Systemy rozproszone

Zwielokrotnianie i spójność

Cezary Sobaniec

Zwielokrotnianie

Zwielokrotnianie polega na utrzymywaniu wielu kopii danych (obiektów) na niezależnych serwerach

Cele zwielokrotniania

1. zwiększenie efektywności
2. zwiększenie niezawodności (i dostępności)

Główne problem – spójność

- modele spójności
- protokoły spójności
- mobilność

Zwielokrotnianie i spójność (2)

Zwielokrotnianie (replikacja, ang. *replication*) jest bardzo ważnym mechanizmem stosowanym w systemach rozproszonych. Najogólniej polega ona na utrzymywaniu wielu kopii tych samych danych lub obiektów na wielu, niezależnych serwerach. Podstawową motywacją dla stosowania zwielokrotniania jest potencjalne zwiększenie efektywności oraz niezawodności przetwarzania. Osiągnięcie tych celów wymaga jednak rozwiązania wielu problemów, w śród których najważniejszym jest kwestia spójności danych. Modyfikacje wprowadzone są bowiem do pojedynczej kopii, a użytkownicy odwołują się do wszystkich, co może spowodować obserwowanie niespójności i w konsekwencji błędy w aplikacjach. Zdefiniowano wiele **modeli spójności**, szczególnie w ramach prac nad rozproszoną pamięcią współdzieloną. Modele te jednak często słabo się skalują, co powoduje, że trzeba dążyć do ich uproszczenia lub osłabienia, dając tym samym szansę na efektywniejszą implementację. Oddzielną kwestią jest mobilność uczestników przetwarzania, która dodaje nowy wymiar do problemu zarządzania spójnością. Istnieje klasa modeli spójności uwzględniająca ten aspekt z punktu widzenia pojedynczego klienta.

Model spójności jest tylko pewnego rodzaju opisem „matematycznym” własności systemu. Jego implementacja wymaga zaprojektowania odpowiedniego protokołu spójności czyli algorytmu rozproszonego dostarczającego gwarancji modelu spójności. Konstrukcja protokołu musi uwzględnić lokalizację replik i mechanizmów ich aktualizacji.

Po co zwielokrotniać?

Niezawodność

- odporność na awarie

Efektywność

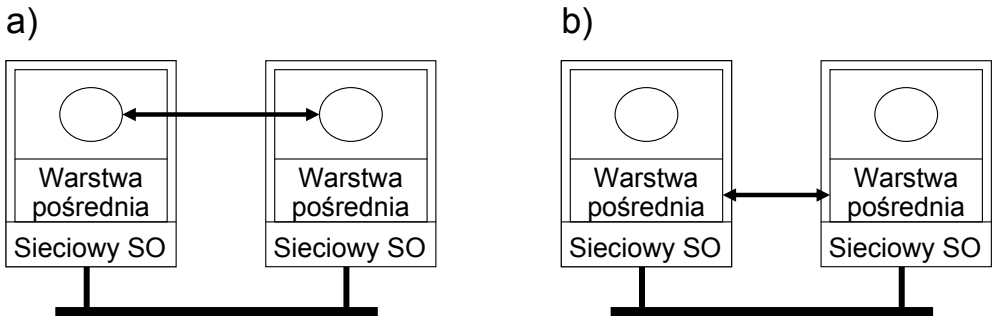
- współbieżny dostęp do wielu serwerów
  - równoważenie obciążenia
  - skalowalność w sensie liczbowym
- wykorzystanie bliższych serwerów
  - skalowalność w sensie geograficznym
  - mniejsze opóźnienia

Zwielokrotnianie może być wykorzystane do zwiększenia **niezawodności** systemu rozproszonego. Serwer, który ulega awarii może być zastąpiony innym, który jest dostępny, a który przechowuje te same dane. Co więcej, zmiana serwera w przypadku awarii może się dokonywać w sposób całkowicie przezroczysty dla użytkownika. Repliki można też wykorzystywać do maskowania błędów poprzednich zapisów. Mając 3 kopie i odczytując z nich wszystkich, można stwierdzić różnice i maskować błąd jednego serwera.

Zwiększanie **efektywności** przetwarzania może być rozpatrywane w dwóch kontekstach. Przy wzrastającej liczbie żądań pojedynczy serwer może być niewystarczający. Należy w takiej sytuacji system skalować w sensie liczbowym, wprowadzając serwery-repliki. Umożliwi to współbieżne wykorzystanie mocy obliczeniowych tych jednostek przez wielu klientów. Oczywiście należy zadbać, aby w miarę możliwości obciążenie serwerów było równomierne.

Z drugiej strony systemy rozproszone mogą wymagać skalowania w sensie geograficznym, co może zaowocować zbliżeniem się serwerów do użytkowników końcowych. W efekcie będą oni obserwować mniejsze opóźnienia komunikacyjne. Trzeba jednak pamiętać, że serwery muszą utrzymywać dane w stanie spójnym. Jeżeli są odległe od siebie, to koszty komunikacji związanej z synchronizacją danych będą odpowiednio wyższe, a w skrajnym przypadku mogą przekraczać zyski z lokalizacji bliżej klienta.

Zwielokrotnianie obiektów



Podejście obiektowe do tworzenia systemów informatycznych jest bardzo efektywne; pozwala na proste obudowanie danych metodami przetwarzającymi te dane, co pozwala na wyraźnie rozgraniczenie poszczególnych komponentów systemu. To samo podejście próbuje się zastosować w systemach rozproszonych. W tym kontekście problem zwielokrotniania obiektów staje się bardzo ważny.

Zwielokrotnienie obiektu musi być pewien sposób nadzorowane tak, aby uniknąć problemów ze spójnością. Jednym z możliwych rozwiązań jest np. wymuszenie wykonywania metod na poszczególnych kopiach w tej samej kolejności. Przy założeniu determinizmu przetwarzania w obiektach, stan końcowy po serii wywołań tych samych metod z tymi samymi argumentami powinien być identyczny. Synchronizację taką można przeprowadzić na dwa sposoby. W pierwszym podejściu (rysunek a) obiekt replikowany jest świadomy istnienia innych kopii i sam współuczestniczy w koordynacji aktualizacji poszczególnych kopii. System operacyjny w takim rozwiązaniu nie musi wspierać w żaden sposób zwielokrotniania. Wywołania metod są synchronizowane samodzielnie przez obiekt. Zaletą tego podejścia jest możliwość adaptacji strategii zwielokrotniania do faktycznych potrzeb danego obiektu bez potrzeby rekonfiguracji całego systemu.

