Wprowadzenie

Cezary Sobaniec
Jerzy Brzeziński

Rozwój technologiczny

Postęp w ostatnim półwieczu

- 10 mln dolarów, 1 instrukcja na sekundę
- 1000 dolarów, 100 mln instrukcji na sekundę
- 12^{12} razy lepszy współczynnik cena/efektywność
- sieci komputerowe LAN i WAN

Systemy komputerowe rozwijane są od roku 1945. Niestety do czasu wynalezienia minikomputerów ich ceny były bardzo wysokie. Sytuację zmieniły mikroprocesory 8-, 16-, 32- i w końcu 64-bitowe, charakteryzujące się dużą mocą obliczeniową, często przewyższającą moc dostępnych wcześniej komputerów.

Rozwój systemów komputerowych był rzeczywiście oszałamiający, powodując wzrost współczynnika cena/efektywność o 12 rzędów wielkości.

Drugim osiągnięciem, które przyczyniło się do zaistnienia i rozwoju systemów rozproszonych były sieci komputerowe, zarówno lokalne (LAN – ang. *Local-Area Networks*), jak i rozległe (WAN – ang. *Wide-Area Networks*). Szybkości przesyłanych danych wahają się od kilkudziesięciu Kb/s (kilobitów na sekundę) do dziesiątek Gb/s.

Definicja systemu rozproszonego

System rozproszony — zestaw niezależnych komputerów, sprawiający na jego użytkownikach wrażenie jednego, logicznie zwartego systemu.

Aspekty

- sprzęt: maszyny są autonomiczne
- oprogramowanie: wrażenie pojedynczego systemu

Nie istnieje jedna, ogólnie przyjęta definicja systemu rozproszonego. Powodem tego jest między innymi różnorodność aspektów, w jakich systemy te są rozpatrywane. Powyższa definicja zwraca uwagę na integralność systemu rozproszonego, sprawiającego na użytkowniku końcowym wrażenie pracy z systemem scentralizowanym. Jest to własność wysoce pożądana ze względu na złożoność systemów rozproszonych.

Cechy systemów rozproszonych

- Ukrycie przed użytkownikami:
  - różnic pomiędzy poszczególnymi komputerami
  - sposobów komunikowania się komputerów
  - wewnętrznej organizacji systemu rozproszonego
- Jednolity i spójny interfejs dla użytkownika — niezależnie od czasu i miejsca interakcji
- Łatwość rozszerzania (skalowania)

Uzyskanie spójnego i zwartego obrazu systemu rozproszonego wymaga ukrycia przed użytkownikami wielu szczegółów związanych z jego organizacją. Oznacza to, że różnice pomiędzy komputerami (m.in. architektura komputera, lokalny system operacyjny), różne sposoby komunikowania się komputerów (technologie komunikacyjne, protokoły sieciowe) oraz organizacja systemu rozproszonego są (lub powinny być) niewidoczne dla użytkownika końcowego. Ujednolicenie takie daje w efekcie możliwość korzystania z zasobów systemu w jednolity sposób, korzystając z tego samego interfejsu, niezależnie od czasu i miejsca dokonywanej interakcji.

Kolejna cecha systemów rozproszonych to łatwość ich rozszerzania, co w konsekwencji powinno prowadzić do zapewnienia skalowalności systemu. Łatwość ta jest efektem niezależności komputerów, przy jednoczesnym ukrywaniu szczegółów dotyczących funkcji pełnionych przez poszczególne komputery w systemie. Ze względu na powielanie pewnych funkcji przez wiele komputerów, system rozproszony może sprawiać wrażenie nieustannej dostępności jego zasobów i usług. Dzieje się tak pomimo przejściowych awarii poszczególnych jego komponentów, które są jednak maskowane przejęciem obsługi przez inne komputery. Podobnie dodanie nowych komputerów i/lub użytkowników pozostaje niezauważone przez dotychczasowych użytkowników.

Architektura systemu rozproszonego

- Rozbudowa własności lokalnych systemów operacyjnych (SO)
- Usługi dla aplikacji rozproszonych
- Oprogramowanie warstwy pośredniej (ang. *middleware*)

Wrażenie jednolitości systemu można uzyskać poprzez zastosowanie architektury warstwowej w odniesieniu do oprogramowania. System taki bazuje na lokalnych systemach operacyjnych i nadbudowuje warstwę pośrednią (ang. *middleware*), dostarczającą ujednoliconego interfejsu dostępu do usług dla aplikacji rozproszonych.

Cele systemów rozproszonych

- Łatwe połączenie użytkownik-zasoby
- Ukrywanie faktu rozproszenia zasobów
- Otwartość
- Skalowalność

Systemy rozproszone są tworzone ze względu na istotne *potencjalne* zalety tych systemów. Efektywne zagospodarowanie tych zalet może sprawić, że będą one bardziej atrakcyjne dla użytkownika końcowego niż systemy scentralizowane.

Łączenie użytkowników i zasobów

- Ekonomia: współdzielony dostęp do zasobów jest tańszy
  - drukarki
  - specjalizowane komputery
  - szybkie pamięci
  - bazy danych
  - zdalne dokumenty
- Bezpieczeństwo

Systemy rozproszone powstają po to, by ułatwić użytkownikom dostęp do zdalnych zasobów, ukrywając fakt ich rozproszenia i umożliwiając współdzielenie. Pojęcie zasób powinno tu być rozumiane szeroko: począwszy od urządzeń (drukarki, skanery), a skończywszy na plikach, stronach WWW. Współdzielony dostęp do zasobów jest uzasadniony głównie ekonomicznie: wiele zasobów jest drogich, zarówno w zakupie jak i w późniejszym utrzymaniu. Korzystanie ze wspólnych zasobów daje też możliwość szybkiej wymiany informacji pomiędzy samymi użytkownikami.

Łatwy dostęp do zasobów nie powinien jednak oznaczać rezygnacji z zapewniania bezpieczeństwa realizowanych operacji. Dotyczy to zarówno poufności, spójności przesyłanych danych, jak i niezaprzeczalności. Problemy bezpieczeństwa ujawniają się również w kontekście samej komunikacji. Monitorowanie aktywności użytkowników i budowanie profili zachowań również może naruszać naszą prywatność.

Przezroczystość (I)

System przezroczysty — (transparentny, ang. *transparent*) sprawia wrażenie systemu scentralizowanego (dla użytkowników i aplikacji).

Przezroczystość dostępu (ang. *access transparency*) ujednolicanie metod dostępu do danych i ukrywanie różnic w reprezentacji danych.

Rozproszenie zasobów i procesów na fizycznie rozłącznych maszynach może być maskowane tak, aby system był postrzegany jako jedna spójna całość. Mówimy o takim systemie, że jest *transparentny*. Przezroczystość może być jednak postrzegana na różnych poziomach, a poziomy te mogą być mniej lub bardziej istotne dla użytkownika końcowego. Poniżej przedstawiono podstawowe poziomy przezroczystości rozpatrywane w systemach rozproszonych:

Przezroczystość dostępu oznacza ujednolicenie metod dostępu do danych i ukrywanie różnic w reprezentacji danych. Użytkownik korzysta cały czas z tego samego interfejsu dostępu do danych. Różnice w reprezentacji danych mogą wynikać z zastosowania różnych architektur komputerowych. Przykładem jest reprezentacja liczb --- procesory Intela stosują tzw. kodowanie *little endian* a np. procesory Sun SPARC stosują kodowanie *big endian*. Różnica polega na kolejności ułożenia poszczególnych bajtów liczby w pamięci.
Innym przykładem różnic w reprezentacji danych są różne konwencje nazewnictwa plików stosowane w różnych systemach operacyjnych.

Przezroczystość (II)

Przezroczystość położenia (ang. *location transparency*) – użytkownicy nie mogą określić fizycznego położenia zasobu (np. na podstawie jego nazwy czy identyfikatora).

