Raport projektu celowego
Obliczenia wielkiej skali i wizualizacja do zastosowań w wirtualnym laboratorium z użyciem klastra SGI

Zadanie WP 2.1. Zdalny dostęp do bibliotek naukowych

Przegląd cech i funkcjonalności systemów udostępniania bibliotek matematycznych, raport z testów i analizy kodów źródłowych,

raport z wykonania instalacji testowej

Maciej Brzeźniak

Poznańskie Centrum Superkomputerowo-Sieciowe

Poznań, 30 czerwca 2003 r.

Spis treści

Wstęp

Rozwój sieci komputerowych, szczególnie w zakresie ich przepustowości, daje rzeczywistą możliwość rozproszonego wykonywania obliczeń dużej skali. Szereg inicjatyw naukowych i komercyjnych prowadzi do budowy, korzystających z tych nowych możliwości, systemów obliczeniowych o dużej wydajności i przepustowości.

W badaniach naukowych zachodzi potrzeba prowadzenia obliczeń, które wymagają użycia zaawansowanych bibliotek matematycznych. Na komputerach dużej mocy pracujących w centrach naukowych w Polsce zainstalowane są wysoko-wydajne, zoptymalizowane dla poszczególnych maszyn wersje bibliotek obliczeniowych. Istnieje potrzeba stworzenia systemu udostępniającego te specjalizowane biblioteki naukowe szerokiemu gronu naukowców. Jedną z głównych cech takiego systemu miałaby być prostota dostępu oraz możliwość uzyskania rzeczywistego przyspieszenie działania wykorzystującej go aplikacji.

Na świecie prowadzonych jest kilka projektów, w toku których rozwijane są środowiska udostępniania rozproszonych zasobów obliczeniowych. Należą do nich m.in. *Ninf*, *NetSolve*, *RCS* oraz *NEOS*. W ramach projektu celowego *Obliczenia wielkiej skali i wizualizacja do zastosowań w wirtualnym laboratorium z użyciem klastra SGI* opracowane będzie rozwiązanie umożliwiające udostępnianie bibliotek matematycznych zainstalowanych na komputerach pracujących w polskich centrach obliczeniowych. Rozwiązanie bazować będzie na jednym z dwóch systemów *Ninf* lub *NetSolve*.

W raporcie zaprezentowane są wyniki analizy systemów *Ninf* i *NetSolve.* Opis funkcjonalności systemów zawarty jest w punkcie pierwszym. W punkcie drugim przedstawiono porównanie cech i funkcji systemów *Ninf* i *NetSolve*. W punkcie trzecim skomentowano instalację testową oraz przeprowadzone przez PCSS testy funkcjonalności systemów a także kwestię wyboru systemu, który posłuży jako podstawa do opracowywanego rozwiązania. W punkcie czwartym przedstawionon informacje o instalacji testowej.

1. Przegląd cech i funkcji systemów zdalnego wykonywania funkcji matematycznych

W tym punkcie raportu zostaną przedstawione ogólne informacje o projektach Ninf i NetSolve, zasadzie ich działania, a także omówione zostaną rozważane w dokumentach powstałych w ramach projektów problemy i zagadnienia techniczne. Z powodu złożoności tematyki, autor raportu uznał za zasadną szczegółową prezentację niektórych problemów technicznych i ich rozwiązań.

1.1. Specyfika systemów zdalnego wykonywania funkcji

Dla naświetlenia specyfiki systemów działających według schematu zdalnego wykonania funkcji bibliotecznych, przytoczona zostanie definicja środowiska przetwarzania funkcyjnego, zamieszczona w pracy: [1]: „Środowisko przetwarzania funkcyjnego to klasa projektów warstwy *middleware* środwiska *Grid*, zapewniającej niezbędne funkcje i narzędzia obsługujące większość nisko-poziomowych operacji w imieniu użytkownika. Interfejsem użytkownika do struktury *Grid*, jest w tych systemach wywołanie zdalnej funkcji (ang. *functional remote procedure call*). Warstwa *middleware* przechwytuje wołania procedur i traktuje je jako żądania obsługi. Argumenty procedury są upakowywane i wysyłane do zasobów *Grid*, które w danej chwili są w stanie najlepiej obsłużyć żądanie i, kiedy żądanie zostanie obsłużone, wyniki są przesyłane z powrotem do użytkownika, a wywoływana przez niego funkcja wykonuje powrót. Warstwa *middleware* jest odpowiedzialna za zarządzanie obsługą w środowisku *Grid*: wybór zasobów i ich przydzielenie, przenoszenie danych, operacje wejścia-wyjścia i odporność na uszkodzenia.”

Powyższa definicja określa pewne ogólne założenia dla środowisk przetwarzania funkcyjnego. Poniżej przedstawione zostaną wybrane informacje o strukturze i schemacie działania systemów zdalnego wykonywania obliczeń, które pozwolą lepiej umiejscowić omawiane w pracy magisterskiej rozwiązania w stosunku do innych, istniejących systemów sieciowego przetwarzania.

1.1.1. Jednostka pracy

System zdalnego wykonywania obliczeń jest podobny w swojej strukturze i ogólnym schemacie działania do systemu RPC (ang. *Remote Procedure Calls* – zdalne wywołania procedur) – w istocie jest jego rozwinięciem. Klient systemu wykonuje zdalne wywołanie

pojedynczej funkcji bibliotecznej (lub, w pewnych przypadkach grupy funkcji) na systemie obliczeniowym. Wołanie takie jest realizowane przez jeden z serwerów obliczeniowych przy użyciu jego lokalnych bibliotek programowych.

Dla porównania, np. w systemach uruchomieniowych dla zadań wsadowych i systemach kolejkowych (*LSF, NQE, Load Leveller* itp.) jednostką pracy jest nie wywołanie jednej funkcji lecz zadanie, rozumiane jako kod wykonywalny w postaci programu, który może składać się z szeregu wołań funkcji bibliotecznych, wywołań operacji wejścia-wyjścia itd. W systemie kolejkowym gotowy kod wykonywalny umieszczany jest w kolejce, a system, odpowiednio do zdefiniowanej polityki, przydziela mu zasoby (procesor, pamięć, zasoby przechowywania, urządzenia wejścia-wyjścia).

1.1.2. Zarządzanie zadaniami i zasobami

Dla pokazania specyfiki zarządzania zadaniami i zasobami w systemach zdalnego wywołania procedur znów podane zostanie odniesienie do systemów kolejkowych.

Rozdział zasobów odbywa się w systemach kolejkowych nie tylko statycznie, jednorazowo, przy uruchomieniu zadania, lecz jest dynamiczny, zachodząc również podczas działania funkcji-zadania. I tak, możliwe jest przenoszenie zadań pomiędzy systemami i węzłami systemu obliczeniowego, w celu uzyskania lepszej wydajności obliczeń lub optymalizacji wykorzystania sprzętu i wyrównania jego obciążenia (ang. *load balancing*). Jest to wykonalne dzięki technikom takim jak mechanizm punktów kontrolnych (ang. *checkpointing*), który umożliwia zapisywanie stanu zadań podczas ich działania, daje możliwość późniejszego restartowania ich od wybranego punktu na tej samej lub (w połączeniu z mechanizmami migracji) innej maszynie.

W systemach zdalnego wykonywania funkcji bibliotecznych przydział zasobów (i ewentualne wyrównywanie obciążenia systemów obliczeniowych lub minimalizacja czasu obliczeń) odbywa się „z góry”, jednorazowo. Wprawdzie, z uwagi na fakt, że nawet wywołania pojedynczych funkcji obliczeniowych mogą być bardzo długo wykonywane, rozwijane są mechanizmy służące do dynamicznego rozkładu obciążenia, wykonywania punktów kontrolnych i migracji zadań podczas ich wykonywania, jednak nie są one zbyt zaawansowane. Mechanizmy rozdziału zadań skupiają się więc na przewidywaniu dostępności zasobów w pewnym, przyszłym przedziale czasowym i stosownym do prognozy przydziale zasobów do zadań. Ważną rolę w systemach zdalnego wykonywania zajmują systemy prognozowania obciążenia zasobów.

1.2. System Ninf

Pierwszym omawianym systemem przetwarzania funkcyjnego będzie japoński system *Ninf* zaprojektowany i rozwijany w Instytucie Technologii w Tokio.

1.2.1. Koncepcja

Celem autorów systemu *Ninf* było dostarczenie platformy dla obliczeń naukowych prowadzonych przy użyciu zasobów rozproszonych po całym świecie.

Podstawową koncepcją systemu *Ninf* jest podejście oparte na architekturze klient-serwer. Zasoby obliczeniowe są dostępne jako *zdalne biblioteki Ninf* na zdalnych maszynach obliczeniowych. Zdalne biblioteki mogą być wołane poprzez globalną sieć z programu klienta napisanego w jednym z popularnych języków takich jak *Fortran*, *C* czy *C++*. Parametry, włączając tablice dużych rozmiarów są odpowiednio (efektywnie) kodowane, pakowane i wysyłane do *serwera Ninf* na zdalnej maszynie, który z kolei wykonuje żądane funkcje biblioteczne i odsyła klientowi wyniki. Protokół *Ninf RPC* (ang. *remote procedure call*) zapewnia interfejs, który jest bardzo przyjazny dla programistów operujących wyżej wspomnianymi językami. Programista może budować globalne systemy obliczeniowe używając zdalnych bibliotek systemu *Ninf* jako składowych, nie martwiąc się złożonością i kłopotliwością programowania sieciowego.

1.2.1.1. Główne cechy systemu Ninf

Poniżej zostaną omówione główne cechy systemu *Ninf*.

Klient może wykonywać najbardziej czasochłonną część swojego programu na wielu, zdalnych, wysoko-wydajnych komputerach, takich jak superkomputery wektorowe i komputery MPP (ang. *Massive Parralel Processor* – komputer masywnie równoległy), bez konieczności spełniania dodatkowych wymagań, co do specjalnego sprzętu czy systemu operacyjnego. Jeśli tylko takie komputery są osiągalne poprzez szybką sieć, aplikacja będzie działać znacznie szybciej. *Ninf* zapewnia więc jednolity dostęp do wielu, różnego rodzaju maszyn obliczeniowych.

Interfejs programistyczny *Ninf* jest tak zaprojektowany, by być bardzo łatwy w użyciu dla programistów popularnych, istniejących języków. Użytkownik może wywoływać zdalne funkcje biblioteczne bez szerokiej wiedzy o programowaniu sieciowym i dzięki temu łatwo modyfikować istniejące, lokalne aplikacje, które używają popularnych bibliotek obliczeniowych takich jak np. LAPACK.

Mechanizm RPC systemu *Ninf* może być asynchroniczny i automatyczny: *meta-serwer Ninf* utrzymuje informacje o serwerach *Ninf* znajdujących się w sieci i automatycznie rozdziela zdalne wołania funkcji pomiędzy serwery obliczeniowe, zgodnie z odpowiednią polityką szeregowania (ang. *scheduling*) i wyrównywania obciążeń (ang. *load balancing*).

Ninf zawiera również API (ang. *Application Programming Interface* – interfejs programistyczny dla aplikacji) umożliwiający zagregowane wykonanie wielu zdalnych obliczeń.

Sieciowy serwer bazy danych Ninf zapewnia możliwość przepytywania bazy danych o pewne zdefiniowane wartości, takie jak np. wartości stałych fizycznych i chemicznych. Wołania funkcji *Ninf* mogą przyjmować jako argumenty nie tylko stałe i zmienne programowe ale także adresy sieciowe pewnych wartości w postaci URL (ang. *Unified Resource Locatior* - zunifikowany lokalizator zasobu). Również wyniki mogą być składowane na stronach www (dzięki wykorzystaniu polecenia PUT protokołu HTTP).

System *Ninf* daje użytkownikowi dostęp do lokalnych zasobów programowych i sprzętowych zdalnych maszyn. Pomijając kwestię dostępności sprzętu o dużej mocy obliczeniowej, przyjęte w systemie mechanizmy i rozwiązania pozwalają na wykorzystanie zoptymalizowanych dla danej architektury a przez to bardzo wydajnych bibliotek matematycznych. Przy założeniu, że sieć łącząca klienta z urządzeniami obliczeniowymi jest odpowiednio szybka (w sensie dostępnego pasma i wnoszonego opóźnienia) prowadzenie obliczeń na zdalnych maszynach może się okazać dla klienta bardzo opłacalne. Kwestie opłacalności przenoszenia obliczeń na zdalne maszyny zostaną szczegółowo omówione poniżej.

1.2.2. Architektura systemu Ninf

W tym punkcie zaprezentowana zostanie architektura systemu *Ninf*. Opiera się ona koncepcyjnie na mechanizmie zdalnego wywołania procedur.

1.2.2.1. Wywołanie funkcji bibliotecznych

Podstawowy system *Ninf* oparty jest o model klient-serwer. Serwer i klient mogą być połączeni poprzez sieć lokalną lub przez sieć *Internet*. Maszyny mogą być heterogeniczne: dane są podczas komunikacji tłumaczone na wspólny, sieciowy format.

Proces serwera *Ninf* działa na maszynie obliczeniowej. Zdalne biblioteki systemu *Ninf* są zaimplementowane jako wykonywalne programy, które zawierają, jako główną funkcję, *stopkę* (ang. *stub*). Bibliotekami zarządza proces serwera. Kiedy program klienta wywołuje

funkcję biblioteczną, serwer *Ninf* szuka funkcji w postaci wykonywalnej *Ninf* (ang. *Ninf executable*), która jest skojarzona z nazwą z żądania klienta, wykonuje jej kod, zestawiając przedtem odpowiednie połączenie z klientem. Funkcja-stopka obsługuje komunikację serwera *Ninf* i jego klienta, m.in. upakowuje argumenty i wyniki.

1.2.2.2. Przygotowanie funkcji bibliotecznych i zasobów obliczeniowych dla systemu Ninf

Wykonywalny kod funkcji bibliotecznej może być napisany w dowolnym istniejącym języku (*Fortran*, *C*, itd.) a następnie skompilowany do postaci, w której może być uruchamiany na danej maszynie.

Na udostępnienie funkcji bibliotecznych i zasobów obliczeniowych dla systemu *Ninf* składa się kilka kroków:

a) przygotowanie niezbędnego opisu interfejsu każdej funkcji bibliotecznej w języku opisu interfejsu (*Ninf IDL* - ang. *Ninf Interface Description Language*);
b) kompilacja opisu w języku IDL, której produktem są pliki nagłówkowe (ang. *header files*) i kody stopki (ang. *stub code*);
c) kompilacja biblioteki kompilatorem języka, w którym napisane zostały funkcje biblioteczne;
d) połączenie (ang. *linking*) obiektów programowych z funkcjami *Ninf* RPC;
e) rejestracja funkcji bibliotecznych w serwerze *Ninf*.

Po tym krokach, programista może w sposób przeźroczysty, używać bibliotek poprzez *Ninf* RPC. Niektóre istniejące biblioteki, jak np. LAPACK, zostały już przystosowane dla systemu *Ninf* opisaną powyżej metodą.

1.2.2.3. Protokoły komunikacyjne

W bieżącej implementacji, komunikacja pomiędzy klientem i serwerem odbywa się poprzez wykorzystanie protokołu TCP/IP dla zapewnienia wiarygodnego i pewnego przesyłu danych pomiędzy procesami. W środowiskach heterogenicznych, *Ninf* używa jako domyślnego protokołu formaty danych zgodnego ze standardem *Sun XDR*.

1.2.2.4. Mechanizm wywołania zwrotnego

Możliwe jest zdefiniowanie przez klienta funkcji wywołania zwrotnego (ang. *call back functions*) inicjalizujących po stronie klienta pewne czynności, takie jak np. interaktywną wizualizację danych, operacje wejścia-wyjścia (dla danych) itd.

1.2.3. Interfejs programistyczny

Jedynym interfejsem klienta do serwerów i baz danych systemu *Ninf* jest funkcja `Ninf_call()`. Dla zilustrowania interfejsu programistycznego systemu rozpatrzmy prostą funkcję mnożenia macierzy w programie *C* z następującym interfejsem:

```
double A [N][N],B[N][N],C[N][N]; /* deklaracja tablicy */
....
dmmul(N,A,B,C);                  /* wywołuje mnożenie macierzy, C=A*B */
```

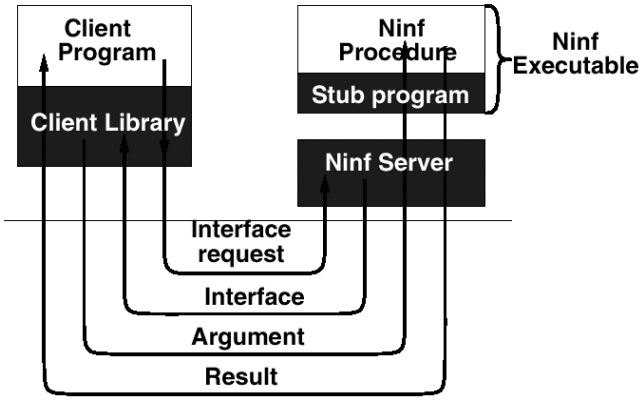
Kiedy funkcja `ddmul` jest dostępna na serwerze *Ninf*, program kliencki może wywołać funkcję zdalnej biblioteki używając funkcji `Ninf_call`. Odbywa się to w następujący sposób:

```
Ninf_call(„dmmul”,N,A,B,C);      /* wywołuje funkcję zdalnej biblioteki Ninf */
```

`ddmul` jest nazwą funkcji bibliotecznej zarejestrowanej jako funkcja wykonywalna na serwerze a A, B, C, N są argumentami funkcji. Jak tu widzimy, użytkownik systemu specyfikuje nazwę funkcji tak, jakby to było wywołanie funkcji lokalnej; funkcja `Ninf_call()` automatycznie określa stopień funkcji (tzn. wielkość danych wejściowych i wejściowych, np. rozmiar macierzy) i typ każdego z argumentów. Odpowiednio upakowuje i koduje (ang. *marshals*) argumenty, wykonuje zdalne wołanie do serwera, otrzymuje wyniki, umieszcza je w odpowiednich argumentach i zwraca je do klienta. W ten sposób, mechanizm *Ninf* RPC daje klientowi złudzenie, że argumenty są współdzielone pomiędzy klientem a serwerem (tak jakby były we współdzielonym segmencie pamięci). Zauważmy, że fizyczna lokalizacja serwera nie jest tu wyspecyfikowana. Serwer obliczeniowy jest przydzielany przez moduł szeregujący (ang. *scheduler*).

1.2.3.1. Szczegóły mechanizmu Ninf RPC

By ukazana powyżej prosta interfejsu programistycznego klienta była możliwa, wprowadzono do systemu *Ninf* RPC zasadę, że wszystkie informacje o interfejsie zdalnej funkcji bibliotecznej pozyskiwane są, przez funkcję klienta, od serwera *Ninf*, w momencie wywołania funkcji `Ninf_call` (ang. *at runtime*). Chociaż musi być, w związku z tym, poniesiony dodatkowy koszt czasowy, uznano, że typowe aplikacje naukowe są do tego stopnia złożone obliczeniowo i danowo (ang. *compute and data intensive*), że ów narzut nie ma większego znaczenia. Co więcej, faza pozyskiwania informacji o interfejsie podczas wołania zdalnej funkcji może zostać pominięta dzięki buforowaniu tej informacji (ang. *caching*) w programie klienta. Informacja o interfejsie zawiera liczbę parametrów, ich typ i rozmiary oraz tryb dostępu do nich (odczyt lub zapis). Używając tej informacji, *Ninf* RPC automatycznie dokonuje upakowania argumentów i generuje sekwencje operacji wysyłania i odbierania danych z i do serwera Ninf.



Rysunek 1.1. Zdalne wywołanie funkcji w systemie Ninf

Źródło: praca [1]

Jak pokazano na rysunku 1.1, wywołanie funkcji przez klienta powoduje wysłanie żądania udzielenia informacji o interfejsie funkcji do serwera *Ninf*. Klient, na podstawie uzyskanej w ten sposób od serwera informacji, interpretuje i upakowuje argumenty „leżące" na stosie. Dla argumentów w postaci tablic zmiennych rozmiarów, język definicji interfejsów (ang. IDL – *Interface Definition Language*) musi specyfikować wyrażenie, które zawiera wejściowy, skalarny argument, na podstawie którego może zostać obliczony rozmiar tablicy. To podejście różni się od tradycyjnego RPC, takiego jak np. RPC firmy *Sun* lub systemu CORBA, gdzie

generowanie stopki musi być wykonane po stronie klienta na etapie kompilacji (a nie w momencie uruchomienia). Dzięki automatycznemu pozyskiwaniu interfejsu, *Ninf* RPC nie wymaga w ogóle podobnych czynności na etapie kompilacji, zwalniając użytkowników z obowiązku „utrzymywania" kodu (ang. *code maintenance*), tak by był on dopasowany do zmieniającej się konfiguracji środowiska obliczeniowego.

1.2.3.2. Dostęp do serwerów www

Serwer obliczeniowy systemu *Ninf* ma możliwość odczytywania i zapisywania danych na stronach *www*. Umożliwia to używanie danych znajdujących się na stronach *www* jako argumentów funkcji bibliotecznych zarejestrowanych w systemie *Ninf*. Używając łańcuchów znaków zawierających URL jako danych, użytkownicy mogą używać danych, które są umieszczone w systemie *web* pod podanym adresem. Poniższy kod używa takich danych jako drugiego argumentu tablicowego:

```
Ninf_call("dmmul",N,A,"http://.../..",C)
```

Kod klienta *Ninf* automatycznie wykrywa, że argument określa URL i wysyła ten URL do serwera. Serwer *Ninf* odbiera URL i łączy się z serwerem *web*, pobiera wyspecyfikowane dane używają komendy GET protokołu HTTP, składuje je w pamięci i wywołuje funkcję obliczeniową.

Użytkownicy mogą również umieszczać wyniki na stronach *www*. Program korzystający z systemu *Ninf* może wczytywać dane na serwery *web*. Ta funkcja może być używana, np. dla przechowywania pośrednich wyników obliczeń na serwerze *www*.

1.2.3.3. Asynchroniczne wywołania funkcji

Zdalne wywołanie funkcji w systemie *Ninf* może również być wykonane asynchronicznie, dla wykorzystania możliwości prowadzenia współbieżnych obliczeń w sieci. Funkcja `Ninf_call_async` jest podobna do funkcji `Ninf_call`, z tą różnicą, że nie oczekuje na zakończenie zdalnych obliczeń, tylko natychmiast wykonuje powrót, zwracając identyfikator żądania (ang. *request ID*). Używając identyfikatora, programista może czekać lub cyklicznie odpytywać serwer *Ninf* o odpowiedź na żądanie. Jest możliwe wysłanie żądania do serwera *Ninf* a następnie kontynuowanie dalszych obliczeń lokalnych. Możliwe jest również kierowanie wielu żądań RPC do różnych serwerów. Z powyższych powodów, asynchroniczne RPC systemu *Ninf* jest ważnym mechanizmem umożliwiającym programowanie współbieżne. Istnieje kilka funkcji API służących do oczekiwania

na odpowiedź serwera obliczeniowego, umożliwiających programiście pozyskiwanie wyników wykonania dowolnego zbioru żądań *Ninf* w dogodny dla niego sposób.

1.2.3.4. Wywołania zagregowane – transakcje Ninf

Ninf zapewnia również interfejs wywołań zagregowanych, który traktuje określony przez użytkownika blok kodu, nazywany transakcją, jako jedną jednostkę szeregowania. Blok kodu otoczony dyrektywami `Ninf_transaction_begin` i `Ninf_transaction_end` nie jest wykonywany natychmiastowo – dopiero na końcu bloku meta-serwer szereguje obliczenia odpowiednio do kilku serwerów obliczeniowych. Kiedy wszystkie obliczenia są wykonane, meta-serwer zwraca wyniki do klienta. Ten mechanizm jest przydatny dla współbieżnego wykonania zadań, które mogą być łatwo podzielne na mniejsze pod-problemy. Transakcje mogą być również użyte do wykonania obliczeń wymagających dużych przepływów danych. W obrębie transakcji są automatycznie wykrywane zależności danowe pomiędzy poszczególnymi wołaniami funkcji systemu *Ninf*; te zależności są brane pod uwagę podczas szeregowanie obliczeń przez meta-serwer *Ninf*.
Dla przykładu, rozważmy dodawanie czterech macierzy A, B, C i D (wynik ma znaleźć się w macierzy G):

```
Ninf_transaction_begin();
  Ninf_call("madd",n,A,B,E);
  Ninf_call("madd",n,C,D,F);
  Ninf_call("madd",n,E,F,G);
Ninf_t0ransaction_end();
```

Ponieważ dodawanie macierzy A i B oraz macierzy C i D może być wykonywane współbieżnie, jest właśnie w ten sposób szeregowane, natomiast dodawanie macierzy E i F jest wykonywane po zakończeniu tych operacji.

