6: Efektywność systemu w zależności od liczby węzłów użytych w trakcie obliczeń dla zadania o rozmiarze 432 jednostek

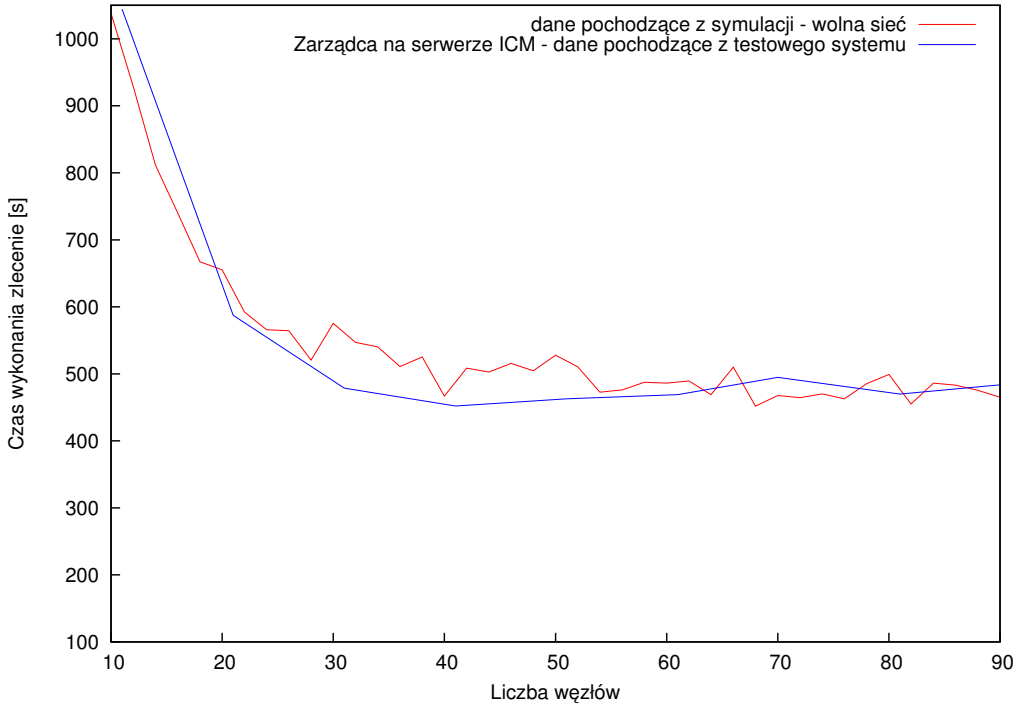
Różnica ta jest wyraźniejsza na wykresach 2 i 4. Zostały one sporządzone dla zadania o sumarycznym rozmiarze 423 jednostek. Dla małego zadania narzut na komunikację sieciową jest znaczny. Stąd duża różnica pomiędzy wykonaniem zadania w systemie z zarządcą połączonym bezpośrednio siecią InfiniBand lub Gigabit Ethernet, a wykonaniem analogicznego zadania z zarządcą połączonym wolniejszym łączem. Wynik jest zgodny z symulacjami opisanymi w rozdziale VII, przeprowadzonymi z uwzględnionym obciążeniem sieci (wykres 12 strona 73). Powyżej pewnej ilości węzłów, wraz z dalszym zwiększaniem ich liczby, nie następuje poprawa wydajności.

Różnice w wydajności są jeszcze wyraźniejsze na wykresie 6, przedstawiającym zależność efektywności od liczby węzłów. Efektywność systemu zależy od tego, czy sieć ma wąskie gardła – jak w przypadku zarządcy oddzielonego połączeniem ICM – PL-grid, czy nie. Zależność ta jest oczywiście znacznie mniejsza dla obliczenia składającego się z większych zadań – jak przedstawiono na wykresie 5.

W celu lepszego sprawdzenia jakości symulatora przeprowadzono obliczenia z parametrami odpowiadającymi systemowi eksperymentalnemu.