Druga metoda (rysunek b) zakłada, że obiekty nie są świadome zwielokrotniania, co oznacza, że cały ciężar zarządzania spójnością spada na warstwę pośrednią (ang. *middleware*). System musi w tym przypadku zagwarantować globalne uporządkowanie wszystkich wywołań metod na wszystkich obiektach, co nie zawsze jest konieczne. Jest natomiast wygodne dla programistów samych obiektów, ponieważ nie muszą ich programować w żaden specjalny sposób.

Zwielokrotnianie a skalowalność

Kompromis

- skracanie czasu dostępu do serwerów
- utrzymywanie kopii w stanie spójnym
  - koszt utrzymywania nieużywanych kopii
- spójność odwzorowująca system scentralizowany
  - transakcyjna aktualizacja wszystkich kopii
  - globalna synchronizacja
- osłabienie modelu spójności
  - charakter aplikacji i zwielokrotnianych danych

Skalowalność w dużej mierze sprowadza się do konieczności zapewnienia satysfakcjonującej efektywności przetwarzania przy wzrastającej liczbie jego uczestników. Zastosowanie zwielokrotniania może wspomóc osiąganie skalowalności poprzez umieszczenie kopii w pobliżu użytkowników końcowych, co skraca czas dostępu do danych. Z drugiej jednak strony nowe kopie danych muszą być aktualizowane tak, aby uniknąć problemu spójności. Oznacza to, że trzeba ponieść koszty związane z dodatkową komunikacją pomiędzy poszczególnymi serwerami utrzymującymi repliki. Co więcej komunikacja ta musi być zsynchronizowana, co stawia jeszcze większe wymagania odnośnie przepustowości. Okazuje się więc, że zwielokrotnianie daje zyski jedynie potencjalnie. Faktyczne zwiększenie efektywności jest możliwe przy starannym opracowaniu protokołów spójności, uwzględniającym charakter i faktyczne wymagania aplikacji.

Rozproszona pamięć dzielona

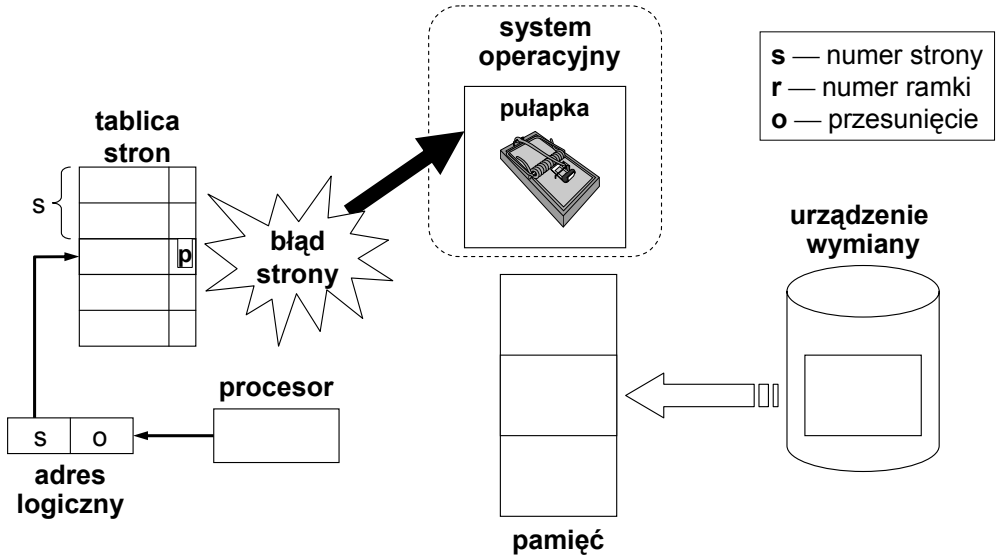
DSM (ang. *Distributed Shared Memory*) — wspólna wirtualna przestrzeń adresowa, dostępna dla wszystkich węzłów systemu rozproszonego.

Zalety

- wygodny paradygmat programowania równoległego
- skalowalność i łatwość rozbudowy
- dostęp do fizycznej pamięci wszystkich węzłów
- środowisko uruchomieniowe dla programów równoległych pisanych dla maszyn wieloprocesorowych

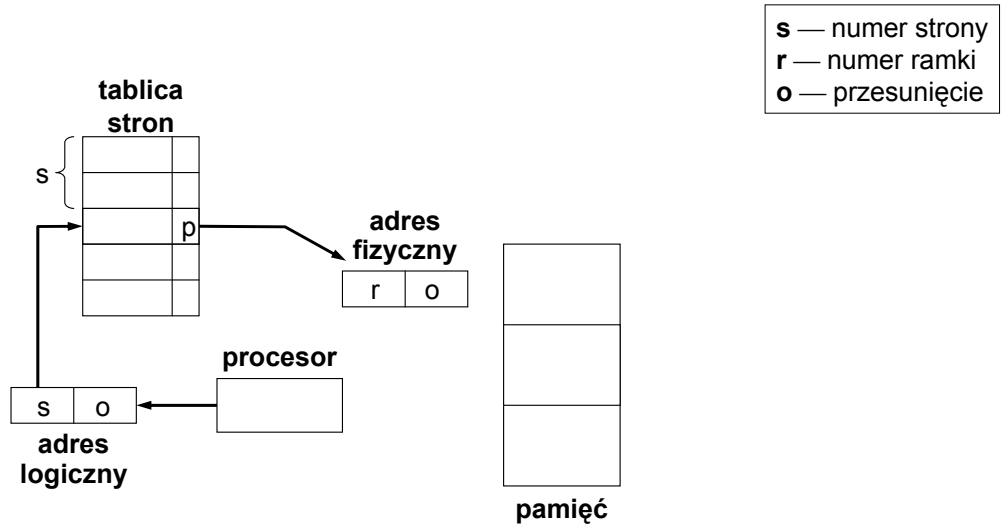
Programowanie multikomputerów jest znacznie trudniejsze od programowania wieloprocesorów; wymaga bowiem stosowania wymiany komunikatów. Wynika to z zupełnie innego sposobu programowania: zamiast operowania na pojęciach dobrze znanych programistom, takich jak pamięć, procesy, semafory czy monitory, trzeba w przypadku multikomputerów przesyłać surowe komunikaty. Jest to podejście o wiele trudniejsze i w praktyce będące źródłem wielu błędów programistycznych. Dodatkowo w przypadku systemów rozproszonych trzeba brać pod uwagę takie zagadnienia, jak: buforowanie, blokowanie czy zawodną komunikację. Jedną z propozycji ułatwiających budowanie aplikacji rozproszonych jest emulacja pamięci dzielonej występującej w wieloprocesorach w środowisku rozproszonym. Celem jest tu uzyskanie środowiska podobnego do wieloprocesorów a realizowanego na wielokomputerach. Stworzenie takiej *rozproszonej pamięci dzielonej* umożliwiłoby naturalne przenoszenie oprogramowania dostępnego dla wieloprocesorów do środowiska multikomputerów.