1.2.3.5. Wołania zwrotne Ninf RPC

RPC systemu *Ninf* daje możliwość określenia funkcji, która będzie, zwrotnie, wywoływana przez serwer obliczeniowy podczas wykonywania obliczeń zainicjowanych przez RPC (ang. *call-back function*). Funkcje biblioteczne systemu *Ninf* mogą przyjmować argumenty specyfikujące funkcje, które mają być wykonane zwrotnie a są dostarczone przez klienta (ang. *user-supplied routines*). Rozważmy następujący przykład: w systemie *Ninf* wykonywana jest symulacja naukowa. Typowy program symulacyjny inicjalizuje pewien stan i, w toku symulacji, aktualizuje podgląd tego stanu lub pliki go opisujące, zapisując np. wartości

zmiennych stanowych w pewnych chwilach czasowych w pliku logu. Mechanizm wywołania zwrotnego może być użyty, by wywołać dostarczone przez użytkownika funkcje służące do zapisywania danych do pliku lub aktualizacji podglądu procesu,

1.2.3.6. Dostępność interfejsu programistycznego dla różnych języków programowania

Interfejs programistyczny klienta systemu *Ninf* jest tak zaprojektowany, by być niezależny od języka na tyle, na ile to możliwe. Dzięki temu klienci systemu *Ninf* mogą być zaprogramowani w jednym wielu popularnych języków. Aktualnie zaimplementowane są funkcje `Ninf_call` dla języka *C*, *Fortran* i *Java*. Dodatkowo, stworzony jest interfejs programistyczny niższego poziomu (ang. *sub-API*), które umożliwia implementację API dla systemu *Ninf* we wszystkich językach, które wspierają obce interfejsy do programów w języku C. Jednakże, należy zaznaczyć, że bardzo trudna jest implementacja API systemu *Ninf* w językach, w których reprezentacja tablic jest zupełnie inna niż w *C* (np. *Lisp*).
W celu ułatwienia implementacji API systemu *Ninf*, jego autorzy stworzyli pod-API nazywane *Ninf CIM* (ang. *Common Interface Module* – wspólny moduł interfejsu, wspólny interfejs). Wykorzystując to „pod-API", implementujący API muszą jedynie stworzyć kilka podstawowych funkcji zwanych „przemierzaczami" (ang. *traversers*). Te funkcje „przemierzają" naturalne dla danego języka struktury danych i pobierają/ustawiają wartości danych w odpowiednich miejscach. Funkcja `Ninf_cim_main()`, wersja funkcji `Ninf_call()` wykorzystująca mechanizm CIM, upakowuje/rozpakowuje dane poprzez wywołanie tych „funkcji-przemierzaczy". API dla programów *Excel*, *Mathematica* oraz języka *Lisp* zostały opracowane właśnie z użyciem mechanizmów CIM.

1.2.4. Język opisu interfejsu

Dostawca biblioteki *Ninf* opisuje interfejs biblioteki funkcji bibliotecznych w języku opisu interfejsu *Ninf* IDL (ang. *Interface Description Language*). Ponieważ system *Ninf* jest przeznaczony dla aplikacji obliczeniowych, obsługiwane w *Ninf* typy danych są ograniczone do niezbędnych w tych zastosowaniach, tzn. do skalarów i ich wielowymiarowych tablic. Z drugiej strony, co jest ważne dla aplikacji numerycznych, możliwe jest używanie wyrażeń dotyczących argumentów wejściowych, pozwalających na obliczenie rozmiarów tablic, stwierdzanie jakie tymczasowe tablice muszą być utworzone na serwerze obliczeniowym itd.

Dla przykładu, opis interfejsu funkcji mnożenia macierzy (patrz punkt 1.2.3.4) wygląda następująco:

```
Define dmmul(long mode_in int N,        mode_in double A[N][N],
             mode_in double B[N][N],    mode_out double C[N][N])
"... opis ..."
Required "libxxx.o"        /* określa biblioteki wymagane przez tę funkcję */
Calls "C" dmmul(N,A,B,C);  /* używa konwencji wołań języka C */
```

gdzie specyfikatory dostępu (ang. *access specifiers*) `mode_in` i `mode_out` określają, czy argumenty są odczytywane czy zapisywane. Dla określenia rozmiaru każdego z argumentów mogą zostać użyte inne argumenty (w trybie `mode_in`), które pozwolą sformułować wyrażenie, na podstawie którego może zostać obliczony rozmiar argumentu. W powyższym przykładzie dla obliczenia rozmiaru macierzy A, B i C używana jest wartość argumentu N.
Dodatkowo, poza definicją interfejsu funkcji bibliotecznej, opis w języku IDL zawiera informacje niezbędne do kompilacji i łączenia (ang. *link*) bibliotek.

1.2.4.1. Generator interfejsów systemu Ninf

Opis interfejsu jest kompilowany przy użyciu generatora interfejsów systemu *Ninf* (ang. *Ninf interface generator*), który tworzy programy-stopki dla każdej opisanej funkcji bibliotecznej. Tworzy on także pliki *makefile*, które służą do tworzenia obiektów wykonywalnych *Ninf* poprzez łączenie programów-stopek i funkcji bibliotecznych.
Użytkownicy zdalni mogą, za pomocą przeglądarki *web*, przeglądać dostępne na serwerze *Ninf* funkcje biblioteczne, ich interfejsy i inne informacje wygenerowane przez generator.

1.2.5. Szeregowanie w systemie Ninf

Jednym z największych zagadnień badawczych związanych z systemami „obliczeń globalnych” (ang. *global computing*) jest problem szeregowania zadań. Szeregowanie to powinno zapewnić efektywne wykorzystanie zasobów (lub minimalizację czasu obliczeń). W tej części opisany zostanie schemat szeregowania w systemie *Ninf*, meta-serwer systemu, który zawiera moduł szeregujący (ang. *scheduler*) a także monitor zasobów (ang. *resource monitor*). Opisany zostanie również model symulacyjny, który jest używany do wyznaczenia najbardziej optymalnego algorytmu szeregowania.

1.2.5.1. Wymagania dla modułu szeregującego

Moduł szeregujący musi zbierać (rozproszone) informacje dotyczące stanu (zajętości – ang. *utilization*) zasobów obliczeniowych i sieciowych. Bazując na zebranej informacji, która reprezentuje jedynie obraz przeszłego stanu całego systemu, moduł szeregujący musi dokonać predykcji aktualnego obciążenia zasobów. By dokonanie takiej predykcji było możliwe, wszystkie zebrane informacje muszą rezydować u globalnego zarządcy bazy danych.

1.2.5.2. Moduł szeregujący – części składowe

Moduł szeregujący składa się z następujących komponentów:

- monitora obciążenia zasobów obliczeniowych (np. serwera obliczeniowego),
- monitora przepustowości (i opóźnienia) sieci,
- zarządcy bazy danych o zasobach,
- modułu predykcji stanu zasobów,
- modułu rozdziału zasobów (ang. *resource scheduler*), realizującego pewien algorytm szeregowania.

Zauważmy, że dla monitora przepustowości sieci, pasmo pomiędzy klientem i węzłem obliczeniowym powinno być mierzone przez każdego klienta, ponieważ np. w środowiskach sieci rozległej (np. *Internet*), przepustowość pomiędzy dwoma węzłami jest trudna do obliczenia na podstawie innych informacji o przepustowości. Z drugiej strony, monitorowanie obciążenia może być przeprowadzane przez tylko jeden, reprezentatywny dla grupy węzłów obliczeniowych, serwer.

Monitory okresowo raportują stan zasobów zarządcy bazy danych. Na podstawie umieszczonej w nim informacji predyktor stanu zasobów przewiduje aktualną i przyszłą dostępność zasobów. Z kolei te prognozy, w połączeniu z informacjami z bazy danych są wykorzystywane przez moduł szeregujący do przydziału zasobów, tzn. przydzielania odpowiedniego węzła obliczeniowego do obsługi każdego z żądań klientów

1.2.5.3. Meta-serwer systemu Ninf

Biorąc pod uwagę powyższe zagadnienia, autorzy projektu zaimplementowali prototypowy moduł szeregujący nazywany meta-serwerem. Meta-serwer składa się z następujących części:

- monitor obciążenia zasobów obliczeniowych (ang. *load monitor for computing resources*),
- agent proxy dla klienta (ang. *client proxy*),
- baza danych informacji o zasobach (ang. *resource information database manager*),
- moduł szeregujący zasoby (ang. *resource scheduler*).

Proxy klienta ma dwie role: po pierwsze, monitoruje przepustowość sieci („w imieniu" całej „lokalizacji" – grupy węzłów znajdujących się w danym obszarze sieci); po drugie, działa jako *serwer-proxy* dla sieci zabezpieczonych systemami *firewall*.

Meta-serwer działa w następujący sposób:

a) monitor obciążenia obserwuje obciążenie serwerów obliczeniowych i składuje dane w bazie informacji o zasobach;
b) *proxy* klienta, periodycznie, monitoruje stan łączy komunikacyjnych (m.in. obciążenie sieci) pomiędzy klientami (w imieniu których działa) a serwerami obliczeniowymi i utrzymuje wyniki pomiarów w swojej pamięci;
c) klient wysyła żądanie szeregowania wywołania funkcji bibliotecznej *Ninf* do *proxy* klienta;
d) *proxy* klienta, z kolei, zadaje zapytanie do modułu szeregującego, dołączając jego, wewnętrznie utrzymywaną, informację o stanie łączy komunikacyjnych pomiędzy nim a serwerem;
e) moduł szeregujący otrzymuje informację o obciążeniu serwerów z bazy informacji o zasobach i wybiera najbardziej odpowiedni serwer dla wykonania funkcji obliczeniowej, w zależności od charakterystyki żądanej funkcji (zazwyczaj, serwer, który osiąga najlepszy czas odpowiedzi, utrzymując przy tym sensowną ogólną wydajność); następnie informuje *proxy* klienta o tym, który serwer został wybrany;
f) *proxy* klienta przesyła żądanie wykonania obliczeń do przydzielonemu klientowi serwera.

Jeśli wszystkie kontrolowane przez meta-serwer serwery obliczeniowe są obciążone, meta-serwer wstrzymuje wywoływanie funkcji obliczeniowych – czyta i składuje wszystkie dane potrzebne do ich uruchomienia i czeka na moment, w którym obciążenie jednego (lub więcej) serwera spadnie na tyle, by można było wysłać mu zadanie obliczeniowe. Ten proces jest całkowicie niewidoczny dla użytkownika, dzięki czemu klienci mogą wywołać wiele zadań, nawet gdy w danej chwili żaden serwer obliczeniowy nie jest dostępny.

1.2.5.4. Modyfikowalność, możliwość konfiguracji i zastępowalność modułu szeregującego

Dla maksymalizacji przenośności, wszystkie moduły meta-serwera są zaimplementowane w języku *Java* a główne moduły szeregujące są tak skonstruowane, by były łatwo modyfikowalne i zastępowalne, tak by można było zmieniać politykę szeregowania „w locie" poprzez odpowiednie protokoły sieciowe.

1.2.5.5. Infrastruktura meta-serwerów

System *Ninf* umożliwia prowadzenie obliczeń przy wykorzystaniu bardzo wielu zasobów obliczeniowych i sieciowych. Taka rozległość i wielość kontrolowanych zasobów wymaga odpowiedniej, modułowej konstrukcji systemu. Infrastruktura systemu *Ninf* składa się ze zbiorów serwerów obliczeniowych i meta-serwerów rozproszonych w sieci. Meta-serwery, okresowo, wymieniają informacje pomiędzy sobą. Np. jeśli nowy serwer obliczeniowy *Ninf* jest dodawany do meta-serwera, lokalizacja nowego serwera jest propagowana wśród meta-serwerów. W sytuacji, gdy dany meta-serwer nie posiada pewnej informacji, np. dotyczącej dostępności pewnej biblioteki na określonym serwerze obliczeniowym, propaguje zapytanie do innych meta-serwerów, a następnie, odebraną informację przekazuje klientowi lub podejmuje odpowiednią do żądania klienta decyzję.

Meta-serwer *Ninf* utrzymuje następujące informacje o serwerach obliczeniowych:

- lokalizację serwera (adres IP i numer portu),
- listę funkcji bibliotecznych zarejestrowanych na serwerze,
- dystans do serwera z uwzględnieniem przepustowości sieci,
- informacje o zdolności obliczeniowej serwera (prędkość zegara, wskaźnik *Flops* – ang. *floating point operations per second*),
- stan serwera, włączając średnie obciążenie (ang. *load average*).

1.2.5.6. Wyrównywanie obciążenia realizowane przez meta-serwer

Meta-serwer w procesie szeregowania dąży nie tylko do przydzielenia zadaniu użytkownika najlepszego możliwego serwera obliczeniowego, ale również do wyrównania obciążenia poszczególnych maszyn.

Podstawowym obszarem, w którym może zostać zastosowany *load balancing* są transakcje *Ninf*, czyli zagregowane wywołania kilku funkcji obliczeniowych. Podczas wykonania, te sekwencje funkcji są analizowane pod kątem zależności pomiędzy ich argumentami.

Niezależne wywołania funkcji są wykonywane współbieżnie na różnych serwerach obliczeniowych.

Dla klienta, transakcja wygląda jak atomowe wykonanie zagregowanych wołań funkcji. W momencie uruchomienia jednak, każde wywołanie pojedynczej funkcji (`Ninf_call`) wewnątrz bloku transakcji nie powoduje rzeczywistego wykonania zdalnej funkcji, lecz zamiast tego, powoduje skonstruowanie pewnej części grafu przepływu danych. Dopiero po osiągnięciu przez klienta dyrektywy kończącej transakcję, skonstruowany graf jest przesyłany do meta-serwera. Wtedy meta-serwer szereguje wołania funkcji bibliotecznych opisane grafem, przydzielając je do odpowiednich serwerów obliczeniowych. Kiedy wszystkie, składowe obliczenia są zakończone, meta-serwer zwraca ostateczny wynik do klienta.

1.2.5.7. Implementacja mechanizmu dynamicznego rozdziału zasobów za pomocą meta-serwera

Dynamiczny *load balancing* różni się, w ujęciu autorów systemu *Ninf*, od statycznego tym, że przydział zadań obliczeniowym składających się na większe zadanie (czyli pojedynczych wywołań funkcji `Ninf_call`) do zasobów nie jest wykonywany jednorazowo, z góry, lecz czyniony jest na bieżąco, wraz postępem obliczeń, czyli po zakończeniu każdej operacji elementarnej. Taki mechanizm jest konieczny w przypadku wykonywania obliczeń, które są, owszem, podzielne, jednak nie znany (i nieprzewidywalny) jest *a priori* czas trwania poszczególnych części. Stosowanie statycznego *load balancing*'u nie gwarantowałoby więc równomiernego obciążenia poszczególnych węzłów obliczeniowych. Dynamiczny *load balancing* pozwala na przydzielanie pracy tym węzłom, które wcześniej skończyły poprzedni etap obliczeń, przez co uzyskuje się pewne wyrównanie obciążenia i minimalizację czasu wykonania całego zadania.

W systemie *Ninf* dynamiczny *load balancing* wykonywany może być na dwa sposoby:

a) wykorzystanie wywołań zagregowanych – całe zadanie wysyłane jest, w postaci transakcji, do meta-serwera, który dokonuje rozdziału składowych elementów transakcji pomiędzy maszyny obliczeniowe; i tak, gdy wykonanie którejś z funkcji bibliotecznych wchodzących w skład transakcji zakończy się a w kolejce czekają inne, nie wykonane jeszcze wywołania funkcji bibliotecznych, meta-serwer wykonuje wywołania tych funkcji na zwolnionym węźle obliczeniowym;
b) wykorzystanie wywołań zwrotnych – po wykonaniu każdej funkcji składającej się na całe zadanie wykonywane jest wołanie zwrotne funkcji po stronie klienta; funkcja ta może

inicjalizować następny etap obliczeń a meta-serwer przydzieli temu żądaniu odpowiednie, dostępne zasoby.

Techniki te mają zasadnicze różnice. W pierwszym przypadku, nie jest wymagany, poza ujęciem bloku operacji w transakcję, żaden dodatkowy wysiłek ze strony programisty – całość pracy związanej z wyrównywaniem obciążeń jest wykonywana przez meta-serwer. W drugim przypadku, programista musi dostarczyć funkcji, które będą zwrotnie wywoływane i inicjalizować będą następne etapy obliczeń.

1.2.5.8. Ocena poprawności i kosztów dynamicznego szeregowania przez meta-serwer

W pracy [5] autorzy systemu *Ninf* przeprowadzili ocenę poprawności, wydajności i kosztów szeregowania przez meta-serwer.

Koszt szeregowania przeanalizowano w dwóch przypadkach: podstawowego kosztu szeregowania oraz kosztu dynamicznego *load balancing*'u.

Pierwszy eksperyment polegał na porównaniu czasu wykonania obliczeń rozdzielanych między węzły obliczeniowe przez meta-serwer i rozdzielonych, *a priori*, przez użytkownika (zadanie obliczeniowe było dobrze i „równomiernie" podzielne) stwierdzono, że:

- koszt meta-serwera wzrasta proporcjonalnie do liczby węzłów obliczeniowych, pomiędzy które rozdzielane są zadania elementarne; co ciekawe, jest niezależny od rozmiaru zadania;
- koszt jest na tyle mały, że może być, w przypadku zadań złożonych obliczeniowo, zignorowany.

Drugi eksperyment porównywał dynamiczny *load balancing* wykonywany przez meta-serwer, *load balancing* wykonywany metodą wywołań zwrotnych a także prosty, cykliczny rozkład obciążenia (czyli rozkład, *a priori*, całego zadania na części i kolejne ich wykonywanie). Zadanie obliczeniowe było dobrze podzielne, jednak nieznane były z góry czasy wykonania poszczególnych segmentów zadania. Eksperyment doprowadził do następujących wniosków:

- czas wykonania zadania w przypadku stosowania jednego węzła obliczeniowego zwiększa się wraz ze wzrostem liczby dekompozycji (czyli operacji podziału dużego zadania na mniejsze); spowodowane jest to, zdaniem autorów, zwiększonym narzutem na łączenie kilku wywołań funkcji `Ninf_call` w jedno wołanie tej funkcji;
- w miarę zwiększania liczby serwerów obliczeniowych, czas obliczeń, monotonicznie zmniejsza się;

- dobór odpowiedniej liczby dekompozycji do liczby serwerów obliczeniowych prowadzi do uzyskania pewnego optimum wydajności: dla 8 węzłów – 32 dekompozycje, dla 16 węzłów 64 dekompozycje itd.; z dużej liczby dekompozycji wynika lepsze wyrównanie obciążenia, ale może ona nie być optymalna ze względu na zwiększony narzut po stronie meta-serwera;
- dynamiczny *load balancing* przy użyciu meta-serwera obfituje lepszą wydajnością niż statyczny, cykliczny rozkład zadania, jednak, w porównaniu z jawnym, wykonywanym po stronie klienta, rozdziałem pracy (z wykorzystaniem mechanizmu wywołania zwrotnego), narzut szeregowania przez meta-serwer jest większy.

Autorzy systemu *Ninf* spekulują, że zjawisko „spowolnienia" obserwowane przy stosowaniu *load balancing*'u po stronie meta-serwera spowodowane jest następującymi zjawiskami:

- liczba wywołań (`Ninf_call`) jest równa liczbie węzłów, które wykonują wołanie zwrotne, podczas gdy liczba ta jest równa liczbie dekompozycji wykonywanych na meta-serwerze (a te dekompozycje kosztują);
- obliczenia „składowe" nie rozpoczną się wcześniej niż nie zakończy się buforowanie argumentów wszystkich funkcji transakcji w meta-serwerze; pociąga to za sobą pewne opóźnienia oraz niemożność „nakładania" (w czasie) obliczeń i komunikacji, tak jak dzieje się to przy przetwarzaniu strumieniowym (ang. *pipelining*); podobnie, meta-serwer nie zwraca wyników zanim wszystkie składowe obliczenia danego, złożonego zadania nie wykonają się.

Zdaniem autorów systemu *Ninf*, drugie z wymienionych zjawisk (w przeciwieństwie do pierwszego, które „jest podstawowe dla tego podejścia"), jest możliwe do wyeliminowania przez lepszą implementację systemu. Poza tym, narzut wydajności na szeregowanie w meta-serwerze jest równoważony przez fakt, że całość działań „wyrównawczych" wykonywana jest przez meta-serwer systemu *Ninf* i nie angażuje programisty [4]. A przecież łatwość stosowania jest ważnym aspektem systemu *Ninf*.

1.2.6. Model środowiska obliczeniowego

Dla oceny wydajności obliczeń potrzebnej do szeregowania zadań autorzy systemu Ninf stworzyli model środowiska obliczeniowego.

1.2.6.1. Wady eksperymentalnej oceny algorytmów szeregowania

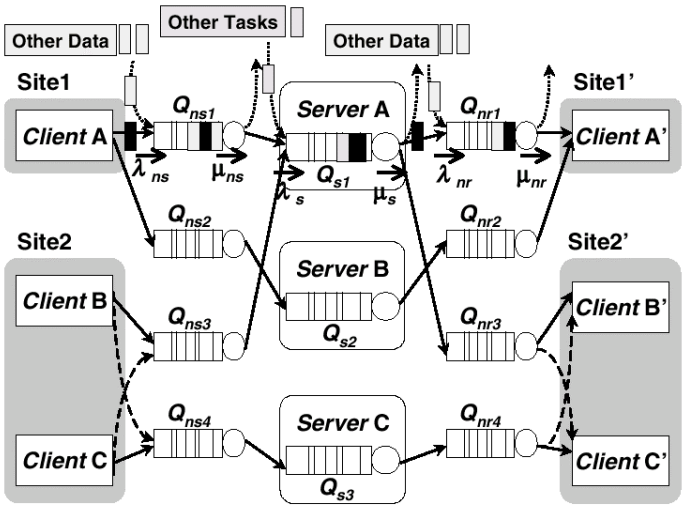
W świecie naukowym prowadzonych jest wiele prac badawczych dotyczących szeregowania w systemach globalnych obliczeń. Do takich inicjatyw należą między innymi *AppLeS*, *Prophet* i *Condor* oraz *Nimrod*. Autorzy systemu Ninf zakładają jednak, że systemy rozwijane obrębie tych projektów nie dostarczają wystarczająco wiarygodnych algorytmów szeregowania, szczególnie dla środowiska WAN. Ich zdaniem, [4] i [3], eksperymenty na istniejących systemach nie są wystarczające, by rozważać ogólną wydajność algorytmów szeregowania. Ponieważ np. przepustowość sieci i jakość usług przez nie oferowanych oraz dostępność zasobów obliczeniowych podlega zarówno krótko jak i długoterminowym zmianom, testy wydajności nie są wystarczające dla oceny algorytmów szeregowania.

Testy przeprowadzone w środowiskach sieci lokalnej i sieci rozległej dowodzą, że algorytmy szeregowania, które sprawdzają się w jednej konfiguracji (np. LAN albo WAN) mogą okazać się bezużyteczne w innej [4]. Co więcej, szeregowanie powinno uwzględniać cechy wykonywanych zadań, przede wszystkim zaś ich złożoność komunikacyjną i obliczeniową. I tak, dla zadań złożonych komunikacyjnie wiodącym wskaźnikiem, przy określaniu przydziału zasobów, jest uwzględnienie pojemności sieci łączącej klienta i serwer obliczeniowy, natomiast przy zadaniach złożonych obliczeniowo bardziej istotne staje się obciążenie samego serwera obliczeniowego. Nie do pominięcia jest również aktualne rozmieszczenie klientów i serwerów obliczeniowych w sieci: testy dowodzą [4] np., że dla zadań złożonych komunikacyjnie, fakt że klienci danego serwera obliczeniowego znajdują się w jednej lokalizacji lub w różnych lokalizacjach decyduje o tym, jak bardzo sieć ma tendencję do nasycania się, z powodu równoczesnego przesyłania danych i wyników od i do różnych klientów, co, oczywiście, bardzo obniża wydajność zdalnych obliczeń. Porównanie wydajności obliczeń złożonych komunikacyjnie dla konfiguracji „wielu klientów w jednej lokalizacji" i „wielu klientów w różnych lokalizacjach" potwierdza przypuszczenie, że mniej istotna jest w tej klasie zadań konfiguracja sieci a bardziej decydująca o wydajności obliczeń jest wydajność samego serwera obliczeniowego (prędzej nastąpi jego „przeciążenie" niż łączy komunikacyjnych). Dobre i skuteczne algorytmy szeregowania muszą uwzględniać wymienione powyżej czynniki (sieć, obciążenie serwerów i konfigurację), dostosowując wagi

przypisywane poszczególnym wartościom do klasy rozwiązywanego zadania. Część istniejących systemów szeregowania traktuje wymienione zagadnienia fragmentarycznie. Autorzy pracy [3], stwierdzili: „nie zostały, jak dotychczas, stworzone modele dla szeregowania zadań w środowiska globalnego przetwarzania".