Wykres 7: Porównanie wyników czasu wykonania zlecenia w systemie testowym z wynikami symulacji. Porównanie dla przypadku zarządcy podłączonego łączem Gigabit Ethernet



Wykres 8: Porównanie czasu wykonania zlecenia w systemie testowym z wynikami symulacji

Testowano dwa przypadki – zarządcy połączonego połaczeniem Gigabit Ethernet i zarządcy w ICMie (wolne łącze). Testowano zadanie o rozmiarze 432 jednostki (mniejsze zadanie). Porównanie wyników symulacji z wynikami rzeczywistymi jest przedstawione na wykresach 7 i 8. Na wykresie 8 pojawiają większe wahania czasu wykonania zleceń dla symulacji niż dla rzeczywistego systemu. Jest to spowodowane za dużymi założonymi wahaniami przepustowości sieci.

Na wykresach 7 i 8 widać bardzo dobre przybliżenie systemu rzeczywistego przez symulację. Zmiana szybkości sieci, a w szczególności podłączenia zarządcy, ma zasadniczo przyspieszyć obliczenia. Wniosek ten, uzyskany w symulacjach opisanych w części 3 rozdziału VII, jest zgodny z zachowaniem rzeczywistego systemu i pokazuje faktyczną wartość modelu oraz symulatora. Oznacza to, że istotnie sieć w pewnych warunkach jest wąskim gardłem w systemie typu desktop grid.

Rozbieżności pomiędzy symulacjami a rzeczywistym systemem pojawiające się dla szybkiej sieci wskazują na to, że rzeczywisty system ma nieco większy narzut na komunikację niż przewidywano. Narzut ten wynika z konieczności gromadzenia informacji o stanie gridu np. sprawdzania które węzły są aktualnie aktywne.

3. Skalowalność systemu

We wstępie postawiono pytanie, na ile dobrze może działać system typu desktop grid oparty na oprogramowaniu warstwy pośredniej gridu usługowego. W trakcie wykonywania testów opisanych w części 2 uzyskane zostały dane umożliwiające częściową odpowiedź na to pytanie. Wykres 4 przedstawia przyspieszenie obliczeń w zależności od położenia zarządcy. Przyspieszenie to wynosi 10 dla 11 węzłów, co jest sytuacją prawie idealną. Wraz ze wzrostem liczby węzłów przyspieszenie oddala się od krzywej idealnej. Jest to spowodowane wpływem ostatnich podzadań na całkowity czas trwania obliczenia. Istotną część czasu system poświęca na wykonanie kilku ostatnich najdłuższych podzadań. W tym czasie obliczenia przeprowadza tylko kilka węzłów, zaś pozostałe są nieobciążone. Przedstawione wyniki pokazują, że zarządca jest w stanie pracować wydajnie, a wąskim gardłem staje się sieć, już przy jednoczesnym użyciu około 90 węzłów liczących. Przy średniej dostępności węzła na poziomie 33% system powinien zatem dobrze sobie radzić nawet z gridem o rozmiarze 270 węzłów – co jest już wielkością zadowalającą dla systemu typu desktop grid. Wyniki te uzasadniają stwierdzenie, że oprogramowanie UNICORE nadaje się do budowy systemu typu desktop grid.

4. Podsumowanie eksperymentów

W wyniku eksperymentów udało się stwierdzić, że symulator dobrze oddaje rzeczywistość. Badania pokazują, że powyżej pewnej liczby węzłów dalsze jej zwiększanie nie przyśpiesza obliczeń, co jest zgodne z wynikami symulacji. Wydajność sieci, tak jak w symulacjach, może mieć istotny wpływ na szybkość obliczeń, szczególnie przy małych zadaniach. Występują drobne rozbieżności między wynikami symulacji, a danymi rzeczywistymi. Są one związane z brakiem możliwości ustalenia dokładnych parametrów rzeczywistego systemu w trakcie przygotowania symulacji.

Stwierdzono, że wydajność stworzonego systemu jest dostateczna dla potrzeb systemów typu desktop grid. Parametry wydajnościowe stworzonego systemu są zgodne z oczekiwaniami, np. większe zadania skaluje się efektywniej niż mniejsze.

