1. Architektura systemu Comcute

Mariusz Matuszek

Politechnika Gdańska,

Wydział Elektroniki, Telekomunikacji i Informatyki,

Katedra Architektury Systemów Komputerowych

e-mail: mrm@eti.pg.gda.pl

Streszczenie

W rozdziale przedstawiono architekturę systemu Comcute realizującego masywne przetwarzanie rozproszone wykorzystujące powszechny wolontariat użytkowników komputerów w sieciach rozległych.

Słowa kluczowe: systemy rozproszone, architektura systemów, wolontariat obliczeniowy, przetwarzanie rozproszone.

1.1. Wprowadzenie

Założenia projektowe:

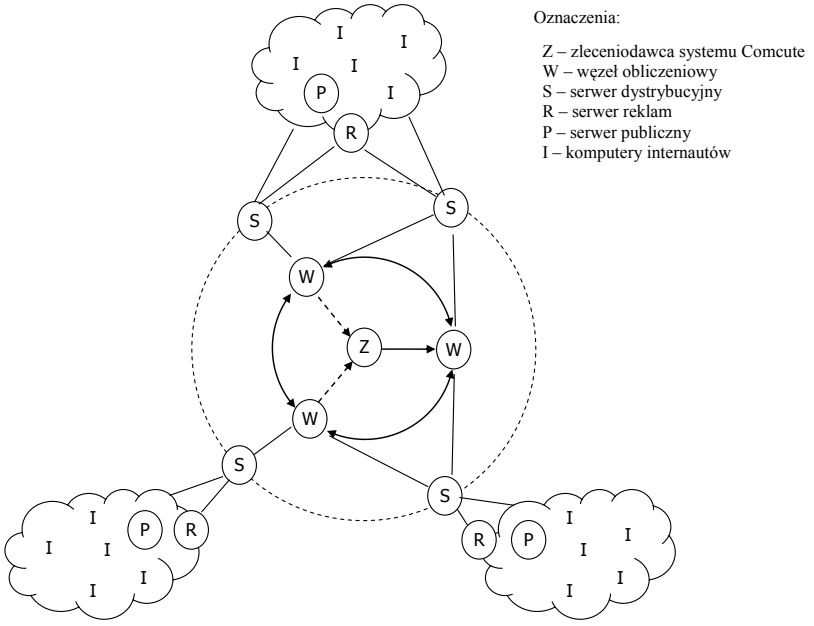
1. System przetwarzania rozproszonego wykorzystujący powszechny wolontariat internautów musi zapewniać możliwość przetwarzania dużej ilości danych przy wykorzystaniu efektu skalowania obliczeń wykonywanych przez bardzo dużą liczbę komputerów.
2. Dane przetwarzane przez system mogą mieć charakter zarówno w pełni jawny jak też poufny i tajny, należy więc dokonać takich decyzji projektowych, które z jednej strony maksymalnie uproszczą dołączanie mocy obliczeniowej internautów do systemu, a z drugiej strony zabezpieczą infrastrukturę systemu przed działaniami złośliwymi.
3. Dane są dostarczane do internautów i przetwarzane przez nich w sposób jawny, przez zgłoszenie chęci uczestnictwa w projekcie obliczeniowym. Wyjątkiem może być ogłoszenie stanu kryzysowego i przejście w tryb przetwarzania wymuszonego.
4. System jest systemem rozproszonym składającym się z wielu węzłów.
5. System musi być odporny na ataki z zewnątrz, zarówno na przechwytywanie danych, ataki typu DoS, jak i celowe zniekształcanie wyników.
6. System musi zapewnić możliwość kontynuowania obliczeń nawet wówczas, gdy większa część węzłów zostanie zaatakowana i wyeliminowana, w skrajnym wypadku aż do momentu, gdy jeden węzeł ma możliwość działania.

7. Zleceniodawca (użytkownik zlecający obliczenia dla systemu) może zgłosić się do dowolnego węzła i odebrać wyniki z dowolnego innego węzła (działającego).
8. Obliczenia mogą być czasochłonne – odstęp czasowy pomiędzy zleceniem obliczeń a odebraniem wyników może być długi, a moment odebrania wyników zależy od zleceniodawcy.
9. System powinien zapewniać weryfikację wyników obliczeń np. przez porównanie rezultatów otrzymywanych od różnych internautów. Wyjątkiem są obliczenia, które mają charakter niedeterministyczny.
10. Mogą istnieć sytuacje, w których zadanie obliczeniowe uznaje się za ukończone nawet wówczas, gdy wyniki nie zostaną policzone w 100%.