1.2.6.2. Model środowiska obliczeniowego – model sieci kolejkującej

Wymienione wyżej stwierdzenia doprowadziły autorów systemu *Ninf* do wniosku, iż potrzebny jest abstrakcyjny i logiczny model środowiska obliczeniowego, który pozwoli symulować takie środowisko, jego konfigurację, cechy i zachowanie, a dzięki temu oceniać algorytmy szeregowania, również jeśli środowisko obliczeniowe jest siecią rozległą. Ten model symulacyjny oparty jest o model sieci kolejkującej (ang. *queueing network model*). Rysunek 1.2 daje pogląd na taki model symulacyjny dla typowego, opartego o mechanizm RPC, globalnego systemu obliczeniowego. Na rysunku, klienci A i A', B i B' oraz C i C' oznaczają tych samych klientów, najpierw w roli nadawców, potem w roli odbiorców.



Rysunek 1.2. Model sieci kolejkującej

Źródło: praca [6]

W modelu, wywołanie i przesłanie argumentów od klienta do serwera, przetwarzanie na serwerze i zwracanie odpowiedzi od serwera do klienta jest modelowane jako kolejki, odpowiednio, Q_{ns_i}, Q_{nr_i} oraz Q_{s_i} . $\lambda_{ns}, \lambda_{nr}$ są prędkościami napływania żądań w sieci innych niż `Ninf_call`, innymi słowy mówiąc, modelują one przeciążenia (ang. *congestions*) i opóźnienia zachodzące w sieci rozległej. Zakłada się, że $\lambda_{ns}, \lambda_{nr}$ mają rokład Poisona, jednak

brane pod uwagę są również inne, bardziej odpowiednie dla środowiska WAN rozkłady. μ_{ns}, μ_{nr} są prędkościami obsługi kolejek, tj. reprezentują przepustowość sieci (zakłada się rozład wykładniczy). λ_s, μ_s, podobnie, reprezentują stopień przeciążenia serwerów i ich moc obliczeniową.

Dla przykładu, na rysunku 1.2 klienci B i C są zlokalizowani w na tym samym węźle sieci; dodatkowo zakłada się, że współdzielą segment sieci do dowolnego danego serwera obliczeniowego. Dla zareprezentowania tego faktu w modelu, dane transmitowane z i do obu tych węzłów trafiają do tej samej kolejki reprezentującej sieć.

1.2.7. Symulacja środowiska obliczeniowego na bazie modelu sieci kolejkującej

Autorzy systemu *Ninf* wychodzą z założenia, że stosowane powszechnie procedury testowania algorytmów szeregowania pozwalają wprawdzie na ich ocenę w konkretnym środowisku, jednak nie dostarczają wiedzy, która mogłaby być wykorzystana w innych przypadkach. Ich zdaniem, trudne jeśli nie niemożliwe, jest formułowanie ogólnych tez dotyczących algorytmów szeregowania oraz porównywanie poszczególnych algorytmów i ich przydatności w skomplikowanym środowisku obliczeniowym sieci rozległej. Każda eksperymentalna ocena metodyki szeregowania odbywa się w niepowtarzalnych warunkach a przez to dostarcza wyników, które nie mogą być wykorzystane w innych środowiskach obliczeniowych. Poza tym, z powodu wysokich kosztów, skala rzeczywistych, możliwych do przeprowadzenia eksperymentów, jest, zdaniem Japończyków, ograniczona. Wynikający z tego brak całościowych i ogólnych a także porównywalnych podejść do problemu szeregowania jest wielką przeszkodą na drodze rozwojów systemów globalnych obliczeń.

Dla wyjścia z tego impasu, autorzy systemu *Ninf* proponują system oceny wydajności *Bricks* [6], który będzie pozwalał na analizę i porównanie różnych schematów szeregowania w typowym, wysoko wydajnym, globalnym środowisku obliczeniowym.

1.2.7.1. System Bricks

Bieżąca wersja systemu *Bricks* skupia się na ocenie różnych algorytmów i schematów szeregowania opartych na kanonicznym modelu wysoko-wydajnego środowiska obliczeniowego (czyli modelu sieci kolejkującej). System napisany jest w języku *Java*.

Składa się z symulowanego globalnego środowiska obliczeniowego (ang. *Global Computing Environment*) i jednostki szeregującej (ang. *Scheduling Unit*), pozwalających na symulację zachowania:

- algorytmów szeregowania zasobów,
- programów szeregujących,
- topologii sieci i wzajemnego rozmieszczenia serwerów i klientów,
- schematów przetwarzania dla sieci i serwerów obliczeniowych.

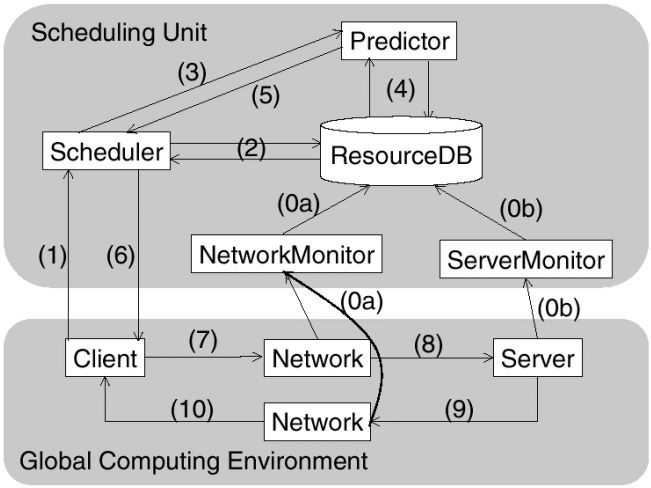
Konfiguracja i parametry globalnego środowiska obliczeniowego mogą być łatwo opisywane za pomocą skryptów systemu *Bricks* (ang. *Bricks script*). Dla przeprowadzania użytecznych symulacji użytkownicy mogą konstruować lub zmieniać skrypty używając składowych skryptów zwanych przez autorów systemu „cegiełkami" (ang. *bricks*).

System *Bricks* zawiera ramy i elementy składowe, które monitorują zasoby środowiska obliczeniowego i przewidują ich obciążenie. Informacje te potrzebne są jako dane wejściowe dla procesów dokonujących szeregowania zasobów. System *Bricks* dostarcza pewnych domyślnych komponentów dla monitorowania, predykcji i szeregowania zadań w symulowanej sieci. Ponieważ te komponenty są tak zaprojektowane, by być niezależnymi od ich (starannie zdefiniowanych) interfejsów programistycznych (ang. *component APIs*), możliwa jest ich podmiana innymi napisanymi w języku *Java* komponentami: np. można opisać nowy algorytm szeregowania w języku *Java*, zgodnie z interfejsem dostarczyciela usługi (ang. *Service Provider Interface*) i testować go pod systemem *Bricks*. Co więcej, składowe (komponenty) mogą być zewnętrzne, w szczególności mogą być składowymi rzeczywistych systemów szeregujących. System *Bricks* może dostarczać symulowany czas jak również rozmaite monitorowane, symulowane informacje do zewnętrznych, powiązanych z zasobami systemów i odbierać od nich wyniki podjętych decyzji, uwzględniając je w symulacji. Chociaż, zdaniem autorów systemu, jest jeszcze zbyt wcześnie, by stwierdzić, że system *Bricks* może łatwo integrować się z dowolnymi składowymi systemów globalnego przetwarzania, to jednak zauważyć trzeba, że udało się system Bricks zintegrować z istniejącym systemem NWS (ang. *Network Weather Service* – patrz punkt 2.1) dzięki

zdefiniowaniu dla tego systemu API dla języka *Java*. System *Bricks* oparty jest na modelu sieci szeregującej (patrz punkt 1.2.6).

1.2.7.2. Architektura systemu Bricks

W systemie *Bricks*, symulator globalnego środowisko obliczeniowego (na rysunku: ang. *Global Computing Environment*) i jednostka szeregująca (na rysunku: ang. *Scheduling Unit*) wspólnie symulują zachowanie środowiska obliczeniowego. Ogólnie rzecz ujmując, system *Bricks* działa jako dyskretny symulator zdarzeń w sieci kolejkującej w wirtualnym czasie. Poszczególne kroki symulacji w systemie pokazuje rysunek 1.3.



Rysunek 1.3. Kroki symulacji w systemie Bricks

Źródło: praca [6]

1.2.7.3. Składowe systemu Bricks

Symulator globalnego środowiska obliczeniowego składa się z następujących części:

- klient – reprezentuje maszynę klienta, na której inicjalizowane są zadania obliczeniowe użytkowników,
- sieć – reprezentuje (rozległą) sieć łączącą klienta i serwer obliczeniowy; może być ona parametryzowana, np. przez pasmo, obciążenie i ich zmienność w czasie,
- serwer – oznacza zasoby obliczeniowe danego środowiska obliczeniowego i jest parametryzowany przez m.in. wydajność, obciążenie i ich zmienność w czasie.

Zarówno sieć jak i serwer są modelowani jako kolejki, których schemat przetwarzania może być zmieniany. Model zadania wprowadzanego przez maszynę klienta, modele komunikacyjne sieci i modele serwera będą omówione poniżej.

Model zadania

Symulator powinien zarządzać a także rozpoznawać informację o czasie trwania komunikacji i obliczeń związanych z danym zadaniem. W bieżącej implementacji systemu Bricks zadanie jest reprezentowane przez:

- ilość danych przesyłanych z i do serwera razem z zadaniem a także wielkość komunikacji wykonywanej w czasie obliczeń na serwerze;
- liczbę wykonywanych w zadaniu instrukcji (operacji).

Modele komunikacyjne

System *Bricks* może symulować rozmaite modele komunikacji w sieci dzięki ich specyfikacji w skryptach. Aktualnie, są dwie główne rodziny modeli wspierane, obsługiwane (ang. *supported*) przez system. Pierwsza rodzina zakłada, że obciążenie sieci jest reprezentowane przez modyfikację wartości określających ilość danych związanych z ubocznym ruchem w sieci generowanym przez inne węzły systemu. W tym przypadku należy określić idealną przepustowość, średnią aktualnej przepustowości, średni rozmiar danych z zewnętrznych węzłów i ich zmienność. Określanie małych pakietów będzie skutkować większą dokładnością symulacji przy wzroście kosztów symulacji. W drugiej rodzinie, zmienność przepustowości sieci w każdym przedziale czasowym jest określana przez obserwowane parametry rzeczywistego środowiska sieciowego. Koszt takiej symulacji jest dość niski (stosuje się metody interpolacyjne włączając metody liniowego i nieliniowego dopasowania) a symulator sieci zachowuje się, jakby był rzeczywistą siecią. Obie te rodziny modeli służą do generowania bogatego zbioru modeli zachowania sieci w globalnym środowisku obliczeniowym. Jest to możliwe dzięki temu, że mogą być określane rozmaite parametry (takie jak np. różne probabilistyczne funkcje prędkości przybywania pakietów). Trwają prace nad rozszerzeniem systemu *Bricks* tak, by uwzględniał jeszcze więcej rodzin modeli. Ma to doprowadzić do zwiększenia dokładności symulacji, zwiększenia jej efektywności, wygody dla użytkowników itd.

Modele serwerów

Aktualnie, system *Bricks* modeluje serwery obliczeniowe jako przetwarzające zgodnie z zasadą FCFS (ang. *First Came First Served* – pierwszy przyszedł, pierwszy obsłużony) i działa jak kolejka. Jego obciążenie może być określone i symulowane nie tylko jako prędkość przybywania zadań od innych użytkowników (to jest zewnętrznych zadań) ale również może być określana przez obserwowane parametry rzeczywistego środowiska.

Autorzy systemu *Bricks* deklarują wolę rozszerzenia modelu zadania tak, by umożliwić mu reprezentację różnych typów zadań, i tak, dążą do tego, by można było reprezentować np. aplikacje równoległe. Podobnie, modele serwerów powinny być tak rozszerzone, by obrazować różne architektury serwerów, takie jak SMP czy a także różne schematy przetwarzania i szeregowania w systemach obliczeniowych pracujących pod kontrolą zarządców zasobów takich jak LSF (ang. *Load Sharing Facility*).

Inną, główną częścią systemu *Bricks* jest jednostka szeregująca. Składowe jednostki szeregującej reprezentują wspólne cechy globalnych systemów obliczeniowych w następujący sposób:

- monitor sieci (ang. *Network Monitor*) – mierzy przepustowość sieci i jej opóźnienie w globalnym środowisku obliczeniowym; mierzone wartości składowane są w bazie danych o zasobach (ang. *ResourceDB*);
- monitor serwera (ang. *Server Monitor*) – mierzy wydajność, obciążenie i dostępność serwerów; mierzone wartości również umieszczane są w bazie danych o zasobach;
- baza danych zasobów (ang. *ResourceDB*) – działa jako baza danych potrzebnych do szeregowania, składując wartości rozmaitych pomiarów; mierzone wartości są pobierane przez moduł predykcji (dla przewidywania dostępności zasobów) i moduł szeregujący (dla podejmowania decyzji o rozdziale zasobów).
- moduł predykcji (ang. *Predictor*) – odczytuje pomiary dotyczące zasobów z pewnego okresu czasu z bazy danych zasobów i przewiduje dostępność zasobów; przewidywania te są zazwyczaj używane dla szeregowania;
- moduł szeregujący (ang. *Scheduler*) – przydziela nowe zadania wywoływane przez klientów do odpowiednich dla nich maszyn; decyzje o przydziale zasobów są podejmowane w oparciu i informacje otrzymane z bazy danych zasobów i od modułu predykcji.

Podobnie jak w symulatorze globalnego środowiska obliczeniowego, składowe jednostki szeregującej napisane są w języku *Java*. Interfejsy dostawcy usługi (ang. *SPIs - Service Provider Interfaces*) pozwalają na podmianę składowych przez alternatywne, dostarczone przez użytkownika moduły. Dla przykładu możliwa jest zamiana jednostki szeregującej tak, by stosowała nowe algorytmy szeregowania lub wymiana modułu predykcji tak, by wcielał zewnętrzne moduły predykcji takie jak np. NWS (patrz punkt 2.1).

1.2.7.4. Wcielanie istniejących elementów środowisk obliczeniowych do systemu Bricks

Jak wspomniano powyżej, system *Bricks* pozwala na wcielanie zewnętrznych komponentów, włączając istniejące elementy środowisk obliczeniowych, pozwalając na ich ocenę i testowanie w symulowanych, powtarzalnych warunkach. Cel ten jest osiągany głównie poprzez zamianę modułów jednostki szeregującej i używanie interfejsów zewnętrznych modułów (ang. *SPIs*) dla przekazywania im i otrzymywania od nich rozmaitych informacji o szeregowaniu, takich jak wartości mierzone przez monitory itd.

Interfejs dostarczyciela usługi jednostki szeregującej

Każdy element jednostki szeregującej może być zamieniony na inny, napisany w języku *Java* element, implementujący interfejs SPI.

1.2.8. Dążenie systemu Ninf do architektury GRID

W roku 2000 grupa autorów systemu Ninf opublikowała dokument [7]. Poniżej przedstawione zostaną, omówione w dokumencie, podstawowe (zdaniem autorów systemu *Ninf*) zagadnienia projektowe dotyczące systemów obliczeniowych dostosowanych do architektury *Grid*, a także rozpatrywane w tej publikacji możliwe decyzje i ich implikacje, szczególnie te, które dotyczą metod i protokołów komunikacyjnych stosowanych w systemie a także mechanizmów bezpieczeństwa.

1.2.8.1. Pojęcie serwera sieciowego

Serwer sieciowy, otwarty na sieć (ang. *Network Enabled Server*, w skrócie NES) jest ważnym elementem infrastruktury obliczeniowej *Grid*. Jest systemem *Grid* działającym w oparciu o schemat RPC, gdzie klienci żądają wykonania pewnych zadań przez serwer. NES dostarcza łatwego w użyciu, intuicyjnego interfejsu, co pozwala potencjalnym użytkownikom systemu na łatwe przystosowanie ich aplikacji do używania struktur *Grid*. Dlatego właśnie autorzy wspomnianego dokumentu uznają model NES za „ważną abstrakcję pewnej warstwy niskopoziomowego segmentu usług *Grid* takich jak *Globus* czy *Legion*.”

W drodze projektowania, implementacji i użytkowania systemu Ninf, autorzy zdobyli doświadczenie i wiedzę o ważnych aspektach systemów NES, które odróżniają te systemy od konwencjonalnych systemów RPC takich jak np. CORBA, a także uzyskali znajomość rozmaitych zagadnień związanych z projektowaniem tego typu systemów. Z tych powodów autor pracy magisterskiej zdecydował się przytoczyć obszerne fragmenty pracy [7], również

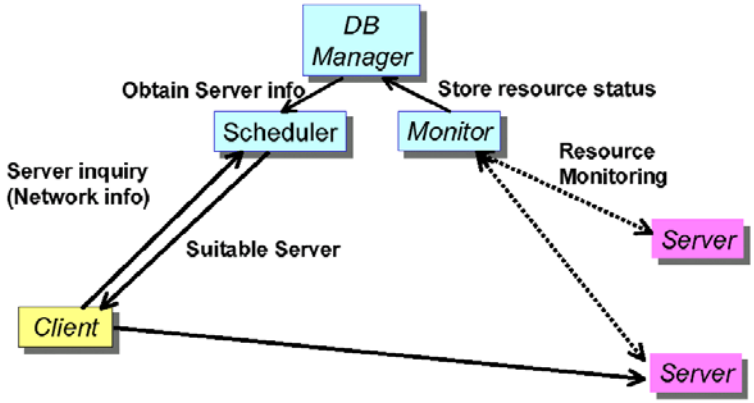
w części dotyczącej decyzji projektowych (i motywacji tych decyzji) podjętych przy projektowaniu drugiej wersji systemu *Ninf*.

1.2.8.2. Podstawy systemów NES

System NES składa się z następujących części:

- klienci (ang. *clients*) – żądają od serwera wykonania dostosowanych do *Grid* (ang. *grid-enabled*) funkcji bibliotecznych i/lub aplikacji;
- serwery (ang. *servers*) – odbierają żądania od klientów i wykonują funkcje biblioteczne lub aplikacja dla klientów;
- jednostka szeregująca (ang. *scheduler*) – wybiera serwer do wykonania zadania, zgodnie z informacją otrzymaną z bazy danych o zasobach;
- monitory (ang. *monitors*) – monitorują stan rozmaitych zasobów, takich jak zasoby obliczeniowe, komunikacyjne etc. i rejestrują wyniki obserwacji w bazie danych o zasobach;
- baza danych o zasobach (ang. *resource database*) – składuje i utrzymuje informacje o stanie monitorowanych zasobów.

Użytkownicy systemu *Grid* modyfikują swoje aplikacje tak, by używały serwerów obliczeniowych za pomocą interfejsów programistycznych klienta (ang. *client APIs*) lub narzędzi na bazie tych API skonstruowanych.



Rysunek 1.4. Schemat systemu NES

Źródło: praca [7]

Klient zapytuje jednostkę szeregującą o odpowiedni dla zadania serwer. Jednostka szeregująca, z kolei, pozyskuje informacje o zasobach, wybiera odpowiedni serwer zgodnie z pewnym algorytmem szeregowania i zwraca wyniki decyzji klientowi. Wtedy klient zdalnie

wywołuje funkcję biblioteczną lub aplikację na wytypowanym serwerze, wysyłając odpowiednie dane wejściowe zadania (argumenty). Serwer przeprowadza obliczenia i zwraca wyniki do klienta (patrz rysunek 1.4).

1.2.8.3. Zagadnienia projektowe

Jest kilka zagadnień projektowych dotyczących konstrukcji systemu NES, włączając metody łączenia klientów i serwerów, protokoły komunikacyjne oraz bezpieczeństwo. Ważny jest również problem uczynienia systemu otwartym na przyszłe rozszerzenia.

Metody łączenia klientów i serwerów

Pierwszym problemem jest zagadnienie, czy połączenia klient-serwer powinny być ustanawiane na cały czas obliczeń, czy powinny być wykonywane na żądanie, tylko wtedy, gdy następuje rzeczywista wymiana informacji. Metoda połączenia ciągłego jest często stosowana w typowych realizacjach mechanizmu RPC. Jest łatwa do implementacji przy użyciu interfejsu gniazd protokołu TCP/IP (ang. *TCP/IP sockets*). Zaletą tej metody jest również fakt, że ewentualne awarie i błędy serwerów są łatwo wykrywalne, objawiając się rozłączeniem strumienia. Wadą tego podejścia jest ograniczenie maksymalnej liczby zdalnych zadań, które mogą być równocześnie wykonywane, spowodowane, typowym dla większości systemów operacyjnych, ograniczeniem na ilość otwartych przez pojedynczy proces deskryptorów plików. To ograniczenie nie było problemem w przypadku tradycyjnych systemów RPC, ponieważ większość transakcji była krótkotrwała i/lub liczba połączeń była mała z powodu faktu, że zadania użytkowników były długie. Stosowanie metody stałych (trwałych) połączeń pociąga za sobą wymaganie, by klienci cały czas byli *on-line* i nie dopuszcza przerwy w komunikacji. Z tego powodu klienci nie mogą rozłączyć się z systemem ani umyślnie ani z powodu przypadku, awarii. Nawet chwilowy błąd w komunikacji (taki, którego nie jest w stanie „przetrwać" połączenie TCP/IP) powodowałoby przerwę w obliczeniach. To zjawisko również jest poważnym ograniczeniem, ponieważ niektóre funkcje bibliotek obliczeniowych włączonych do systemów *Grid* mogą wykonywać się bardzo długo (godzinami lub nawet całymi dniami).

Dla odmiany, przyjęcie modelu połączeń w razie potrzeby (ang. *connection-by-necessity*), polegającej na tym, że klient wysyła żądanie do serwera i rozłącza się natychmiast po przesłaniu argumentów zadania a ponowne połączenie nawiązywane jest przez serwer po zakończeniu obliczeń w celu zwrócenia wyników, omija ograniczenia metody ciągłych połączeń, jednak pociąga za sobą następujące negatywne zjawiska:

- protokół staje się bardziej skomplikowany z powodu konieczności ponownego ustanawiania (bezpiecznego, co kosztuje) połączenia;
- zachodzi potrzeba stosowania dodatkowych metod wykrywania awarii serwera;
- może ucierpieć ogólna wydajność systemu z uwagi na wysoki koszt nawiązywania połączeń.

Drugim ważnym zagadnieniem projektowym dotyczącym komunikacji klient-serwer w systemie NES jest, czy łączyć klienta i serwer bezpośrednio czy, dla różnych celów (utrzymywanie połączenia, monitorowanie wydajności, obchodzenie systemów *firewall*) stosować dedykowanych agentów *proxy*. Dla przykładu „stary" (do wersji 1.2) system *Ninf* dokonywał połączeń za pośrednictwem agentów *proxy*. Dzięki temu, że cały ruch sterowany i negocjowany był przez *proxy-agenta* (w rzeczywistości komunikacja z modułem szeregującym dla wyboru serwera była przeprowadzana przez agenta *proxy* a nie przez samego klienta) możliwe było uproszczenie bibliotek samego klienta. Z drugiej strony, kierowanie komunikacji przez pośrednika, szczególnie w systemach *Grid*, gdzie dane dużych rozmiarów nie są niczym dziwnym, może skutkować obniżeniem wydajności.

Protokoły komunikacyjne dla mechanizmu RPC systemów Grid

Ogólnie rzecz ujmując, protokoły komunikacyjne dzielą się na takie, w których składowe protokołu mają format binarny lub format tekstowy. Formaty binarne pozwalają na łatwą analizę składniową (ang. *parsing*) sekwencji komend ale są trudne do strukturalizacji, analizy i rozszerzania. Natomiast dobrze zaprojektowane formaty tekstowe mogą być dobrze strukturalizowane, łatwe do zrozumienia i rozszerzenia, ale są mniej wydajne i wymagają większego nakładu na programową analizę składniową.

Jest tendencja do stosowania języka XML dla definiowania protokołów tekstowych. Mimo że język ten wymaga większych nakładów (po stronie oprogramowania) na analizę składniową, ma one wiele zalet. Możliwe jest np. proste i przejrzyste definiowanie składowych protokołu przy użyciu DTD. Poza tym, koszt poniesiony z powodu konieczności transmisji dodatkowych (w stosunku do samych danych) jednostek języka, może być zamortyzowany przez, relatywnie duże, rozmiary przesyłanych danych.

