1. Rozpraszanie obliczeń za pomocą serwerów dystrybucyjnych

Jerzy Balicki, Tomasz Bieliński, Piotr Brudło, Jacek Paluszak, Julian Szymański

Politechnika Gdańska,

Wydział Elektroniki, Telekomunikacji i Informatyki,

Katedra Architektury Systemów Komputerowych

e-mail: balicki@eti.pg.gda.pl, tombieli@student.pg.gda.pl, piotr.brudlo@eti.pg.gda.pl, jacekpaluszak@gmail.com, julian.szymanski@eti.pg.gda.pl

Streszczenie

W rozdziale omówiono zasady funkcjonowania serwerów dystrybucyjnych w systemie obliczeniowym klasy grid pracującym w trybie volunteer computing*. Omówiono sposoby zwiększania wydajności tej warstwy systemu za pomocą zarządzania strumieniem paczek danych. Odniesiono się także do koncepcji* Map-Reduce *w implementacji przetwarzania równoległego.*

Słowa kluczowe: systemy klasy grid, volunteer computing, *przetwarzanie równoległe*

1.1. Paradygmat obliczeń za pomocą e-wolontariatu a korporacyjne obliczenia dedykowane w warunkach kryzysowych

Systemy gridowe działające zgodnie z paradygmatem tzw. *volunteer computing* opierają się na serwerach dystrybucyjnych, które służą do interakcji z komputerami internautów. Dlatego też omówione zostaną warianty możliwych trybów pracy serwerów dystrybucyjnych w systemach gridowych. Ponieważ jednak rola tej klasy jednostek obliczeniowych zależy od paradygmatu obliczeń zespołowych internautów, warto omówić ich genezę i stan aktualny.

Wolontariat obliczeniowy (ang. *volunteer computing*) jest rodzajem przetwarzania rozproszonego, w którym internauci i inni właściciele komputerów udostępniają bezpłatnie zasoby swoich komputerów (moc obliczeniową, zasoby dyskowe lub inne zasoby) w celu automatyzacji obliczeń jednego lub więcej projektów. Termin *volunteer computing* wprowadził Luis Sarmenta podczas programowania aplikacji w projekcie *Bayanihan* [8]. Alexandrov, Ibel, Schauser i Scheiman wdrożyli system prowadzenia *volunteer computing* jako SuperWeb, a obliczenia w takim systemie z wykorzystaniem języka Java – nazwali globalnymi (ang. *Java-Based Global Computing*) [1].

Pierwszym projektem zrealizowanym w oparciu o paradygmat *volunteer computing* był *Great Internet Mersenne Prime Search* (GIMPS) w 1996 [7]. Liczby *Mersenne'a* (od nazwiska francuskiego matematyka *Marina Mersenne'a*) to liczby wyznaczane z zależności $M_p=2^p-1$, gdzie p jest liczbą naturalną. Jeśli liczba Mersenne'a jest liczbą pierwszą, to wykładnik p jest także liczbą pierwszą, ale nie odwrotnie. Jeśli p jest

liczbą pierwszą, to nie jest to warunek wystarczający, ale jedynie konieczny do tego, że liczba Mersenne'a M_p jest także liczbą pierwszą.

Liczb pierwszych Mersenne'a jest 47, a największa z nich została wyznaczona dla p=43 112 609 przez Edsona Smitha 23 sierpnia 2008 roku w ramach projektu fridowego GIMPS. Największa liczba Mersenne'a składa się z 12 978 189 cyfr. Poszukiwaniem liczb pierwszych Mersenne'a i rozkładaniem liczb złożonych Mersene'a na czynniki pierwsze zajmują się badacze w projektach wykorzystujących e-wolontariat. Najbardziej efektywnym projektem jest właśnie GIMPS, w ramach którego odkryto dwanaście największych znanych liczb pierwszych Mersenne'a.

W 1997 roku w ramach środowiska gridowego *distributed.net* rozpoczęto obliczenia związane z deszyfrowaniem zakodowanego tekstu za pomocą algorytmu RSA dla klucza 56-biotowego. Obecnie wiele pokrewnych zagadnień jest liczonych na tym gridzie. Kolejne projekty to *Bayanihan, Popcorn* i *Charlotte* [4]. W 1999 roku rozpoczęto obliczenia w ramach projektów *SETI@home* i *Folding@home* z udziałem setek tysięcy internautów, co było niespodziewanie pozytywnym efektem dla projektantów i wywołało falę medialnych dyskusji nad możliwościami technologii przetwarzania w Internecie.

W 1998 także w komercyjnych projektach rozpoczęto wykorzystywanie e-wolontariatu, co dotyczy szczególnie projektów *Popular Power* (poszukiwanie szczepionki przeciwko grypie), *Porivo, Entropia* (liczby Mersenne'a jeszcze raz, szczepionka przeciwko AIDS) i *United Devices* [8].

W 2002 roku w projekcie *Berkeley Open Infrastructure for Network Computing* (BOINC) rozpoczęto realizację obliczeń, dostarczając oprogramowanie warstwy pośredniej dla *volunteer computing*, włączając w to oprogramowanie klienta, GUI, środowisko pracy aplikacji, oprogramowanie serwera i oprogramowanie strony projektu. Pierwszym zaawansowanym projektem na platformie BOINC był *Predictor@home*, który rozpoczęto w 2004 roku, a kolejnymi projektami: *SETI@home* i *ClimatePrediction.net*. Popularne projekty realizowane na tej platformie to *Rosetta@home* i *Einstein@home* [1].