Przezroczystość wędrówki (ang. *migration transparency*) – można przenosić zasoby pomiędzy serwerami bez zmiany sposoby odwoływania się do nich.

Przezroczystość przemieszczania (ang. *relocation transparency*) – zasoby mogą być przenoszone nawet podczas ich używania.

Przezroczystość położenia oznacza, że użytkownicy nie mogą określić fizycznego położenia zasobu (np. na podstawie jego nazwy czy identyfikatora). Warunkiem koniecznym osiągnięcia przezroczystości położenia jest stosowanie nazewnictwa zasobów, które abstrahuje od ich położenia. Z tego między innymi powodu adresy dokumentów w usłudze WWW zaczęto określać nie jako adresy URL (ang. *Universal Resource Locator* --- uniwersalny lokalizator zasobów) a raczej jako adresy URI (ang. *Universal Resource Identifier* --- uniwersalny identyfikator zasobów). Zaakcentowano w ten sposób bardziej ogólny sposób interpretacji elementów składowych adresu. Np. adres: *http://www.put.poznan.pl/rekrutacja/rekrutacja.html* może być interpretowany jako wskazanie na dokument znajdujący się na serwerze *www.put.poznan.pl* (URL) lub po prostu całościowo jako identyfikator dokumentu (URI).

Przezroczystość wędrówki oznacza, że zasoby mogą być przenoszone pomiędzy serwerami bez potrzeby zmiany sposobu odwoływania się do nich. Uzyskanie przezroczystości wędrówki zakłada oczywiście przezroczystość położenia, gdyż w przeciwnym wypadku po dokonaniu przeniesienia zasobu jego stary identyfikator stawałby się nieaktualny.

Przezroczystość przemieszczania oznacza, że zasoby mogą być przenoszone nawet wtedy gdy są używane przez użytkowników i nie wymaga to informowania użytkowników o zmianie położenia. Z taką sytuacją mamy do czynienia np. w przypadku sieci komórkowych, gdzie użytkownicy pomimo fizycznego przemieszczania się mogą prowadzić stale rozmowę (jest to tzw. *roaming*).

Przezroczystość (III)

Przezroczystość zwielokrotniania (ang. *replication transparency*) ukrywanie przed użytkownikami faktu zwielokrotniania (replikacji) zasobów.

Przezroczystość współbieżności (ang. *concurrency transparency*) możliwość współbieżnego przetwarzania danych nie powodująca powstawania niespójności w systemie.

W systemach rozproszonych zwielokrotnianie (replikacja) jest jednym z podstawowych mechanizmów pozwalających na zwiększenie wydajności i niezawodności takich systemów. Stosowanie zwielokrotniania może skutkować jednak znacznymi utrudnieniami dla użytkowników końcowych, wynikającymi m.in. z niespójności współbieżnie modyfikowanych kopii.

Przezroczystość zwielokrotniania oznacza, że pomimo zwielokrotniania zasobów użytkownicy nie zauważają tego faktu — korzystają z zasobów dokładnie w taki sam sposób jak w systemie nie stosującym zwielokrotniania. W praktyce oznacza to, że przezroczystość zwielokrotniania można uzyskać w systemach gwarantujących przezroczystość położenia, ponieważ dostęp do każdej z kopii zasobu powinien być realizowany z wykorzystaniem tego samego identyfikatora.

System rozproszony umożliwia współdzielenie zasobów. Współdzielenie takie może być realizowane w sposób *kooperatywny* – jak to ma miejsce np. w przypadku komunikacji – lub w sposób *rywalizacyjny*, kiedy wielu użytkowników jednocześnie ubiega się o dostęp do tego samego zasobu.

Przezroczystość współbieżności gwarantuje, że współbieżne odwoływanie się do tego samego zasobu realizowane przez wielu użytkowników nie będzie prowadziło do powstania stanu niespójnego w systemie. Stan spójny można osiągnąć poprzez blokowanie dostępu do wspólnych danych (gwarantujące wyłączny dostęp) lub poprzez zastosowanie mechanizmu przetwarzania transakcyjnego, które jednak jest trudne do zrealizowania w systemie rozproszonym.

Przezroczystość (IV)

Przezroczystość awarii (ang. *failure transparent*) maskowanie przejściowych awarii poszczególnych komponentów systemu rozproszonego.

Przezroczystość trwałości (ang. *persistence transparency*) maskowanie sposobu przechowywania zasobu (pamięć ulotna, dysk).

Kompromis pomiędzy dużym stopniem przezroczystości a efektywnością systemu.

System jako całość powinien działać nawet w przypadku awarii pojedynczych jego komponentów.

Przezroczystość awarii oznacza, że użytkownik nie zauważa faktu uszkodzenia pojedynczych węzłów. Wadliwy komponent powinien zostać zastąpiony poprawnym, a cała operacja nie powinna wpływać na spsób korzystania z systemu. Maskowanie awarii jest zadaniem trudnym, a przy przyjęciu pewnych założeń, nawet niemożliwym. Główna trudność polega na odróżnieniu awarii zdalnego węzła od awarii łączy komunikacyjnych.

Przezroczystość trwałości ukrywa mechanizmy zarządzania zdalnymi zasobami. Przykładem mogą tu być zdalne obiekty, na rzecz których użytkownicy wywołują metody. Obiekt powinien trwale przechować swój stan po zdalnym wywołaniu jego metody. Z drugiej strony odwołanie do tego obiektu powinno być możliwe nawet wtedy, gdy nie jest przechowywany aktualnie w pamięci.

Oczekuje się, że systemy rozproszone będą ukrywać przed użytkownikiem szczegóły swojej wewnętrznej organizacji, jednakże osiągnięcie wysokiego poziomu przezroczystości wiąże się z dużymi kosztami, głównie związanymi z ogólną wydajnością systemu. W praktyce dąży się do osiągnięcia racjonalnego kompromisu pomiędzy obserwowaną przez użytkownika złożonością systemu, a efektywnością jego pracy.

Otwartość systemów rozproszonych

- Usługi zgodne ze standardowymi regułami opisującymi ich składnię i semantykę
  - przykład: protokoły komunikacyjne w sieciach komputerowych
- Specyfikacja usług poprzez interfejsy
  - język opisu interfejsu (ang. *Interface Definition Language*)
- Specyfikacje muszą być *zupełne* (kompletne) i *neutralne*
  - zdolność do współdziałania (ang. *interoperability*)
  - przenośność (ang. *portability*)

Systemy rozproszone, aby mogły być rozbudowywane, muszą być **otwarte**. Oznacza to konieczność standaryzacji stosowanych protokołów i interfejsów komunikacyjnych. Definicje interfejsów obejmują najczęściej opisz składnię usług (nazwy funkcji, typy parametrów i zwracanych wyników). Semantyka usług opisywana jest najczęściej w sposób nieformalny, korzystając z języka naturalnego.

Specyfikacja interfejsu aby mogła być implementowana niezależnie przez wielu dostawców oprogramowania musi być **zupełna** (kompletna) i **neutralna**. Kompletność oznacza, że opis jest wystarczający do stworzenia implementacji. Neutralność oznacza, że specyfikacja nie narzuca żadnych szczegółów dotyczących implementacji. Spełnienie tych warunków jest konieczne dla osiągnięcia **zdolności do współdziałania** (ang. *interoperability*), która oznacza możliwość współpracy ze sobą komponentów programowych pochodzących od różnych dostawców o ile implementują one odpowiednie interfejsy programowe.

Przenośność (ang. *portability*) oznacza możliwość uruchomienia aplikacji stworzonej dla jednego systemu w innym systemie bez konieczności wprowadzania jakichkolwiek modyfikacji.