1.2. Koncepcja architektury systemu

System składa się z czterech warstw (rys. 1.1):

- warstwy *Z* (zleceniodawcy), zapewniającej interfejs użytkownika dla użytkowników zlecających zadania obliczeniowe i separująca ich od warstwy wewnętrznej,
- warstwy *W* (wewnętrznej), składającej się z serwerów *W*, odpowiedzialnej za podział (partycjonowanie) zadań obliczeniowych od zleceniodawcy *Z* na fragmenty, synchronizację i weryfikację wyników obliczeń (arbitraż) oraz ich scalanie i dostarczanie z powrotem do zleceniodawcy,
- warstwy *S* (dystrybucyjnej, brzegowej), składającej się z serwerów *S*, zajmującej się dostarczaniem fragmentów zadań do obliczeń dla internautów i odbieraniem od nich wyników,
- warstwy *I* (publicznej, zewnętrznej), w której komputery internautów *I* wykonują w tle właściwe obliczenia. Komputery internautów są dołączane się do systemu przez serwery publiczne *P* i stowarzyszone z nimi serwery przekierowujące *R*.



Rys. 1.1. Koncepcja architektury systemu przetwarzania rozproszonego

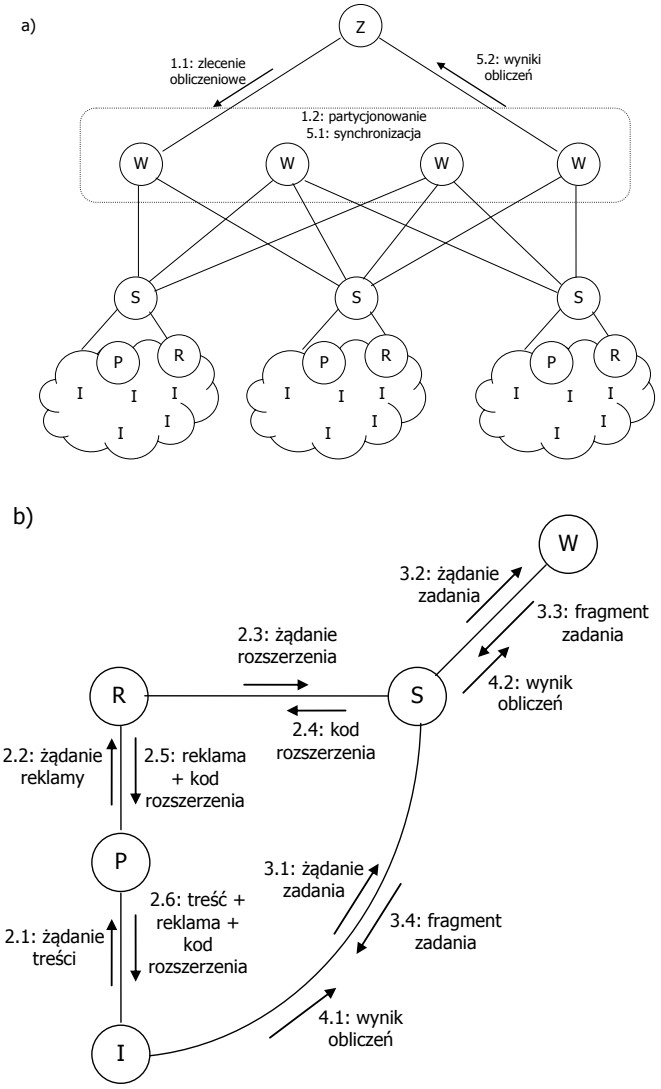
Oddzielenie warstwy dystrybucyjnej od warstwy wewnętrznej umożliwia:

- dopasowanie liczby węzłów *W* do zadania obliczeniowego,
- dopasowanie liczby węzłów *S* do liczby internautów i ich lokalizacji,
- ochronę węzłów *W* dysponujących informacją o całym zadaniu (wraz z wynikami obliczeń) przed zagrożeniem z zewnątrz.