W 2007, społeczność *World Community Grid* (WCG) przeniosła swoje obliczenia z platformy *the United Devices* na BOINC. *World Community Grid* to platforma przetwarzania rozproszonego zarządzana przez IBM. W ramach WCG poszukuje się m.in. wielu potrzebnych szczepionek i leków przeciwko groźnym chorobom [7].

W wyniku prac prowadzonych w ramach projektu Comcute, proponujemy wprowadzenie w warunkach kryzysowych paradygmatu instytucjonalnych *obliczeń obowiązkowych* (ang. *obligatory computing*). W proponowanym podejściu organizacje rządowe i samorządowe udostępniają zasoby komputerowe, inicjując linki do witryn sztabów antykryzysowych. Tej klasy obliczenia nazwaliśmy *korporacyjnymi obliczeniami dedykowanymi w warunkach kryzysowych*. Zasadność tego rodzaju obliczeń potwierdza poniżej opisana sytuacja kryzysowa w energetyce jądrowej.

1.2. E-wolontariat w sytuacji awarii reaktora jądrowego

W 2011 roku w czasie katastrofy elektrowni jądrowej Fukushima nastąpiła seria wypadków jądrowych w wyniku trzęsienia ziemi u wybrzeży Honsiu [3]. W rejonie katastrofy przestały działać superkomputery, na których prowadzono większość obliczeń wspierających działanie reaktorów, a także wspierających przewidywane akcje ratunkowe w wyniku wybuchu reaktorów. W tej sytuacji nie można było

prowadzić obliczeń na dedykowanych superkomputerach i stacjach roboczych. Natomiast możliwy był dostęp do Internetu za pomocą komputerów internautów, a także komputerów znajdujących się w większości urzędów w pozostałej części Japonii. Wobec tego przeniesiono część oprogramowania antykryzysowego na *ad-hoc* zorganizowany grid i kontynuowano kluczowe obliczenia.

Warto podkreślić, że podczas tej katastrofy jedną z awarii reaktora jądrowego oceniono najwyższym stopniem w siedmiostopniowej międzynarodowej skali INES [6]. W niektórych reaktorach doszło do stopienia rdzeni. Ponadto nastąpiła emisja substancji promieniotwórczych do środowiska, w tym przedostanie się do środowiska skażonej wody morskiej stosowanej do chłodzenia reaktorów. Prądy morskie do dnia dzisiejszego roznoszą skażenia po całym Pacyfiku. Szczególnie narażone są ryby, które znacznie dłużej przechowują skażenia promieniotwórczymi izotopami niż rośliny czy woda morska [3].

W Niemczech z powodu obaw o bezpieczeństwo elektrowni jądrowych, kilka elektrowni jądrowych starszego typu zamknięto. Ocenia się, że z punktu widzenia intensywności emisji materiałów radioaktywnych katastrofa japońska nie była mniejsza niż katastrofa jądrowa w Czarnobylu. Natomiast ilość uwolnionego materiału promieniotwórczego była na poziomie ok. 10% tego, co zostało uwolnione w trakcie katastrofy w Czarnobylu [4].

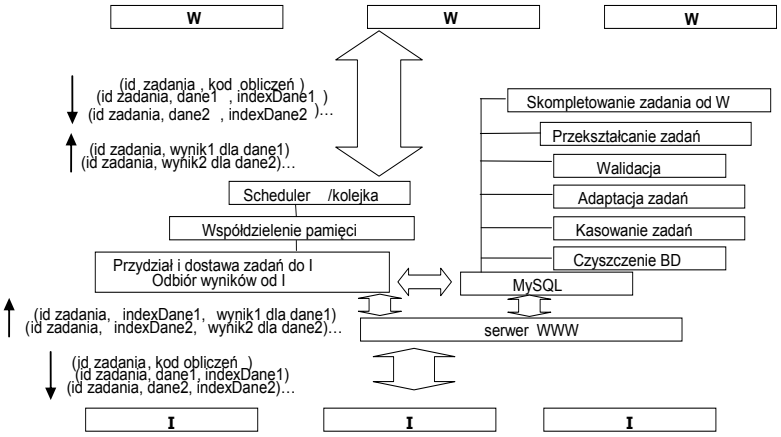
W sytuacji tego rodzaju katastrof niezbędne jest prowadzenie szybkiej i sprawnej akcji ratowniczej, co wiąże się z wykorzystaniem przygotowanego wcześniej odpowiedniego oprogramowania antykryzysowego. W wypadku awarii supercentrów komputerowych rozsądną alternatywą jest „przeniesienie" obliczeń na komputery internautów i pracowników administracji za pomocą systemu gridowego, np. Comcute JEE.

1.3. Funkcjonowanie serwerów dystrybucyjnych realizujących paradygmat obliczeń za pomocą e-wolontariatu

W systemie gridowym Comcute JEE (akronim CJEE) istotną rolę odgrywają serwery dystrybucyjne. Schemat funkcjonalny serwera dystrybucyjnego pokazano na rys. 1.1.