Systemy elastyczne

Systemy elastyczne
- łatwość konfiguracji
- łatwość rekonfiguracji (np. wymiana poszczególnych komponentów)

Organizacja systemu rozproszonego
- projekt monolityczny
- logiczne wydzielenie składowych
- podział na autonomiczne komponenty (koszt: wydajność)
- oddzielenie polityki od mechanizmu

Dobrze zaprojektowany system otwarty powinien być **elastyczny**, co w praktyce oznacza łatwość budowy i przebudowy systemu składającego się z komponentów pochodzących od różnych dostawców. Osiągnięcie wysokiej elastyczności jest możliwe poprzez podział systemu rozproszonego na wysoce autonomiczne komponenty komunikujące się poprzez precyzyjnie opisane interfejsy. Daje to możliwość wymiany poszczególnych komponentów bez konieczności wyłączania całego systemu.

Otwartość i elastyczność systemów rozproszonych może być wspierana poprzez oddzielenie polityki od mechanizmu. Polityka określa deklaratywnie cele, które chce osiągnąć użytkownik, a mechanizm dostarcza narzędzi do osiągania tych celów. Przykładem może być stosowanie pamięci podręcznych w przeglądarkach internetowych. Jest to mechanizm zwiększający dostępność zasobów w usłudze WWW. Ważne jednakże są tu parametry wejściowe sterujące pracą tej pamięci (czas przechowywania dokumentów, strategia aktualizacji, wybór dokumentów). Parametry te określają właśnie politykę jaką chce się narzucić użytkownik i która może być potencjalnie realizowana z wykorzystaniem innego mechanizmu.

Skalowalność

Wymiary skalowalności

- skalowalność pod względem rozmiaru
  - decentralizacja: danych, usług i algorytmów
- skalowalność geograficzna
- skalowalność pod względem administracyjnym

Algorytmy zdecentralizowane

- brak informacji globalnej
- decyzje na podstawie informacji lokalnych
- odporność na awarie pojedynczych maszyn
- brak założeń dot. istnienia globalnego zegara

Rozwój sieci komputerowych i liczba komputerów przyłączanych do Internetu wzrastają tak szybko, że jednym z najważniejszych celów projektowych staje się obecnie zapewnienie skalowalności. Skalowalność może być rozważana w trzech różnych wymiarach. Po pierwsze skalowalność pod względem **rozmiaru**, oznaczająca możliwość dodawania do systemu nowych użytkowników i zasobów. Po drugie skalowalność **geograficzna**, umożliwiająca rozrzucenie poszczególnych użytkowników po całym świecie. Po trzecie w końcu skalowalność pod względem **administracyjnym** oznacza, że zarządzanie systemem pozostaje równie proste pomimo zwiększania jego rozmiaru i pomimo dystrybucji odpowiedzialności na wiele jednostek administracyjnych.

Skalowalność pod względem rozmiaru może być ograniczona poprzez stosowanie rozwiązań scentralizowanych. Każdy serwer, który jest jedynym pełniącym określoną funkcję, staje się wąskim gardłem w warunkach wzrastającej liczby żądań. Stosowanie przetwarzania scentralizowanego staje się niekiedy jednak nieuniknione. Przykładem może być przechowywanie poufnych danych, których powielenie powodowałoby zwiększenie ryzyka ich ujawnienia (centralizacja danych). Z centralizacją usług mamy do czynienia wtedy, gdy przetwarzanie zleceń jest realizowane przez pojedynczy serwer. Dobrym przykładem usługi, która jest doskonale zdecentralizowana jest usługa DNS (ang. *Domain Name Service*), zajmująca się m.in. odwzorowaniami nazw domenowych na adresy IP. W usłudze tej grupa rozproszonych serwerów realizuje wspólnie wspomnianą funkcję. Rozproszenie przetwarzania jest tu koniecznością, ponieważ liczba żądań przychodzących do systemu jest tak duża, że żaden serwer nie byłby w stanie jej obsłużyć.

Wysoką skalowalność systemów można osiągnąć poprzez zastosowanie algorytmów zdecentralizowanych. Charakteryzują się one następującymi własnościami:

1. Żadna z maszyn nie posiada pełnej informacji o stanie systemu.
2. Decyzje podejmowane są tylko na podstawie informacji lokalnych.
3. Uszkodzenie pojedynczych maszyn nie powoduje blokady całego systemu.
4. Nie ma niejawnych założeń dot. globalnego zegara.

Ostatnie założenie jest konsekwencją ogólnie przyjętej charakterystyki systemów rozproszonych, w których komunikacja ma charakter *asynchroniczny*. Oznacza to, że komunikaty docierają do odbiorcy w czasie skończonym, ale bliżej nieokreślonym. Konsekwencją tego założenia jest niemożliwość dokładnego zsynchronizowania lokalnych zegarów komputerów pracujących w sieci (przede wszystkim w sieci rozległej).

Skalowalność geograficzna jest ograniczana głównie przez dostępne mechanizmy komunikacyjne, które wprowadzają znaczne opóźnienia i charakteryzują się wysoką zawodnością. Opóźnienia powodują, że akceptowalne w sieciach lokalnych przetwarzanie *synchroniczne* staje się nieakceptowalne w sieciach rozległych, gdyż wprowadza zbyt duże przestoje.

Metody zapewniania skalowalności

- Ukrywanie opóźnień komunikacji
  - komunikacja asynchroniczna (systemy wsadowe)
  - przeniesienie części obliczeń na stronę klienta
- Rozpraszanie (ang. *distribution*)
  - DNS
- Zwielokrotnianie (replikacja – ang. *replication*)
  - zwiększenie dostępności
  - równoważenie obciążenia
  - zwiększenie niezawodności
  - przechowywanie podręczne (ang. *caching*)
  - problem **spójności** danych (ang. *consistency*)

Ponieważ skalowalność jest tak wysoce pożądaną cechą systemów rozproszonych, bardzo ważnym staje się pytanie o ogólne metody zapewniania wysokiej skalowalności. Generalnie istnieją trzy metody zwiększania skalowalności: ukrywanie opóźnień komunikacyjnych, rozpraszanie i zwielokrotnianie.

Ukrywanie opóźnień komunikacyjnych oznacza w praktyce stosowanie na szeroką skalę komunikacji asynchronicznej, w której zlecający operację nie czeka na jej wynik a wykonuje dalsze przetwarzanie nie wymagające wyniku ostatniej operacji. Efekt taki można uzyskać stosując przetwarzanie współbieżne po stronie klienta, w którym oczekiwanie na wynik będzie blokowało oddzielny wątek przetwarzania. Niestety komunikację asynchroniczną można stosować jedynie w systemach nieinteraktywnych. W systemach interaktywnych można próbować redukować ilość przesyłanych informacji poprzez przeniesienie części przetwarzania do klienta, np. w celu lepszej weryfikacji danych wejściowych. Można w tym celu wykorzystać np. dynamiczne przesyłanie kodu weryfikującego z serwera (aplety Javy).

Rozpraszanie (ang. *distribution*) polega na podziale zadań komponentu programowego na wiele jednostek i rozproszenie tych jednostek w sieci. Przykładem może być system DNS, w którym nie występuje pojedynczy serwer przechowujący całość informacji o konfiguracji odwzorowań nazw na adresy. Informacja ta jest rozproszona pomiędzy wszystkie *serwery nazw*, które odpowiadają za obsługę pojedynczych *domen* nazewniczych. Translację wybranej nazwy dokonuje serwer nazw dla domeny, z której pochodzi testowana nazwa.

Zwielokrotnianie (ang. *replication*) daje możliwość nie tylko zwiększenia dostępności zasobów, ale i również równoważenia obciążenia. W efekcie systemy stosujące replikację (potencjalnie) charakteryzują się większą wydajnością. Co więcej, zwielokrotniony serwer może być zastąpiony innym w przypadku awarii. Jeżeli zwielokrotnione serwery są dodatkowo rozproszone, to średnia „odległość" do najbliższego serwera ulega skróceniu, co powoduje dodatkowo maskowanie opóźnień komunikacyjnych.