Założenia projektowe systemu zakładają, że istnieje pełna kontrola nad budową i sposobem funkcjonowania węzłów *W* i serwerów *S*. Serwery *R* nie są projektowane w ramach systemu, lecz muszą być z nim „zaprzyjaźnione", tzn. muszą mieć informacje o położeniu serwerów *S* i sposobie komunikacji z nimi. Nie ma żadnej kontroli nad serwerami *P* i komputerami internautów *I*.

1.3. Współdziałanie komponentów

Koncepcję współdziałania komponentów należących do różnych warstw ilustruje rys. 1.2.



Rys. 1.2. Koncepcja działania systemu Comcute: a) zlecanie i partycjonowanie zadań, b) dystrybucja zadań

Działanie systemu składa się z pięciu faz:

1. **Faza zlecenia zadania**:

   1.1. Zleceniodawca *Z* zleca zadanie obliczeniowe do dowolnego węzła warstwy *W*.

   1.2. Węzeł, który przyjmuje zadanie, rozsyła je do innych węzłów. Wszystkie węzły *W* dzielą zadanie na fragmenty wg tego samego deterministycznego algorytmu. Dzięki temu wszystkie węzły W mogą niezależnie oferować fragmenty zadania do wykonania dla internautów.

2. **Faza nawiązywania połączenia** internautów z systemem:

   2.1. Internauta zgłaszający się do serwisu publicznego *P* żąda dostarczenia pewnej użytecznej treści.

   2.2. Serwis *P* łączy się z serwerem *R* w celu dołączenia dodatkowej treści (np. reklamy w technologii Flash) do treści żądanej przez internautę.

   2.3. Serwer *R* zgłasza się do „zaprzyjaźnionego" serwera S z żądaniem podania kodu rozszerzającego serwowaną treść o skrypt pełniący rolę loadera zadania obliczeniowego.

   2.4. Kod rozszerzenia zostaje dołączony do treści oferowanej przez R.

   2.5. Wzbogacona o kod rozszerzenia treść zostaje dostarczona do serwera *P*.

   2.6. Użyteczna treść wraz treścią dodatkową i kodem rozszerzenia zostaje dołączona do przeglądarki internauty.

3. **Faza dystrybucji zadań**:

   3.1. Kod rozszerzenia (*loadera*) wykonuje się w ramach technologii oferowanych przez przeglądarkę internauty. Łączy się z serwerem *S* zgłaszając gotowość na przyjęcie zadania obliczeniowego.

   3.2. Serwer *S* przegląda dostępne serwery *W* w poszukiwaniu zadań do obliczenia.

   3.3. Zadanie składa się z fragmentów, z których pierwszy zawiera kod obliczeniowy i dane do obliczenia, a następne mogą zawierać tylko dane. Serwer S pobiera fragment zadania obliczeniowego z serwera *W*…

   3.4. ...i dostarcza zadanie do kodu rozszerzenia w przeglądarce internauty, gdzie to zadanie jest wykonywane.

4. **Faza zwracania wyników**:

   4.1. Po obliczeniu fragmentu kod obliczeniowy zwraca wynik do węzła *S*,

   4.2. a ten zwraca je do tego węzła *W*, z którego pobrał fragment zadania do obliczenia.

5. **Faza kompletowania zadania**:

   5.1. Węzły *W* luźno synchronizują stan fragmentów zadania. Ten sam fragment może (a czasami musi) być przydzielony do różnych internautów. Jeśli różne węzły otrzymują różne wyniki tych samych fragmentów, to muszą stosować algorytm arbitrażu dla ustalenia, który wynik jest prawidłowy.