Inicjacja zadania następuje po ściągnięciu na komputer internauty *I* strony webowej pobranej z portalu WWW stowarzyszonego z serwerem dystrybucyjnym *S*. Strona może zawierać banner reklamowy zawierający kod *loadera*, który wysyła żądanie zadania do serwera *S*.

Żądanie zadania jest adresowane do modułu równoważącego obciążenia serwerów *S*, a następnie jest przekierowywane do najmniej obciążonego serwera *S*. Istnieje kilka metod równoważenia obciążenia (na różnych warstwach modelu sieci) [2]. W zaimplementowanym prototypie przyjęto, że równoważenie obciążeń między serwerami *S* odbywa się na wyróżnionym serwerze *S*.



Rys. 1.1. Schemat funkcjonalny serwera dystrybucyjnego

Żądanie komputera internauty *I* jest przez serwer *S* kierowane do skojarzonego z nim węzła *W*. Jeśli węzeł *W* ma jakąś paczkę zadania do przetworzenia, to przesyła tę paczkę z powrotem przez serwer *S* do komputera internauty *I*. Paczka zadania zawiera: identyfikator zadania, kod obliczeń, identyfikator paczki danych oraz samą paczkę danych.

Możliwy jest inny sposób inicjacji obliczeń polegający na bezpośrednim dołączeniu kodu obliczeniowego do kodu strony WWW. Kod inicjujący obliczenia rezydujący na komputerze internauty może także (w ramach przepisów prawa) przenieść się na komputer innego internauty i z tego nowego komputera zainicjować żądanie obliczeń do warstwy *S*. W szczególności możemy rozważać propagację kodu obliczeń podobną do „rozmnażania się" wirusów lub robaków internetowych. Oczywiście taki sposób propagacji obliczeń powinien być regulowany przepisami odnośnie przetwarzania w warunkach kryzysowych państwa.

Po wykonaniu zadania komputer *I* zwraca wynik do serwera dystrybucyjnego wraz z identyfikatorem zadania oraz identyfikatorem paczki danych. Następnie komputer internauty *I* może otrzymać kolejną paczkę danych wejściowych, może też otrzymać nowy kod nowego zadania albo zakończyć pracę.

1.4. Zwiększenie wydajności przesyłania zadań

Można zwiększyć wydajność poprzez przesyłanie dwóch (lub więcej) paczek danych podczas pierwszego uruchomienia zadania na komputerze internauty. Wówczas komputer internauty po przetworzeniu pierwszej paczki danych i odesłaniu wyników realizowałby zadanie dla drugiej paczki czekając na kolejną porcję danych. To rozwiązanie nazywa się *przeplotem komunikacyjnym* [5].

Innym sposobem zwiększenia wydajności jest przesłanie z węzła *W* do serwera *S* stosunkowo dużej ilości danych, np. stu paczek danych. Wówczas serwer *S* tworzy dla komputera internauty *strumień danych* wejściowych, a ten przetwarzając każdą paczkę danych zwraca do serwera *S strumień wyników*.

Ze względu na zawodność obliczeń na komputerach internautów serwery w warstwie *W* mogą kopiować strumień danych dwukrotnie lub trzykrotnie, wysyłając kod obliczeń i dane do dwóch lub trzech grup rozłącznych serwerów *S*.

1.5. Partycjonowanie danych i dystrybucja kodu obliczeń

Zakładamy, że kod obliczeń jest definiowany i dostarczany przez zleceniodawcę obliczeń *Z*. w wybranej technologii programistycznej. Jest on przesyłany do komputera internauty *I* poprzez serwery *W* i *S*. Kod obliczeń nie ulega zmianom podczas dystrybucji.

Podział danych i ich dystrybucja w warstwie *W* jest realizowana za pomocą algorytmu *partycjonera*. Z kolei scalanie wyników cząstkowych odbywa się również w warstwie *W* za pomocą algorytmu *linkera*. Trzeba podkreślić, że prawie każdy nowy kod zadania wymaga napisania nowego partycjonera i linkera paczek danych. Aby ograniczyć tę konieczność warto zdefiniować takie algorytmy partycjonera i linkera, które będą mogły być stosowane w różnych zadaniach, ale należących do tej samej *klasy zadań*.

Załóżmy, że na podstawie jednego partycjonera następuje zrealizowanie zadania *A* klasy *K*, a także zadania *B* należącego do tej samej klasy *K*. Tak rozumiana uniwersalność partycjonera jest ograniczona do danej klasy zadań. Mamy zatem klasy rozwiązywanych problemów $K_1, K_2, \ldots, K_K$, które są zależne od partycjonera.

Przykładowo, niech *A* będzie zadaniem polegającym na weryfikacji pewnego zbioru liczb naturalnych czy są pierwsze, a zadanie *B* weryfikuje hipotezę Collatza. Danymi wejściowymi w obu wypadkach jest zbiór liczb naturalnych ograniczony od góry pewnym elementem maksymalnym. *Pracę obliczeniową* (*kod programu, dane wejściowe*) dekomponujemy na mniejsze jednostki składające się z kodu programu i paczki danych, a następnie wysyłamy do komputerów internautów. Strumień paczek danych wejściowych generowany jest jednakowo dla obu prac obliczeniowych, tj. wyznaczanie liczb pierwszych i problem *3n+1*. Każda paczka danych to para (*a,b*) opisująca przedział liczb naturalnych. Sekwencja kolejnych przedziałów właśnie generuje strumień, który *partycjoner* rozsyła do komputerów internautów.