Błąd strony w systemie pamięci wirtualnej



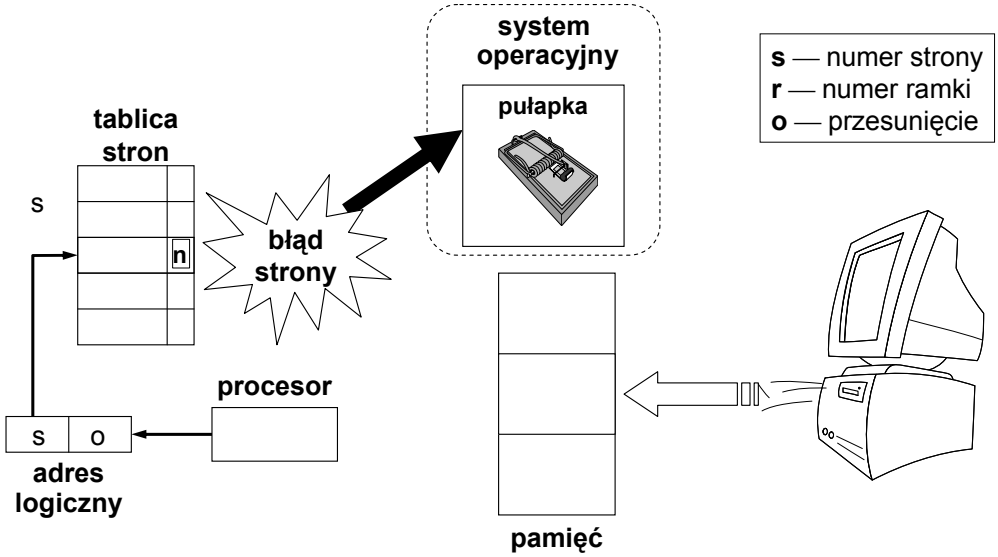
Rozproszona pamięć dzielona jest realizowana jako rozbudowa pamięci wirtualnych poszczególnych węzłów. Pamięć wirtualna jest realizowana najczęściej jako pamięć stronicowana. Cała przestrzeń adresowa jest podzielona na strony o stałym rozmiarze (zwykle 4 lub 8 KB). Adresy w pamięci wirtualnej składają się z dwóch części. Pierwsza (*s*) jest indeksem w tablicy stron. Druga część adresu (*o* -- *offset*) to przesunięcie na stronie. Tłumaczenie adresu wirtualnego na fizyczny wymaga odczytania z tablicy stron na pozycji *s* numeru ramki (*r*), gdzie dana strona jest przechowywana. Numer ramki jest częścią adresu fizycznego. Cały adres fizyczny powstaje poprzez sklejenie numeru ramki z przesunięciem na stronie. Ponieważ wirtualna przestrzeń adresowa jest znacznie większa od fizycznej przestrzeni adresowej, niemożliwe jest przechowywanie wszystkich ramek jednocześnie w pamięci. Powyższy rysunek prezentuje sytuację, gdy ramka *s* jest niedostępna (*p*). Odwołanie do takiej strony powoduje powstanie *błędu strony*, które musi być obsłużone przez system operacyjny. Obsługa polega na wczytaniu brakującej strony z urządzenia wymiany (najczęściej dysk). Po wczytaniu strony (do wolnej ramki), następuje aktualizacja tablicy stron, aby pozycja *s* wskazywała na ramkę zawierającą właśnie załadowaną stronę.

Błąd strony w systemie pamięci wirtualnej



Rysunek przedstawia sytuację, kiedy strona, do której następuje odwołanie znajduje się w pamięci. Adres wirtualny (*s*, *o*) zostaje przetłumaczony na adres fizyczny (*r*, *o*).

Obsługa błędu strony w systemie DSM



Obsługa błędu w systemie rozproszonej pamięci współdzielonej przebiega bardzo podobnie jak w przypadku pamięci wirtualnej. Jedyna różnica polega na tym, że strona nie jest sprowadzana z dysku lokalnego komputera a ze zdalnego systemu (z jego pamięci wirtualnej). Operacja taka oczywiście będzie obciążona większym kosztem wykonania, ponieważ pobranie strony wymaga wysłania komunikatu, przetworzenia żądania po drugiej stronie i wysłania odpowiedzi. Oznacza to, że wczytanie niedostępnej strony będzie realizowane znacznie dłużej; czas przesłania komunikatu w sieci jest kilka rzędów wielkości większy od czasu dostępu do lokalnej pamięci dynamicznej. W efekcie algorytmy zarządzania wymianą stron muszą bardzo precyzyjnie określać, które strony i kiedy mają być wczytywane. Sama organizacja pamięci powinna też w miarę możliwości minimalizować konieczność przesyłania stron.