Rozdział X

Podsumowanie i wnioski

Całokształt przeprowadzonych rozważań badawczych, analiz, testów modelu oraz uzyskane wyniki upoważniają do przedstawienia, następujących wniosków:

1. Symulacje mogą być użyte do optymalizacji działania systemu typu desktop grid. Umożliwiają one nie tylko wskazanie elementu mającego największy wpływ na wydajność, ale również pozwalają stwierdzić jaka jego poprawa będzie wystarczająca dla poprawienia jakości pracy systemu. Pozwalają dzięki temu uniknąć niepotrzebnych kosztów. Przedstawiono w rozdziale VII przykład symulacji który pokazuje, że przyspieszenie podłączenia węzła centralnego do sieci może przyspieszyć działanie systemu zdecydowanie bardziej niż zwiększanie innych parametrów takich jak: ilości węzłów i zwiększanie szybkości podłączenia węzłów. Przyspieszenie podłączenia węzła centralnego może być porównywalne ze zdecydowanie droższym przyśpieszaniem każdego z węzłów z osobna.

2. W przedstawionych w pracy symulacjach systemów typu desktop grid uwzględniono wyłączenia węzłów. W połączeniu z uwzględnieniem limitowanej zmiennej wydajności sieci pozwala to odnajdywać zasady, którym podlegają systemy typu desktop grid. Na przykład:

   - w nieobciążonym systemie typu desktop grid, a także w systemach z bardzo szybką siecią zwielokrotnianie obliczenia jest bardzo dobrą polityką alokacji zasobów,
   - w przypadku przeciążania sieci – w szczególności pełnego obciążenia połączenia zarządcy, opóźnianie wykonania obliczeń i nie wysyłanie podzadań do gotowych węzłów może poprawić wydajność systemu.

3. Przedstawiony w pracy model sieci, oparty na szeregach czasowych dobrze oddaje rzeczywistość. Po obserwacjach, potwierdza założenie, że wydajność

sieci zależy od jej najwolniejszego elementu. Model ten pozwala bardzo skutecznie symulować obciążenie sieci bez konieczności tworzenia bardzo skomplikowanego modelu zawierającego pełną topologie sieci. Użyty zamiast tego model drzewiasty, gdzie każdemu połączeniu odpowiada szereg czasowy przepustowości daje bardzo dobre rezultaty.

4. Poważnym problemem istniejących systemów typu desktop grid jest brak zaawansowanych i elastycznych mechanizmów autoryzacji. Integracja systemu bezpieczeństwa istniejących systemów typu desktop grid z gridami usługowymi jest bardzo trudnym zadaniem. Problemy te można rozwiązać poprzez stworzenie systemu typu desktop grid na bazie oprogramowania dla gridów usługowych.

5. Brakuje obecnie możliwości łatwej integracji gridów usługowych z systemami typu desktop grid. Zaimplementowane rozwiązanie pokazuje możliwość dokonania takiej integracji w oparciu o oprogramowanie UNICORE. Stworzone rozwiązanie ma system bezpieczeństwa pochodzący z bazowego oprogramowania. Umożliwia to łatwą integrację z gridem usługowym opartym na tym oprogramowaniu.

6. W systemie typu desktop grid węzłami obliczeniowymi są komputery osobiste - jednostki o dość niskiej wydajności. Konieczna jest ich duża ilość, aby osiągnąć akceptowalną wydajność całego systemu. W gridach usługowych węzły mają dość dużą wydajność, a w przypadku klastrów złożonych ze słabszych komputerów, cały klaster jest widoczny jako jeden węzeł gridu. Powoduje to że większość znanych gridów usługowych nie była testowana w sytuacji gdy konieczna jest obsługa setek węzłów. Mogą się pojawić wtedy problme z wydajnością oprogramowanie do zarządzania gridami usługowymi.

7. Stworzony system typu desktop grid, oparty na oprogramowaniu UNICORE ma wydajność wystarczająca na potrzeby systemu typu desktop grid. Potwierdzają to testy wydajności opisane w rozdziale IX. Wskazują one, że system powinien sobie poradzić przy jednym zarządcy i ilości węzłów na poziomie 300. Jest to ilość wystarczająca dla systemów ograniczonego wielkością do kilku instytucji. Powyżej tej wielkości konieczne będzie zwiększenie ilości zarządców.