   5.2. Wyniki po skompletowaniu są odsyłane do zleceniodawcy z dowolnego węzła *W*, który uczestniczył w obliczeniach.

1.4. Sposoby dystrybucji zadań obliczeniowych

Ponieważ system nie może w żaden sposób zagwarantować, że internauta nie zamknie przeglądarki lub nie przełączy się na inną stronę przed zakończeniem wykonywania swojego fragmentu zadania obliczeniowego, potencjalnie te same fragmenty zadań będą realizowane wielokrotnie przez węzły *W* aż do uzyskania odpowiedzi (od własnych internautów lub od innych węzłów). Taki mechanizm jest potrzebny do zapewnienia niezawodności systemu. Dla zapewnienia wiarygodności (uodpornienia na ataki podstawieniowe) obliczenia muszą być powtarzane (potencjalnie przez różne węzły), zaś wyniki porównywane ze sobą i wybierane przez głosowanie (min. 3 odpowiedzi). Węzły dystrybuują zadania wg algorytmu pseudolosowego. Jeśli algorytm wskazuje na zadanie, które ma już wyniki, to węzeł *W* prześle zadanie do realizacji wówczas, gdy ilość odpowiedzi jest mniejsza od minimalnej. Wszystkie wyniki są rejestrowane wraz z ilością „oddanych głosów". Po skompletowaniu wszystkich wyników i scalaniu rozwiązania problemu wybierane są wyniki z największą ilością głosów (rys. 1.3).

W problemach niedeterministycznych nie da się zagwarantować wiarygodności tą metodą (wyniki obliczeń nie są znane z góry, a porównywanie ich nic nie da, bo odpowiedzi mogą być różne). Dlatego ze względu na koszt czasowy obliczeń, jeśli algorytm wyboru zadania przez węzeł *W* wskaże zadanie, które ma już odpowiedź, to węzeł pominie to zadanie. Nie da się w całości wyeliminować powtórzeń tą metodą, ale ilość powtórzeń będzie minimalna (algorytm pseudolosowy będzie powodował inicjowanie różnych zadań w tym samym czasie a algorytm synchronizacji daje duże prawdopodobieństwo, że to samo zadanie nie zostanie ponownie zainicjowane).

Założeniem jest dystrybuowanie zadań do internautów w sposób jawny, ale mało inwazyjny w normalne korzystanie z serwisów internetowych. Internauci zgłaszając się do serwerów publicznych pobierają z nich treści interesujące dla siebie (np. serwisy wiadomości, aplikacje do komunikacji *peer-to-peer*, aplikacje gier internetowych, muzyka, zdjęcia i filmy), a wraz z tymi treściami pobierają zadania obliczeniowe do wykonania.

Treści pobierane przez internautów można podzielić na następujące kategorie:

1. statyczne strony HTML – praktycznie bardzo krótko goszczą internautów, którzy po ewentualnym zapoznaniu się z treścią przechodzą do kolejnej strony,
2. dynamiczne strony HTML – tworzone dynamicznie po stronie serwera w oparciu np. o technologie JSP, ASP, AJAX; dominują we współczesnych serwisach typu

serwis wiadomości, serwis pogodowy, rozkład jazdy, rezerwacja biletów; są już ciekawsze dla internautów, jednak ich czas przebywania na takiej stronie jest ograniczony do kilku minut,
3. reklamy, bannery reklamowe w technologii Flash – stanowią często treści niepożądane z punktu widzenia internauty; jeśli są wkomponowane w treść strony, to jeszcze są tolerowane; jeśli zasłaniają właściwą treść strony, to są najczęściej jak najszybciej zamykane lub blokowane a priori,
4. zdjęcia, muzyka, filmy – ściągane przez programy typu *peer-to-peer* do odtworzenia na wbudowanej w system klienta przeglądarce multimedialnej lub z wyspecjalizowanych serwisów do odtwarzania na bieżąco w multimedialnej wtyczce do przeglądarki internetowej; na tych stronach internauci przebywają przez dość długi czas, a jeśli słuchają muzyki i oglądają filmy, to nawet kilkadziesiąt minut bez przerwy,
5. gry internetowe – są aplikacjami najczęściej graficznymi działającymi po stronie klienta (np. w technologii Java), sieć jest wykorzystywana do komunikowania się między graczami (np. w grach zespołowych); internauci potrafią spędzać na takich stronach nawet wiele godzin,
6. wiadomości od innych internautów – w formie wiadomości pocztowych (email) lub komunikatów przesyłanych na bieżąco przez specjalne komunikatory (typu Gadu-Gadu) wbudowane w przeglądarkę lub zainstalowane osobno w systemie; forma komunikatorów jest często sprzężona z 4. i 5. kategorią treści. W niektórych przypadkach tego typu aplikacje działają praktycznie bez przerwy,
7. aplikacje usługowe działające po stronie klienta – oparte o najnowsze technologie Flex i Silverlight; jeśli internauci potrzebują usług wymagających względnie dużej mocy obliczeniowej lub działających na względnie dużych danych, to zamiast wysyłać dane do obróbki po stronie serwera pobierają aplikację działającą po stronie klienta i w ten sposób rozproszony system wykorzystuje rozproszoną moc obliczeniową; internauci spędzają na tych stronach tyle czasu, ile potrzebują do wykonania usługi.