Serwer *W* dysponuje kodami obliczeniowymi wszystkich rozwiązywanych problemów i aktualizuje tabelę kodów obliczeniowych (*identyfikator, zawartość kodu obliczeniowego*) na podstawie zleceń z warstwy *Z*. Zleceniodawca obliczeń *Z* może dostarczyć kod rozwiązujący nową klasę problemu bądź też zrezygnować z prowadzenia obliczeń wybranej klasy. Może także dostarczyć kod jedynie na potrzeby jednorazowego uruchomienia.

1.6. Zastosowanie koncepcji *Map-Reduce* do implementacji przetwarzania gridowego

Koncepcja *Map-Reduce* to alternatywne podejście do przetwarzania strumienia danych. Paradygmat programowania rozproszonego *Map-Reduce* został zrealizowany w Google w 2004 roku, aby uczynić łatwiejszym pisanie aplikacji operujących na dużych ilościach danych. Ukrywa przed programistą większość problemów programowania współbieżnego. Wykorzystywany jest przez takie firmy jak Google, Yahoo, Amazon czy Facebook [9].

Hadoop MapReduce to framework wspomagający pisanie aplikacji, które przetwarzają równolegle znaczące ilości danych (rzędu terabajtów) za pomocą dużych klastrów (tysiące węzłów) w sposób niezawodny i odporny na uszkodzenia. Wszystkie dane nazywane są *pracą*, która zazwyczaj dzielona jest na niezależne paczki danych. Paczki są przetwarzane przez *zadania klasy map* w sposób całkowicie równoległy. Następnie sortowane są wyniki z *zadań klasy map*, które teraz stanowią dane wejściowe do *zadań klasy reduce*. Zazwyczaj zarówno dane wejściowe jak i wyjściowe z zadania są przechowywane w systemie plików. Framework zajmuje się szeregowaniem zadań, ich monitorowaniem i zapewnia ponowne wykonanie nieprawidłowo zakończonych zadań.

Zazwyczaj węzły obliczeniowe i węzły przechowujące dane są takie same, gdyż framework *MapReduce* i *Hadoop Distributed File System* są uruchamiane uruchomione na tym samym zbiorze węzłów. Taka konfiguracja umożliwia skutecznie szeregowanie zadań w węzłach, gdzie dane są już dostępne, co skutkuje bardzo wysoką wydajnością.

Framework *MapReduce* składa się z JobTrackera typu master i TaskTrackera typu slave w każdym węźle klastra. *Master* jest odpowiedzialny za szeregowanie zadań, ich monitorowanie i ponownie wykonanie nieprawidłowo zakończonych zadań. *Slave* realizuje zadania zgodnie z zaleceniami *mastera*.

Aplikacje określają lokalizacje danych wejściowych i wyjściowych oraz wspierają funkcje *map* i *reduce* za pomocą implementacji odpowiednich interfejsów lub klas abstrakcyjnych. Te i inne parametry *pracy* stanowią *konfigurację pracy*. *Klient zadania* następnie przesyła pracę (plik JAR, wykonywalny itp.) oraz konfigurację do JobTrackera, który następnie przejmuje odpowiedzialność za dystrybucję oprogramowania/konfiguracji do *slave*-ów, szeregowanie zadań oraz ich monitorowanie, zapewniając status oraz informacje diagnostyczne do klienta pracy.

Chociaż framework Hadoop jest realizowany w Javie, to aplikacje *MapReduce* nie muszą być napisane w Javie.

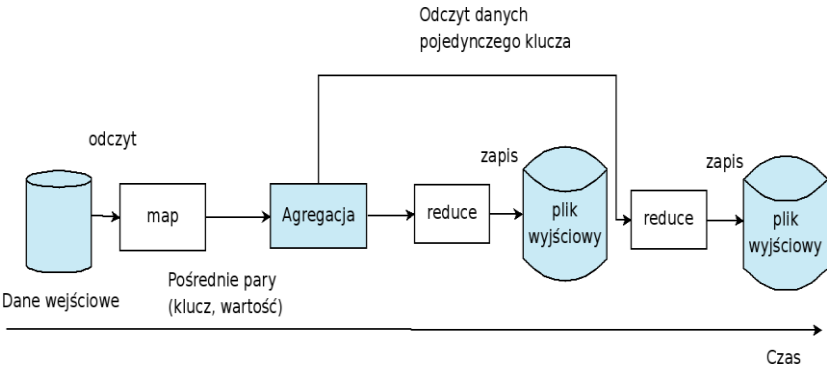
- *Hadoop Streaming* to narzędzie, które pozwala użytkownikom na implementację i uruchamianie prac za pomocą wszelkich plików wykonywalnych (np. narzędzia powłoki) jak mapper lub reduktor.
- *Hadoop Pipes* jest narzędziem klasy SWIG zgodnym z API *C++* do wdrożenia aplikacji w środowisku *MapReduce* (nie opiera się na *Java Native Interface*).