8. Wykazanie, że system stworzony na podstawie oprogramowania UNICORE nadaje się na oprogramowanie warstwy pośredniej dla systemów typu desktop grid potwierdziło wysoką wydajność sytemu UNICORE. Podkreśla to unikalność tego systemu i stanowi poważny argument za stosowaniem tego systemu do tworzenia dużych gridów usługowych.

9. Porównanie wyników symulacji z testowym wdrożeniem systemu w ICM wskazuje na to, że właściwości systemów typu desktop grid odnajdywane w trakcie symulacji, są spełnione także w rzeczywistym systemie. Przykładem właściwości która dotyczyła obu systemów było wydajność sieci, którą podłączony był węzeł centralny (zarządca) do sieci w której znajdowały się węzły obliczeniowe.

Podsumowując, udało się odpowiedzieć pozytywnie na trzy pytania zawarte we wstępie:

Czy istnieje możliwość stworzenia modelu systemu typu desktop grid o takich samych właściwościach jak system, w szczególności modelu systemu, w którym węzły ulegają awariom i przypadkowym wyłączeniom?

Czy system stworzony na bazie istniejącego oprogramowania warstwy pośredniej, używanego w gridach usługowych, może być wystarczająco wydajny do pracy w systemie typu desktop grid?

Czy istnieje możliwość stworzenia wydajnego systemu typu desktop grid z akceptowalnym poziomem bezpieczeństwa?

Stworzony na potrzeby pracy model systemu ma właściwości takie jak rzeczywisty system. Dobrze symuluje on awarie, szczególnie interesujące okazały się właściwości wynikające z ograniczonej, zmiennej przepustowości sieci i wyłączeń węzłów.

Opisany w pracy system UDG, będący systemem typu desktop grid bazuje na oprogramowaniu UNICORE. Badania pokazały, że jest on dostatecznie wydajny, co odpowiada pozytywnie na drugie pytanie.

System UDG ma ten sam poziom bezpieczeństwa co system UNICORE. Bezpieczeństwo w systemie UNICORE jest jednym z najlepiej rozwiązanych wśród systemów gridowych a jego poziom jest całkowicie akceptowalny. Zatem poziom bezpieczeństwa systemu UDG także jest całkowicie akceptowalny, co odpowiada pozytywnie na pytanie trzecie.

Konieczna jest dalsza praca nad rozwojem opisanego systemu mająca na celu stworzenie wygodnego interfejsu użytkownika, wykorzystanie pełni możliwości bezpieczeństwa systemu UNICORE. Umożliwienie pracy systemu z kilkoma serwerami zarządzającymi oraz wieloma magazynami danych powinno znacząco zwiększyć wydajność umożliwiając podłączanie większej ilości węzłów. Po wdrożeniu systemu do pracy z zadaniami zlecanymi przez naukowców warto przeprowadzić nowe symulacje mające na celu optymalizacje pracy systemu i dokładniej dopasować parametry modelu do rzeczywistych systemów.

Bibliografia

[1] Strona domowa projektu UNICORE. http://www.unicore.eu.

[2] World Wide Web consortium. http://www.w3c.org.

[3] Web Services Resource 1.2 (WS-Resource). Technical report, OASIS Standard, http://docs.oasisopen.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, 1 april 2006.

[4] Emmen A. AlmereGrid, the world's first city supercomputer, is taking shape. *EnterTheGrid - PrimeurMagazine*, 2003.

[5] N. Abdennadher and R. Boesch. Towards a Peer-To-Peer Platform for High Performance Computing. *International Conference on High Performance Computing and Grid in Asia Pacific Region*, pages 354–361, 2005.

[6] A. Acharya, G. Edjlali, and J. Saltz. The utility of exploiting idle workstations for parallel computation. *SIGMETRICS Perform. Eval. Rev.*, 25(1):225–234, 1997.