Z punktu widzenia celu systemu interesujące są te kategorie treści, które spełniają łącznie następujące warunki:

1. po stronie klienta jest uruchamiany kod wykonywalny w pewnym języku skryptowym,
2. umożliwiają w sposób nieinwazyjny uruchomienie procedur skryptowych, których wykonanie nie będzie powodowało istotnego spadku wydajności ani pogorszenia funkcjonalności zasadniczego kodu klienta,
3. angażują czas procesora klienta na tyle dużo, aby użytkownik nie odczuł wykonania zadania obliczeniowego w tle, ale jednocześnie na tyle mało, aby zadanie obliczeniowe mogło zostać wykonane,
4. angażują czas internautów na tyle długo, aby procedura skryptowa zdążyła wykonać dane zadanie i odesłać wyniki na serwer zanim internauta przełączy się na inną stronę,
5. umożliwiają dynamiczne dopasowanie się do zmieniających się zadań obliczeniowych po stronie serwera.

Stąd wynika, że w dalszych pracach trzeba skoncentrować się na treściach z kategorii 4–7. Wszystkie one działają w oparciu o aplikacje pracujące po stronie klienta i w taką aplikację muszą być wbudowane moduły obliczeniowe (wymaga to uzgodnienia między organizacją zarządzającą systemem obliczeniowym a kadrą zarządzającą serwerami *R* i jej administratorami).

Nie znaczy to, że należy całkowicie pominąć pierwsze trzy kategorie treści. Chociaż czas przebywania internauty na stronach HTML statycznych i dynamicznych (z kategorii 1 i 2) jest dość krótki, to zysk ze stosowania systemu przetwarzania wolontariatowego wynika ze skali obliczeń – bardzo dużej ilości często krótkich, podstawowych obliczeń. Dlatego również wykorzystanie statycznych i dynamicznych strony może być opłacalne. Jeśli są one zaopatrzone w bannery reklamowe Flash (kategoria 3), to można treść bannera wzbogacić o komendy ActionScript tworząc w ten sposób miniaplikację działającą po stronie klienta.