Koncepcje dostępu do danych

- Dostęp zdalny – zawsze poprzez sieć
  - prostota koncepcji i implementacji
  - opóźnienia komunikacyjne
- Relokacja – fizyczna zmiana lokalizacji obiektu
  - zmniejszenie czasu dostępu
  - koszt przenoszenia obiektu między węzłami
  - opłacalne przy wielokrotnych, zgrupowanych odwołaniach
- Zwielokrotnianie – kopie w lokalnych węzłach
  - zmniejszenie czasu dostępu
  - problem spójności

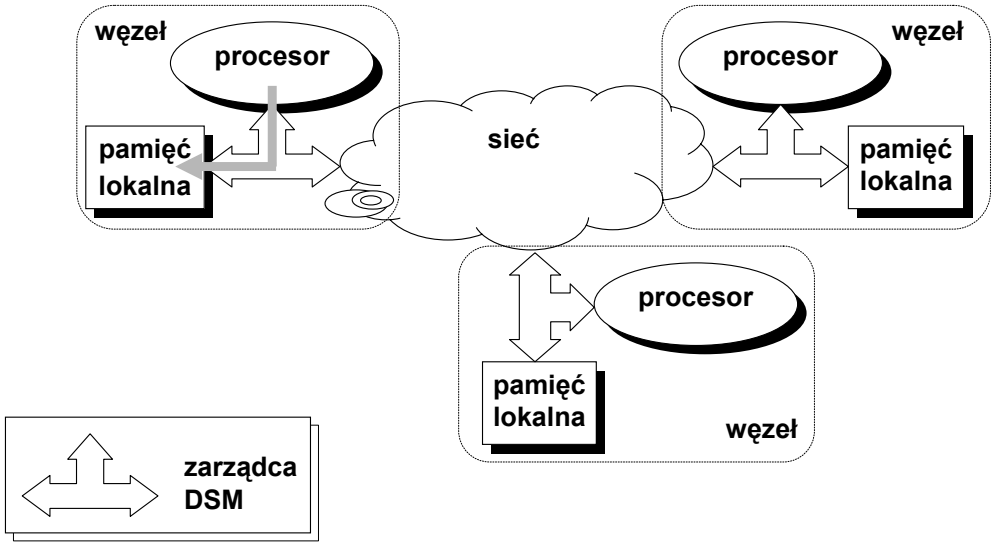
W systemie rozproszonym istnieje problem dostępu do zdalnych danych, niedostępnych lokalnie. Generalnie istnieją trzy możliwe rozwiązania tego problemu.

Pierwsze polega na wykonywaniu **zdalnego dostępu** do danych. W podejściu tym każdy dostęp do współdzielonego obiektu, zlokalizowanego fizycznie w pamięci lokalnej innego węzła, odbywa się przez sieć. Wymaga to oczywiście przesłania komunikatów, co wiąże się ze znacznymi opóźnieniami. Zaletą tego podejścia jest jego prostota. W systemie jest jedna kopia danych, do której się wszyscy odwołują.

Drugie rozwiązanie próbuje zaradzić sytuacji, gdy do obiektu odwołuje się bardzo często pojedynczy proces lub grupa procesów zlokalizowanych na innym komputerze niż sam obiekt. Jeżeli założymy, że możliwa jest zmiana fizycznej lokalizacji współdzielonego obiektu (**relokacja**), czyli umieszczenie go w pamięci lokalnej węzła, w którym pojawiło się żądanie dostępu, to potencjalnie możemy znacząco skrócić czas kolejnych dostępów do tych danych. Podejście to będzie opłacalne w przypadku aplikacji, w których występują zgrupowane, seryjne odwołania do obiektów z pojedynczych węzłów. W takiej sytuacji opłacalne będzie ponoszenie kosztów przenoszenia (potencjalnie dużego) obiektu między węzłami.

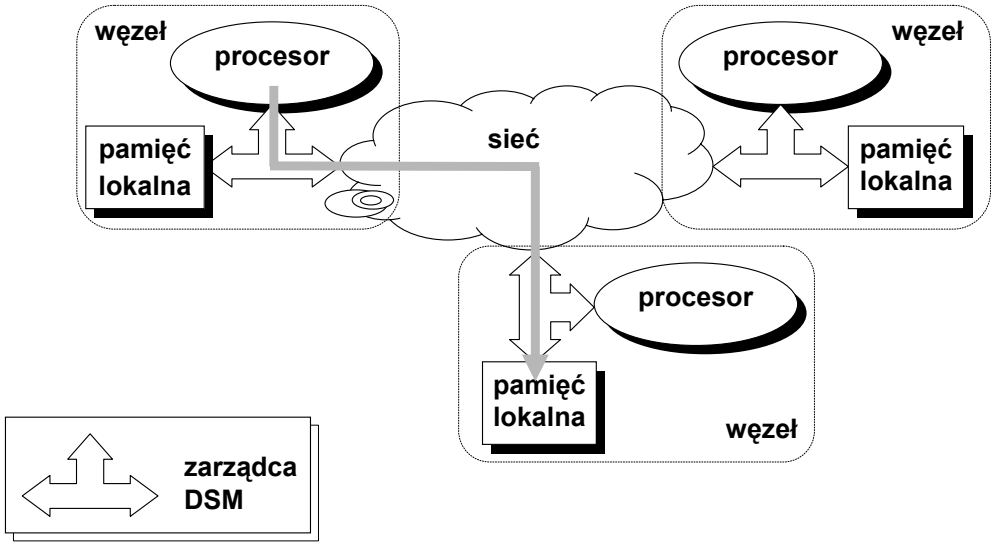
Trzecie rozwiązanie polega na zastosowaniu **zwielokrotniania**; obiekt logiczny może być jednocześnie zlokalizowany fizycznie w pamięci lokalnej wielu węzłów, co umożliwia równoległy dostęp do tego obiektu w wielu węzłach. Podstawowym problemem jest w tym przypadku sygnalizowana wcześniej spójność danych.