[7] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[8] D. P. Anderson. Volunteer computing: the ultimate cloud. *Crossroads*, 16(3):7–10, 2010.

[9] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[10] D. P. Anderson and J. McLeod. Local Scheduling for Volunteer Computing. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA*, pages 1–8. IEEE, 2007.

[11] A. Anjomshoaa, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification, Version 1.0. Technical report, http://www.gridforum.org/documents/GFD.56.pdf, November 2005.

[12] Z. Balaton, Z. Farkas, G. Gombas, P. K. Kacsuk, R. Lovas, A. C. Marosi, A. Emmen, G. Terstyanszky, T. Kiss, I. Kelley, I. Taylor, and F. Araujo. EDGeS, the common boundary between service and desktop grids. *Parallel Processing Letters*, 18(3):433–445, September 2008.

[13] J. Basney, R. Raman, and M. Livny. High Throughput Monte Carlo. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, TX, March 1999.

[14] S. Basu, S. Klivansky, and A. Mukherjee. Time Series Models for Internet Traffic. Technical report, Georgia Institute of Technology, Technical Report GIT-CC-95-27, 1996.

[15] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of Dynamic Grid Replication Strategies in OptorSim. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, pages 46–57, London, UK, 2002. Springer-Verlag.

[16] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 661, Washington, DC, USA, 2003. IEEE Computer Society.

[17] K. Benedyczak, P. Bala, S. van den Berghe, R. Menday, and B. Schuller. Key aspects of the UNICORE 6 security model. *Future Generation Computer Systems*, 27(2):195–201, 2011.

[18] G.E.P. Box and G. M. Jenkins. *Time series analysis: Forecasting and control*. Holden Day, San Francisco, 2nd edition, 1976.

[19] B. Calder, A. A. Chien, J. Wang, and D. Yang. The entropia virtual machine for desktop grids. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 186–196, New York, NY, USA, 2005. ACM.

[20] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid.

Future Generation Computer Systems, 21(3):417–437, 2005. P2P computing and interaction with grids.

[21] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *CCGRID*, First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia, pages 430–441. IEEE Computer Society, 2001.

[22] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, mar 2008.

[23] G. F. Cecile, G. Fedak, C. Germain, and V. Neri. XtremWeb : A Generic Global Computing System. In *In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'01*, pages 582–587, 2001.

[24] P. E. Ceruzzi. *A history of modern computing*. History of computing. MIT Press, 2003.

[25] C. Chen, K. Salem, and M. Livny. The DBC: Processing Scientific Data Over the Internet. In *16th International Conference on Distributed Computing Systems*, pages 673–682, Hong Kong, May 1996.

[26] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003. Special Issue on Computational Grids.

[27] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the Global Internet. *Computing in Science and Engg.*, 1(1):42–50, 1999.

[28] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6:86–93, 2002.

[29] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, D. Snelling, and S. Tuecke. The WS-Resource Framework. 2004.

[30] D. L. Eager, J. Zahorjan, and E. D. Lozowska. Speedup Versus Efficiency in Parallel Systems. *IEEE Trans. Comput.*, 38(3):408–423, 1989.

[31] P. Eerola, T. Ekelöf, M. Ellert, J. R. Hansen, A. Konstantinov, B. Kónya, J. L. Nielsen, F. Ould-Saada, O. Smirnova, and A. Wäänänen. The NorduGrid architecture and tools. *CoRR*, physics/0306002, 2003. informal publication.

[32] Y. El-khatib, C. (Ed.) Edwards, D. Damjanovic, W. Heiß, M. Welzl, B. Stiller, P. Gonçalves, P. Loiseau, P. V. Primet, L. Fan, J. Wu, Y. Yang, Y. Zhou, Y. Hu, L. Li, S. Li, S. Liu, X. Ma, M. Yang, L. Zhang, W. Kun, Z. Liu, Y. Chen, T. Liu, C. Zhang, and L. Zhang. Survey of Grid Simulators and a Network-level Analysis of Grid Applications. Technical report, University of Innsbruck, April 2008.

[33] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation Comp. Syst.*, 23(2):219–240, 2007.