Mechanizmy bezpieczeństwa

Bezpieczeństwo jest istotnym zagadnieniem systemów *Grid*. Przy założeniu globalnego wykorzystywania takich systemów, muszą zostać zapewnione środki chroniące przed złośliwymi czy niepowołanymi klientami. Dodatkowo, z powodu faktu, że duże systemy obliczeniowe wykorzystywane są do badań naukowych, musi być zachowany odpowiedni poziom poufności danych. Dla autoryzacji klientów można użyć rozmaitych technik, takich jak *Kerberos* czy SSL (ang. *Secure Sockets Layer*).

Otwartość systemu i zdolność do współpracy z innymi systemami Grid

Ważną kwestią projektową jest decyzja, na ile system ma być otwarty, szczególnie na inne, bardziej ogólne elementy infrastruktury *Grid*, a także na jakim poziomie można będzie ten system konfigurować i dostosowywać do specyficznych wymagań.

Poprzez używanie istniejących podsystemów i komponentów, możliwe jest bezpośrednie wykorzystanie funkcjonalności, które były wypróbowane i przetestowane i podlegają ciągłym ulepszeniom. Z drugiej strony, ponieważ otwarte systemy są projektowane na wyższym poziomie ogólności są bardziej „toporne" i mogą być trudne do zarządzania. Co więcej, obsługiwane platformy sprzętowe i programowe będą częścią wspólną tych, które obsługiwane są przez poszczególne, indywidualne podsystemy.

1.2.8.4. Projekt i szczegóły implementacyjne systemu Ninf w wersji drugiej

Powyżej zaprezentowano pewne ogólne zagadnienia projektowe związane z systemami obliczeniowymi opartymi o mechanizm RPC. Poniżej, na bazie opisu systemu *Ninf* w wersji drugiej zamieszczonego w pracy [7], przedstawione zostaną przykładowe decyzje projektowe systemu, ich motywacje i implikacje. Autor pracy magisterskiej uznał, że ponieważ decyzje te podjęte zostały przez doświadczonych w projektowaniu i użytkowaniu systemów obliczeniowych ludzi, ciekawe i pouczające będzie ich przytoczenie.

Przegląd decyzji projektowych

Podstawowym założeniem, jest to, by projektowany system był elastyczny i rozszerzalny i posiadał zdolność do współpracy z istniejącymi podsystemami internetowymi i *Grid*.

Ponieważ, zdaniem projektantów systemu, w systemie NES z reguły uruchamiane są zadania, które są złożone obliczeniowo, decyzje projektowe powinny przede wszystkim dawać systemowi ową otwartość i elastyczność, mniejszą wagę przywiązując do narzutów komunikacyjnych (oczywiście powinny ograniczać je na tyle, by mogły być amortyzowane

przez inne korzyści). To podstawowe założenie ma skutki w dalszych, szczegółowych decyzjach.

Połączenia klient-serwer. W celu pomieszczenia wielu, odpornych na błędy, długotrwałych wywołań funkcji, które są charakterystyczne dla środowiska *Grid*, przyjęto model łączenia w razie potrzeby. Poza tym zdecydowano się, dla utrzymania prostoty modułów programowych klienta, na dokonywanie połączeń poprzez agenta *proxy*, jednakże, w celu uniknięcia wąskich gardeł, agenci *proxy* jedynie pośredniczą w przesyle komend pomiędzy klientem i serwerem; argumenty zdalnych wywołań są transmitowane bezpośrednio, chyba że zachodzi potrzeba ominięcia systemu *firewall*.

Komendy protokołu komunikacyjnego. Dążąc do elastyczności, rozszerzalności i otwartości systemu, zdecydowano się na stosowanie komend tekstowych opartych o XML. Za tym wyborem przemawiał dodatkowo fakt, że dostępne są darmowe analizatory składniowe (ang. *parsers*) dla języków *C* i *Java*, które upraszczają i ułatwiają implementację systemu.

Mechanizmy bezpieczeństwa. Dla umożliwienia systemowi Ninf działania w globalnym środowisku *Grid*, zdecydowano się stworzyć, podobną do stosowanej w systemie *Globus*, opartej o SSL warstwę autoryzacji (ang. *authentication and authorization*), która pozwala na delegacje autoryzacji wzdłuż tzw. łańcucha bezpieczeństwa (ang. *security chain*). Na SSL zdecydowano się dlatego, że technologia stała się komercyjnym standardem i istnieje wiele darmowych implementacji bibliotek dla *C* i języka *Java*.

Otwartość systemu i zdolność współpracy z innymi systemami *Grid*. Ten zakres zagadnień był, zdaniem autorów systemu *Ninf*, obszarem, w którym decyzje były najtrudniejsze, ponieważ zalety i wady poszczególnych rozwiązań są niejednoznaczne.

Jako kompromis uznano dostarczenie domyślnej implementacji wszystkich podstawowym modułów systemu Ninf, z jednoczesnym zapewnieniem dobrze zdefiniowanych interfejsów tak, by moduły te mogły być łączone z elementami istniejących środowisk operacyjnych. Dla przykładu, mimo że domyślnie zarządca bazy danych zasobów realizuję usługę LDAP do przeglądania informacji o zasobach, może on również bezpośrednio używać usług MDS systemu *Globus*, jeśli w danym środowisku obliczeniowym są one dostępne.

Przegląd nowych cech systemu Ninf

Poniżej przedstawione zostaną nowe cechy systemu *Ninf* w wersji drugiej, ze szczególnym uwzględnieniem tych, które bezpośrednio wynikają z omówionych wcześniej decyzji projektowych. System *Ninf*, jako taki został omówiony w poprzednich punktach, dlatego w tym miejscu uwaga skupiać się będzie na tym, co nowe i ciekawe z punktu widzenia projektowania systemów w architekturze *Grid* w ogólności.

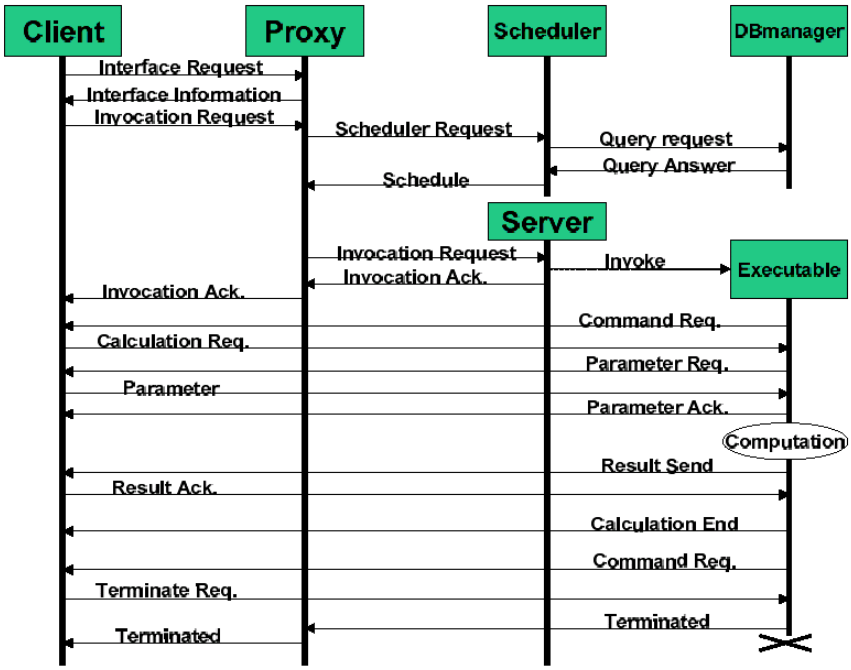
W stosunku do wersji poprzednich zaszła pewna zmiana w spojrzeniu na system obliczeniowy. Wyróżniono nowe elementy:

- składnicę danych (ang. *data storage*) – jest ona miejscem tymczasowego składowania pośrednich rezultatów obliczeń (np. tych, które wymieniane są pomiędzy serwerami);
- zarządcę awarii (ang. *fault manager*) - przeprowadza odpowiednie akcje w sytuacji wystąpienia awarii, które mają wpływ na funkcjonowanie systemu *Ninf*, np. jeśli stwierdza, że serwer obliczeniowy uległ awarii, usuwa ten serwer z bazy danych zasobów.

Niektórym elementom dodano również nowe możliwości. Np., jak wcześniej wspomniano, możliwa jest współpraca zarządcy bazy danych o zasobach z istniejącą w danym środowisku infrastrukturą baz danych (dzięki wykorzystaniu technologii LDAP).

Nowości pojawiły się również w mechanizmach szeregowania. Moduł szeregujący, przy wyborze odpowiedniego dla danego zadania serwera, bierze pod uwagę nie tylko informacje o zasobach takich jak stan (obciążenie) serwerów obliczeniowych czy sieci, ale także o lokalizacji plików używanych w obliczeniach. Nowa jest również większa otwartość modułu szeregującego, w którym można wymienić algorytm.

Rozszerzono również sam protokół związany z uruchamianiem zdalnego zadania. Każde przesłanie informacji (argumentów, wyników lub informacji o zdarzeniu, np. zakończeniu obliczeń) jest potwierdzane przez odbiorcę odpowiednim komunikatem. Schemat przepływu komunikatów w nowym systemie *Ninf* przedstawia rysunek 1.5.



Rysunek 1.5. Schemat przepływu komunikatów w nowym systemie *Ninf*

Źródło: praca [7]

Warstwa bezpieczeństwa

Duże zmiany zaszły w systemie *Ninf* szczególnie w kwestii infrastruktury bezpieczeństwa. Powstał podsystem NAA (ang. *NES Authentication Authorization* – autoryzacja dla NES). NAA używa mechanizmów technologii SSL (dokładniej certyfikatów i podpisów cyfrowych) i obsługuje delegację identyfikacji i autoryzacji za pomocą mechanizmu tzw. łańcucha certyfikacji (ang. *certificate chain*). Delegacja autoryzacji odbywa się automatycznie, klient musi jedynie określić swój certyfikat cyfrowy. Stosowane mechanizmy zapewniają z jednej strony autoryzację samego klienta wobec danego serwera, z drugiej potwierdzenie, że dany serwer, występujący do innego serwera z żądaniem wykonania pewnej usługi rzeczywiście reprezentuje określonego klienta.

Po potwierdzeniu tożsamości klienta (ang. *authentication*) dokonywana jest przez NAA autoryzacja, czyli stwierdzenie, jakie uprawnienia ma dany klient w systemie. Ten proces opiera się na koncepcji tzw. klas polityk (ang. *policy class*) stosowanych w języku *Java*. Polityka jest zbiorem struktur, które są grantami, które z kolei są zbiorami praw, zezwoleń (ang. *permissions*) dla klienta. Biblioteka NAA zarządza strukturami polityk i identyfikuje użytkowników w systemie. Przestrzeń nazw NAA ma strukturę drzewa zgodnie ze standardem X.509. Serwer przegląda bazę zezwoleń sprawdzając, czy pewne zezwolenie

może być zastosowane do danego klienta. Biblioteka sprawdza politykę, czy istnieją granty które zawierają poszczególne zezwolenia. Każde zezwolenie składa się z atrybutów, klasy, celu i akcji. Klasa wskazuje operację, którą może wykonać klient. Cel i akcja wyznacza przedmiot operacji wraz z typem operacji do wykonania. W systemie *Ninf* polityki są opisane przy użyciu języka XML. Dla przykładu przestawiony zostanie plik DTD dla polityk bezpieczeństwa (rysunek 1.6) oraz przykładowy opis polityki (rysunek 1.7).

```
<!ELEMENT policy (grant)*>
<!ELEMENT grant (permission)*>
<!ATTLIST grant userid CDATA #REQUIRED>
<!ELEMENT permission EMPTY>
<!ATTLIST permission class CDATA #REQUIRED>
<!ATTLIST permission target CDATA #REQUIRED>
<!ATTLIST permission action CDATA #REQUIRED>
```

Rysunek 1.6. DTD dla polityk bezpieczeństwa w systemie Ninf

Źródło: praca [7]

```
<policy>
  <grant userid="c=jp,o=etl">
    <permission class = "stubexec"
        target = "test/entry0" action="100 20"/>
    <permission class = "stubexec"
        target = "test/entry1" action="100 20"/>
  </grant>
  <grant userid=H c=jp,o=etl,CN=nakada">
    <permission class = "stubexec"
        target = "test/entry3" action="100 20"/>
  </grant>
</policy>
```

Rysunek 1.7. Opis polityki bezpieczeństwa w systemie Ninf

Źródło: praca [7]

Drzewiasty układ struktur związanych z bezpieczeństwem pozwala na kontrolę dostępu do zasobów na różnych poziomach „ziarnistości” (ang. *fine-grained*). Pierwsze doświadczenia z mechanizmami opartymi o te struktury pokazały, że nie wnoszą one znaczącego narzutu i nie powodują dużego obniżenia wydajności, tak długo jak wielkość ziarna jest wystarczająco duża, by narzut mógł być zamortyzowany (inaczej mówiąc: jak długo operacje, które wymagają autoryzacji trwają odpowiednio długo – zdaniem autorów powyżej 10 sekund – tak długo narzut czasowy na autoryzację jest kompensowany).

1.3. System NetSolve

System *NetSolve* rozwijany jest na Uniwersytecie Tennessee w Stanach Zjednoczonych.

1.3.1. Koncepcja

System realizuje, podobnie jak *Ninf* ideę serwera sieciowego (ang. *Network-enabled Server*). Struktura dwóch systemów, amerykańskiego i japońskiego jest bardzo podobna, podobny jest również ogólny schemat ich działania. Zostało to, zresztą wyrażone przez autorów systemu *Ninf* słowami: „Istnieją dwa główne środowiska przetwarzania funkcjonalnego. Są nimi *NetSolve* i *Ninf*".

Prezentując niektóre rozwiązania techniczne systemu *NetSolve*, dla wykazania ich specyfiki, autor porównywać będzie je z tymi, które zostały przyjęte w systemie *Ninf*. Również problemy i zagadnienia sygnalizowane przez projektantów systemu *NetSolve* zostaną odniesione do systemu *Ninf*.

1.3.2. Architektura

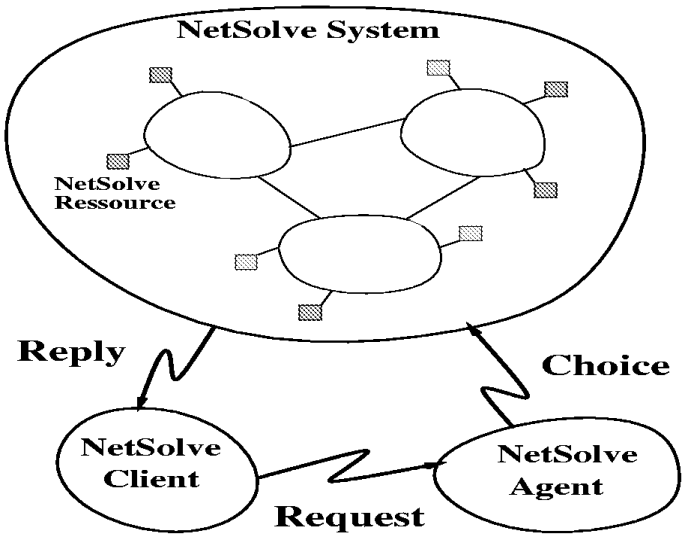
W tym punkcie przedstawione zostaną podstawowe elementy i mechanizmy systemu *NetSolve*.

1.3.2.1. Struktura systemu

System *NetSolve* jest zbiorem luźno powiązanych maszyn, które mogą znajdować się w tej samej, lokalnej sieci lub w sieci rozległej. System *NetSolve* może działać w heterogenicznych środowiskach.

Aktualne implementacje widzą system jako pełny graf bez hierarchicznej struktury. Istnieje pomysł utworzenia zbioru niezależny systemów *NetSolve* w różnych lokalizacjach, być może zapewniających odmienne usługi. Użytkownik mógłby wtedy kontaktować się z wybranym systemem, w zależności od tego, jakiej usługi potrzebuje. W celu efektywnego zarządzania pulą serwerów rozsianych w rozległej sieci, przyszłe implementacje będą zapewniały większą strukturalizację (np. strukturę drzewa), która ograniczy i pogrupuje komunikacje i interakcje.

Rysunek 1.8 obrazuje koncepcję systemu *NetSolve*. Na rysunku klient *NetSolve* wysyła żądanie do agenta *NetSolve*. Agent wybiera najlepsze zasoby systemu *NetSolve* zależnie od rozmiaru i natury problemu obliczeniowego. W sieci może istnieć wiele instancji agentów *NetSolve*. Dobrą, zdaniem autorów, strategią jest posiadanie w każdej lokalizacji, w której są klienci systemu *NetSolve*, jednego agenta.



Rysunek 1.8. Struktura systemu NetSolve

Źródło: praca [8]

Na każdej maszynie systemu *NetSolve* uruchomiony jest serwer obliczeniowych (ang. *computational server*), zwany również zasobem (ang. *resource*). Serwer obliczeniowy ma dostęp do pakietów naukowych (bibliotek lub niezależnych systemów).

Ważnym aspektem systemu opartego o serwery jest fakt, że każda instancja agenta ma swój własny ogląd (ang. *view*) systemu. Dlatego niektórzy agenci mogą znać więcej szczegółów niż inni, w zależności od ich lokalizacji. W końcu jednak system osiąga stan stabilny, w którym każda instancja agenta posiada taką samą informację o systemie.

1.3.2.2. Protokoły komunikacyjne

Komunikacja wewnątrz systemu *NetSolve* odbywa się poprzez protokół TCP/IP. Zdaniem projektantów systemu fakt, że proces jest ograniczony do pewnej liczby równoczesnych połączeń nie jest problemem.

Dla zapewnienia prawidłowego działania w środowisku heterogenicznym, do wymiany danych pomiędzy maszynami o niekompatybilnej reprezentacji danych *NetSolve* używa protokołu *Sun XDR*. Nie jest on używany tam, gdzie jest to zbędne, ponieważ przy transmisji dużych wolumenów zachodziłaby konieczność konwersji dużych jednostek danych, co wnosiłoby niepotrzebne narzuty czasowe i obciążało zasoby.

1.3.2.3. Zarządzanie systemem

Jak wspomniano wyżej, każdy serwer *NetSolve* (czyli instancja agenta lub serwer obliczeniowy) jest niezależną jednostką. System może być dzięki temu modyfikowany bez narażania na szwank jego integralności. Dowolny serwer *NetSolve* może być usunięty lub utworzony w dowolnym momencie.

Zarządzanie takim systemem nie jest jednak łatwe. Potrzebne są pewne narzędzia, by możliwe było uzyskanie zcentralizowanego widoku. Autorzy systemu stworzyli, umieszczane na stronie *www* skrypty CGI, pobierające, jako dane wejściowe, lokalizację agenta (nazwę maszyny na której działa agent) w celu identyfikacji systemu *NetSolve*, który ma zostać sprawdzony. Jeden ze skryptów podaje listę instancji agentów lub zasobów obliczeniowych dostępnych w systemie (lista nazw maszyn i adresów IP). Drugi pokazuje listę problemów (zadań), które wewnątrz danego systemu *NetSolve* można rozwiązać.

1.3.3. Specyfikacja problemu i zarządzanie serwerem

W tym punkcie zdefiniowane zostanie pojęcie problemu systemu *NetSolve* a także opisana zostanie procedura konfiguracji, kompilacji i uruchamiania zasobu obliczeniowego wewnątrz systemu *NetSolve*.

1.3.3.1. Specyfikacja problemu

Autorzy systemu dostarczyli formalną metodę opisu problemu.

Problem jest definiowany jako krotka <*name*, *inputs*, *outputs*>, gdzie:

- *name* jest ciągiem znaków zawierającym nazwę problemu,
- *inputs* jest listą obiektów wejściowych,
- *outputs* jest listą obiektów wyjściowych.

Obiekt, z kolei jest opisany jako: <*object*, *data*>, gdzie:

- *object* może być: ‘MATRIX’, ‘VECTOR’ or ‘SCALAR’,
- *data* może być dowolnym typem danych języka *Fortran*.

Ta definicja, zdaniem autorów systemu, jest wystarczająca dla wymiany danych z rozmaitymi pakietami oprogramowana.

W idealnym przypadku, było by pożądane, by użytkownicy, którzy używają już pakietów oprogramowania naukowego w swoich programach napisanych w językach *C* lub *Fortran*, byli w stanie, bez modyfikacji kodu przejść na używanie systemu *NetSolve*. Z tego punktu widzenia, przedstawiony powyżej opis problemu nie jest wystarczający. Potrzebna jest

również definicja tzw. formatu problemu, która polega na opisaniu sekwencji wołania wykonywanego z języka *C* lub *Fortran* do systemu *NetSolve*. W systemie *NetSolve* problem może mieć różne sekwencje wołania i użytkownicy mogą wybierać między nimi.

1.3.3.2. Tworzenie serwera

Jednym z założeń projektowych systemu *NetSolve* było zapewnienie mu rozległego zakresu zastosowania. Stąd wynikała potrzeba zapewnienia mu możliwości dodawania nowych problemów do serwera obliczeniowego. Uznano za nie akceptowalną konieczność modyfikacji kodu samego serwera *NetSolve* dla dodania do niego nowego problemu.

Zdaniem autorów systemu, możliwe były dwa rozwiązania: uruchamianie programów w postaci wykonywalnej lub jawne wywołanie programów naukowych z kodu serwera. Pierwsze rozwiązanie wydaje się projektantom systemu *NetSolve* łatwiejsze do zaimplementowania: serwer obliczeniowy może mieć dostęp do katalogu zawierającego wszystkie programy w postaci wykonywalnej dla każdego obsługiwanego problemu. Jednakże to rozwiązanie, ich zdaniem, zdaje się mieć następujące wady: po pierwsze, zarządzanie takim katalogiem może nie być łatwe w środowisku rozproszonego systemu plików lub w przypadku, gdy część serwerów chce zapewnić dostęp jedynie do części swoich problemów obliczeniowych. Po drugie, taki kształt systemu wymagałby istnienia oddzielnego pliku wykonywalnego dla każdego problemu obliczeniowego. Co więcej, ponieważ większość pakietów programowych, które aktualnie współpracują z *NetSolve* mają postać biblioteki procedur, stosowanie rozwiązania, w którym serwery obliczeniowe oddzielnie wywołują programy wykonywalne, wydaje się nadmiarowe.

Dlatego zdecydowano się na przyjęcie drugiego podejścia, w którym serwery bezpośrednio wywołują podległe im oprogramowanie.

Autorzy *NetSolve* stworzyli narzędzie wspomagające generację kodu dla tego podejścia. Program ten, pobiera jako dane wejściowe plik konfiguracyjny opisujący każdy problem obliczeniowy w formalny sposób i generuje kod w języku *C* dla procesu obliczeniowego odpowiedzialnego za rozwiązywanie problemu. Dzięki temu nowe problemy mogą być dodawane do *NetSolve*, bez konieczności martwienia się o wewnętrzne struktury danych systemu.

W pierwszej wersji pseudo-kompilator wymaga pewnego wysiłku od administratora systemu *NetSolve*. Wynika to z faktu, że z założenia, ma istnieć możliwość wcielania do systemu *NetSolve* dowolnych pakietów oprogramowania. Jednakże system dostarcza administratorowi

techniki dostępu do parametrów problemu. W szczególności wołania funkcji bibliotecznych muszą być napisane w języku *C* z użyciem predefiniowanego zbioru makr.

1.3.3.3. Język opisu problemów

Do opisu każdej, oddzielnej numerycznej funkcjonalności serwera obliczeniowego został stworzony język opisowy (ang. *descriptive language*). Jak wspomniano, stworzony w tym języku opis problemu poddawany jest kompilacji do modułów obliczeniowych wykonywalnych na platformie *Unix* lub NT. Te pliki są faktycznie tzw. *wrapperami* (ang. *wrapper* – opakowanie, obwoluta) *NetSolve* do bibliotek naukowych. Opis problemu zawiera informacje takie jak: jaka biblioteka numeryczna ma być używana do jego rozwiązywania, jakie są oczekiwane dane wejściowe i jakie są spodziewane dane wyjściowe dla zadania. Plik opisowy musi również zawierać przybliżone określenie złożoności obliczeniowej algorytmu w funkcji rozmiaru problemu. Takie oszacowanie może być dobrze znane (np. dla algorytmów dokładnych, skończonych – ang. *direct solvers*) lub może zależeć od danych wejściowych (dla algorytmów iteracyjnych – ang. *iterative solvers*). Drugi przypadek jest „wciąż otwartą kwestią w *NetSolve*", jak twierdzą autorzy.

Jako wsparcie dla procesu tworzenia opisów problemów projektanci systemu *NetSolve* stworzyli narzędzie z graficznym interfejsem użytkownika, wspomagające takie czynności szczególnie przy opisie złożonych problemów.