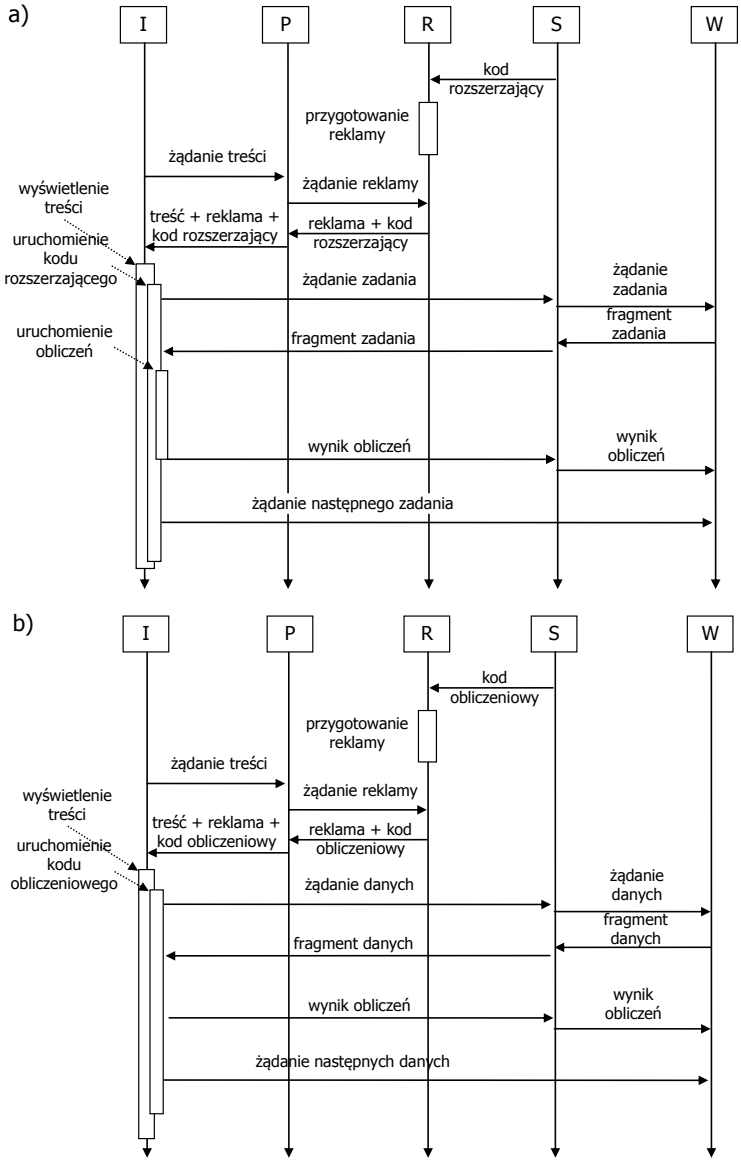
Elastyczne dopasowywanie się systemu do zmiennych zadań obliczeniowych trzeba zapewnić przez dołączanie zmieniających się modułów obliczeniowych do użytecznej treści WWW. Ważne jest, aby treść WWW udostępniana przez serwery *P* odwoływała się do serwerów *R*, które będą wzbogacały ją o kod rozszerzający pełniący funkcję *loadera*, który będzie pobierał z serwera *W* poprzez serwer *S* moduł obliczeniowy zawierający kod i dane do obliczenia Możliwe jest też inne rozwiązanie. Można do internauty dostarczyć gotowy kod obliczeniowy, który będzie pobierał jedynie dane. To rozwiązanie jest mniej elastyczne, ale nie wymaga tworzenia specjalnego środowiska uruchomieniowego po stronie internauty.

Tryb pracy w sytuacji szczególnego zagrożenia

W sytuacji szczególnego zagrożenia założenie o dobrowolnym uczestnictwie w dystrybucji i przetwarzaniu zadań staje się nieaktualne. Internauci świadomi zagrożenia instytucji państwowych mogą zdecydować się na udział w obliczeniach z pobudek patriotycznych, ale również odpowiednie przepisy prawne mogą usankcjonować narzucenie wykorzystywania ich mocy obliczeniowej. Wówczas serwisy *S* mogą zaoferować specjalną aplikację, która będzie jawnie wykonywać zadania obliczeniowe po stronie klienta na komputerach internautów, co pozwoli na pominięcie mechanizmu doklejania dodatkowego kodu do treści oferowanych przez serwery *P*.

Komercyjny tryb pracy

W zastosowaniach komercyjnych można zastosować obliczenia jawne i zwiększyć obciążenie komputerów internautów. Można też zwiększyć zaufanie do aplikacji i w ten sposób częściowo zapewnić możliwość obliczeń *peer-to-peer* (do pewnego stopnia, bo ograniczenia są przez zapory sieciowe).



Rys. 1.3. Sekwencja dystrybucji zadania: a) dla uniwersalnego kodu rozszerzającego, b) dla wyspecjalizowanego kodu obliczeniowego

1.5. Literatura

1. Kuchta Jarosław, Matuszek Mariusz, Czarnul Paweł, Szpryngier Piotr: *Projekt architektury środowiska laboratoryjnego systemu Comcute* - 2011, Raport techniczny WETI nr 32/2011
2. Use-It-Better: *Analiza możliwości rekrutacji mocy obliczeniowej w Internecie*. dokument wewnętrzny opracowany w ramach projektu Comcute, 2011
3. Couloris G., Dollimore J., Kindberg T.: *Distributed Systems: Concepts and Design*. Addison Wesley Longman Ltd., London, 1994