[34] D. W. Erwin and D. F. Snelling. UNICORE: A Grid Computing Environment. In *Euro-Par 2001 Parallel Processing*, volume 2150 of *Lecture Notes in Computer Science*, pages 825–834. Springer Berlin / Heidelberg, 2001.

[35] T. Estrada, M. Taufer, K. Reed, and D. P. Anderson. EmBOINC: An emulator for performance analysis of BOINC projects. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[36] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pages 2–13, 2005.

[37] I. Foster and C. Kesselman(Ed.). *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[38] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002.

[39] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, CCS '98, pages 83–92, New York, NY, USA, 1998. ACM.

[40] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid - Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

[41] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Cluster Computing*, pages 237–246, 2001.

[42] N. K. Groschwitz and G. C. Polyzos. A Time Series Model of Long-Term NSFNET Backbone Traffic. In *In Proceedings of the IEEE International Conference on Communications (ICC'94*, pages 1400–4, 1994.

[43] R. Hempel. The MPI Standard for Message Passing. In W. Gentzsch and U. Harms(Ed.), *High–Performance Computing and Networking, InternationalConference and Exhibition, Proceedings, Volume II: Networking and Tools*, volume 797 of *lncs*, pages 247–252. sv, 1994.

[44] M. D. Hill. What is scalability? *SIGARCH Comput. Archit. News*, 18(4):18–21, 1990.

[45] Sciutto S. J. *AIRES A system for air shower simulations*, Lipiec 2002.

[46] J. Jurkiewicz, P. Bała, and Nowiński K. Prediction of the Jobs Execution on the Community Grid with added network latency. In P. Kacsuk, R. Lovas, and Z. Nemeth(Ed.), *Distributed and Parallel Systems In Focus: Desktop Grid Computing*. Springer, 2008.

[47] J. Jurkiewicz, P. Bała, and K. Nowiński. Numerical simulations of the resource utilization in the community grids. *Polish Journal of Enviromental Studies*, 17(3B):1230–1485, 2008.

[48] J. Jurkiewicz, K. Nowiński, and P. Bała. Prediction of the Jobs Execution on the Community Grid. In M. Bubak, M. Turala, and K. Wiatr(Ed.), *CGW'07 Proceedings*, Kraków, 2008.

[49] M. Keller, J. Kovacs, and Brinkmann A. Desktop Grids Opening up to UNICORE. In M. Romberg, P. Bala, R. Müller-Pfefferkorn, and D. Mallmann (Ed.), *UNICORE Summit 2011 Proceedings, 7–8 July 2011| Torun, Poland*, 2011.

[50] L Kleinrock. *Queueing Systems: Volume I - Theory*. Wiley Interscience, 1975.

[51] D. Kondo. *Scheduling task parallel applications for rapid turnaround on desktop grids*. PhD thesis, La Jolla, CA, USA, 2005.

[52] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@ Home and Genome@Home: Using Distributed Computing to Tackle Previously

Intractable Problems in Computational Biology. *Computational Genomics*, 2003.

[53] E. Laure, C. Gr, S. Fisher, A. Frohner, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, R. Byrom, L. Cornwall, M. Craig, A. Di Meglio, A. Djaoui, F. Giacomini, J. Hahkala, F. Hemmer, S. Hicks, A. Edlund, A. Maraschini, R. Middleton, M. Sgaravatto, M. Steenbakkers, J. Walk, and A. Wilson. Programming the Grid with gLite. In *Computational Methods in Science and Technology*, page 2006, 2006.

[54] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, Peter Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo. Middleware for the next generation Grid infrastructure. (EGEE-PUB-2004-002):4, 2004.

[55] O. Lodygensky, G. Fedak, V. Neri, M. Livny, and D. Thain. XtremWeb and Condor: Sharing Resources Between Internet Connected Condor Pool. In *In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'03) Workshop on Global Computing on Personal Devices*, 2003.

[56] A. C. Marosi, G. Gombás, and Z. Balaton. Secure application deployment in the Hierarchical Local Desktop Grid. In *Distributed and Parallel Systems*, pages 145–153. Springer US, 2007.