MapReduce nie wykorzystuje *Java Native Interface* (JNI) macierzystego interfejsu programistycznego dla Javy. JNI umożliwia uruchamianie kodu w Javie wewnątrz wirtualnej maszyny Javy, we współpracy z aplikacjami i bibliotekami napisanymi w C, C++ czy asembler.

Natomiast SWIG to narzędzie, które łączy programy napisane w C/C++ z różnymi językami wysokiego poziomu, w tym z językami skryptowymi Perl, PHP, Python, Tcl i Ruby. Ponadto możliwe jest łączenie z językami nieskryptowymi: C#, Common Lisp (CLISP, Allegro CL, CFFI, UFFI), D, Go, Java, Lua, Modula-3, OCAML, Octave i R. Także wspierane są różne interpretowane lub kompilowane implementacje klasy Scheme (Guile, MzScheme/Racket, Chicken). SWIG jest najczęściej wykorzystywany tworzenia środowisk do interpretowania lub kompilacji programów napisanych w językach wysokiego poziomu.

SWIG umożliwia także implementację interfejsów użytkowników, a także może być wykorzystywany jako narzędzie do testowania i prototypowania aplikacji napisanych w języku C/C++. SWIG jest zazwyczaj stosowany do parsowania interfejsu C/C++

i generowania kodu w innym języku. SWIG może także eksportować parsowane drzewo w kod języka XML lub s-wyrażenia Lispa. SWIG jest oprogramowaniem bezpłatnym.

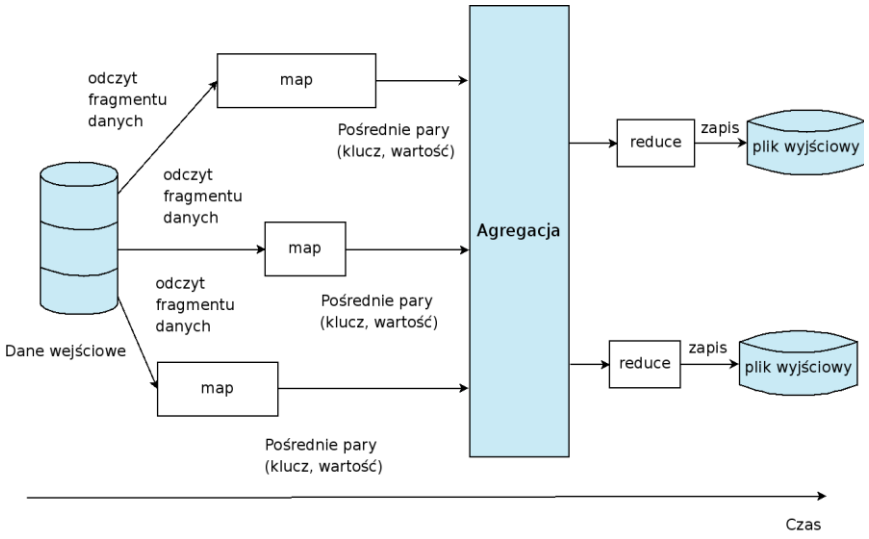


Rys. 1.2. Schemat przetwarzania sekwencyjnego w paradygmacie *map-reduce*

Zgodnie z koncepcją *Map-Reduce* (rys. 1.2) programista dostarcza implementacje dwóch funkcji: *map* oraz *reduce*:

- Funkcja *map* pobiera fragment danych wejściowych i produkuje na ich podstawie ciąg par *(klucz, wartość)*.
- Funkcja *reduce* otrzymuje na wejściu wszystkie pary dzielące wspólny klucz, a następnie na ich podstawie generuje pary wyjściowe.

Koncepcję tę łatwo zastosować do przetwarzania równoległego (rys. 1.3). Wówczas tworzy się wiele instancji funkcji *map*. Dane wejściowe są automatycznie dzielone na fragmenty, które są przesyłane do poszczególnych instancji funkcji *map*.



Rys. 1.3. Diagram przetwarzania równoległego w paradygmacie *map-reduce*

Koncepcja *Map-Reduce* jest w praktyce realizowana w warstwie *W* systemu Comcute JEE, przy czym funkcja *map* jest implementowana jako kod *partycjonera*, a funkcja *reduce* jako kod *linkera*.

1.7. Komunikacja między *S* a warstwą *W*

Poniżej zaprezentowano dwie koncepcje komunikacji między warstwą *S* i *W*. W wersji pierwszej internauta otrzymuje gotowy, zdefiniowany i ustalony *kod programu* realizujący określone operacje (funkcje) i kod ten jest niezmienny. Internauta pobiera natomiast *dane parametryzujące* wykonanie posiadanego kodu.

W wersji drugiej internauta otrzymuje zarówno kod, jak i dane. W tej wersji zakłada się, iż na komputerze internauty (w serwerze w sensie architektury klient-serwer) realizowana będzie interpretacja (lub kompilacja) kodu w celu jego finalnego wykonania w lokalnym środowisku.

Wariant pierwszy jest łatwiejszy w realizacji, natomiast w sposób istotny ogranicza zakres realizowanych zadań. W wersji tej zakłada się, iż do przeglądarki internauty dostarczane są gotowe fragmenty kodu w wersji finalnej (skompilowanej) w celu ich wykonania. W praktyce ogranicza to elastyczność systemu do z góry założonych i na wstępie opracowanych algorytmów bez możliwości ich dynamicznych modyfikacji. Inną zaletą takiego rozwiązania jest większa (niż w drugiej koncepcji) efektywna szybkość wykonywania założonych z góry zadań, które są przesyłane do komputerów internautów w stanie gotowości do uruchomienia.