Pewną formę zwielokrotniania stanowi stosowanie pamięci podręcznych (ang. *cache*), które są również kopią oryginalnych danych. O ile jednak w zwielokrotnianiu decyzję o utworzeniu kopii podejmuje właściciel zasobu, o tyle w przypadku przechowywania podręcznego decyzję taką podejmuje sam klient.

Tworzenie kopii zasobów powoduje jednak powstawanie problemu spójności danych, jeżeli jedna z kopii zostanie zmodyfikowana. Częste modyfikacje i ich propagacje do pozostałych serwerów mogą spowodować znaczne ograniczenie skalowalności systemu stosującego replikację. Z drugiej strony użytkownicy mogą odwoływać się do danych, które nie są już aktualne, co nie zawsze będzie akceptowalne (np. w przypadku operacji bankowych). Użytkownicy mogą formułować swoje oczekiwania co do spójności zwielokrotnianych danych poprzez tzw. *modele spójności* opisujące gwarancje, których udziela system. Dotyczą one propagacji zmian do pozostałych kopii oraz uporządkowania operacji realizowanych współbieżnie na wielu serwerach. Globalne porządkowanie operacji może jednak wymagać stosowania scentralizowanego przetwarzania, co oczywiście jest rozwiązaniem nieskalowalnym. Nie zawsze więc zwielokrotnianie jest właściwą metodą na zwiększanie skalowalności systemu.

Zagadnienia sprzętowe

Wieloprocesorowe systemy komputerowe

- wieloprocesory (ang. *multiprocessors*)
  - posiadają pamięć dzieloną (wspólną)
- multikomputery (ang. *multicomputers*)
  - brak pamięci dzielonej

Architektura połączeń

- szyna (ang. *bus*)
- przełącznik (ang. *switch*)

Systemy rozproszone budowane są z pojedynczych komputerów połączonych siecią. Z punktu widzenia programisty nie jest obojętne jaka jest budowa pojedynczych maszyn w systemie rozproszonym, gdyż rzutuje to na sposób programowania takich systemów. Generalnie można wyróżnić dwie klasy systemów komputerowych: wieloprocesory i multikomputery. Zasadnicza różnica polega na innej organizacji dostępu do pamięci. W wieloprocesorach wszystkie procesory mają dostęp do jednej, wspólnej przestrzeni adresowej. W multikomputerach każda jednostka ma swoją lokalną pamięć.

Drugim ważnym parametrem wyznaczającym charakter systemu rozproszonego jest architektura połączeń pomiędzy poszczególnymi węzłami. Połączenia mogą być realizowane poprzez centralną szynę lub w technologii przełączanej. Połączenia szynowe w wieloprocesorach ułatwiają zarządzanie spójnością danych, ale z drugiej strony bardzo ograniczają skalowalność; szyna przy niedużej liczbie procesorów staje się wąskim gardłem. Często stosowanym rozwiązaniem w takim przypadku jest zastosowanie pamięci podręcznych. Algorytmy wymiany zawartości pamięci podręcznej pozwalają na uzyskiwanie wysokich *współczynników trafień* (ang. *hit rate*), co pozwala na znaczne ograniczenie częstotliwości odwołań do głównej szyny. Z drugiej jednak strony stosowanie pamięci podręcznej powoduje powstawanie problemu spójności kopii tych samych danych przechowywanych na różnych węzłach.

Homogeniczne systemy multikomputerowe

Sieci systemowe (ang. *System Area Networks*)

- grupa komputerów homogenicznych połączonych siecią
- architektura połączeń: szyna lub przełącznik
- topologia połączeń: siatki (kraty, ang. *meshes*, *grids*) i hiperkostki

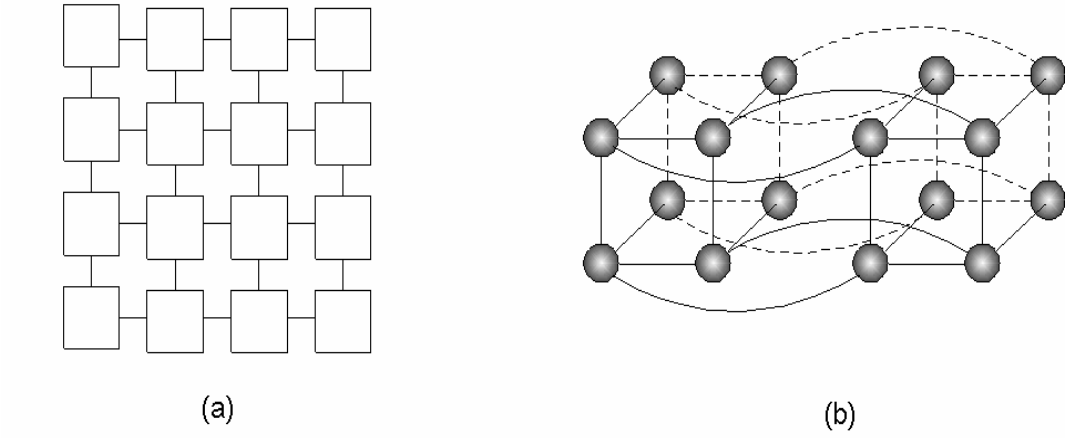
Realizacje

- procesory o masywnej równoległości (ang. *Massively Parallel Processors – MPP*)
- grona, grupy stacji roboczych (ang. *Clusters of Workstations – COW*)

Wieloprocesory budowane są poprzez połączenie wielu autonomicznych komputerów siecią. Jeżeli wykorzystywane są komputery o tej samej architekturze, to mamy do czynienia z siecią systemową. Połączenia mogą być również realizowane w architekturze szynowej lub przełączanej. W przypadku architektury przełączanej stosuje się *wyznaczanie tras* (*trasowanie*) komunikatów przez sieć połączeń. Dwie najpopularniejsze topologie połączeń pomiędzy węzłami to siatki i hiperkostki (zobacz następny slajd). Kraty, ponieważ są dwuwymiarowe, są łatwiejsze do implementacji na płaskiej płytce obwodu drukowanego. Hiperkostka jest sześcianem *n*-wymiarowym. Przykład z następnego slajdu pokazuje kostkę 4-wymiarową.

Praktyczne realizacje koncepcji multikomputera korzystają bądź ze specjalizowanej (często opatentowanej) sieci połączeń, bądź ze standardowych rozwiązań stosowanych w sieciach komputerowych (np. technologia Ethernet). Pierwsze podejście, oczywiście dużo droższe, stosowane jest w komputerach o masywnej równoległości. Multikomputery bazujące na standardowych technologiach zwane są z kolei gronami, grupami stacji roboczych lub po prostu klastrami.