Dostęp lokalny



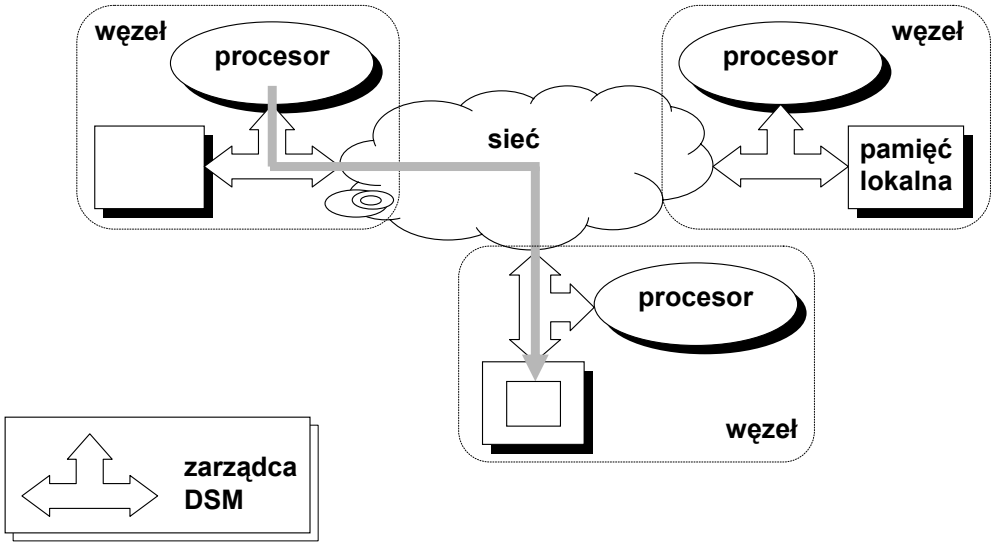
Dostęp lokalny nie wymaga komunikacji sieciowej. Jest więc realizowany z pełną szybkością. Generalnie należy dążyć do tego, aby zdecydowana większość dostępów do pamięci mogła być realizowana właśnie w ten sposób.

Dostęp zdalny



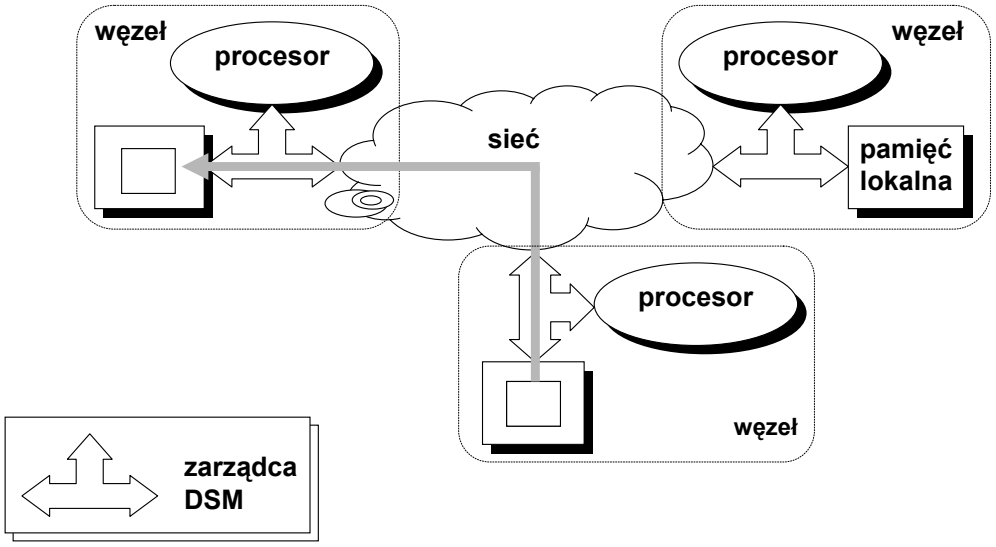
Dostęp zdalny wymaga przesyłania komunikatów: do i z węzła zdalnego. Komunikacja taka jest wykonywana przy każdym dostępie.

Relokacja

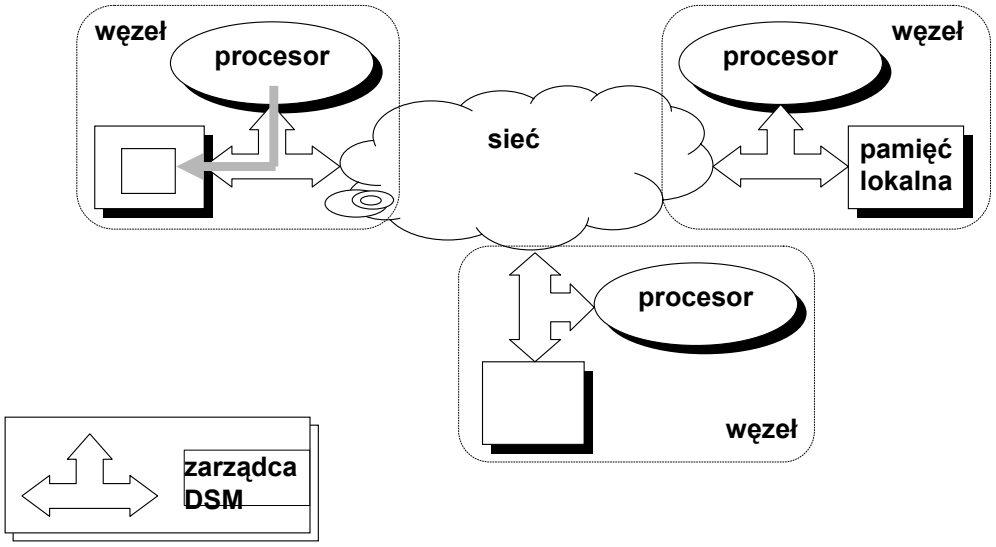


Relokacja wymaga komunikacji, ale – w przeciwieństwie do dostępu zdalnego – po przeniesieniu obiektu do lokalnego węzła kolejne dostępy będzie można wykonywać z pełną szybkością. Warunkiem jest tu oczywiście brak współbieżnych odwołań do tego samego obiektu, które mogłyby spowodować ponowne przeniesienie obiektu w inne miejsce.