Wariant drugi zakłada dynamiczne dostarczanie zarówno kodu, jak i danych do przeglądarki internauty. Dynamiczne dostarczanie kodu do wykonania powoduje to, że system staje się w pełni otwarty i konfigurowalny dla pojawiających się zadań. W wariancie tym internauta otrzymuje do przeglądarki kod zadania wyrażony w pewnym języku programowania, musi ten kod zinterpretować (lub skompilować) i następnie go wykonać. Występuje tu konieczność rozbudowy środowiska uruchomieniowego u internauty. Wnosi to też pewien narzut czasowy na interpretację (lub kompilację) kodu, ale z drugiej strony nie zamyka zakresu wykonywanych zadań tylko do uprzednio określonego zdefiniowanego zbioru.

W opracowaniu rozważane jest potencjalne wykorzystanie obu metod, ze zwróceniem uwagi na zalety oraz wady. Wariant pierwszy jest prostszy implementacyjnie oraz szybszy, wariant drugi natomiast jest elastyczny.

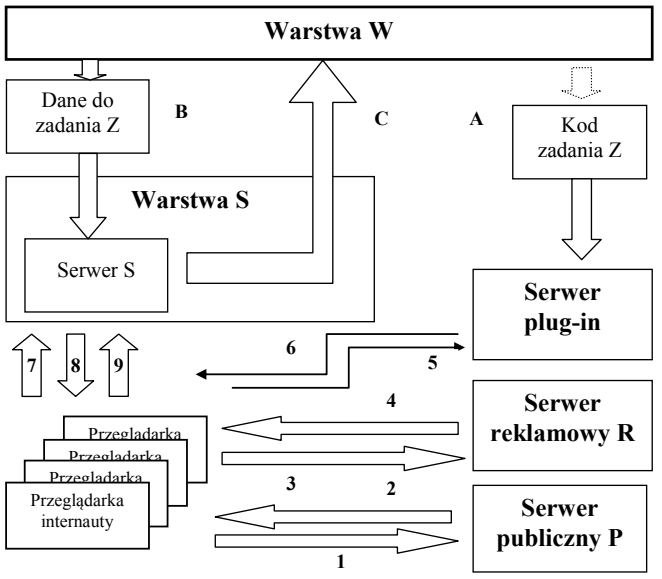
1.7.1. Kooperacja między warstwą *W* a *S* przy statycznym zestawie zadań

Schemat blokowy całości koncepcji w wariancie pierwszym przedstawiony został na rys. 1.4. Wyszczególnione zostały poszczególne elementy koncepcyjne oraz opisany został dynamiczny schemat komunikacji.

W tej koncepcji budowy systemu zakładamy istnienie zamkniętej statycznej grupy zadań do realizacji. Kody zadań zostały wcześniej przygotowane, opracowane i zakodowane. Ich ciała wykonawcze (właściwe kody, bez danych) są dołączone do wtyczek służących do odtwarzania zawartości reklam. Całość operacji w ujęciu wykonawczym odbywa się następująco:

- Dostarczenie i zapisanie kodów wykonawczych do właściwych wtyczek. (programów odtwarzających) Wtyczki powiązane są z określonymi reklamami i w przypadku pobierania określonej reklamy ładowana jest właściwa wtyczka z kodem wykonawczym.
- Dane do określonego zadania przesyłane są z warstwy *W* do warstwy *S* i oczekują na pobranie w serwerze *S*.

- Internauta łączy się poprzez przeglądarkę z serwerem publicznym *P*. Przeglądarka wysyła żądanie przesłania strony.
- Serwer *P* przesyła zawartość strony WWW wraz z umieszczonym w niej linkiem do pobrania określonej reklamy.
- Przeglądarka kontaktuje się z serwerem reklam, gdzie wysyła żądanie pobrania określonej reklamy.
- Serwer reklam przesyła wymaganą reklamę, w określonym formacie. Przeglądarka po odebraniu reklamy potrzebuje określonej wtyczki do odtworzenia reklamy.
- Przeglądarka wysyła żądanie pobrania określonego programu odtwarzającego. W ramach programu odtwarzającego (oprócz kodu do odtworzenia reklamy) znajduje się wcześniej zapisany kod zadania właściwego systemu Comcute.
- Program odtwarzający jest pobierany. U internauty odtwarzana jest reklama, a jednocześnie wykonywany jest kod właściwy systemu Comcute.
- Nawiązywane jest połączenie z serwerem S (przez wtyczkę). Wysyłana jest identyfikacja komputera internauty oraz identyfikator zadania, które będzie wykonane na komputerze internauty.
- Serwer S wysyła dane dla tego zadania (o ile je posiada, jeżeli nie posiada to wysyła polecenie zakończenia). Serwer *S* może nie posiadać żadnych zadań do wykonania, gdyż mógł nie otrzymać żadnych zleceń od warstwy *W*.
- Po wykonaniu zadania rezultaty są przesyłane do serwera *S* (wymagane jest podanie danych identyfikacyjnych, aby warstwa *S* mogła sklasyfikować otrzymane rezultaty).
- Serwer *S* komunikuje się z warstwą *W* i wysyła rezultaty (wymagane jest podanie danych identyfikacyjnych zadania, aby warstwa W była w stanie określić jakiego zadania wyniki dotyczą).