Architektury połączeń multikomputera



Oprogramowanie dla syst. rozproszonych

Systemy operacyjne dla komputerów rozproszonych

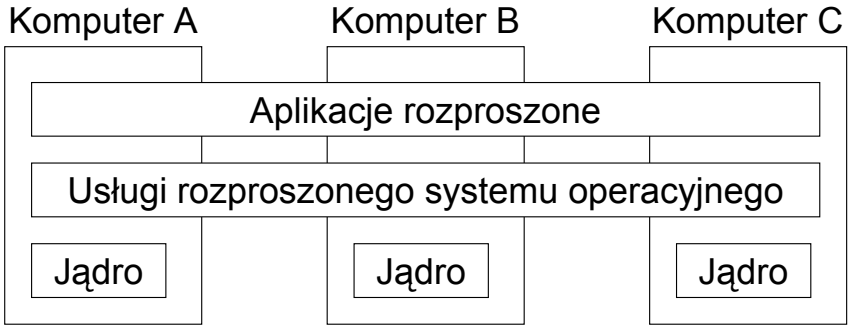
- systemy ściśle powiązane (ang. *tightly-coupled*)
  - wieloprocesory, multikomputery homogeniczne
- systemy luźno powiązane (ang. *loosely-coupled*)
  - sieciowe systemy operacyjne (ang. *network operating system*)
  - oprogramowanie warstwy pośredniej (ang. *middleware*)

Zadania systemu operacyjnego dla systemu rozproszonego są podobne do zadań klasycznego systemu operacyjnego i w dużej mierze sprowadzają się do zarządzania zasobami (ang. *resource management*), takimi jak procesory, pamięci, urządzenia zewnętrzne, sieci. Rozproszone systemy operacyjne mogą próbować zarządzać wszystkimi zasobami systemu rozproszonego globalnie – są to systemy ściśle powiązane. Systemy luźno powiązane to w zasadzie zbiór komputerów z lokalnymi systemami operacyjnymi, które ze sobą współpracują. Systemy ściśle powiązane projektowane są do obsługi wieloprocesorów i multikomputerów homogenicznych. Systemy operacyjne dla heterogenicznych systemów multikomputerowych określane są często jako **sieciowe systemy operacyjne** (ang. *network operating system*). Rozbudowa sieciowego systemu operacyjnego w celu osiągnięcia lepszej przezroczystości zasobów prowadzi do utworzenia **oprogramowania warstwy pośredniej** (ang. *middleware*).

Systemy operacyjne dla systemów rozproszonych

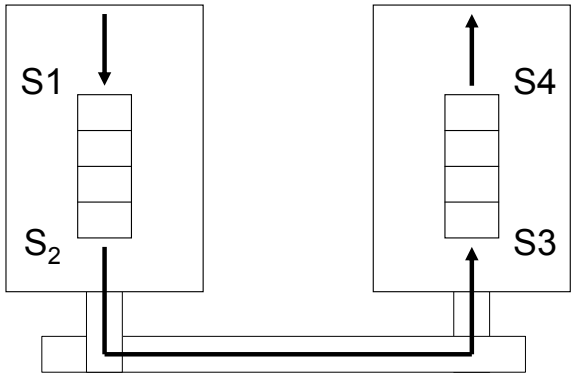
System	Opis	Główny cel
DOS	Ściśle powiązany system operacyjny dla wieloprocesorów i multikomputerów homogenicznych	Ukrywanie zasobów sprzętowych i zarządzanie nimi
NOS	Luźno powiązany system operacyjny dla multikomputerów heterogenicznych (sieci LAN i WAN)	Oferowanie lokalnych usług klientom zdalnym
Middleware	Dodatkowa warstwa na szczycie systemu NOS, realizująca usługi ogólnego przeznaczenia	Zapewnianie przezroczystości rozproszenia

Struktura wielokomputerowego systemu operacyjnego



System operacyjny dla multikomputera składa się z lokalnych systemów operacyjnych, które mają własne jądro, zawierające moduły zarządzania lokalnymi zasobami. Każdy węzeł ma również moduł komunikacyjny umożliwiający wysyłanie i odbiór komunikatów do innych węzłów. Powyżej każdego jądra znajduje się warstwa oprogramowania, która dostarcza dla aplikacji rozproszonych *maszyny wirtualnej* umożliwiającej równoległe i współbieżne wykonywanie zadań. Warstwa ta może również dostarczać programowej implementacji pamięci dzielonej. Inne zadania tej warstwy to: szeregowanie zadań, maskowanie awarii, zapewnianie przezroczystości pamięci.

Wymiana komunikatów



Systemy wielokomputerowe nie oferujące pamięci dzielonej udostępniają jedynie wymianę komunikatów. Mechanizm wymiany komunikatów wykorzystuje wewnętrznie buforowanie i wprowadza niekiedy blokowanie poszczególnych operacji. Buforowanie może być realizowane zarówno po stronie nadawczej jak i odbiorczej. Blokowanie nadawcy (punkt S1) może się zdarzyć, gdy bufor nadawczy jest przepełniony. Jeżeli bufora nadawczego nie ma, nadawca nadal może być blokowany do czasu faktycznego wysłania komunikatu (punkt S2). Blokowanie może być rozciągnięte w czasie do momentu przybycia komunikatu do odbiorcy (punkt S3) lub do momentu dostarczenia komunikatu (punkt S4). Blokowanie nadawcy do momentu przybycia komunikatu do odbiorcy w praktyce oznacza zagwarantowanie **niezawodnej** komunikacji (nadawca dowiaduje się, że komunikat dotarł).

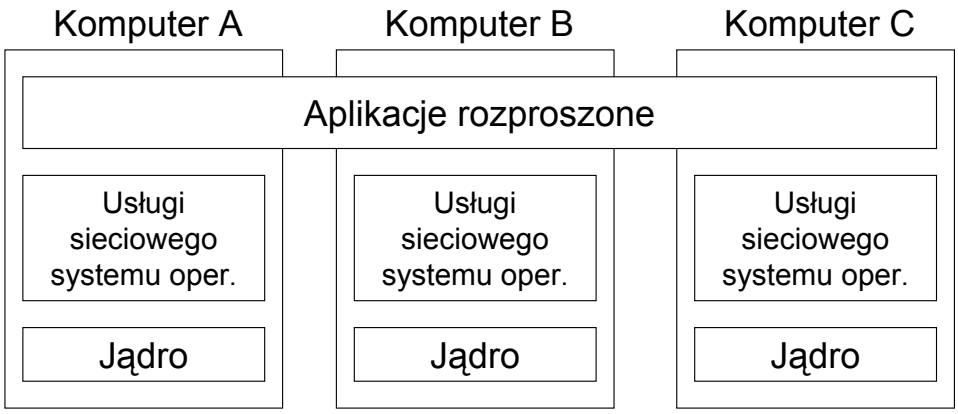
Rozproszona pamięć dzielona

- Programowanie multikomputerów jest trudniejsze niż programowanie wieloprocesorów
- Stronicowana rozproszona pamięć dzielona (ang. *Distributed Shared Memory*)
- Problemy z efektywnością
  - zwielokrotnianie stron przeznaczonych do odczytu
  - zwielokrotnianie wszystkich stron
  - rezygnacja ze ścisłej spójności
  - rozmiar stron (komunikacja vs. fałszywe dzielenie)

Jedną z form komunikacji pomiędzy procesami w systemie rozproszonym jest zastosowanie pamięci dzielonej. W przypadku multikomputerów jest to możliwe poprzez emulowanie działania takiej pamięci. Celem jest utworzenie wirtualnej pamięci, rozproszonej pomiędzy wieloma komputerami, która umożliwi tworzenie aplikacji dla takiego systemu rozproszonego analogicznie jak dla wieloprocesorów. Efekt pamięci dzielonej uzyskuje się poprzez podział wirtualnej przestrzeni adresowej na strony, których lokalizacje rozrzucone są po wszystkich komputerach. Odwołanie do strony nieobecnej lokalnie powoduje wystąpienie pułapki i sprowadzenie strony z węzła posiadającego odpowiednie dane. Idea jest więc taka sama jak w przypadku stronicowania, z tym że zamiast lokalnego dysku jako pamięci pomocniczej wykorzystywana jest zdalna pamięć RAM.

Efektywne zagospodarowanie koncepcji rozproszonej pamięci dzielonej napotyka jednak na poważne problemy, związane głównie z efektywnością pracy takiego systemu. Sprowadzenie strony jest operacją bardzo kosztowną, stąd tworzy się lokalne kopie stron, które nie są modyfikowane. Zwielokrotnianie stron modyfikowanych może prowadzić do powstania niespójności poszczególnych kopii, niespójności, które będą utrudniały pracę programiście, komplikując pierwotnie prosty interfejs. Oddzielną kwestią jest problem doboru wielkości strony. Duże strony optymalizują kwestie komunikacji, ale mogą prowadzić do nasilonego zjawiska **fałszywego dzielenia**. Zjawisko to pojawia się gdy dwa procesy odwołują się do różnych zmiennych, ale ulokowanych na tej samej stronie, co powoduje niepotrzebne przesyłanie zawartości tej strony w celu przeprowadzenia aktualizacji.