Planowane jest także stworzenie repozytorium opisów problemów na stronie *www*, tak by można je było łatwo uzyskać i użyć do konfiguracji dowolnych serwerów obliczeniowych.

Rejestracja zasobów obliczeniowych w systemie

Agenci systemu *NetSolve*, pośród różnych pełnionych przez nich funkcji, śledzą, jakie zasoby obliczeniowe są dostępne na serwerach obliczeniowych i gdzie dokładnie się znajdują. Agenci utrzymują bazę danych informacji o różnych funkcjach obliczeniowych udostępnionych na poszczególnych maszynach użytkownikom systemu *NetSolve*. Baza danych jest automatycznie aktualizowana podczas włączania systemów obliczeniowych do systemu *NetSolve* (jak również przy ich wyłączaniu). Zawsze, gdy uruchamiany jest nowy serwer, wysyła on komunikat do agenta, który zawiera ogólne informacje o serwerze (włączając jego lokalizację) i listę funkcji numerycznych, którą chce on wnieść do systemu. Po tej czynności funkcje serwera są dostępne dla użytkowników.

1.3.4. Interfejsy klienta

Jednym z celem systemu *NetSolve* było dostarczenie klientowi wielu interfejsów, prostych na tyle, na ile to możliwe. I tak, system posiada interaktywne interfejsy dla systemu MATLAB, powłoki systemowej (ang. *shell interface*) a także system oparty o graficzny interfejs użytkownika umieszczony na stronie *www*, działający w technologii *Java* (ang. *Web-based Java GUI*). Poza tym, dostępne są interfejsy programistyczne dla języków *Fortran*, *C* i *Java*. W przeciwieństwie do interfejsów interaktywnych, stosowanie interfejsów programistyczny wymaga pewnego programistycznego wysiłku od użytkownika. Jednakże biblioteki systemu *NetSolve* dostarczają do tego celu kilku funkcji, których używanie nie jest zbyt trudne. Przykłady interfejsów dla języków *C* i *Fortran* przedstawiają poniższy rysunki (1.9 i 1.10).

```
double A[100*100];                /* Matrix A                          */
double Real[100],Imaginary[100]; /* real and imaginary parts of A's    */
                                  /* eigenvalues                       */
int request;                      /* NetSolve request number           */
int is_finished;                  /* Flag giving the computation status */

/* Blocking call        */
request = netsl("eig",              /* Eigenvalues problem             */
                A,100,              /* One matrix in input : A 100x100 */
                Real,Imaginary);    /* Two vectors in output :         */
                                    /*   Real and Imaginary            */

/* Asynchronous call    */
request = netslnb("eig",A,100,
                     Real,Imaginary);

...    Some computations

is_finished = netslpb(request);  /* Probing */

...    Some computations

is_finished = netslwt(request);  /* Waiting */
```

Rysunek 1.9. Intefejs programistyczny systemu NetSolve dla języka C

Źródło: praca [8]

Zarówno interfejsy interaktywne, jak i programistyczne zapewniają możliwość synchronicznego i asynchronicznego wywoływania funkcji obliczeniowych. Dzięki asynchronicznemu interfejsowi wołań możliwe jest jawne wymuszanie przez użytkownika współbieżne wykonywanie funkcji obliczeniowych na różnych zasobach.

```
      INTEGER LDA,N
      PARAMETER(LDA = 100, N = 100)
      DOUBLE PRECISION A(LDA,N), R(N),I(N)
      INTEGER REQUEST,ISREADY

      * Blocking Call
      CALL NETSL('eig',REQUEST,
     $             A,LDA,N,R,I)

      * Asynchronous  Call
      CALL NETSLNB('eig',REQUEST,
     $               A,LDA,N,R,I)

...    Some computations

      * Probing *
      CALL NETSLPB(REQUEST,ISREADY)

...    Some computations

      * Waiting *
      CALL NETSLWT(REQUEST,ISREADY)
```

Rysunek 1.10. Interfejs programistyczny systemu *NetSolve* dla języka *Fortran*

Źródło: praca [8]

1.3.5. Szeregowanie w systemie NetSolve

Podczas uruchamiania zadania w systemie *NetSolve* jedna z maszyn pracujących pod kontrolą tego systemu jest wybierana jako najbardziej odpowiednia dla rozwiązania danego problemu. Poniżej zostaną przedstawione kryteria brane w procesie przydziału zasobów obliczeniowych pod uwagę.

1.3.5.1. Obliczanie „najlepszej maszyny”

Hipotetyczną najlepszą maszyną jest ta, która osiąga najkrótszy czas wykonania zadania *T* dla danego problemu *P*. Zatem zachodzi potrzeba estymacji tego czasu na każdej maszynie *M* systemu *NetSolve*.

Czas T jest dzielony na T_n i T_c, gdzie:

- T_n jest czasem potrzebnym na wysłanie danych do M i odebranie wyniku poprzez sieć,
- T_c jest czasem wykonywania obliczeń.

Czas transmisji T_n może być obliczony przy znajomości następujących elementów:

- opóźnienie sieci i pasmo pomiędzy lokalną maszyną a maszyną *M*,
- rozmiar danych do wysłania,
- rozmiar wyników do odebrania.

Obliczenie czasu obliczeń T_c wymaga znajomości:

- rozmiaru problemu,
- złożoności obliczeniowej problemu,
- wydajności *M*, która z kolei zależy od obciążenia maszyny *M* i jej „surowej" wydajności (ang. *raw perfomance*).

1.3.5.2. Model wydajności

Autorzy systemu *NetSolve* skonstruowali prosty, teoretyczny model pozwalający na estymację wydajności przy danej „surowej" wydajności i obciążeniu. Ten model daje estymowaną wydajność *p*, jako funkcję obciążenia *w*, „surowej" wydajności P, oraz liczby procesorów na maszynie, *n*:

$$p = \frac{P \times 100 \times n}{100 \times n + \max(w - 100 \times (n-1), 0)}$$

Model ten został poddany walidacji na drodze kilku eksperymentów. Pokazują one, że model teoretyczny jest bliski wynikom eksperymentów. Model nie uwzględnia jednak narzutów systemu operacyjnego na migrację procesów w przypadku, gdy maszyna, na której wykonywane są obliczenia jest wieloprocesorowa. Natomiast dobrze obrazuje np. fakt, że spadek wydajności obliczeń w takiej konfiguracji sprzętowej w stosunku do „surowej" wydajności maszyny zaczyna się dopiero po obciążeniu wszystkich procesorów.

1.3.5.3. Obliczanie T

Obliczanie *T* odbywa się w instancji agenta dla każdego serwera obliczeniowego *M*, na każde żądanie rozwiązania problemu. To obliczenie używa wszystkich wymienionych wyżej parametrów. Rozróżnia się następujące ich klasy:

- parametry zależne od klienta:
  - rozmiar danych do wysłania,
  - rozmiar wyników do odebrania,
  - rozmiar problemu;
- statyczne parametry, zależne od serwera:
  - charakterystyka sieci pomiędzy lokalną maszyną a serwerem *M*,
  - złożoność algorytmu używanego na maszynie *M*,
  - „surową" wydajność (ang. *raw performance*) *M*;
- dynamiczne parametry serwera:
  - obciążenie maszyny *M*.

Parametry zależne od klienta znajdują się wewnątrz żądania wysyłanego przez klienta agentowi. Ich szacowanie jest więc sprawą bardzo prostą.

Parametry statyczne, zależne od serwera, ogólnie, są oszacowywane jednorazowo, kiedy nowy serwer kontaktuje się z innymi działającymi już serwerami *NetSolve*.

Charakterystyka sieci jednak oceniana jest wielokrotnie, tak aby oszacować sensowną, średnią wartość opóźnienia i pasma, które może zostać w niej uzyskane. Zdaniem autorów systemu *NetSolve* charakterystyka sieci jest parametrem statycznym, ponieważ nie zmienia się ona wielce w stosunku do obliczonej wartości średniej.

Złożoność algorytmu. Kiedy nowy obliczeniowy serwer przyłącza się do systemu *NetSolve*, rozsyła złożoność obliczeniową wszystkich problemów, które chce rozwiązywać. Ta złożoność nie zmienia się w czasie, ponieważ zależy jedynie od oprogramowania używanego przez serwer obliczeniowy.

„Surowa" wydajność. Przez to pojęcie rozumiana jest wydajność maszyny bez innych, dodatkowych procesów używających procesora. Jego wartość jest określana na każdym serwerze obliczeniowym w czasie uruchamiania. Autorzy systemu *NetSolve* używają narzędzia *LINPACK benchmark* dla oceny wskaźnika *Kflop/s*. *LINPACK* oblicza tzw. czas użytkownika (ang. *user time*) uzyskany dla zadania i w ten sposób dostarcza danych o „surowej" wydajności serwera.

1.3.5.4. Model obciążenia

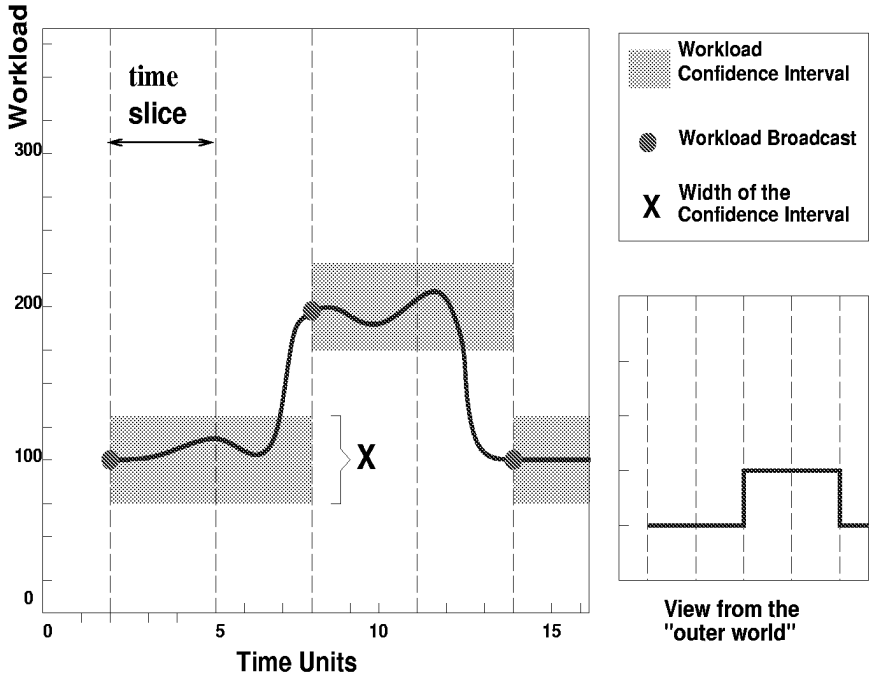
Parametry bieżącego obciążenia są jedynymi parametrami zależnymi od serwera, które są wymagane dla wykonania obliczeń dla predykcji czasu wykonania *T*.

Każda instancja agenta posiada buforowaną wartość obciążenia każdego serwera obliczeniowego. Przez „buforowaną" (ang. *cached*) rozumie się fakt, że wartość ta jest periodycznie aktualizowana i używana dla każdego obliczenia *T* (nie jest pobierana na żądanie). Co prawda wartość ta może być przeterminowana i przez to powodować czasem błędną estymację wartości *T*, niemniej jednak zdecydowano się na ryzyko dokonywania, od czasu do czasu, błędnej estymacji zamiast ponoszenia kosztu ciągłego pobierania dokładnej wartości obciążenia.

Autorzy systemu *NetSolve* dokonali minimalizacji liczby operacji rozgłaszania (ang. *broadcast*) bieżącego obciążenia przez maszyny pracujące w systemie *NetSolve*. I tak, dana maszyna M, rozgłasza swoje obciążenie okresowo (rozsyłana wartość jest średnią obciążenia z poprzedniego okresu). Rozgłaszanie następuje jednak tylko w przypadku, gdy zmiana

obciążenia jest większa niż pewna zdefiniowana wartość (czyli wykraczająca poza pewien przedział ufności).

Zasadę rozgłaszania obciążenia ilustruje poniższy rysunek (1.11).



Rysunek 1.11. Zasada rozgłaszania obciążenia w systemie *NetSolve*

Źródło: praca [8]

Na rysunku (1.11) przedziały ufności oznaczone są szarymi prostokątami, natomiast chwile, w których wykonywane jest rozgłoszenie bieżących wartości obciążenia oznaczone są kropkami.

Wielkość szczeliny czasowej (ang. *time slice*) i szerokość przedziału ufności muszą być odpowiednio dobrane. I tak, zbyt wąska szczelina czasowa powoduje bardzo częste ocenianie obciążenia, co konsumuje czas procesora. Zbyt wąski przedział ufności z kolei, powoduje zbyt częste rozgłaszanie wartości obciążenia. Wartości te powinny więc zostać dobrane na drodze eksperymentalnej lub powinny zostać stworzone narzędzia umożliwiające ich dynamiczne dostosowywanie.

1.3.6. Tolerowanie uszkodzeń

Tolerowanie uszkodzeń jest ważnym zagadnieniem każdego, luźno powiązanego, rozproszonego systemu, jakim jest *NetSolve*. W rozległych, sieciowych środowiskach obliczeniowych czasowa niedostępność niektórych zasobów może wynikać nie tylko z awarii sieci lub maszyn obliczeniowych ale także z planowanych wyłączeń systemów lub ich okresowego przeznaczania do wyłącznego użytku instytucji będących zarządcą, właścicielem lub dzierżawcą systemów. Podatność na awarie i planowana, czasowa niedostępność zasobów wymusza na projektantach systemu *NetSolve* traktowanie wszystkich udostępnionych systemowi zasobów jako przejściowe. Dotyczy to nie tylko zasobów obliczeniowych ale także urządzeń służących do składowania i przechowywania danych (serwery plików itp.). Niedostępność jednego lub większej liczby komponentów systemu nie powinno powodować żadnych katastroficznych błędów. Co więcej, jej skutki powinny być jak najmniejsze, by ograniczyć spadek wydajności systemu *NetSolve*.

W *NetSolve* mechanizmy tolerowania uszkodzeń działają na różnych poziomach. Poniżej zostaną wymienione niektóre, związane z tymi mechanizmami, decyzje projektowe omówione w pracy [9]. Zagadnienia związane z tolerowaniem uszkodzeń (jak również z wyrównywaniem obciążeń systemów) w *NetSolve* rozwinięte zostaną w punkcie 1.3.6.5.

1.3.6.1. Wykrywanie uszkodzeń

Proces systemu *NetSolve* (klient, serwer itd.) wykrywa błędy (awarie sieci, zniknięcie lub uszkodzenie serwera) przez stwierdzenie niemożności połączenia się z którymś z elementów systemu lub zerwania istniejącego połączenia. W takim wypadku proces informuje o awarii agenta *NetSolve*, który bierze to uszkodzenie pod uwagę, tzn. zaznacza w swoich strukturach danych, że serwer jest uszkodzony i niedostępny (ale nie usuwa danych o nim – usuwane są one dopiero po długotrwałej niedostępności maszyny).

Jednym z podstawowych założeń systemu *NetSolve* był fakt, że serwer systemu może być bezpiecznie zatrzymany i uruchomiony w dowolnym momencie. Dlatego, gdy serwer powróci do normalnego działania jest automatycznie „uaktywniany” przez agenta systemu *NetSolve*.

1.3.6.2. Odporność na uszkodzenia

Ważnym aspektem tolerowania uszkodzeń jest fakt, iż wpływ awarii na system powinien być jak najmniejszy. W systemie *NetSolve* wprowadzono mechanizm omijania uszkodzeń, który jest do pewnego stopnia przeźroczysty dla użytkownika (użytkownik nie wie, że wystąpiła awaria, choć może zauważyć pewne opóźnienie wykonania zadania).

Kiedy agent *NetSolve* odbiera od klienta żądania rozwiązania pewnego problemu, odsyła mu listę maszyn, na których problem ten może być wyliczony, posortowaną od najbardziej do najmniej odpowiedniego. Klient „wypróbowuje" kolejne serwery z listy aż do momentu, gdy któryś z nich przyjmie zadanie do rozwiązania. Jeśli jednak po przejrzeniu całej listy okaże się, że żaden z serwerów nie był w stanie przyjąć zadania (np. z powodu ich uszkodzenia czy niedostępności), klient prosi agenta o inną listę serwerów, równocześnie informując agenta o uszkodzonych maszynach, jeżeli takie zostały stwierdzone.

Jednak nawet jeśli zostanie ustanowione połączenie z serwerem obliczeniowym, nie ma gwarancji, że problem zostanie rozwiązany, np. proces obliczeniowy na zdalnej maszynie może zginąć z różnych powodów. W takim przypadku uszkodzenie jest wykrywane przez klienta i problem jest wysyłany do innego dostępnego serwera obliczeniowego (to, oczywiście, jak wspomniano wyżej, wiąże się z pewnym opóźnieniem).

1.3.6.3. Uwzględnianie uszkodzeń

Agenci systemu *NetSolve* śledzą stan (dostępny/niedostępny) serwerów obliczeniowych. Śledzą również stan procesów systemu *NetSolve* (działa, zatrzymany, uszkodzony) na tych maszynach. Kiedy serwer jest niedostępny przez ponad 24 godziny, usuwany jest odpowiadający mu wpis w strukturach danych agenta.

Śledzona przez agentów *NetSolve* jest również liczba błędów stwierdzona przy używaniu poszczególnych serwerów obliczeniowych. I tak, gdy któraś maszyna przekracza pewną wartość progową liczby błędów, jest usuwana z listy serwerów systemu *NetSolve* (umożliwia to eliminację maszyn, na których źle działa serwer obliczeniowy, np. niepoprawnie wykonuje wywołania funkcji bibliotecznych itp.)

1.3.6.4. Planowane mechanizmy dla tolerowania uszkodzeń

Z powodu tego, że wdrożone aktualnie mechanizmy tolerowania uszkodzeń są dość prymitywne, planowane jest (z punktu widzenia pracy [9]) stworzenie bardziej zaawansowanych technik, takich jak mechanizm punktów kontrolnych (ang. *checkpointing*) oraz składowanie danych zadania na wydzielonych serwerach usługi przechowywania danych (ang. *storage servers*).

1.3.6.5. Rozwinięcie tematyki tolerowania uszkodzeń w NetSolve

W poniższym punkcie przedstawione zostaną zaprezentowane w pracy [10] zagadnienia badawcze związane z mechanizmami tolerowania uszkodzeń dla systemu *NetSolve*.

Podstawowym założeniem, branym pod uwagę przy projektowaniu technik tolerowania uszkodzeń i wyrównywania obciążenia, jest fakt, iż mechanizmy te muszą działać w sposób absolutnie przeźroczysty dla użytkownika.

Projektanci systemu *NetSolve*, w pracy [10], dokonali klasyfikacji technik tolerowania uszkodzeń i wyrównywania obciążeń, dzieląc je na dwie kategorie: techniki między-serwerowe (ang. *inter-server*) i wewnątrz-serwerowe (ang. *intra-server*). Dla przykładu, agent może śledzić działania serwerów; jeśli serwer nie dokonuje postępu w obliczeniach w sensownym tempie agent (lub klient) może zlecić innemu serwerowi obliczeniowemu wykonywanie zadania. Przy stosunkowo małym wysiłku, agent może być w stanie uruchomić obliczenia na zastępczym serwerze od miejsca, w którym skończyły się postępy poprzedniej maszyny (wymaga to mechanizmu pobierania i zapamiętywania pośrednich stanów obliczeń). Przedstawiony przykład obrazuje między-serwerowe mechanizmy tolerowania uszkodzeń i wyrównywania obciążenia. Mechanizmy wewnątrz-serwerowe mogą opierać się na rozmaitych technikach stosowanych na różnych platformach sprzętowych i programowych, mających na celu uczynienie maszyn obliczeniowych i środowisk przetwarzania odpornymi na błędy oraz zwiększenie ich wydajności.

Techniki między-serwerowe

Jak wspomniano wcześniej, kiedy serwer obliczeniowy ulega uszkodzeniu lub nie wykonuje powierzonego mu zadania z zadowalającą prędkością, jest pożądane, by agent wykrywał ten fakt i przenosił obliczenia na inną maszynę.

Rozwiązania wdrożone

W implementacji systemu z 1999 roku zaimplementowany jest podstawowy mechanizm tolerowania uszkodzeń. Agenci okresowo zbierają informacje o serwerach i ich stanie. Kiedy serwer, który w danym momencie wykonuje obliczenia staje się niedostępny, wtedy agent wybiera inny serwer, który przejmuje obliczenia. W tej wersji systemu odbywa się to w następujący sposób: klient kontaktuje się z nowym serwerem i rozpoczyna obliczenia od początku.

Agenci używają rozmaitych technik, dla określania dostępności serwerów obliczeniowych, włączając standardowe programy systemu *Unix* takie jak *uptime* lub, bardziej zaawansowane, jak *Globus Hearbeat Monitor*. Planowane jest również wykorzystanie do obserwacji dostępności systemów pakietu *Network Weather Service*.

Wykorzystanie zintegrowanych usług przechowywania danych (ang. *Integrated Storage Services*)

Prostym sposobem ulepszenia opisanego wyżej mechanizmu jest stworzenie serwerowi, który przejmuje obliczenia, możliwości zapytania serwera, który oddał obliczenia i jest dostępny, choć przeładowany, o stan obliczeń i wystartowania ich od tego punktu.

Ta prosta metoda jest istotnym ulepszeniem w stosunku do wdrożonych w *NetSolve* rozwiązań. Jednak jest ona ograniczona przez fakt, że uszkodzony serwer musiałby w jakiś sposób przekazać informacje o stanie obliczeń serwerowi, który je przejmuje. Dla rozwiązania tego problemu, projektanci systemu *NetSolve* wprowadzili pojęcie serwera usługi przechowywania danych (ang. *storage server*). Na wysokim poziomie abstrakcji, serwery składowania danych, tak jak „zwyczajne", obliczeniowe serwery są zarządzane przez agentów *NetSolve*. Jednakże ich funkcją jest nie liczenie, lecz przechowywanie danych.

Serwery składowania mają przechowywać punkty kontrolne (ang. *checkpoints*) obliczeń prowadzonych na serwerach obliczeniowych. Kiedy serwer obliczeniowy decyduje, iż powinien zapisać swój stan, wybiera serwer przechowywania danych (przy pomocy agenta *NetSolve*) i wysyła swój punkt kontrolny do niego.

Serwer składowania może przechowywać powierzone mu dane w pamięci fizycznej (operacyjnej) lub w pamięci dyskowej, natomiast dla zapewnienia większej wiarygodności, może replikować dane.
Przedstawiona technika eliminuje zależność przejmującego obliczenia serwera obliczeniowego od, być może uszkodzonego i niedostępnego serwera, który dotąd je wykonywał.

Techniki wewnątrz-serwerowe

W poniższym punkcie przedstawione zostaną dyskutowane w dokumencie [10] techniki zapewniające tolerowanie uszkodzeń i wyrównywanie obciążeń, których wdrożenie rozważają autorzy *NetSolve*. Są on nazywane wewnątrz-serwerowymi (ang. *intra-server*), ponieważ są implementowane w obrębie jednego serwera lub systemu obliczeniowego.
W punkcie tym zostaną również omówione techniki mieszane (między-serwerowe i wewnątrz-serwerowe) zapewniania odporności na uszkodzenia.

Rozwiązania wdrożone do 1999 roku

W implementacji z 1999 roku, jedyną realizowaną wewnątrz-serwerową techniką tolerowania uszkodzeń było wykorzystanie odpowiednich mechanizmów systemu *Condor*, jeśli system ten był zainstalowany na maszynach obliczeniowych. Jednakże to podejścia ma bardzo poważne ograniczenia, ponieważ system *Condor*:

- może migrować zadania jedynie pomiędzy maszynami o tej samej architekturze,
- może migrować zadania jedynie w obrębie kontrolowanych przez niego serwerów,
- działa tylko z sekwencyjnymi (nie równoległymi) programami,
- nigdy nie dokonuje faktycznej migracji zadania z pierwotnej maszyny, tzn. wszystkie wołania systemowe są przesyłane przez tę maszynę do serwera, który przejął zadanie.

Konieczne jest stworzenie bardziej uniwersalnych i otwartych mechanizmów tolerowania uszkodzeń i wyrównywania obciążeń. Poniżej omówione zostaną przedstawione przez autorów pracy [10] techniki, które mogą zostać wykorzystane w systemie *NetSolve*.

Mechanizm koordynowanych punktów kontrolnych i odtwarzanie wsteczne

Najprostszymi mechanizmami zapewniającymi tolerowanie uszkodzeń dla równoległych programów wydają się być, autorom wspomnianej publikacji, mechanizm koordynowanych punktów kontrolnych (ang. *coordinated checkpoints*) oraz odtwarzania wstecznego (ang. *rollback recovery*).