Relokacja


węzeł
procesor
sieć
procesor
węzeł
pamięć
lokalna
procesor
węzeł
zarządca
DSM

Relokacja


węzeł
procesor
sieć
procesor
węzeł
pamięć
lokalna
procesor
węzeł
zarządca
DSM

Relokacja — charakterystyka

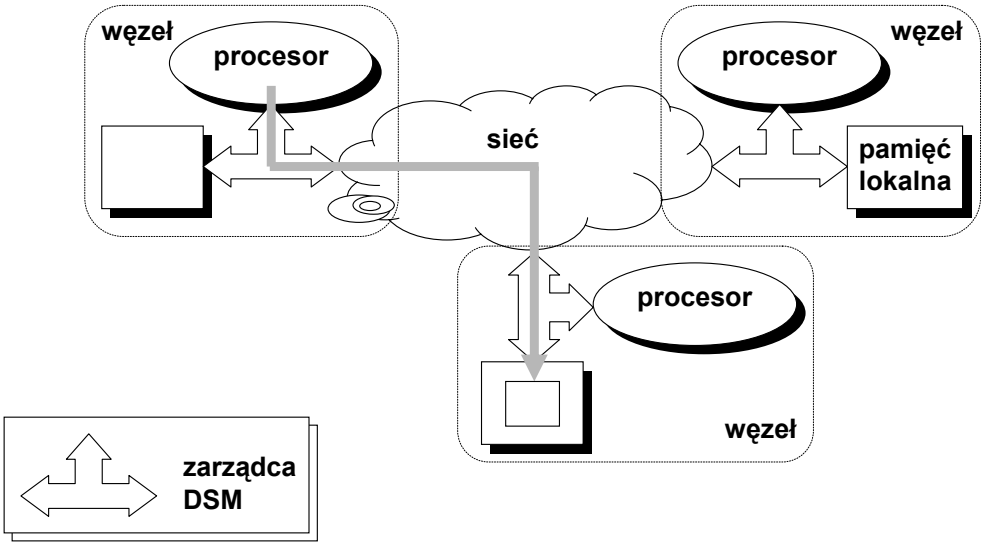
1. Problem lokalizacji
   - adres obiektu zmienienia się w czasie
2. Problem rozmiaru i struktury przemieszczanej jednostki
   - małe obiekty → duży poziom współdzielenia
   - duże obiekty → mały narzut administracyjny
3. Problem migotania (ang. *trashing*, *ping-pong effect*)
   - naprzemienne odwołania kilku procesów

Relokacja danych jest atrakcyjna z tego powodu, że pozwala na zwiększenie wydajności dostępu dodanych, nie powodując jednocześnie powstawania problemu spójności. Generuje jednak inne problemu, m.in. problem **lokalizacji** obiektów, do których kierowane są odwołania. Ponieważ jednostka zmienia swoją lokalizację (adres), przy każdym odwołaniu konieczne jest ponowne jego ustalenie, co może zredukować potencjalne zyski.

Innym problemem związanym ze stosowaniem relokacji jest kwestia właściwego doboru **rozmiaru i struktury obiektu**. Utrzymywanie małych obiektów umożliwia podział wspólnych danych na serwery, w których występują odwołania do nich. Z drugiej jednak strony z każdym obiektem związane są pewne struktury danych przechowujące informację zarządzającą (bieżący adres, tryb dostępu, właściciel itp.). Stosowanie dużych obiektów nie obciąża systemu dużymi kosztami administracyjnymi, ale wymaga sporych nakładów podczas przenoszenia obiektów z węzła na węzeł. W efekcie mamy tu do czynienia z kompromisem między zyskiem związanym z lokalnym dostępem a kosztami związanymi z zarządzaniem i relokacją obiektów. Pewnym rozwiązaniem jest stosowanie podejścia strukturalnego (hierarchicznego) do tworzenia obiektów. Obiekt jako całość może mieć wyodrębnione swoje – w miarę niezależne – części, które mogą być przemieszczane w oderwaniu od obiektu głównego.

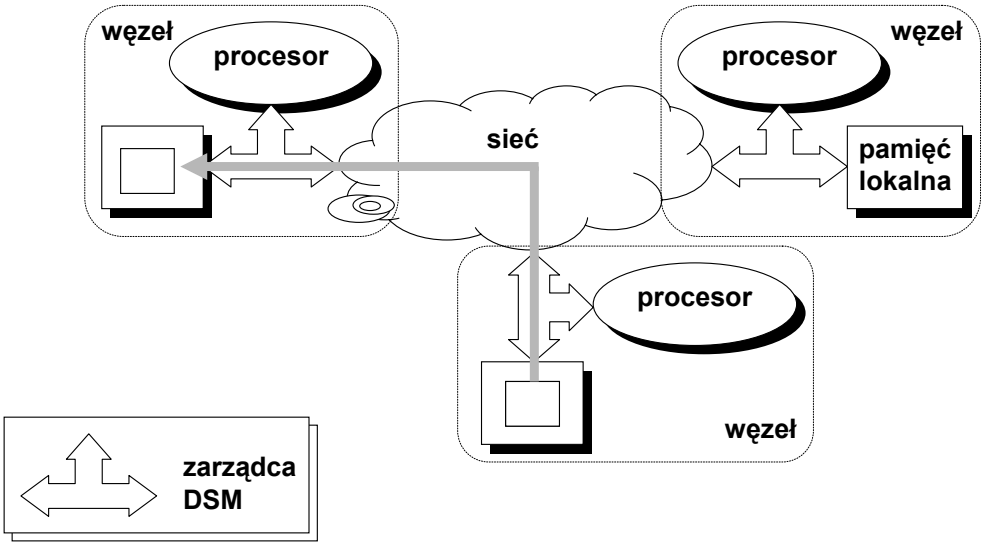
Ostatnim istotnym problemem relokacji jest problem **migotania**, polegający na nieustannym przenoszeniu obiektu między dwoma lub większą liczbą węzłów, z powodu występujących na tych węzłach odwołań. Sytuacja taka może mieć miejsce, gdy procesy naprzemiennie odwołują się do wspólnych danych. W tym przypadku bardziej opłacalne może się okazać pozostawienie obiektu w jednym z tych węzłów i realizowanie pozostałych odwołań zdalnie. Chcąc uniknąć takiego przypadku, system musi monitorować charakterystykę odwołań do obiektu i szacować prawdopodobieństwo lokalizacji przyszłych odwołań.