Sieciowy system operacyjny



Sieciowe systemy operacyjne nie stawiają sobie tak ambitnych celów jak rozproszone systemy operacyjne. System nie musi wyglądać jakby był całością. Zakłada się, że poszczególne węzły są obsługiwane przez lokalne systemy operacyjne, które udostępniają użytkownikom zbiór usług sieciowych. Systemy wchodzące w skład całego systemu rozproszonego nie muszą być jednorodne. Użytkownicy są świadomi istnienia poszczególnych maszyn w systemie i mogą wykonywać swoje zadania wskazując na system, który ma tego dokonać.

Sieciowe systemy operacyjne stawiają większe wymagania użytkownikom, gdyż użytkownicy są świadomi rozproszenia i są zmuszeni do posługiwania się systemem w nieco inny sposób. Ponieważ systemy wchodzące w skład sieciowego systemu są autonomiczne, zarządzanie nimi też musi być realizowane osobno, co jest dodatkowym utrudnieniem dla administratora i użytkowników (np. zmiana hasła musi być realizowana oddzielnie na każdym z serwerów).

Usługi sieciowego systemu operacyjnego

- Praca zdalna
  - np. uniksowy `rlogin`
- Kopiowanie plików
  - np. uniksowy `rcp`
- Sieciowy system plików
  - serwer plików (ang. *file server*)
  - klient

Typowe usługi sieciowych systemów operacyjnych to:

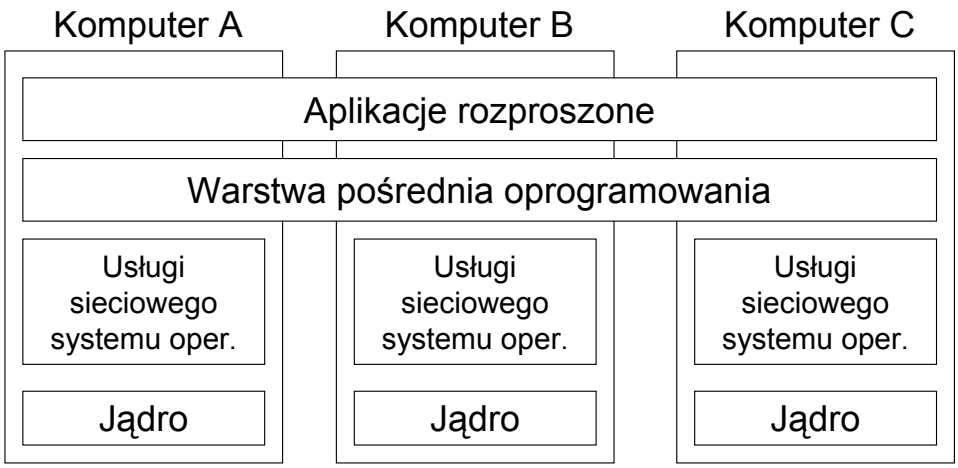
•zdalne logowanie (np. realizowane komendą *rlogin*)

•kopiowanie plików pomiędzy systemami (np. *rcp*)

•sieciowe systemy plików (serwery plików i klienci)

Sieciowe systemy plików są bardzo wygodne w użyciu gdyż przypominają lokalne systemy plików. Różni klienci mogą mieć różny obraz zawartości pojedynczego serwera plików w zależności od lokalnego systemu operacyjnego.

Oprogramowanie warstwy pośredniej



Rozproszony system operacyjny służy do obsługi multikomputerów, a więc komputerów ściśle ze sobą powiązanych. Sieciowy system operacyjny nie oferuje z kolei przezroczystości rozproszonego systemu operacyjnego. Uzyskanie systemu rozproszonego zgodnego z podaną wcześniej definicją wymaga wprowadzenia dodatkowej warstwy oprogramowania; warstwy pośredniej (ang. *middleware*), nadbudowującej nad usługami sieciowymi usługi dla systemu rozproszonego. W zakresie komunikacji np. sieciowe systemy operacyjne oferują interfejs gniazd (ang. *sockets*), umożliwiający komunikację pomiędzy rozproszonymi procesami, ale wymagający wskazania lokalizacji poszczególnych procesów (np. poprzez adresy IP). W systemie rozproszonym warstwa pośrednia może dostarczać mechanizmów transparentnej komunikacji, w które procesy identyfikowane są w sposób abstrakcyjny, niezależny od lokalizacji (potencjalnie zmiennej) procesów.

Zadania warstwy pośredniej dotyczą integracji systemów, które ze swojej strony są autonomiczne. Oznacza to, że zarządzanie zasobami lokalnymi jest powierzone lokalnemu systemowi operacyjnemu. Warstwa pośrednia nadbudowuje na tej bazie nowe usługi.

Warstwa pośrednia powinna ukrywać fakt heterogeniczności systemów składowych. Dzieje się to poprzez definicje nowych interfejsów komunikacyjnych, niezależnych od interfejsów lokalnych systemów operacyjnych. Aplikacje rozproszone powinny posługiwać się tylko tymi interfejsami i nie powinny korzystać bezpośrednio z usług lokalnego systemu operacyjnego.

Paradygmaty warstwy pośredniej

- „Wszystko jest plikiem” (Unix, Plan 9)
  - rozproszone systemy plików – sensowna skalowalność
- Zdalne wywołania procedur (ang. *Remote Procedure Calls*)
  - ukrywanie komunikacji
- Obiekty rozproszone (ang. *distributed objects*)
  - obiekt na jednej maszynie, interfejs do niego na wielu
- Model dokumentów rozproszonych (WWW)

Oprogramowanie warstwy pośredniej tworzone jest bardzo często wokół pewnych *paradygmatów*, a więc wzorców, do których dopasowuje się rożne fragmenty systemu. Jednym z nich jest traktowanie wszystkiego jako pliki – podobnie jak to ma miejsce w systemie Unix. Przy takim podejściu komunikacja staje się równoważna zapisowi do pliku, ponieważ pliki są widziane przez wszystkie procesy w taki sam sposób. Pewnym ograniczeniem tej koncepcji jest rozproszony system plików, gdzie transparentność dotyczy jedynie klasycznie rozumianych plików.

Innym podejściem jest koncepcja **zdalnych wywołań procedur**, gdzie główny nacisk położono na ukrywanie faktu komunikacji sieciowej. Wysłanie komunikatu do serwera i odbiór odpowiedzi jest traktowane jako wywołanie metody, co czyni tą koncepcję zbliżoną do programowania w systemach scentralizowanych (lokalne wywołanie procedury).

Pewną rozbudową koncepcji zdalnych wywołań procedur są obiekty rozproszone. Obiekty takie najczęściej są uruchamiane fizycznie na jednej maszynie, ale dostęp do nich jest możliwy w sposób transparentny z wielu innych maszyn. Jest to możliwe poprzez udostępnienie interfejsów do obiektów rozproszonych. Wywołanie metody obiektu powoduje tu również przesłanie komunikatu do i z serwera i zdalne wykonanie przetwarzania.

Popularność usługi WWW może być przykładem jak skrajne uproszczenie interfejsu i koncepcji może przyczynić się do upowszechnienia usługi. Dokumenty (nie tylko tekstowe) zawierają odnośniki do innych dokumentów tworząc w ten sposób ogólnoświatową sieć połączeń. Adres dokumentu jest w zasadzie jego identyfikatorem, nie wskazującym bezpośrednio na jego lokalizację.