[57] A. C. Marosi, G. Gombás, Z. Balaton, P. Kacsuk, and T. Kiss. SZTAKI Desktop Grid: Building a scalable, secure platform for Desktop Grid Computing. In *Making Grids Work Proceedings of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments 12-13 June 2007*, Heraklion, Crete, Greece , 2007.

[58] M. W. Mutka. Estimating Capacity for Sharing in a Privately Owned Workstation Environment. *IEEE Trans. Softw. Eng.*, 18(4):319–328, 1992.

[59] S. Naqvi and M. Riguidel. Grid Security Services Simulator (G3S) - a simulation tool for the design and analysis of grid security solutions. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–428, july 2005.

[60] B. C. Neuman. Proxy-based authorization and accounting for distributed systems . In *Distributed Computing Systems, 1993., Proceedings the 13th International Conference on*, pages 283–291, may 1993.

[61] D. A. Patterson, D. E. Culler, and T. E. Anderson. A case for NOW (networks of workstation). In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, page 17, New York, NY, USA, 1995. ACM.

[62] B. Quetier and F. Cappello. A survey of Grid research tools: simulators, emulators and real life platforms.

[63] Buyya R. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, School of Computer Science and Software Engineering Monash University, Melbourne, Australia, April 2002.

[64] M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M. S. Memon, A. S. Memon, B. Demuth, and T. Lippert. Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. In *EDOCW '07: Proceedings of the 2007 Eleventh International IEEE EDOC Conference Workshop*, pages 57–60, Washington, DC, USA, 2007. IEEE Computer Society.

[65] G. F. Riley. The Georgia Tech Network Simulator. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12, New York, NY, USA, 2003. ACM.

[66] M. Romberg. The UNICORE Architecture: Seamless Access to Distributed Resources. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 44, Washington, DC, USA, 1999. IEEE Computer Society.

[67] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bała, E. del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt, and K. Baldridge. Chemomentum - UNICORE 6 Based Infrastructure for Complex Applications in Science and Technology. In L. Bougé, M. Forsell, J. Träff, A. Streit, W. Ziegler, M. Alexander, and S. Childs(Ed.), *Euro-Par 2007 Workshops: Parallel Processing*, volume 4854, chapter Lecture Notes in Computer Science, pages 82–93. Springer Berlin / Heidelberg, 2008.

[68] D. F. Snelling, S. van den Berghe, and V. Qian. Explicit trust delegation: Security for dynamic grids. *FUJITSU SCIENTIFIC & TECHNICAL JOURNAL*, 2(40):282–294, 2004.

[69] P. Stefan. The European KNOWARC project. In A. Cormack, editor, *Virtuality into Reality, The 21th Trans European Research and Education Networking*

Conference, June 8-11, 2009, Malaga, Spain, Selected Papers. TERENA, June 2009.

[70] A. Sulistio, G. Poduval, R. Buyya, and C. Tham. Constructing a Grid Simulation with Differentiated Network Service Using GridSim. In *In 6th International Conference on Internet Computing (ICOMP 2005), Las Vegas, NV*, 2005.

[71] A. Sulistio, C. S. Yeo, and R. Buyya. Simulation of Parallel and Distributed Systems: A Taxonomy and Survey of Tools.

[72] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency: Pract. Exper.*, 2(4):315–339, November 1990.

[73] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 97–104, 1999.

[74] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. J. Teller. SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 189–197, Washington, DC, USA, 2007. IEEE Computer Society.

[75] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[76] D. Thain, T. Tannenbaum, and M. Livny. How to Measure a Large Open Source Distributed System. *Concurrency and Computation: Practice and Experience*, 2006.

[77] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, J. Gawor, S. Meder, and F. Siebenlist. X.509 proxy certificates for dynamic delegation. In *In Proceedings of the 3rd Annual PKI R&D Workshop*, 2004.

[78] M. Wu, X. Sun, and H. Jin. Performance under failures of high-end computing. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–11, New York, NY, USA, 2007. ACM.

[79] J. W. Young. A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17(9):530–531, 1974.

[80] G. P. Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, 2003.