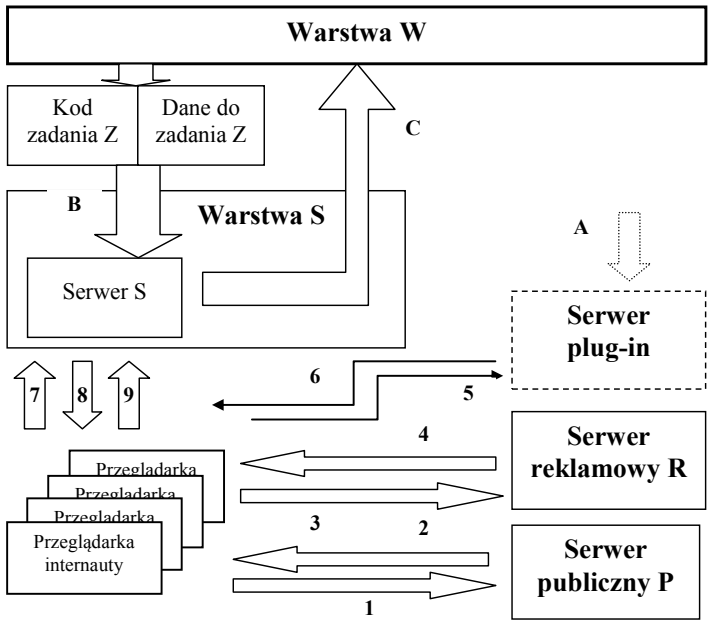
Rys. 1.4. Schemat blokowy realizacji zadań w systemie Comcute – wariant 1.

Podstawowym wyzwaniem realizacyjnym w wariancie pierwszym jest głębokie osadzenie kodu zadań Comcute w programach odtwarzających treści reklamowe. Dla każdej poszczególnej reklamy wymagane jest opracowanie dedykowanego programu

odtwarzającego, w którym zawarty jest określony kod zadania Comcute. Po pobraniu określonej reklamy konieczne jest pobranie dedykowanego dla niej odtwarzacza. Prowadzi to do konieczności realizacji serwera *plug-in* oraz utrzymywania go. Nie można wykorzystać tutaj standardowych rozwiązań. Zaletą jest natomiast niskie obciążenie internauty (przeglądarki). Dostaje jedynie gotowy kod do wykonania oraz dane do przetworzenia.

1.7.2. Kooperacja między warstwą *W* a *S* przy dynamicznym zestawie zadań

Schemat blokowy całości koncepcji w wariancie drugim przedstawiony został na rys. 1.5. Wyszczególnione zostały poszczególne elementy koncepcyjne oraz opisany został dynamiczny schemat komunikacji. Zwrócenia uwagi wymaga odmienny sposób przesłania kodu zadania do przeglądarki internauty.



Rys. 1.5. Schemat blokowy realizacji zadań w systemie Comcute – wariant 2.

W tej koncepcji budowy systemu zakładamy elastyczność w zakresie potencjalnych zadań do realizacji. Kody zadań mogą być zarówno brane z bazy algorytmów, które zostały wcześniej przygotowane, opracowane i zakodowane, ale również mogą być wprowadzane dynamicznie do systemu. Kody zadań do wykonania przesyłane są do realizacji w momencie potrzeby ich uruchomiania i nie są stałym elementem. Ich ciała wykonawcze (właściwe kody) wraz z danymi są przesyłane do interpretacji i wykonania w przeglądarkach. Całość operacyjna w ujęciu schematu blokowego odbywa się następująco:

A. Dostarczenie i zapisanie pojedynczej wtyczki (programu odtwarzającego) Wtyczka jest jednakowa dla wszystkich reklam, jej zadaniem jest umożliwienie stworzenia środowiska dla wykonania kodu otrzymanego przez przeglądarkę internauty od warstwy *S*.