Usługi warstwy pośredniej

- Komunikacja
  - RPC, zdalne obiekty, przezroczysty dostęp do rozproszonych plików, baz danych, dokumenty WWW
- Nazewnictwo (ang. *naming*)
  - lokalizacja zasobów – skalowalność
- Trwałość (ang. *persistency*)
  - pliki, bazy danych, rozproszona trwała pamięć dzielona
- Transakcje rozproszone (ang. *distributed transactions*)
  - własności ACID, dane na wielu maszynach, maskowanie awarii
- Bezpieczeństwo (ang. *security*)

Jedną z podstawowych usług warstwy pośredniej jest **komunikacja**, ukrywająca fakt rozproszenia poszczególnych maszyn. Może być realizowana z wykorzystaniem wspomnianych wcześniej zdalnych wywołań procedur, obiektów rozproszonych lub poprzez transparentny dostęp do danych (plików czy baz danych). Dostęp do rozproszonych dokumentów WWW może być też traktowany jako jednokierunkowa forma komunikacji serwer→użytkownik.

Kolejną ważną usługą warstwy pośredniej jest **usługa nazewnicza**, umożliwiająca publikowanie i wyszukiwanie informacji. Trudności w implementacji tej usługi wiążą się głównie z koniecznością zapewnienia jej wysokiej skalowalności. Czas dostępu do informacji, a więc np. odwzorowania identyfikatora na poszukiwany obiekt, powinien być praktycznie niezależny od wielkości systemu.

Z punktu widzenia aplikacji biznesowych istotną usługą jest możliwość zapewniania **trwałości** generowanym przez aplikację danych. Usługa ta jest oferowana w sposób naturalny w przypadku rozproszonych systemów plików czy rozproszonych baz danych, ale może też występować niezależnie jako element innych usług, np. rozproszonej pamięci dzielonej.

Przetwarzanie danych w systemach rozproszonych może wymagać stosowania **rozproszonych transakcji**. Jedną z własności transakcji jest atomowość, gwarantująca wykonanie wszystkich operacji składowych transakcji w sposób niepodzielny. Atomowa realizacja zbioru operacji modyfikujących na wielu różnych serwerach, które dodatkowo mogą ulegać awarii, jest zadaniem bardzo trudnym, a często wręcz niemożliwym do zrealizowania. Transakcje rozproszone generują również problemy związane ze skalowalnością.

Last but not least – czyli problem **bezpieczeństwa**, który jest często pomijany w prototypowych implementacjach, a który jest niezwykle istotny z praktycznego punktu widzenia. System rozproszony nie może polegać na mechanizmach bezpieczeństwa oferowanych i zarządzanych przez lokalne systemy operacyjne. Współpraca tych systemów wymaga, aby mechanizmy zapewniające bezpieczeństwo były realizowane od nowa i w sposób kompleksowy w warstwie pośredniej.

Otwartość warstwy pośredniej

- Nadbudowa nad SO → niezależność od SO
- Zależność od warstwy pośredniej
- Niekompletność interfejsów warstwy pośredniej
- Zgodność warstwy pośredniej ze standardem, ale nieprzenośność aplikacji

Oprogramowanie warstwy pośredniej jest budowane najczęściej dla pewnego zbioru systemów operacyjnych, a więc jego zastosowanie powinno nas uniezależnić od systemu operacyjnego. W praktyce jednak aplikacje muszą wykorzystywać interfejsy warstwy pośredniej, co powoduje powstanie zależności od tej warstwy. Nawet jeżeli interfejsy warstwy pośredniej zostały ustandaryzowane, nie daje to gwarancji przenośności aplikacji między różnymi implementacjami tego standardu. Wynika to z jednej strony z niekompletności interfejsów, powodujących konieczność odwoływania się bezpośrednio do systemu operacyjnego, a z drugiej strony z niezgodności zastosowanych protokołów (pomimo zgodności interfejsów).

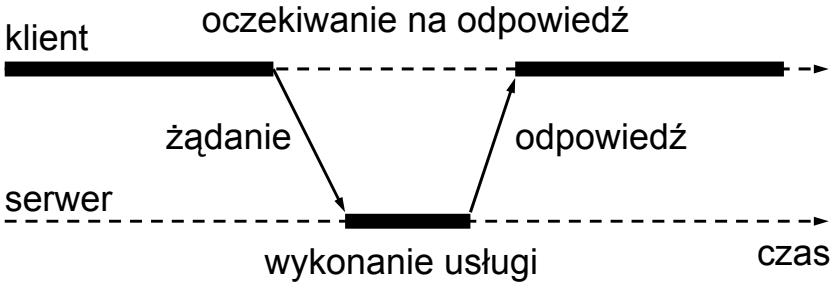
Porównanie systemów

	Rozproszony SO		Sieciowy SO	Middleware
	wielo-procesorowy	wielo-komputerowy		
Przezroczystość	bardzo duża	duża	mała	duża
Jeden SO	tak	tak	nie	nie
Liczba kopii SO	1	N	N	N
Komunikacja	pamięć dzielona	komunikaty	pliki	zależna od modelu
Zarządzanie zasobami	globalne, centralne	globalne, rozproszone	lokalne	lokalne
Skalowalność	nie	umiarkowana	tak	zmienna
Otwartość	zamknięty	zamknięty	otwarty	otwarty

Przedstawione w tabeli klasy systemów operacyjnych pokazują, że w praktyce nie ma systemów idealnych, a więc takich, które jednocześnie oferowałyby wysoką przezroczystość, były otwarte, wysoko wydajne i oferowały dobrą skalowalność. Systemy wieloprocesorowe nie są otwarte, ale za to bardzo efektywne, bo ten właśnie aspekt jest w nich najważniejszy. Sieciowe systemy operacyjne są otwarte i skalowalne, ale stosowane w nich mechanizmy komunikacyjne nie są zbyt wygodne i nie oferują one zadowalającej przezroczystości. Systemy oparte na warstwie pośredniej oferują dużą przezroczystość, ale kosztem jest tu ograniczona skalowalność.

Rozproszenie procesów

Model klient-serwer



Organizację systemu rozproszonego można opisywać na wiele sposobów. Jednym z nich jest wyróżnienie procesów, czy funkcji pełnionych przez określone procesy, poprzez wskazanie na serwery i klientów. Serwer jest procesem usługowym, udostępniającym określoną funkcjonalność. Klient zleca wykonanie akcji serwerowi. Funkcje klientów i serwerów mogą się nakładać na siebie. Schemat interakcji klient-serwer (przedstawiony na rysunku) opisywany jest jako zachowanie żądanie-odpowiedź (ang. *request-reply behaviour*).

Komunikacja pomiędzy klientem a serwerem może być realizowana z wykorzystanie protokołu bezpołączeniowego jak i połączeniowego. W pierwszym przypadku zyskujemy na efektywności przetwarzania (nie trzeba nawiązywać połączenia), ale trzeba uwzględnić możliwość utraty komunikatu w sieci. Co więcej: klient nie jest w stanie rozróżnić sytuacji zaginięcia komunikatu z żądaniem od zaginięcia komunikatu z odpowiedzią. W efekcie, powtarzając żądanie wykonania zdalnej usługi, może dojść do powielonego wykonania tych samych działań, co nie zawsze będzie akceptowalne.

Warstwy aplikacji

1. Poziom interfejsu użytkownika
   - np. dokument w przeglądarce
2. Poziom przetwarzania
   - np. przetworzenie zapytania w wyszukiwarce internetowej
3. Poziom danych
   - indeks stron WWW w bazie danych

Opisywanie aplikacji poprzez rozróżnianie funkcji serwera i klienta jest niekiedy niewygodne i nieodpowiednie. Bardziej adekwatny jest podział warstwowy aplikacji, wskazujący na funkcje poszczególnych modułów. Najczęściej spotykanym podejściem jest wyróżnienie w aplikacji wielu warstw odpowiedzialnych za różne poziomy przetwarzania: poziom interfejsu użytkownika, poziom przetwarzania i poziom danych.