W ustalonych interwałach procesy biorące udział w obliczeniach zapisują swoje stany na trwałym medium (dyskowym). To właśnie jest nazywane koordynowanym punktem kontrolnym. W przypadku wystąpienia uszkodzenia jednego lub więcej komponentów systemu, równa liczba procesów używa stanów, zapisanych na trwałym medium, do restartowania obliczeń z punktu kontrolnego. Istnieje kilka implementacji bibliotek umożliwiających wykonywanie koordynowanych punktów kontrolnych w rozmaitych środowiskach przetwarzania współbieżnego (np. MIST i *CoCheck* zapewniające *checkpointing* dla sieci stacji roboczych pracujących pod kontrolą systemów PVM i MPI), w sposób całkowicie przeźroczysty, przeźroczy tylko częściowo lub całkowicie jawny. Poziom przeźroczystości jest tu rozumiany jako wskaźnik poziomu zaangażowania programisty w proces wykonywania punktów kontrolnych.

Mechanizmy przeźroczyste a nie przeźroczyste wykonywania punktów kontrolnych

W pierwszym kroku, do systemu *NetSolve* wdrożone ma być nieprzeźroczyste wykonywanie punktów kontrolnych. Tak metoda ma następującą zaletę: pozwala uczynić mechanizm punktów kontrolnych niezależnym od platformy, co pozwala na odtwarzanie obliczeń w konfiguracji odmiennej od tej, w której punkt kontrolny wykonano (np. na innej liczbie procesorów lub na maszynie o odmiennej architekturze). W połączeniu z użyciem serwerów przechowywania, metoda ta umożliwia restartowanie obliczeń na dowolnej maszynie obliczeniowej udostępnionej systemowi *NetSolve*.

Jednakże metoda ta posiada poważną wadę: wymaga ingerencji w kod używanej biblioteki obliczeniowej – operacje wykonywania punktów kontrolnych muszą zostać jawnie wywołane przez uruchomienie odpowiednich funkcji (wykonujących punkt kontrolny) z wnętrza funkcji obliczeniowych. Poza tym składowanie punktów kontrolnych na medium trwałym (np. dysku twardym) lub serwerze przechowywania danych pociąga za sobą narzuty czasowe.

Techniki bezdyskowe wykonywania punktów kontrolnych

Metodą polepszenia wydajności mechanizmu koordynowanych punktów kontrolnych jest ominięcie medium trwałego, co prowadzi do mechanizmu tzw. bezdyskowego *checkpointing*'u (ang. *diskless checkpointing*). W tym przypadku, punkty kontrolne są składowane w pamięci procesorów aplikacyjnych, kodowane odpowiednimi, umożliwiającym korekcję błędów (ubytków) algorytmami. Technika ta, w połączeniu ze stosowaniem dodatkowych procesorów dla punktów kontrolnych (ang. *checkpointing processors*), zapewnia, że utrata pewnej, ograniczonej liczby jednostek przetwarzających może być tolerowana przez system obliczeniowy.

Ta technika nie jest specyficzna dla systemu *NetSolve*, znajduje bowiem szersze zastosowanie. W systemie *NetSolve* jest ona eksploatowana w postaci mieszanej, wewnątrz-serwerowej i między-serwerowej metody dla tolerowania uszkodzeń. W adaptacji tej metody dla NetSolve, zamiast składowania kodów korekcyjnych w specjalnych procesorach serwera, składuje się je na serwerach usługi przechowywania danych. To, poza eliminacją konieczności przeznaczania zasobów serwera obliczeniowego do przechowywania punktów kontrolnych, umożliwia łatwą migrację aplikacji obliczeniowych na inne maszyny (przy założeniu, że sam punkt kontrolny wykonany jest w sposób, który tej możliwości migracji na inne maszyny nie zamyka).

Inne wewnątrz-serwerowe techniki tolerowania uszkodzeń

Autorzy pracy [10] stwierdzają, iż techniki tolerowania uszkodzeń zaimplementowane w specyficznych systemach obliczeniowych takich jak np. systemy oparte o model pamięci współdzielonej, systemy z odporną na błędy współdzieloną przestrzenią krotek oraz systemy oparte o przekazywanie komunikatów i ich rozmaite mutacje, mogą być eksploatowane, w postaci „zagnieżdżonej" wewnątrz tych systemów, przez system *NetSolve*.

Nie rozwiązany jednak nadal pozostaje problem konstrukcji uniwersalnego, między-serwerowego *checkpointing*'u. Rozważane są techniki takie jak stosowanie specjalnych preprocesorów dla włączanych do systemu bibliotek obliczeniowych, osadzających w ich kodzie struktur programowych wspomagających wykonywanie punktów kontrolnych, używanie specjalnych języków programowania, które zawierają mechanizmy wspomagające niezależny od architektury *checkpointing* osadzone w kompilatorze i systemie uruchomieniowym a także zatrudniające specjalne „checkpoint'owalne" struktury danych. Jednakże, ponieważ w czasie powstawania pracy [10] wymienione techniki były w fazie testowej, autorzy pracy nie sformułowali wyraźnych tez dotyczących ich użyteczności i przydatności.

1.3.7. Wydajność

Jednym z wyzwań projektu *NetSolve* było połączenie łatwości użytkowania z dobrą wydajnością systemu. Do mechanizmów wpływających na podniesienie wydajności systemu bez angażowania użytkownika należą *load balancing* (*NetSolve* stara się wybrać najlepszy, z punktu widzenia użytkownika, serwer do wykonania danego zadania) i równoczesne używanie wielu zasobów (interfejs nie blokujących wołań funkcji obliczeniowych umożliwia współbieżne wykonywanie programu klienta na różnych maszynach obliczeniowych).

Ważnym aspektem przy planowaniu rozdziału zasobów powinno być, poza wyborem najodpowiedniejszego serwera obliczeniowego dla danego zadania, określenie planu wykonania powiązanych ze sobą grup zadań, który minimalizowałby przesyły danych w sieci. Zagadnienia związane z optymalizacją przesyłu danych w środowiskach przetwarzania sieciowego oraz techniki stosowane w systemie *NetSolve* wychodzące naprzeciw tym wyzwaniom zostały omówione w pracy [11].

1.3.7.1. Optymalizacja przesyłu danych

Autorzy publikacji [11] zauważyli, że ciągły, znaczący wzrost wydajności procesorów nie pociąga za sobą równie szybkiego rozwoju infrastruktury sieciowej a przesyłanie danych w sieci wciąż powoduje duże narzuty czasowe w wielu aplikacjach wysoko-wydajnych środowisk obliczeniowych. Istnieją jednak, ich zdaniem, stosunkowo proste metody przyspieszania takich aplikacji poprzez optymalizację komunikacji międzyprocesowej.

Ogólnie, aplikacje wykazują następujące, wspólne cechy: dane wejściowe są wielkimi wolumenami i istnieją duże zależności pomiędzy kolejnymi operacjami obliczeniowymi. Zdaniem autorów pracy, nie przywiązywano, jak dotychczas, zbytniej uwagi do sposobów efektywnego rozpraszania danych aplikacji pomiędzy różnymi składowymi komponentami systemów *Grid*. I, o ile istniały dobrze zbadane metody rozdziału danych dla aplikacji równoległych, o tyle nie rozwinięto podobnych metod dla systemów, w których fragmenty obliczeń wykonywane są współbieżnie. Często powodowało to niepotrzebne, wielokrotne przesyłanie tych samych danych pomiędzy tymi samymi modułami.

Autorzy zaproponowali technikę, którą nazwali „sekwencjonowaniem" żądań (ang. *request sequencing*). Termin ten obejmuje zarówno interfejs do grupowania serii żądań, jak i techniki szeregowania, które używają skierowanych acyklicznych grafów (ang. *Direct Acyclic Graph – DAG*) do reprezentowania modułów obliczeniowych.

Model DAG

Poniżej przedstawimy definicję modelu DAG sformułowaną przez Y.Kwok'a, cytowaną w pracy [11]:

„DAG jest ogólnym modelem programu równoległego składającego ze zbioru procesów (węzłów), pomiędzy którymi zachodzą zależności. Węzeł w grafie reprezentuje zadania, które z kolei jest zbiorem instrukcji, które muszą być kolejno (bez wyprzedzeń) wykonane na tym samym procesorze. Węzeł ma jedno lub więcej wejść. Kiedy wszystkie wejścia są dostępne, węzeł może rozpocząć przetwarzanie. Graf ma również skierowane krawędzie reprezentujące

relację częściowego porządku określoną na zadaniach. Częściowy porządek wprowadza ograniczony relacją poprzedzania, acykliczny graf i powoduje, że jeśli $n_i \rightarrow n_j$, to n_j potomkiem, który nie może rozpocząć działania zanim jego rodzic n_i nie zakończy swojego i nie wyśle swoich danych do n_j."

Analiza danych i DAG

Dla zbudowania grafu DAG konieczna jest analiza każdej wartości wejściowej i wyjściowej w sekwencji żądań. Dwa parametry uznawane są za te same, jeśli odnoszą się do tego samego obszaru w pamięci. Pole „rozmiar" i wskaźnik odniesienia używane są do obliczania, kiedy parametry wejściowe przekraczają przestrzeń pamięci. *NetSolve* obsługuje wiele typów danych, włączając macierze, wektory i skalary. Zdecydowano się na sprawdzanie pod kątem wielokrotności wystąpień jedynie macierzy i wektorów (uznano, że są to jedyne obiekty, które mają tendencję do bycia wystarczająco dużymi, by koszt poniesiony na analizę mógł się zwrócić). Jednakże parametry wszystkich typów są sprawdzane pod kątem wzajemnych zależności, ponieważ mogą one wpływać na kształt grafu wykonania.
Taka analiza prowadzi do konstrukcji grafu DAG. Graf jest acykliczny, ponieważ struktury pętlowe nie są dozwolone wewnątrz sekwencji wołań i przez to dowolny węzeł nigdy nie będzie własnym potomkiem.

Interfejs dla grupowania wołań

Jako dodatek do funkcji używanych do wysyłania żądań obsługi, zaimplementowano dwie funkcje, których celem jest oznaczanie początku i końca sekwencji żądań. Funkcja `begin_sequence()` nie pobiera argumentów i nie zwraca żadnej wartości. Powiadamia ona system o konieczności rozpoczęcia analizy danych. `Funkcja end_sequence()` oznacza koniec sekwencji; w tym punkcie sekwencja zebranych żądań jest wysyłana do serwera, by wykonał on szeregowanie wykonania. Jako rozszerzenie, ta funkcja pobiera również zmienną liczbę argumentów opisujących, które parametry wyjściowe nie mają być zwracane do klienta. Oznacza to, że pośrednie wyniki nie są potrzebne dla żadnych lokalnych obliczeń i nie muszą być zwracane. To użytkownik musi określić, które wyniki są obowiązkowe a które są niepotrzebne. Rysunek 1.12 pokazuje, jak może wyglądać wywołanie sekwencji obliczeń. Dwie cechy tego przykładu są godne odnotowania: po pierwsze, dla wszystkich żądań jedynie ostatni parametr jest wyjściowy; po drugie, użytkownik instruuje system, by nie zwracał pośrednich wyników operacji `command1` i `command2`.

```
...
begin_sequence();
submit_request("command1", A, B, C);
submit_request("command2", A, C, D);
submit_request("command3", D, E, F);
end_sequence(C, D);
...
```

Rysunek 1.12. Wywołanie sekwencji obliczeń w systemie *NetSolve*

Źródło: praca [11]

Dla zapewnienia prawidłowego zachowania systemu na oprogramowanie korzystające z mechanizmu grupowanych wołań funkcji narzucone muszą być następujące ograniczenia:

- niedozwolone wewnątrz sekwencji wołań są struktury kontrolne, które mogłyby zmieniać ścieżkę wykonania sekwencji;
- zabronione wewnątrz sekwencji są instrukcje, które zmieniałyby wartość dowolnego z parametrów wejściowych funkcji składowych sekwencji.

Szeregowanie sekwencji wywołań w serwerze

Po skonstruowaniu grafu DAG jest on transmitowany do serwera obliczeniowego *NetSolve*. W omawianej w pracy [11] wersji systemu, agent *NetSolve* używa algorytmu szeregowania na dużym poziomie granulacji i decyduje, który serwer powinien wykonać całą sekwencję. Planowane rozwinięcie tego mechanizmu tak, by możliwe było wykonywanie poszczególnych fragmentów sekwencji wołań na rozłącznych serwerach, wymaga, z jednej strony, dostarczenia wszystkim tym serwerom funkcji programowych umożliwiających wykonanie składowych operacji, z drugiej, obfituje możliwością prawdziwie współbieżnego wykonania obliczeń, nawet w przypadku, gdy niedostępne są maszyny równoległe.

Węzeł grafu jest wykonywany, jeśli wszystkie jego dane wejściowe są dostępne i nie ma konfliktu z jego parametrami wyjściowymi.

Dla podziału danych, do wybranego serwera obliczeniowego transmitowana jest unia zbiorów parametrów wejściowych. Dzięki temu wejścia dla wszystkich węzłów grafu, poza tymi, które korzystają z wyników pośrednich z węzłów będących w grafie ich poprzednikami, są dostępne i może rozpocząć się wykonywanie sekwencji.

Algorytm szeregowania wykonania zawiera następujące punkty:

a) wykonywane są wszystkie węzły bez zależności,
b) gdy zakończy się działanie tych węzłów, aktualizowana jest lista zależności,
c) wykonywane jest sprawdzanie zależności dla następnych węzłów do wykonania.

Wstępna ocena przyjętych rozwiązań

Autorzy sytemu pracy [11] zakładają, że redukcja ruchu w sieci powoduje wystarczająco duże polepszenie wydajności, by proces szeregowania był wart zachodu. Wyniki opisanych w tej publikacji eksperymentów zdają się tę tezę potwierdzać.

Konieczna jest jednak, zdaniem autorów, dalsza praca nad algorytmami szeregowania, szczególnie w kwestii uwzględniania, przy opracowywaniu planów wykonania sekwencji żądań, lokalizacji danych. Do tego celu autorzy planują wykorzystać narzędzia takie jak *Internet Backplane Protocol* i *Global Access to Secondary Storage*.

1.3.7.2. Szeregowanie zadań dobrze podzielnych

W pracy [12] opisano adaptacyjne algorytmy szeregowania dla klasy aplikacji obliczeniowych, działających według schematu *master-slave*. Autorzy tego dokumentu nazwali te aplikacje *task farming applications* (aplikacje dobrze podzielnych zadań).

Podstawowe założenia

Autorzy wychodzą z obserwacji, że w środowiskach *Grid* dostępność zasobów obliczeniowych podlega ciągłym zmianom. Dla długotrwałych ale dobrze podzielnych zadań obliczeniowych, konstrukcja odpowiednich, adaptacyjnych algorytmów szeregowania oraz dostarczenie interfejsu użytkownika umożliwiającego ich dokładne specyfikowanie, może, ich zdaniem, doprowadzić do znacznego skrócenia czasu ich wykonywania.

Farming job, tak nazywają tę klasę zadań autorzy pracy [12], składa się dużej liczby niezależnych żądań, które mogą być obsługiwane równolegle; praca taka należy do klasy „kłopotliwie równoległych” (ang. *embarrasingly parallel*) programów, dla których jest jasne, jak należy dokonać podziału na podzadania do wykonania w środowisku przetwarzania współbieżnego.

Bez interfejsu programistycznego dla *farming jobs* odpowiedzialność za zrównoleglenie programu spada na użytkownika. Może on albo wysłać asynchronicznie wszystkie żądania na raz, pozwalając systemowi je uszeregować, albo „ręcznie” zarządzać kolejką gotowości, poprzez utrzymywanie co najwyżej *n* równocześnie wysłanych żądań w danym momencie przetwarzania.

Interfejs programistyczny

Trudno jest, zdaniem autorów pracy [12] zaprojektować API, które jest równocześnie wygodne dla użytkownika i wystarczająco zaawansowane, by sprawdzić się w rzeczywistych zastosowaniach.

Farming API przedstawione w tej publikacji składa się z funkcji `farm()`, za pomocą której użytkownik określa dane dla wszystkich zadań obliczeniowych. Głównym pomysłem jest zastąpienie wielokrotnych wołań funkcji `submit()` jednym wywołaniem funkcji `farm()`, której argumenty są listami argumentów dla funkcji `submit()`.

W pierwszej implementacji tego mechanizmu, zakłada się, że argumenty wołań funkcji `submit()` są liczbami całkowitymi lub wskaźnikami (co jest zgodne ze specyfikacją systemu *NetSolve*). Rozszerzenie wołań tak, by obsługiwały również inne typy argumentów, wydaje się autorom trywialne.

Pierwszy argument funkcji `farm()` określa liczbę żądań poprzez deklarację indukcyjnej zmiennej i definicję jej przedziału. Składnia tego argumentu jest następująca: `„i=%d,%d"` (patrz przykład poniżej, rysunek 1.13). Drugi argument jest identyfikatorem funkcji numerycznej dostępnej w środowisku obliczeniowym. Następnie pojawia się (w postaci zmiennej) liczba argumentów na liście. Implementacja dostarcza trzech funkcji, które muszą być wywołane dla wygenerowania takiej listy argumentów:

a) `expr()` – sprawia, że argumentowi operacji obliczeniowej *i* jest liczba całkowita obliczona jako wyrażanie arytmetyczne zawierające *i*;
b) `int_array()` – sprawia, że całkowity argument operacji obliczeniowej *i* jest elementem tablicy indeksowanej przez wartość wyrażenia arytmetycznego zawierającego *i*;
c) `ptr_array()` jest podobny do `int_array()`, lecz obsługuje argumenty wskaźnikowe.

Wyrażenia arytmetyczne są określane zgodnie ze składnią powłoki *Bourne*'a systemu *Unix* (ang. *Bourne Shell syntax*).

```
double x[10],y[3],z[10];
submit(„foo",2,x,10);
submit(„foo",4,x,30);
submit(„foo",6,x,10);
```

Rysunek 1.13. Zwykłe wywołanie grupy funkcji

Zródło: praca [12]

Prezentowany na rysunku 1.13 fragment kodu zawiera wywołanie funkcji obliczeniowej `foo` dla następującego zbioru argumentów: `(2,x,10)`, `(4,y,30)` i `(6,z,10)`. Zauważmy, że `x`, `y` i `z` przechowują rezultaty wołań *NetSolve*.

```
void *ptrs[3]
int  *ints[3];

ptrs[0] = x;  ptrs[1] = y;  ptrs [2] = z;
ints[0] = 10; ints[1] = 30; ints [2] = 10;

farm("i=0,2","foo",expr("2*($i+1)"),ptr_array(ptrs,"$i"),
                   int_array(ints,"$i"));
```

Rysunek 1.14. Wywołanie zgrupowane

Zródło: praca [12]

Rysunek 1.14 pokazuje fragment kodu, wykonujący te same operacje z wykorzystaniem funkcji `farm()`.

Strategia szeregowania

Podstawowa idea przyświecająca algorytmowi szeregowania przedstawiona została już w punkcie 1.3.7.2, mianowicie jest nią zarządzanie kolejką gotowości. Algorytm zarządza więc kolejką gotowości i dostosowuje się do środowiska przetwarzania, w którym działa, poprzez modyfikację wartość *n* dynamicznie, odpowiednio do pomiarów przepustowości obliczeniowej. Algorytm widzi system jako jednostkę, która w różnym czasie odpowiada na żądanie wykonania tej samej operacji.

```
n = initial guess on the queue size;
α = scheduling factor;
δ = 1;
while (tasks remaining) {
    while (number of pending tasks < n) {
        submit();
    }
    foreach (pending task) {
        poll();
    }
    if (n− number of pending tasks ≥ n × α) {
        if (average task response time has improved) {
            n = n + δ;
            δ = δ + 1;
        }
        else {
            n = n − δ;
            δ = 1;
        }
    }
}
```

Rysunek 1.15. Algorytm szeregowania zadań dobrze podzielnych

Zródło: praca [12]

Rysunek 1.15 przedstawia schemat tego algorytmu. Najpierw, algorytm wybiera pewną początkową wartość *n* oraz ustawia wartość współczynnika szeregowania α (ang. *scheduling factor*), przyjmującego wartości z zakresu [0,1), który określa zachowanie algorytmu. Wartość n może być zmieniona tylko w przypadku, gdy więcej niż *n* zadań zakończy się podczas trwania jednej iteracji najbardziej zewnętrznej pętli algorytmu. Wartość α=1 powoduje, że algorytm jest skrajnie konserwatywny (tylko gdy obsługa *n* żądań jest ukończona wartość *n* może być zmieniona); mniejsza wartość α powoduje, że algorytm częściej będzie próbował modyfikować wartość *n*. Algorytm utrzymuje historię średnich czasów wykonania dla wszystkich żądań w kolejce. Ta historia jest używana do wykrywania zmian wydajności (na lepsze lub na gorsze) i do odpowiedniej modyfikacji wartości *n*.

Ocena mechanizmu szeregowania i możliwości jego ulepszenia

Opisana w pracy [12] realizacja mechanizmu szeregowania dla zadań dobrze podzielnych opiera się na implementacji funkcji `farm()` jako dodatkowej warstwy działającej ponad tradycyjnym interfejsem programistycznym systemu *NetSolve*. Nie jest więc ten mechanizm wcielony w wewnętrzne struktury systemu.

Farming API systemu *NetSolve* jest, w opinii jego autorów, bardzo ogólne i ta ogólność ma prowadzić do uproszczeń. Zagnieżdżony w API mechanizm szeregujący nie jest w stanie zrobić użytku z wiedzy na temat charakterystycznych cech niektórych aplikacji, takich jak

schematy dostępu do danych, wymagania dla operacji wejścia/wyjścia oraz zależności pomiędzy składowymi operacjami. Innym ograniczeniem interfejsu jest fakt, że wywołanie funkcji `farm()` jest atomowe, co jest wprawdzie korzystne z punktu widzenia łatwości obsługi, jednak uniemożliwia bieżące korzystanie z wyników zakończonych już obliczeń składających się na wywołanie (np. dla wizualizacji obliczeń itp.).

Wyniki opisanych w pracy [12] eksperymentów są dość zadowalające. Potwierdzono zdolność algorytmu szeregowania do adaptacji do zmieniających się warunków panujących w środowisku obliczeniowym (czasowa niedostępność serwerów obliczeniowych itd.).

W publikacji zaproponowano też kilka możliwości ulepszenia algorytmu, między innymi tak, by umożliwiał bieżące wykorzystanie wyników tych obliczeń składowych, które w danym momencie przetwarzania zostały zakończone.

1.3.8. Plany rozwojowe dla systemu NetSolve

W artykułach dotyczących systemu *NetSolve* określono następujące zadania, które zamierza się zrealizować w projekcie *NetSolve*.

- stworzenie systemu nazw dla łatwiejszego zarządzania złożonym, obsługującym wielu użytkowników systemem;
- wcielenie systemów *Globus* i *Legion* - spojrzenie na projekt *NetSolve* z uwzględnieniem wiedzy zdobytej z obserwacji i badań tych systemów oraz wcielenie pewnych mechanizmów w tych projektach skonstruowanych, np. *Globus Heartbeat Monitor* dla wykrywania uszkodzeń oraz mechanizmów bezpieczeństwa, kodowania i kompresji danych i kontroli dostępu z systemu *Legion*;
- zbadanie możliwości stworzenia silniejszej (ang. *robust*) implementacji na bazie rozproszonych obiektów CORBY i technologii *Java RMI* (szczególnie chodzi o wykorzystanie mechanizmów *load-balancing* i *fault-tolerance*);
- rozwój agenta (ang. *resource broker*) mający na celu uwzględniania przez niego specyfiki nowo powstających i istniejących już (np. *MPP*) systemów obliczeniowych; agent musi mieć większą zdolność predykcji czasów wykonania zadań na różnych rozproszonych i równoległych systemach obliczeniowych (w zależności od ich konfiguracji sprzętowej i programowej) a także, co bardzo ważne, w systemach wsadowych (ang. *batch systems*).
- z kolei na samym serwerze obliczeniowym wykorzystującym pakiety takie jak SCALAPACK lub LAPACK powinny być, z uwzględnieniem rozmiaru i specyfiki zadania podejmowane decyzje dotyczące konfiguracji systemu obliczeniowego, w której

wykonywane mają być obliczenia (np. liczba procesorów i rozmieszczenie poszczególnych elementów macierzy na procesorach systemu równoległego).

2. Porównanie cech systemów Ninf i NetSolve

Przedstawiony przegląd środowisk udostępniania naukowych zasobów obliczeniowych miał na celu zaprezentowanie możliwości i funkcjonalności systemów oraz związanych z ich projektowaniem, realizacją i użytkowaniem problemów technicznych. Zaprezentowany przegląd systemów przetwarzania funkcyjnego opiera się w dużej mierze na literaturze: artykułach, stronach www i dokumentacji związanych z systemami Ninf i NetSolve.