B. Kod zadania wraz z danymi jest przesyłany z warstwy *W* do warstwy *S* i oczekuje na pobranie w serwerze *S*.

C. Internauta łączy się poprzez przeglądarkę z serwerem *P*. Przeglądarka wysyła żądanie przesłania strony.

1. Serwer *P* przesyła zawartość strony wraz z umieszczonym w niej linkiem do pobrania określonej reklamy.
2. Przeglądarka kontaktuje się z serwerem reklam, gdzie wysyła żądanie pobrania określonej reklamy.
3. Serwer reklam przesyła wymaganą reklamę. Przeglądarka po odebraniu reklamy potrzebuje wtyczki do odtworzenia reklamy. Przy czym rozważane jest wykorzystanie wtyczek standardowych (ewentualnie sparametryzowanych).
4. Przeglądarka wysyła żądanie pobrania określonego programu odtwarzającego. W sytuacji, gdy pobierana jest jedna wersja wtyczki dla wszystkich reklam, operacja ta sprowadza się praktycznie do pojedynczego pobrania przez internautę.
5. Program odtwarzający jest pobierany. U internauty odtwarzana jest reklama. W ramach wykonywania reklamy (w kodzie reklamy) realizowane jest właściwy kod Comcute. Zadaniem kodu jest nawiązanie połączenia z warstwą *S* (serwerem *S*) w celu pobrania kodu zadania *Z* wraz z właściwymi danymi.
6. Nawiązywane jest połączenie z serwerem *S* (w trakcie wykonywania reklamy). Wysyłana jest identyfikacja komputera internauty wraz z informacją o gotowości przyjęcia zadania do wykonania).
7. Serwer S wysyła kod posiadanego zadania *Z* wraz z danymi dla wykonania tego zadania (o ile je posiada, jeżeli nie posiada to wysyła polecenie zakończenia). Serwer S może nie posiadać żadnych zadań do wykonania, gdyż mógł nie otrzymać żadnych poleceń od warstwy *W*.
8. W przeglądarce internauty odbierany jest kod wraz z danymi. Kod jest wykonywany. Po wykonaniu zadania rezultaty są przesyłane do serwera *S* (wymagane jest podanie danych identyfikacyjnych, aby warstwa *S* mogła sklasyfikować otrzymane rezultaty).
9. Serwer S komunikuje się z warstwą *W* i wysyła rezultaty (wymagane jest podanie danych identyfikacyjnych zadania, aby warstwa *W* była w stanie określić jakiego zadania wyniki dotyczą).

Podstawowym wyzwaniem realizacyjnym w wariancie drugim jest umożliwienie wykonania dynamicznie dostarczonego kodu do przeglądarki internauty. Należy spodziewać się, iż ten kod nie będzie mógł być w związku z tym zbyt obszerny. Ponadto należy stworzyć środowisko do wykonania tego kodu. W związku z tym, że środowisko to jest jednolite, rozważane jest wprowadzenie go do wtyczki (programu odtwarzającego), lub też wykorzystanie wtyczek standardowych, o ile posiadają zakładane możliwości wykonawcze. Niewątpliwą zaletą wariantu drugiego jest dynamiczna możliwość realizacji algorytmów. Ponadto algorytmy (zadania) są określane w warstwie *W* podczas pracy całości systemu Comcute, co umożliwia wyższy poziom kontroli funkcjonowania oraz dużą elastyczność operacyjną. Do rozwiązania pozostaje umieszczenie w treści reklam (serwer reklamowy)

fragmentów kodu pozwalającego na skomunikowanie się z serwerem *S* w celu pobrania kodu zadania oraz danych dla jego wykonania.

1.8. Podsumowanie

Przyjęto, że serwer dystrybucyjny *S* może przydzielić kilka milionów zadań dziennie. Wymaga to efektywnego mechanizmu dostarczania zadań do komputerów internautów z równoważeniem obciążenia serwerów *S*. Istnieją dwie koncepcje dostarczania zadań: w postaci statycznego, gotowego do wykonania kodu zadania oraz w postaci kodu dynamicznie interpretowanego lub kompilowanego po stronie klienta w komputerze internauty.

W sytuacji katastrof reaktorów jądrowych niezbędne jest prowadzenie szybkiej i sprawnej akcji ratowniczej, co wiąże się z wykorzystaniem przygotowanego wcześniej odpowiedniego oprogramowania antykryzysowego. Zazwyczaj oprogramowanie tej klasy wymaga zastosowania superkomputerów do prowadzenia obliczeń. Jednakże w wypadku awarii supercentrów komputerowych, tak jak to miało miejsce podczas katastrofy elektrowni jądrowej Fukushima, rozsądną alternatywą jest „przeniesienie" obliczeń na dziesiątki tysięcy komputerów instytucji rządowych za pomocą systemu gridowego Comcute JEE.

1.9. Wykaz literatury

1. Alexandrov A.D., Ibel M., Schause K.E., Scheiman K.E.: *SuperWeb: Research issues in Java-Based Global Computing*. Proceedings of the Workshop on Java for High performance Scientific and Engineering Computing Simulation and Modelling. New York: Syracuse University., 1996
2. Bourke T.: *Server Load Balancing*, O'Reilly, Sebastopol, 2001
3. *Fukushima faced 14-metre tsunami*. World Nuclear News. 24 March 2011. http://www.world-nuclear-news.org/RS_Fukushima_faced_14-metre_tsunami_2303113.html. March 2011.
4. Karaul, M., Kedem, Z., Wyckoff, P. *Charlotte: Metacomputing on the Web*. Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, Sept 1996
5. Kopparapu Ch.: *Load Balancing Servers, Firewalls & Caches*, Wiley, New York 2002
6. Martin, Alex, "Lowdown on nuclear crisis and potential scenarios", *Japan Times*, 20 March 2011, p. 3.
7. Regev O., Nisan N.: *The POPCORN market~Wan online market for computational resources*. Proceedings of the First International Conference on Information and Computation Economies. Charleston, South Carolina, United States: ACM Press, New York, NY. pp. 148–157.October 25 - 28, 1998).
8. Sarmenta L.F.G.: *Bayanihan: Web-Based Volunteer Computing Using Java*. Lecture Notes in Computer Science, 1368, Springer-Verlag, 1998. pp. 444-461. Proc. of the 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998
9. Syme M., Goldie P.: *Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing"*, Prentice Hall PTR, New York 2004