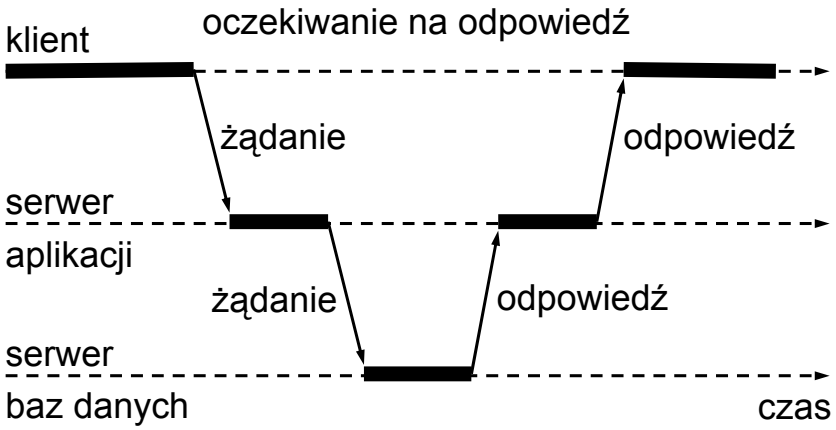
Poziom interfejsu użytkownika najczęściej jest częścią aplikacji-klienta, zawierając wszystko co jest niezbędne do przeprowadzenia bezpośredniej interakcji z użytkownikiem. Stopień skomplikowania interfejsu użytkownika może być bardzo różny: od prostego ekranu znakowego, poprzez interfejs webowy, aż do złożonego interfejsu graficznego.

Poziom przetwarzania to właściwa część aplikacji, zawierająca implementację określonego przetwarzania. Zakres działań wykonywanych przez tą warstwę może być bardzo różny i zależny od konkretnej aplikacji.

Poziom danych ma zapewnić przede wszystkim trwałość przechowywanych danych, oznaczająca możliwość dostępu do tych samych danych po wyłączeniu i włączeniu aplikacji. W najprostszej realizacji poziom danych może być zrealizowany w postaci plików, ale częściej stosowana jest do tego celu baza danych. Baza danych dodatkowo gwarantuje spójność danych pomiędzy wieloma aplikacjami. Dotyczy to zarówno schematu danych, jak i współbieżnego ich przetwarzania. Bazy danych mogą być zarówno typu relacyjnego (zastosowania biznesowe), jak i obiektowego (projektowanie wspomagane komputerowo, multimedia).

Przykład: wyszukiwarka internetowa. Interfejsem użytkownika w tym przypadku jest dokument HTML wyświetlany w przeglądarce. Po wysłaniu żądania do serwera następuje jego przetwarzanie, polegające na interpretacji zapytania, wyszukaniu surowych odpowiedzi, ich poklasyfikowaniu i przygotowaniu odpowiedzi w postaci odpowiedniego dokumentu HTML. Realizacja zapytania wymaga w tym przypadku sięgnięcia do bazy danych przechowującej indeks zawartości stron WWW w Internecie.

Architektura wielowarstwowa

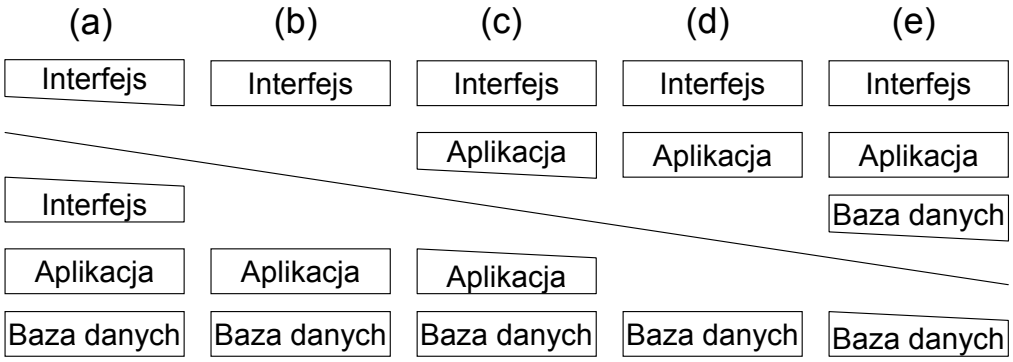


Serwery bardzo często pełnią rolę klientów odwołując się do innych usług. Można więc rozpatrywać budowę złożonego systemu rozproszonego jako architekturę wielowarstwową. Bardzo popularny jest model trójwarstwowy, w którym wyróżnia się **serwer aplikacji** implementujący logikę aplikacji i **serwer baz danych** zajmujący się już tylko składowaniem danych. Serwer aplikacji w takim modelu jest klientem serwera baz danych. Użytkownicy końcowi natomiast kierują swoje żądania tylko do serwera aplikacji. Wydzielenie serwera baz danych potencjalnie pozwala na jego podmianę bez konieczności zmiany kodu w aplikacjach klienckich.

Poszczególne warstwy oprogramowania architektury wielowarstwowej mogą być instalowane i uruchamiane na różnych komputerach, co jest naturalnym przejściem do systemów rozproszonych. Mówi się tu o **rozproszeniu pionowym** (ang. *vertical distribution*). W praktyce jednak znacznie istotniejsze jest **rozproszenie poziome** (ang. *horizontal distribution*), które oznacza, że rozproszeniu podlegają części składowe poszczególnych warstw. W przypadku baz danych np. może oznaczać to rozmieszczenie części danych na różnych serwerach. W efekcie uzyskujemy możliwość współbieżnego przetwarzania danych przez różne serwery, równoważąc w ten sposób obciążenie. Rozproszenie poziome może być też realizowane w formie zwielokrotnienia tych samych danych (usług) na wielu serwerach.

Rozproszenie może również dotyczyć klientów. W przypadku prostych aplikacji współpraca klientów może nawet nie wymagać obecności serwera lub jego funkcje zostaną ograniczone do lokalizacji pozostałych klientów. Mówimy w tym przypadku o **rozproszeniu partnerskim** (ang. *peer-to-peer distribution*).

Architektury klient-serwer



Rozróżnienie funkcji klienta i serwera może mieć różne przełożenie na implementację. W przypadku komputerów typu *mainframe*, gdzie całość przetwarzania odbywa się na centralnym serwerze, zadania klienta sprowadzają się jedynie do prezentacji tego, co wyśle serwer. Mówi się w tym przypadku o tzw. *cieńkim kliencie* (ang. *thin client*). Z drugiej jednak strony duża część przetwarzania może zostać przeniesiona z serwera do klienta celem odciążenia tego pierwszego. W takim przypadku mamy do czynienia z *grubym* lub *bogatym klientem* (ang. *fat/rich client*). Można też rozważać różne rozwiązania pośrednie. Na przedstawionym powyżej rysunku skrajnie po lewej mamy np. architekturę terminalową, gdzie klient realizuje tylko część interfejsu użytkownika (a). W przypadku (b) klient obsługuje w całości interfejs z użytkownikiem. Jeżeli część przetwarzania jest realizowana po stronie klienta, to dostajemy realizację z rys. (c). Przykładem takiej organizacji jest formularz HTML z kodem JavaScript dokonującym wstępnej weryfikacji danych wprowadzanych w przeglądarce. Aplikacja może realizować swoje zadania całkowicie po stronie klienta odwołując się tylko do bazy danych po stronie serwera (d). Przykładem takiej organizacji może być aplet Java pracujący w przeglądarce. W skrajnym przypadku również część danych może być składowana po stronie klienta realizując schemat z rys. (e). Sytuacja taka może mieć miejsce w przypadku aplikacji mobilnych, które mają łączność z serwerem tylko czasowo.