Zwielokrotnianie



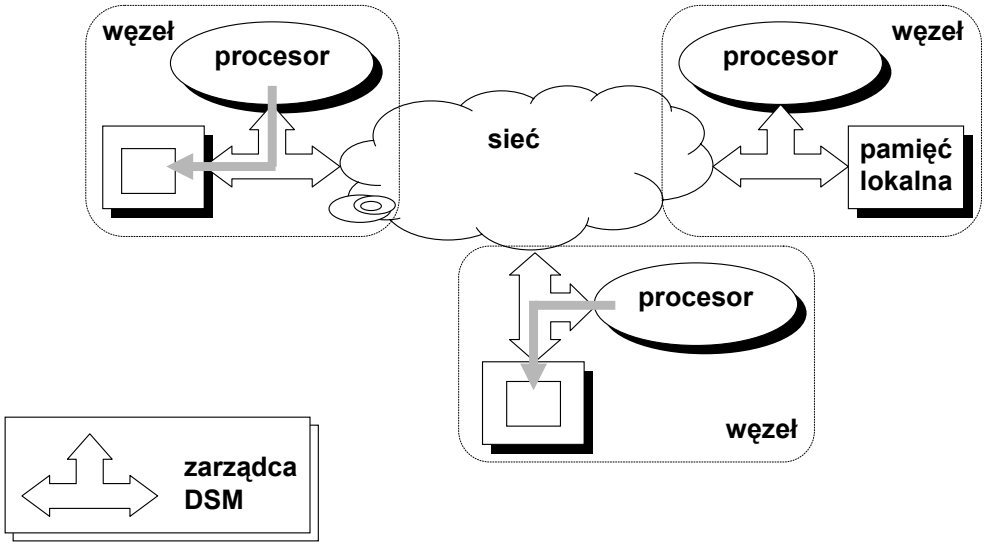
Rysunek przedstawia dostęp do danych w systemie, gdzie stosuje się zwielokrotnianie (replikację). Dane początkowo są dostępne jedynie w węźle dolnym. Żądanie, które zostało wygenerowane w węźle górnym musi zostać przesłane do węzła posiadającego kopię danych.

Zwielokrotnianie



Serwer posiadający odpowiednie dane przesyła ich kopię do serwera na którym pojawiło się żądanie.

Zwielokrotnianie



Po utworzeniu kopii na serwerze, na którym zażądano dostępu do danych, następuje udostępnienie nowej kopii procesowi. Posiadanie wielu kopii danych pozwala wykonywać do nich dostęp współbieżnie na wielu serwerach. Daje to szczególnie dobre efekty w sytuacji, gdy wykonywany jest dostęp typu odczyt. Z reguły dostępy nie modyfikujące stanu obiektów dominują, co jest przesłanką do tworzenia dodatkowych kopii danych. Wprowadzenie modyfikacji wymaga bowiem zsynchronizowania wszystkich kopii dla uniknięcia niespójności danych.

Zwielokrotnianie — charakterystyka

- Problem lokalizacji
  - tworzenie nowych replik
  - usuwanie starych replik
- Problem rozmiaru i struktury obiektów zwielokrotnianych
- Problem migotania nie występuje
  - kopia dla każdego ubiegającego się węzła
- Problem spójności kopii (replik)
  - stosunek liczby zapisów do odczytów

Zwielokrotnianie (replikacja) podobnie jak relokacja powoduje powstawanie problemu **lokalizacji** danych. W typowym systemie lokalizacja poszczególnych replik obiektów może ulegać zmianie; w każdej chwili mogą być utworzone nowe repliki na żądanie, a repliki zbędne mogą być usunięte. Utworzenie kopii danych i usunięcie oryginału to w istocie jest relokacja.

Podobieństwo do relokacji występuje również w kwestii tworzenia kopii obiektów i ich aktualizacji. W przypadku zwielokrotniania podobnie występuje problem doboru odpowiedniego rozmiaru obiektu, a także jego struktury.

Zaletą zwielokrotniania jest to, że w zasadzie eliminuje problem migotania. Jeżeli dane są potrzebne w wielu miejscach jednocześnie (do odczytu), to wystarczy utworzyć odpowiednią liczbę kopii i dalsze przetwarzanie może być kontynuowane bez potrzeby komunikacji.

Problemem specyficznym dla zwielokrotniania jest natomiast spójność danych. Istnienie wielu kopii jest korzystne z punktu widzenia procesów czytających, ale staje się sporym problemem dla systemu, gdy zachodzi konieczność aktualizacji tych wszystkich replik.

Struktura zwielokrotnianej jednostki

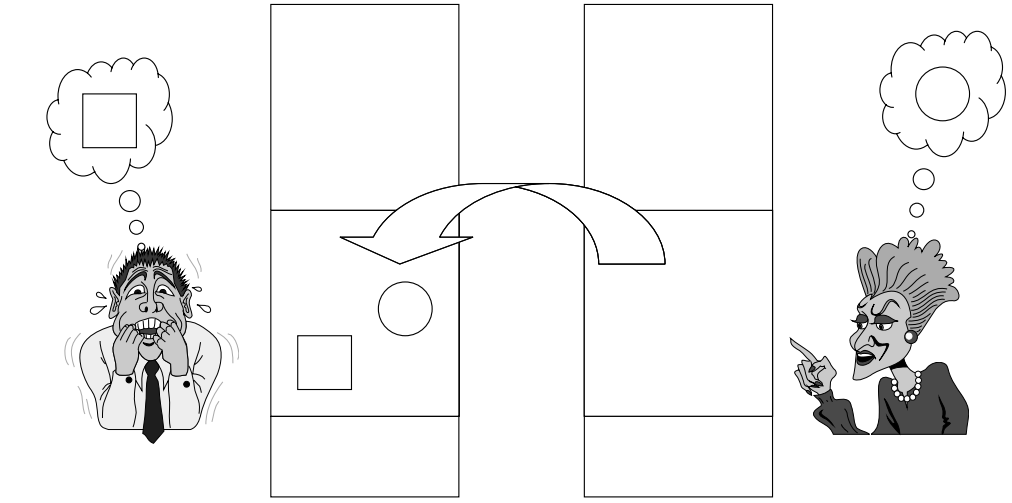
- **Strona** – fizyczne połączenie kilku odrębnych obiektów logicznych w jedną jednostkę udostępnianą jako całość przez **DSM** (*problem fałszywego współdzielenia*)
- **Pojedyncza zmienna** – duży jednostkowy koszt relokacji i utrzymywania spójności
- **Obiekt** (hermetyczna struktura danych udostępniana tylko przez zdefiniowane metody) – możliwość optymalizacji w strategii utrzymywania spójności w związku ze ściśle określonym sposobem dostępu (poprzez metody)