Przegląd dokonany w punkcie 1 miał układ sekwencyjny, omówiono kolejno poszczególne systemy. W tym punkcie przedstawione zostanie porównanie cech systemów Ninf i NetSolve w formie tabeli.

Poniższa tabela (Tab. 1) zbiera informacje z punktu 1 dotyczące cech i funkcji systemów Ninf i NetSolve. Cechy pogrupowano ze względu na aspekty systemów zdalnego przetwarzania, jakich dotyczą.

Tab. 1 Zestawienie cech systemów Ninf i NetSolve

Id	Cecha (feature)	Ninf	NetSolve
A	**Dostęp do systemu**		
A.1	interfejsy programistyczne:	• API: C, Fortran i Java • Sub-API: może być wykorzystane przez wszystkie języki, które obsługują „obce interfejsy" do języka C; • Ninf-CIM: dla języków/środowisk o bardzo odmiennej reprezentacji danych (szczególnie tablic), np. Excel, Mathematica, Lisp; „funkcje-przemierzacze" wywoływane przez funkcję `Ninf_cim_main()` [1.2.3.6]	• API: Fortran, C, Java [1.3.4]
A.2	interfejsy interaktywne	• Mathematica • Excel [1.2.3.6]	• Web based Java Gui • Matlab • shell Unix [1.3.4] • Mathematica [release notes, v.1.4]
A.3	API:	automatyczne określanie stopnia funkcji na podstawie wyróżnionych argumentów funkcji [1.2.3]	brak informacji n.t. automatycznego określania stopnia funkcji na podstawie argumentów funkcji [1.2.3]
A.4	API: asynchroniczne wywołania funkcji	• `Ninf_call_async` [1.2.3.3]	• `Netslnb, netslpb, netswt` [1.3.4]

A.5	API: wywołania zagregowane	„transakcje Ninf"
		`Ninf_transaction_begin();`
		`Ninf_transaction_end();`
		[1.2.3.4]
			„sekwencja żądań"
			`begin_sequence();`
			`end_sequence(...);`
			[1.3.7.1]

A.6	API: mechanizm wywołania zwrotnego	jest [1.2.2.4] i [1.2.3.5]	brak informacji
A.7	możliwość pobierania / umieszczania danych z/na www	jest [1.2.3.2]	brak informacji

B	**Szeregowanie zadań i zarządzanie zasobami**		
B.1	Metaserwery: kooperacja	Metaserwery mogą kooperować, zmieniające się informacje są propagowane pomiędzy metaserwerami, ew. nie posiadane przez dany metaserwer informacje mogą być ściągane od innych na żądanie [1.2.5.5]	Każda instancja agenta może mieć swój własny ogląd systemu. Nie musi być on identyczny. Można będzie ich łączyć w struktury hierarchiczne (plany) [1.3.2.1]
B.2	Moduł szeregujący – otwartość, wymienność algorytmu szeregowania	• zaimplementowany w Javie; • można wymienić algorytm szeregowania [1.2.8.4] • można zmieniać politykę szeregowania „w locie" za pomocą odpowiednich protokołów sieciowych [1.2.5.4]	moduł szeregujący całkowicie zaimplementowany w C; dobry i prosty interfejs do schedulera
B.3	Moduł szeregowania – uwzględnianie rozmieszczenia danych	uwzględnia przy szeregowaniu lokalizację plików używanych w obliczeniach [1.2.8.4]	było w planach [„Wstępna ocena przyjętych rozwiązań", 1.3.7.1]; w [release notes, v.1.4] brak omówienia
B.4	Szeregowanie – prognozowanie dostępności zasobów	metody własne (brak szczegółów) utrzymuje następujące informacje o serwerach obliczeniowych: • lokalizację serwera (adres IP i numer portu), • listę funkcji bibliotecznych zarejestrowanych na serwerze, • dystans do serwera z uwzględnieniem przepustowości sieci, • informacje o zdolności obliczeniowej serwera (prędkość zegara, wskaźnik Flops – ang. floating point operations per second), • stan serwera, włączając średnie obciążenie (ang. load average). [1.2.5.5]	Integracja z NWS [release notes, v. 1.4]
B.5	Szeregowanie – monitorowanie dostępności zasobów	własne metody (brak szczegółów)	• zbierana periodycznie przez agenta (nie na żądanie); serwer obliczeniowy rozgłasza *workload* tylko gdy wielkość zmiany przekracza wielkość przedziału ufności [1.3.5.4] (stare rozwiązanie); • uwaga! w wersji 1.4. jest zintegrowany NWS [release notes, v. 1.4]
B.6	Szeregowanie - informacja o funkcjach	• brak informacji	• plik opisowy funkcji (IDL) zawiera przybliżone określenie złożoności obliczeniowej algorytmu w funkcji rozmiaru problemu [1.3.3.3]
B.7	Szeregowanie - obliczanie najlepszej maszyny	• brak informacji	• stosuje się prosty teoretyczny model wydajności ("surowa wydajność" i obciążenie)
B.8	Zarządzanie systemem (dane o zasobach)	• baza danych zasobów dostępna przez protokół LDAP • opcjonalnie Globus MDS jeśli są dostępne w środowisku [1.2.8.4]	Skrypty CGI na www pozwalające uzyskać: • listę instancji agentów lub zasobów obliczeniowych występujących w danym systemie (lista nazw maszyn i adresów IP) [1.3.2.3]
B.9	Metaserwer – rejestracja zasobów	dynamiczna (wynika to nie wprost z [1.2.5.5])	dynamiczna; przy włączaniu serwerów obliczeniowych; przy rejestracji podawane są informacje takie jak lokalizacja, adres, lista rozwiązywanych problemów [1.3.3.3]
B.10	Metaserwer - spis dostępnych problemów obliczeniowych	• użytkownicy mogą przeglądać spis funkcji dostępnych na serwerze Ninf za pomocą przeglądarki web [1.2.4.1]	• listę problemów (zadań), które można rozwiązać wewnątrz danego systemu NetSolve [1.3.2.3]
B.11	Metaserwer - uruchamianie zadań w sytuacji pełnego obciążenia dostępnych zasobów	Serwery obliczeniowe przydzielane są do zadań aż do wyczerpania. Jeżeli nie ma wolnych serwerów, metaserwer przechowuje wszystkie dane potrzebne do uruchomienia obliczeń aż do momentu zwolnienia się zasobów. (przeźroczyste dla użytkownika) [1.2.5.3]	brak informacji

C	**Serwer obliczeniowy**		
C.1	Przygotowanie funkcji bibliotecznych i zasobów obliczeniowych dla systemu	Opis [1.2.2.2]: • przygotowanie opisu w IDL, • kompilacja IDL (wynikiem są pliki nagłówkowe i kody stopki), • kompilacja biblioteki obl. • łączenie obiektów programowych z funkcjami Ninf RPC • rejestracja funkcji bibliotecznych w serwerze Ninf Generator interfejsów: • pobiera IDL, • tworzy programy-stopki i pliki makefile służące do tworzenia obiektów wykonywalnych Ninf poprzez łączenie programów stopek i funkcji bibliotecznych [1.2.4.1]	Kompilacja IDL: • pseudo-kompilator pobiera plik konfiguracyjny opisujący każdy problem obliczeniowy w sposób formalny i generuje kod w języku C dla procesu obliczeniowego odpowiedzialnego za rozwiązywanie problemu; • wywołania funkcji bibliotecznych muszą być napisane w języku C z użyciem predefiniowanego zbioru makr; • po skompilowaniu opisów powstają moduły obliczeniowe (wykonywalne) – *wrapper*'y – będące opakowaniem do bibliotek naukowych • można przeglądać listę rozwiązywanych problemów i zasobów sprzętowych na www [1.3.3.2]
C.2	sposób uruchamiania oprogramowania na serwerach obliczeniowych	*executable modules* – powstają przez łączenie programów stopek i funkcji bibliotecznych [1.2.4.1]	sprzeczne informacje: • bezpośrednio, poprzez wołanie funkcji bibliotecznej [1.3.3.2] • w [1.3.3.3] stwierdzono, że powstają egzeki-wrappery
C.3	IDL: zawartość pliku z opisem	• lista argumentów, • specyfikatory dostępu (in/out), • parametry (wyrażenia) pozwalające obliczyć rozmiar danych; • informacje niezbędne do kompilacji i łączenia (jaka biblioteka itp.) [1.2.4]	• oczekiwane dane wejściowe i jakie są spodziewane dane wyjściowe • jaka biblioteka, • przybliżone określenie złożoności obliczeniowej [1.3.3.3]
C.4	IDL: typy danych	skalary i wielowymiarowe tablice [1.2.4]	<object, data> object = MATRIX, VECTOR, SCALAR data = dowolny typ danych jezyka Fortran [1.3.1.1]
C.5	IDL: określanie rozmiarów tablic na podstawie argumentów wejściowych	jest [1.2.4]	jest
C.6	Narzędzia dla IDL		• Graficzny edytor IDL [1.3.3.2]: *PDF* (*Problem Description File*) *Generator* + *Java GUI* [release notes, v. 1.4] • Planowane jest [1.3.3.3] stworzenie repozytorium opisów problemów na www, by można było ich używać do konfiguracji dowolnych serwerów obliczeniowych. Brak informacji czy zostało stworzone
C.7	Dostęp do zasobów obliczeniowych: biblioteki i solvery	LAPACK [1.2.2.2]	Dostęp do nowych solverów (więcej szczegółów w dokumencie) [release notes, v. 1.4]
C.8	Dostęp do zasobów obliczeniowych: systemów	• proxy do NetSolve	• proxy dla Globus (możliwe jest zlecanie zadań Globusowi) [release notes, v. 1.4] • proxy dla NetSolve

D	**Bezpieczeństwo**		
D.1	Bezpieczeństwo: autoryzacja, autentyfikacja	• SSL: *certificate chain*, *policy class*, • NAA (NES *Authentication Authorization*), przestrzeń nazw NAA przechowywana w katalogach X.509, granty, pozwolenia, opisy polityk w XML [1.2.8.4]	Kerberos: *single sign-on* do Kerberos, autoryzacja użytkownika w stosunku do serwera [release notes, v. 1.4]
E	**Techniki dla niezawodności**		
E.1	Tolerowanie uszkodzeń: wykrywanie uszkodzeń	Zarządca awarii: np. usuwa serwer, który uległ awarii z bazy danych zasobów [1.2.8.4]	Wykrywanie uszkodzeń [1.3.6]: • brak połączenia z serwerem obliczeniowym powoduje jego „wyrejestrowanie" z systemu [1.3.6.1]; • jeśli serwer jest niedostępny przez 24 h usuwane są z bazy wszystkie wpisy z nim związane; podobnie jeśli wykazuje dużą stopę błędów [1.3.6.3] • do wykrywania uszkodzeń wykorzystać można Globus Heartbeat Monitor i NWS (brak informacji szczegółowych) • agenci NetSolve śledzą stopę błędów dla poszczególnych maszyn (błędy: zanik procesów obliczeniowych, błędy obliczeń) [1.3.6.3]
E.2	Tolerowanie uszkodzeń: reakcje na uszkodzenia	Jeśli wszystkie serwery obl. są obciążone, metaserwer buforuje argumenty obliczeń (przeźroczyste dla klienta) [1.2.5.3]	Omijanie uszkodzeń [1.3.6.2]: • klient otrzymuje od agenta listę serwerów obliczeniowych, klient próbuje uruchomić obliczenia na kolejnych serwerach • jeśli „zaginie" proces obliczeniowy, uszkodzenie wykrywane jest przez klienta (raczej przez agenta, jak wynika z [„Rozwiązania wdrożone", 1.3.6.5] (stan na rok 1999) natomiast klient tylko restartuje obliczenia) i problemy wysyłany jest do innego serwera obliczeniowego
E.3	Tolerowanie uszkodzeń: przechowywanie pośrednich wyników obliczeń	składnice danych (*data storage*) – służą do tymczasowego przechowywania pośrednich rezultatów obliczeń (tych które są wymieniane pomiędzy serwerami) [1.2.8.4] w razie awarii węzła obliczeniowego dane są nadal dostępne	• plany integracji z usługami przechowywania danych [1.3.6.4] • integracja z IBP: pośrednie dane obliczeń składowane w *storage servers* (IBP) [release notes, v. 1.4] w razie awarii węzła obliczeniowego dane zadania nadal dostępne
F	**Techniki dla wysokiej wydajności**		
F.1	Techniki dla wysokiej wydajności: szeregowanie grup wywołań	• Niezależne wywołania funkcji są wykonywane współbieżnie na różnych serwerach obliczeniowych. [1.2.5.6] • brak szczegółów na temat mechanizmu przepływu obliczeń w ramach transakcji! • patrz też „Dynamiczny load balancing" • minus: obliczenia w ramach transakcji nie mogą rozpocząć się wcześniej niż zostaną zbuforowane argumenty wszystkich składowych transakcji; podobnie wynik zostanie zwrócony po zakończeniu wszystkich składników transakcji;	• optymalizacja przesyłu danych dla obliczeń – sekwencjonowanie żądań – *request sequencing*: techniki szeregowania wywołań zgrupowanych wykorzystujące model DAG (ang. Directed Acyclic Graph) [1.3.7.1] Uwaga! W omawianej w pracy [11] wersji systemu NetSolve agent szereguje wykonanie grupy funkcji zawsze na jednym serwerze. W [release notes, v. 1.4] nie ma jawnych informacji, czy nadal jest szeregowane na jedną maszynę czy już to poprawiono. • Szeregowanie zadań dobrze podzielnych – *task farming*: wartswa ponad tradycyjnym API NetSolve – nie wcielone do systemu; więcej

			szczegółów w [1.3.7.2]
F.2	uruchamianie obliczeń: buforowanie informacji o interfejsach funkcji matematycznych w module klienta	jest [1.2.3.1]	brak / brak informacji
F.3	Dynamiczny *load balancing*	• Pozwala na przydzielanie pracy tym węzłom, które wcześniej skończyły poprzedni etap obliczeń, przez co uzyskuje się pewne wyrównanie obciążenia i minimalizację czasu wykonania całego zadania [1.2.5.7] • Możliwy poprzez: odpowiednie wywołania zagregowane (całkowicie Ninf-driven) • mechanizm wywołań zwrotnych (częściowo programista-driven) [1.2.5.7]	Wywołania zagregowane są [1.3.7.1], ale nie mówi się o tym, że służą dynamicznemu load-balancing'owi.
F.4	wysoka wydajność, architektura systemu	składnice danych [1.2.8.4]	Integracja z IBP: Można składować dane do obliczeń w magazynach danych. Możliwe jest tworzenie, usuwanie, odczyt i zapis obiektów poprzez IBP i używanie ich jako danych do obliczeń. Użytkownik może uruchamiać dane na zdalnych danych i odbierać tylko odpowiednie rezultaty [release notes, v. 1.4]
G	**Komunikacja, protokoły**		
F.5	Komunikacja klient-serwer	TCP/IP + Sun XDR [1.2.2.3]	TCP/IP + Sun XDR [1.3.2.2]
F.6	Sposób wykonywania połączeń	*Continous connections* *by proxy* (tylko dane sterujące); argumenty bezpośrednio, chyba, że trzeba omijać firewall [1.2.8.4]	połączenia tymczasowe, bezpośrednio lub przez proxy
F.7	Protokół komunikacyjny	Komendy tekstowe, XML [1.2.8.4]. dodano potwierdzenia odbioru komunikatów [rys. 1.5], [1.2.8.4]	brak informacji
F.8	Proxy klienta	• reprezentuje grupę węzłów przy pomiarze przepustowości sieci; • działa jako serwer-proxy dla sieci zabezpieczonych firewall'ami • przy zapytaniu klienta o uszeregowanie dodaje do tego zapytania swoje dane o stanie sieci; • przejmuje informacje od metaserwera o przydzielonym serwerze i kieruje tam argumenty obliczeń [1.2.5.3]	działa jako serwer-proxy dla sieci zabezpieczonych firewall'ami

W tabeli, w przypadku kilku cech i funkcji systemów zdalnych obliczeń zamieszczono informacje dotyczące jednego z nich, natomiast brakuje informacji dotyczących drugiego. Wynika to z faktu, że tabela bazuje na artykułach i dokumentacji systemów dostępnych do początku roku 2002 a materiały te nie zawierają wszystkich informacji niezbędnych dla porównania systemów.

3. Testy funkcjonalności i analiza kodów źródłowych

Przedstawione porównanie systemów *Ninf* i *NetSolve* bazuje na analizie artykułów, prezentacji i dokumentacji tych systemów.

PCSS przeprowadziło analizę kodów źródłowych systemów a także testy ich funkcjonalności. W ramach testów funkcjonalności przeprowadzono serię eksperymentów zmierzających do sprawdzenia, czy deklarowane w literaturze dotyczącej systemów *Ninf* i *NetSolve* funkcje i mechanizmy są zrealizowane i działają sprawnie.

Analiza kodow źródłowych i testy funkcjonalności przeprowadzono dla pakietu *Ninf* w wersji 1 oraz *NetSolve* w wersji 1.4.1.

3.1. Wyniki analizy i testów

Poniższa tabela uporządkowana jest według cech systemów zestawionych w punkcie 2. Uzupełniona jest o informacje, które uzyskano w trakcie analizy kodów źródłowych i testów. Informacje sprawdzone na drodze analizy kodów źródłowych oznaczono w tabeli słowem "SRC" natomiast informacje sprawdzone w drodze testów oznaczono słowem "TST". Jeżeli informacje podane przez autorów dokumentów zostały potwierdzone pozytywnie w tabeli oznaczone są przedrostkiem "T-", jeśli zaś negatywnie, oznaczone są przedrostkiem "N-".

Informacje, które nie zostały sprawdzone oznaczone są symbolem "X". Na tym etapie prac cechy, których sprawdzenie pominięto nie są istotne dla dalszych prac w ramach projektu.

Tab. 2 Podsumowanie praktycznej analizy cech systemów Ninf i NetSolve

Id	Cecha (feature)	Ninf	NetSolve
A	**Dostęp do systemu**		
A.1	interfejsy programistyczne:	• API: C **[T-TST]**, Fortran **[T-TST]** i Java **[T-SRC]** • Sub-API: może być wykorzystane przez wszystkie języki, które obsługują „obce interfejsy" do języka C; **[T-SRC]** • Ninf-CIM: dla języków/środowisk o bardzo odmiennej reprezentacji danych (szczególnie tablic), np. Excel, Mathematica, Lisp; „funkcje-przemierzacze" wywoływane przez funkcję `Ninf_cim_main()` **[T-SRC]** [1.2.3.6]	• API: Fortran **[T-TST]**, C **[T-TST]**, Java **[N-SRC]** [1.3.4]
A.2	interfejsy interaktywne	• Mathematica **[T-SRC]** • Excel **[T-SRC]** [1.2.3.6]	• Web based Java Gui **[X]** • Matlab **[T-SRC]** • shell Unix [1.3.4] **[N-SRC]** • Mathematica [release notes, v.1.4] **[T-SRC]**
A.3	API:	automatyczne określanie stopnia funkcji na podstawie wyróżnionych argumentów funkcji [1.2.3] **[T-SRC]**	brak informacji n.t. automatycznego określania stopnia funkcji na podstawie argumentów funkcji [1.2.3] **zrealizowane [T-SRC]**
A.4	API: asynchroniczne	• `Ninf_call_async`	• `Netslnb, netslpb, netswt`

	wywołania funkcji	[1.2.3.3] **[T-TST]**	[1.3.4] **[T-TST]**

A.5	API: wywołania zagregowane	„transakcje Ninf”
		`Ninf_transaction_begin();`
		`Ninf_transaction_end();`
		[1.2.3.4] **[T-TST]**
			„sekwencja żądań”
			`begin_sequence();`
			`end_sequence(...);`
			[1.3.7.1] **[T-TST]**

A.6	API: mechanizm wywołania zwrotnego	jest [1.2.2.4] i [1.2.3.5] **[T-TST]**	**zrealizowane [T-SRC,T-TST]**
A.7	możliwość pobierania / umieszczania danych z/na www	jest [1.2.3.2] **[T-SRC]**	brak informacji, **nie ma [T-SRC]**
B	**Szeregowanie zadań i zarządzanie zasobami**		
B.1	Metaserwery: kooperacja	Metaserwery mogą kooperować, zmieniające się informacje są propagowane pomiędzy metaserwerami, ew. nie posiadane przez dany metaserwer informacje mogą być ściągane od innych na żądanie [1.2.5.5] **[T-TST]**	Każda instancja agenta może mieć swój własny ogląd systemu. Nie musi być on identyczny. Można będzie ich łączyć w struktury hierarchiczne (plany) [1.3.2.1] **[T-TST]**
B.2	Moduł szeregujący – otwartość, wymienność algorytmu szeregowania	• zaimplementowany w Javie; **[T-SRC]** • można wymienić algorytm szeregowania [1.2.8.4] **[T-SRC]** • można zmieniać politykę szeregowania „w locie” za pomocą odpowiednich protokołów sieciowych [1.2.5.4] **[X]**	moduł szeregujący całkowicie zaimplementowany w C; dobry i prosty interfejs do schedulera **[T-SRC]**
B.3	Moduł szeregowania – uwzględnianie rozmieszczenia danych	uwzględnia przy szeregowaniu lokalizację plików używanych w obliczeniach [1.2.8.4] **[T-SRC]**	było w planach [„Wstępna ocena przyjętych rozwiązań”, 1.3.7.1]; w [release notes, v.1.4] brak omówienia; **zrealizowano IBP [T-SRC],** **nie potwierdzone informacje, czy rozmieszczenie plików w IBP jest brane pod uwagę przy szeregowaniu żądań**
B.4	Szeregowanie – prognozowanie dostępności zasobów	metody własne (brak szczegółów) utrzymuje następujące informacje o serwerach obliczeniowych: • lokalizację serwera (adres IP i numer portu), • listę funkcji bibliotecznych zarejestrowanych na serwerze, • dystans do serwera z uwzględnieniem przepustowości sieci, • informacje o zdolności obliczeniowej serwera (prędkość zegara, wskaźnik Flops – ang. floating point operations per second), • stan serwera, włączając średnie obciążenie (ang. load average). [1.2.5.5] **[T-SRC]**	Integracja z NWS [release notes, v. 1.4] zrealizowano **[T-SRC]**
B.5	Szeregowanie – monitorowanie dostępności zasobów	własne metody (brak szczegółów) **[T-SRC]**	• zbierana periodycznie przez agenta (nie na żądanie); serwer obliczeniowy rozgłasza *workload* tylko gdy wielkość zmiany przekracza wielkość przedziału ufności [1.3.5.4] (stare rozwiązanie); **[T-SRC]** • uwaga! w wersji 1.4. i późniejszych jest zintegrowany NWS [release notes, v. 1.4]; możliwe również działanie wg metod własnych **[T-SRC]**
B.6	Szeregowanie - informacja o funkcjach	• plik opisowy funkcji (IDL) nie zawiera przybliżonego określenie złożoności obliczeniowej algorytmu w funkcji rozmiaru problemu **[N-SRC]**	• plik opisowy funkcji (IDL) zawiera przybliżone określenie złożoności obliczeniowej algorytmu w funkcji rozmiaru problemu [1.3.3.3]
B.7	Szeregowanie - obliczanie najlepszej maszyny - model wydajności	• prosty teoretyczny model wydajności **[T-SRC]**	• stosuje się prosty teoretyczny model wydajności ("surowa wydajność" i obciążenie) **[T-SRC]**
B.8	Zarządzanie systemem	• baza danych zasobów dostępna przez	Skrypty CGI na www pozwalające

	(dane o zasobach)	protokół LDAP • opcjonalnie Globus MDS jeśli są dostępne w środowisku **[N-SRC]** [1.2.8.4]	uzyskać: • listę instancji agentów lub zasobów obliczeniowych występujących w danym systemie (lista nazw maszyn i adresów IP) **[T-TST]** [1.3.2.3]
B.9	Metaserwer – rejestracja zasobów	dynamiczna (wynika to nie wprost z [1.2.5.5]) **[T-TST]**	dynamiczna; przy włączaniu serwerów obliczeniowych; przy rejestracji podawane są informacje takie jak lokalizacja, adres, lista rozwiązywanych problemów [1.3.3.3] **[T-TST]**
B.10	Metaserwer - spis dostępnych problemów obliczeniowych	• użytkownicy mogą przeglądać spis funkcji dostępnych na serwerze Ninf za pomocą przeglądarki web [1.2.4.1] **[T-SRC]**	• listę problemów (zadań), które można rozwiązać wewnątrz danego systemu NetSolve [1.3.2.3] **[T-TST]**
B.11	Metaserwer - uruchamianie zadań w sytuacji pełnego obciążenia dostępnych zasobów	Serwery obliczeniowe przydzielane są do zadań aż do wyczerpania. Jeżeli nie ma wolnych serwerów, metaserwer przechowuje wszystkie dane potrzebne do uruchomienia obliczeń aż do momentu zwolnienia się zasobów. (przeźroczyste dla użytkownika) [1.2.5.3] **[T-SRC]**	przydzielane są maszyny, które według przewidywań zapewniają największą wydajność, nie ma sytuacji, w której stwierdza, że wszystkie serwery są zajęte **[T-SRC]**
C	**Serwer obliczeniowy**		
C.1	Przygotowanie funkcji bibliotecznych i zasobów obliczeniowych dla systemu	Opis [1.2.2.2]: • przygotowanie opisu w IDL, • kompilacja IDL (wynikiem są pliki nagłówkowe i kody stopki), • kompilacja biblioteki obl. • łączenie obiektów programowych z funkcjami Ninf RPC • rejestracja funkcji bibliotecznych w serwerze Ninf Generator interfejsów: • pobiera IDL, • tworzy programy-stopki i pliki makefile służące do tworzenia obiektów wykonywalnych Ninf poprzez łączenie programów stopek i funkcji bibliotecznych [1.2.4.1] **[T-SRC, T-TST]**	Kompilacja IDL: • pseudo-kompilator pobiera plik konfiguracyjny opisujący każdy problem obliczeniowy w sposób formalny i generuje kod w języku C dla procesu obliczeniowego odpowiedzialnego za rozwiązywanie problemu; **[T-TST]** • wywołania funkcji bibliotecznych muszą być napisane w języku C z użyciem predefiniowanego zbioru makr **[T-TST]** • po skompilowaniu opisów powstają moduły obliczeniowe (wykonywalne) – *wrapper*'y – będące opakowaniem do bibliotek naukowych; **te moduły to biblioteki zawierające wszystkie funkcje danej grupy (zdefiniowane w jednym pliku IDL) [T-TST]** • można przeglądać listę rozwiązywanych problemów i zasobów sprzętowych na www **[T-TST]** [1.3.3.2]
C.2	sposób uruchamiania oprogramowania na serwerach obliczeniowych	*executable modules* – powstają przez łączenie programów stopek i funkcji bibliotecznych [1.2.4.1] **[T-TST]**	wywołanie następuje przez wywołanie funkcji bibliotecznej, z wnętrza programu wykonywalnego zawierającego "wrappery" i złaczonego z biblioteką udostępnianych funkcji **[T-SRC, T-TST]**
C.3	IDL: zawartość pliku z opisem	• lista argumentów, • specyfikatory dostępu (in/out), • parametry (wyrażenia) pozwalające obliczyć rozmiar danych; • informacje niezbędne do kompilacji i łączenia (jaka biblioteka itp.) [1.2.4] **[T-SRC, T-TST]**	• oczekiwane dane wejściowe i jakie są spodziewane dane wyjściowe • jaka biblioteka, • przybliżone określenie złożoności obliczeniowej [1.3.3.3] **[T-SRC, T-TST]**
C.4	IDL: typy danych	skalary i wielowymiarowe tablice [1.2.4] **[T-SRC]**	<object, data> object = MATRIX, VECTOR, SCALAR data = dowolny typ danych jezyka Fortran [1.3.1.1] **[T-SRC]**
C.5	IDL: określanie rozmiarów tablic na podstawie argumentów wejściowych	jest [1.2.4] **[T-SRC]**	jest **[T-SRC]**

C.6	Narzędzia dla IDL	brak **[T-SRC]**	• Graficzny edytor IDL [1.3.3.2]: *PDF* (*Problem Description File*) *Generator* + *Java GUI* [release notes, v. 1.4] • **Istnieje** planowane w [1.3.3.3] stworzenie repozytorium opisów problemów na www, by można było ich używać do konfiguracji dowolnych serwerów obliczeniowych. **[T-TST]**
C.7	Dostęp do zasobów obliczeniowych: biblioteki i solvery	LAPACK [1.2.2.2], LINPACK **[T-SRC]**	Dostęp do nowych solverów (więcej szczegółów w dokumencie); [release notes, v. 1.4] **ogólnie większa liczba wcielonych bibliotek:** **LAPACK, arpack, blas_subset, fftpack, fitpack, itpack, mandelbrot, petsc_user_func, scalapack, sparse_direct_solve, sparse_iterative_solve,** **[T-SRC]**
C.8	Dostęp do zasobów obliczeniowych: systemów	• proxy do NetSolve **[T-SRC]**, **proxy dla Globus w Ninf-G [T-SRC]**	• proxy dla Globus (możliwe jest zlecanie zadań Globusowi) [release notes, v. 1.4] **[T-SRC]**
D	**Bezpieczeństwo**		
D.1	Bezpieczeństwo: autoryzacja, autentyfikacja	• SSL: *certificate chain*, *policy class*, • NAA (NES *Authentication Authorization*), przestrzeń nazw NAA przechowywana w katalogach X.509, granty, pozwolenia, opisy polityk w XML [1.2.8.4] **[T-SRC]**	Kerberos: *single sign-on* do Kerberos, autoryzacja użytkownika w stosunku do serwera [release notes, v. 1.4] **[T-SRC]**
E	**Techniki dla niezawodności**		
E.1	Tolerowanie uszkodzeń: wykrywanie uszkodzeń	Zarządca awarii: np. usuwa serwer, który uległ awarii z bazy danych zasobów [1.2.8.4] **[T-SRC]**	Wykrywanie uszkodzeń [1.3.6]: • brak połączenia z serwerem obliczeniowym powoduje jego „wyrejestrowanie" z systemu [1.3.6.1]; **[T-TST]** • jeśli serwer jest niedostępny przez 24 h usuwane są z bazy wszystkie wpisy z nim związane; podobnie jeśli wykazuje dużą stopę błędów [1.3.6.3] **[X]** • do wykrywania uszkodzeń wykorzystać można Globus Heartbeat Monitor i NWS (brak informacji szczegółowych) **[T-SRC]** • agenci NetSolve śledzą stopę błędów dla poszczególnych maszyn (błędy: zanik procesów obliczeniowych, błędy obliczeń) [1.3.6.3] **[T-SRC]**
E.2	Tolerowanie uszkodzeń: reakcje na uszkodzenia	Jeśli wszystkie serwery obl. są obciążone, metaserwer buforuje argumenty obliczeń (przeźroczyste dla klienta) [1.2.5.3] **[T-SRC]**	Omijanie uszkodzeń [1.3.6.2]: • klient otrzymuje od agenta listę serwerów obliczeniowych, klient próbuje uruchomić obliczenia na kolejnych serwerach **[T-TST]** • jeśli „zaginie" proces obliczeniowy, uszkodzenie wykrywane jest przez agenta [„Rozwiązania wdrożone", 1.3.6.5] natomiast klient tylko restartuje obliczenia) i problemy wysyłany jest do innego serwera obliczeniowego **[T-SRC]**
E.3	Tolerowanie uszkodzeń: przechowywanie pośrednich wyników obliczeń	składnice danych (*data storage*) – służą do tymczasowego przechowywania pośrednich rezultatów obliczeń (tych, które są wymieniane pomiędzy serwerami) [1.2.8.4] w razie awarii węzła obliczeniowego dane są nadal dostępne **[T-SRC]**	• integracja z IBP: pośrednie dane obliczeń składowane w *storage servers* (IBP) [release notes, v. 1.4] w razie awarii węzła obliczeniowego dane zadania nadal dostępne **[T-SRC]**

F	**Techniki dla wysokiej wydajności**		
F.1	Techniki dla wysokiej wydajności: szeregowanie grup wywołań	• Niezależne wywołania funkcji są wykonywane współbieżnie na różnych serwerach obliczeniowych. [1.2.5.6] • brak szczegółów na temat mechanizmu przepływu obliczeń w ramach transakcji • patrz też „Dynamiczny load balancing" • minus: obliczenia w ramach transakcji nie mogą rozpocząć się wcześniej niż zostaną zbuforowane argumenty wszystkich składowych transakcji; podobnie wynik zostanie zwrócony po zakończeniu wszystkich składników transakcji **[X]**	• optymalizacja przesyłu danych dla obliczeń – sekwencjonowanie żądań – *request sequencing*: techniki szeregowania wywołań zgrupowanych wykorzystujące model DAG (ang. Directed Acyclic Graph) [1.3.7.1] Uwaga! W omawianej w pracy [11] wersji systemu NetSolve agent szereguje wykonanie grupy funkcji zawsze na jednym serwerze. W wersji 1.4.1 nadal jest szeregowane na jedną maszynę. **[T-TST]** • Szeregowanie zadań dobrze podzielnych – *task farming*: wartswa ponad tradycyjnym API NetSolve – nie wcielone do systemu; więcej szczegółów w [1.3.7.2] **[T-TST]**
F.2	uruchamianie obliczeń: buforowanie informacji o interfejsach funkcji matematycznych w module klienta	jest [1.2.3.1] **[T-SRC]**	brak **[T-SRC, T-TST]**
F.3	Dynamiczny *load balancing*	• Pozwala na przydzielanie pracy tym węzłom, które wcześniej skończyły poprzedni etap obliczeń, przez co uzyskuje się pewne wyrównanie obciążenia i minimalizację czasu wykonania całego zadania [1.2.5.7] **[T-SRC]** • Możliwy poprzez: odpowiednie wywołania zagregowane (całkowicie Ninf-driven) **[T-SRC]** • mechanizm wywołań zwrotnych (częściowo programista-driven) [1.2.5.7] **[T-SRC]**	Po każdym uszeregowaniu agent aktualizuje informacje o obciążeniu maszyn (powiększa je o przewidywane przez agenta obciążenie związane z uszeregowanym zadaniem). W ten sposób agent dynamicznie "dociąża" mniej obciążone serwery obliczeniowe. **[T-SRC]**
F.4	wysoka wydajność, architektura systemu	składnice danych [1.2.8.4] **[X]**	Integracja z IBP: Można składować dane do obliczeń w magazynach danych. Możliwe jest tworzenie, usuwanie, odczyt i zapis obiektów poprzez IBP i używanie ich jako danych do obliczeń. Użytkownik może uruchamiać dane na zdalnych danych i odbierać tylko odpowiednie rezultaty [release notes, v. 1.4] **[T-SRC, T-TST]**
G	**Komunikacja, protokoły**		
F.5	Komunikacja klient-serwer	TCP/IP + Sun XDR [1.2.2.3] **[T-SRC]**	TCP/IP + Sun XDR [1.3.2.2] **[T-SRC]**
F.6	Sposób wykonywania połączeń	*Continous connections* *by proxy* (tylko dane sterujące); argumenty bezpośrednio, chyba, że trzeba omijać firewall [1.2.8.4] **[T-SRC]**	połączenia tymczasowe, bezpośrednio lub przez proxy **[T-TST]**
F.7	Protokół komunikacyjny	Komendy tekstowe, XML [1.2.8.4]. dodano potwierdzenia odbioru komunikatów [rys. 1.5], [1.2.8.4] **[T-SRC]**	brak informacji **[X]**
F.8	Proxy klienta	• reprezentuje grupę węzłów przy pomiarze przepustowości sieci; **[T-SRC]** • działa jako serwer-proxy dla sieci zabezpieczonych firewall'ami • przy zapytaniu klienta o uszeregowanie dodaje do tego zapytania swoje dane o stanie sieci; **[T-SRC]** • przejmuje informacje od metaserwera o	działa jako serwer-proxy dla sieci zabezpieczonych firewall'ami **[T-TST]**

		przydzielonym serwerze i kieruje tam argumenty obliczeń **[T-SRC]** [1.2.5.3]	

3.2. Wnioski z analizy kodów źródłowych i testów

Przeprowadzone analiza kodów źródłowych i systemów testy potwierdziły, że większość cech systemów *Ninf* i *NetSolve* deklarowanych w artykułach, prezentacjach i dokumentach jest zrealizowana. Przeprowadzone badanie potwierdziły przydatność systemów Ninf i NetSolve do budowy systemu udostępniania bibliotek matematycznych.

Rozszerzenie funkcjonalności systemów udostępniania, które dokonane zostanie w ramach projektu celowego zostanie przetestowane dla jednego z środowisk.

Zdaniem PCSS bardziej nadaje się do tego celu system *NetSolve*. Z dostępnych nam informacji wynika, iż z systemem *NetSolve* zintegrowano większą liczbę bibliotek matematycznych niż z systemem *Ninf*. Jest to ważny argument, gdy wziąć pod uwagę różnorodność zainstalowanego i wykorzystywanego w ośrodkach obliczeniowych w Polsce oprogramowania. Większa liczba dostępnych pakietów pozwoli również dokładniej przetestować sprawność opracowywanych w ramach projektu celowego rozszerzeń i narzedzi.

Za wykorzystaniem systemu *NetSolve* przemawia także bogatszy niż w systemie *Ninf* język opisu interfejsów funkcji (*IDL*). Pozwala on m.in. określać informacje związane ze złożonością czasową algorytmu realizującego daną operację matematyczną. W ramach projektu celowego rozszerzana będzie funkcjonalność modułu szeregującego agenta. Jest to jedno z podstawowych zadań. *IDL* systemu *NetSolve* daje lepsze podstawy dla opracowywanej techniki predykcji czasu wykonania funkcji matematycznych.

Kolejnym argumentem przemawiającym za użyciem systemu *NetSolve* jest fakt, że moduły "robocze" (moduł klienta, agent, moduł serwerowy) zostały całkowicie zaimplementowane w języku *C*. Jedynie narzędzia dla użytkownika końcowego stworzone zostały z wykorzystaniem technologii *Java*. W systemie *Ninf* metaserwer (odpowiednik agenta w *NetSolve*) napisany został w języku *Java*. Zdaniem PCSS może to mieć negatywny wpływ na wydajność meta-serwera przy dużej liczbie zleceń.

Dla prowadzenia dalszych prac stworzono instalację testową na bazie systemu *NetSolve*. System ten został zainstalowany na komputerachw PCSS w Poznaniu oraz w Centrum Informatycznym TASK w Gdańsku. Informacje dotyczące instalacji testowej znajdują się w punkcie dokumentu.

4. Instalacja testowa

Jako system bazowy dla opracowywanego w ramach zadania WP 2.1 projektu celowego wybrano system *NetSolve*. Dla prowadzenia dalszych prac w ramach zadania stworzono instalację testową na bazie tego systemu. System został zainstalowany na komputerach w PCSS w Poznaniu oraz w Centrum Informatycznym TASK w Gdańsku.

Instalacja obejmowała instalację bibliotek matematycznych BLAS i LAPACK na systemach komputerowych, na których byłby one wcześniej niedostępne oraz instalację systemu NetSolve.

Tabela 3 zawiera listę maszyn, które zostały objęte instalacją testową. Na komputerach wymienionych w tabeli 3 zainstalowane zostały kompletne dystrybucje systemu NetSolve. Dla sprawdzenia poprawności konfiguracji przeprowadzono wszystkie testy zawarte w standardowym pakiecie instalacyjnym NetSolve. Tabela 4 zawiera lista komputerów, na których trwają prace związane z konfiguracją systemu (instalacja została przeprowadzona prawidłowo, napotkano jednak problemy przy integracji z bibliotekami matematycznymi instniejącymi na maszynach). Przewidywany termin zakończenia prac to koniec lipca 2003 r.

Tabela 5 zawiera listę maszyn, o które rozszerzona zostanie instalacja testowa do końca lipca 2003 r. Decyzję o rozszerzeniu instalacji testowej ośrodki podjęły w czerwcu 2003 r. Przewidywany termin zakończenia prac - koniec lipca 2003 r.

Tab. 3 Lista komputerów objętych instalacją testową systemu NetSolve (stan na koniec czerwca 2003)

Ośrodek	Komputer	System operacyjny	nazwa domenowa maszyny	Liczba procesorów	Typ procesora	RAM
PCSS	SGI Origin 3800	Irix 6.5	grape.man.poznan.pl	128	R12000, 400 MHz	80 GB
PCSS	SGI Onyx 2	Irix 6.5	cactus.man.poznan.pl	8	R10000, 185 MHz	6 GB
PCSS	SGI Power Challenge 10000 L	Irix 6.5	elder.man.poznan.pl	4	R10000, 195 MHz	384 MB
PCSS	SGI Power Challenge XL	Irix 6.5	tulip.man.poznan.pl	12	R8000, 90 MHz	1 GB
PCSS	Dell Power Edge 6300	Red Hat Linux 8.0	cpcsingle-atrium.man.poznan.pl	1	Intel Xeon 500 MHz	256 MB
PCSS	Dell Power Edge 6300	Red Hat Linux 8.0	cpcdual-atrium.man.poznan.pl	2	Intel Xeon 500 MHz	512 MB
TASK	Klaster IA32 Xeon/Dolphin	Debian Linux 3.0	galera.task.gda.pl.man.poznan.pl	128	Intel Xeon 700 MHz	16 GB

Tab. 4 Lista komputerów, na których trwają jeszcze prace związane z instalacją testową (stan na koniec czerwca 2003)

Ośrodek	Komputer	System operacyjny	nazwa domenowa maszyny	Liczba procesorów	Typ procesora	RAM
PCSS	Cray T3E	UNICOS/mk	lotus.man.poznan.pl	8	450 MHz	1 GB
PCSS	Cray SV1	UNICOS 10.0.0.7	croton.man.poznan.pl	8	SV1 300 MHz	16 GB
PCSS	Cray J916	UNICOS 8.04	pink.man.poznan.pl	16	C90	4 GB

Tab. 5 Lista maszyn, o które zostanie rozszerzona instalacja testowa

Ośrodek	Komputer	System operacyjny	nazwa domenowa maszyny	Liczba procesorów	Typ procesora	RAM
PCSS	SGI Challenge L	Irix 6.5	melisa.man.poznan.pl	4	MIPS R4400, 150 MHz	384 MB
PCSS	IBM/SP2	AIX 4.1.4	clover.man.poznan.pl	14+1 węzłów	POWER2	64MB/węzeł
TASK	SGI Origin 2000 + Onyx 2	Orix 6.5	fregata.task.gda.pl		8x MIPS R1000 195 MHz + 16x MIPS R12000 300 MHz	16 GB
TASK	SGI Octane	Irix - która wersja	pancernik.task.gda.pl		2x MIPS R12000 400 MHz	512 MB, cache L1 65 K, L2 2MB

Bibliografia

[1] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, H. Takagi: Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, HPCN'97 (LNCS-1225), pp. 491-502", 1997" (http://ninf.apgrid.org/papers/hpcn97/hpcn97-paper.pdf)

[2] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, Umpei Nagashima: Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms, 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8), pp. 97-104, 1999, (http://ninf.apgrid.org/papers/hpdc99takefusa/hpdc99takefusa.pdf)

[3] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, U. Nagashima: A Performance Evaluation Model for Effective Job Scheduling in Global Computing Systems, 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8), pp. 97-104, 1999 (http://ninf.apgrid.org/papers/nasa98aida/slide.ppt)

[4] S. Matsuoka, H. Ogawa, A. Takefusa, H. Nakada, K. Aida, U.Nagashima, M. Sato, S. Sekiguchi: Preliminary Evaluation of Scheduling in Ninf: a Global Computing System, International Workshop on Innovative Architectures '97, 1997 (http://ninf.apgrid.org/papers/iwia97/iwia97.ps)

[5] H. Nakada, H. Takagi, S.Matsuoka, E.Nagashima, M. Sato, S. Sekiguchi: Utilizing the Metaserver Architecture in the Ninf Global Computing System, High-Performance Computing and Networking '98, LNCS 1401, pp.607-616, 1998 (http://ninf.apgrid.org/papers/hpcn98/hpcn98.ps.Z)

[6] Y. Tanaka, M. Sato, M. Hirano, H. Nakada, S. Sekiguchi: Resource Manager for Globus-based Wide-area Cluster Computing, 1st IEEE International Workshop on Cluster Computing (IWCC'99), pp.237-244, 1999 (http://ninf.apgrid.org/papers/iwcc99tanaka/IWCC99.pdf)

[7] S. Matsuoka, H. Nakada, M. Sato, S. Sekiguchi: Design issues of Network Enabled Server Systems for Grid, Grid Computing - GRID 2000, Springer-Verlag, LNCS 1971, pp.4-17, 2000, (http://ninf.apgrid.org/papers/grid2000matsuoka/grid2000matsuoka.pdf)

[8] H. Casanova, J. Dongarra: NetSolve: A Network Server for Solving Computational Science Problems, The International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Number 3, pp 212-223, 1997; również: proceedings of Supercomputing'96, Pittsburgh, 1996, (http://icl.cs.utk.edu/netsolve/files/pubs/general.ps)

[9] H. Casanova, J. Dongarra: NetSolve: A Network Enabled Server, Examples and Users. Henri Casanova and Jack Dongarra. Proceedings of the Heterogeneous Computing Workshop, Orlando, Florida, 1998 (http://icl.cs.utk.edu/netsolve/files/pubs/examples-apps.ps)

[10] J. Plank, H. Casanova, M. Beck, J. Dongarra: Deploying Fault-tolerance and Task Migration with NetSolve. To appear in The International Journal on Future Generation Computer Systems, 1999 (http://icl.cs.utk.edu/netsolve/files/pubs/fault-tolerance.ps)

[11] D. Arnold, D. Bachmann, J. Dongarra: Request Sequencing: Optimizing Communication for the Grid. To appear in Euro-Par 2000 - Parallel Processing, 2000, (http://icl.cs.utk.edu/netsolve/files/pubs/sequencing.ps)

[12] H. Casanova, M. Kim, J. Plank, J. Dongarra: Adaptive Scheduling for Task Farming with Grid Middleware. To appear in The International Journal of Supercomputer Applications and High Performance Computing, 1999 (http://icl.cs.utk.edu/netsolve/files/pubs/farming.ps)

[13] P. Arbenz, W. Gander, and M. Oettli: The Remote Computation System [23 Kbyte], In High Performance Computing and Networking, H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot (eds.). Springer-Verlag, Berlin, 1996, pp. 820-825. (Lecture Notes in Computer Science, 1067) (http://www.inf.ethz.ch/~arbenz/HPCN96_rcs.ps.gz)

[14] P. Arbenz, W. Gander, and M. Oettli: The Remote Computation System. Technical Report 245, Institute of Scientific Computing, ETH Zürich, April 1996 (ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/245.ps.gz)

[15] Michael C. Ferris, Michael Mesnier and Jorge J. More NEOS and CONDOR: Solving Optimization Problems over the Internet. Preprint ANL/MCS-P708-0398, March 1998. Mathematical Programming Technical Report 96-08, October 1996. (ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-08.pdf)

[16] S. Fields: Hunting for Wasted Computing Power. New Software for Computing Networks Puts Idle PC's to Work, 1993, (http://www.cs.wisc.edu/condor/doc/WiscIdea.html)

[17] S. Fields: An Overview of the Condor System (http://www.cs.wisc.edu/condor/overview)

[18] Rich Wolski: Dynamically Forecasting Network Performance Using the Network Weather Service. Journal of Cluster Computing, Volume 1, pp. 119-132, January, 1998. (http://www.cs.ucsb.edu/~rich/publications/nws-tr.ps.gz)

[19] R. Wolski, N. Spring, J. Hayes: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768, October, 1999. (http://www.cs.ucsb.edu/~rich/publications/nws-arch.ps.gz)

[20] R. Wolski, N. Spring, J. Hayes: Predicting the CPU Availability of Time-shared Unix Systems. Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8), August, 1999. (http://www.cs.ucsb.edu/~rich/publications/nws-cpu.ps.gz)

[21] R. Wolski, N. Spring, C. Peterson: Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service, in Proceedings of SC97, November, 1997. (http://www.cs.ucsb.edu/~rich/publications/nws-impl.ps.gz)

[22] D. Zagorodnov, F. Berman, R. Wolski: Application Scheduling on the Information Power Grid, 1998, http://apples.ucsd.edu (hpdc98.ps)

[23] F. Berman, R. Wolski: Scheduling from the Perspective of the Application. Proceedings of Symposium on High Performance Distributed Computing, 1996. (http://apples.ucsd.edu/pubs/hpdc96.ps)

[24] F. Berman, R. Wolski: The AppLeS Project: A Status Report. Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997. (http://apples.ucsd.edu/pubs/nec97.ps)

[25] T. Suzumura, T. Nakagawa, H. Nakada, S. Sekiguchi: Are Global Computing Systems Useful? - Comparison of Client-Server Global Computing Systems Ninf, NetSolve versus CORBA. Proc. of International Parallel and Distributed Processing Symposium (http://ninf.apgrid.org/papers/ipdps00suzumura/ipdps00suzumura.pdf)

[26] A. S. Tannenbaum: Rozproszone systemy operacyjne, 1997, Wydawnictwo Naukowe PWN