Organizacja systemu rozproszonej pamięci dzielonej może wyglądać bardzo różnie w zależności od poziomu na jakim jest ona realizowana. W rozwiązaniach sprzętowych lub niskopoziomowych wbudowanych w system operacyjny podstawową jednostką zarządzaną jest **strona** pamięci, równoważna stronie pamięci wirtualnej. Podstawową wadą tej metody jest całkowite abstrahowanie od tego jakie dane będą przechowywane w ramach poszczególnych stron i w jaki sposób będzie przebiegać alokacja pamięci w ramach stron. W efekcie na jednej stronie mogą się znaleźć dwa różne obiekty całkowicie ze sobą niezwiązane lub pojedynczy obiekt może rozciągać się na kilka stron, które będą zarządzane niezależnie. Pierwszy przypadek może prowadzić do efektu *fałszywego współdzielenia* opisanego na następnym slajdzie.

Rozwiązaniem, które jest skrajnie odmienne od przedstawionego powyżej jest zarządzanie danymi na poziomie elementarnych zmiennych. Zaletą tego podejścia jest ścisłe powiązanie z logiką aplikacji i językiem programowania. Zmienne z reguły są jednostkami niepodzielnymi, których modyfikacje mogą być w prosty sposób przekazane do innych węzłów poprzez kopiowanie. Podstawowa wada tej metody to bardzo duży narzut administracyjny wynikający z małego rozmiaru zmiennych. Dane służące do zarządzania dzielonymi zmiennymi mogą zajmować więcej miejsca niż same zmienne.

Pewnym kompromisem w tym względzie jest podejście obiektowe, w którym rozmiar obiektu jest różny, w zależności od złożoności jednostki. Obiekt może zawierać wewnątrz prostą zmienną (np. licznik), ale może równie dobrze reprezentować np. całą bazę danych. Efektywne zarządzanie zwielokrotnianiem obiektów wymaga jednak aby system w jakiś sposób „rozumiał” obiekty, a więc potrafił automatycznie uzyskać informacje dotyczące semantyki poszczególnych metod (np. tylko do odczytu, modyfikująca itd.).

Fałszywe współdzielenie



Fałszywe współdzielenie (ang. *false sharing*) występuje w systemach ze stronicowaną rozproszoną pamięcią współdzieloną. Jeżeli na jednej stronie znajdzie się kilka obiektów, do których odwołują się różni użytkownicy, to system stwierdzi, że ta strona jest między nimi dzielona. W zależności od protokołu spójności zachowanie systemu może być różne: strona może być nieustannie relokowana między serwerami lub jej aktualizacje będą przesyłane z miejsca na miejsce. Oba zachowania nie będą adekwatne do sytuacji, bo najprostszym rozwiązaniem jest zezwolenie na jednoczesny zapis do tych stron przez różnych użytkowników na różnych serwerach, ponieważ zapisy kierowane są do rozłącznych obszarów tej strony. Na końcu wystarczy jedynie przesłać zmodyfikowane fragmenty do pozostałych serwerów w celu uzyskania stanu spójnego. Takie podejście znacząco ograniczy intensywność komunikacji, ale z drugiej strony wymaga precyzyjnego monitorowania odwołań do stron, w celu wychwycenia zmian na poziomie poszczególnych bajtów.

Protokół koherencji

Protokół koherencji (spójności) jest algorytmem rozproszonym realizującym określony model spójności

1. Protokół **unieważniania** danych (ang. *invalidation protocol*)
   - małe komunikaty
   - jednokrotnie unieważnienie
2. Protokół **aktualizacji** danych (ang. *update protocol*) – niespójne repliki są aktualizowane
   - większe komunikaty

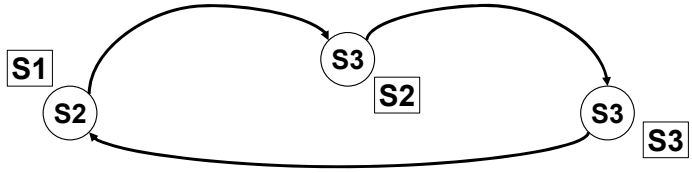
Protokół koherencji (ang. *consistency protocol*) jest algorytmem rozproszonym realizującym określony model spójności, a więc dostarczającym użytkownikom gwarancji odnośnie uporządkowania operacji modyfikujących, które są zgłaszane w systemie. Operacje odczytu nie powodują powstawania problemu spójności danych. Operacje zapisu modyfikują pojedynczą kopię i muszą być przesłane do pozostałych węzłów. Generalnie istnieją dwa podstawowe podejścia do zapewniania spójności: unieważnianie i aktualizacja.

Unieważnianie (ang. *invalidation*) polega na zmianie statusu zdalnych kopii danych tak, aby nie mogły już być wykorzystywane, co efektywnie może być traktowane jako usunięcie repliki. Komunikaty unieważniające są małe ponieważ muszą zawierać jedynie identyfikator strony, która ma ulec unieważnieniu. Zaletą unieważniania jest brak konieczności dalszej komunikacji z serwerem w przypadku wprowadzania następnych modyfikacji – kopia przestała bowiem już istnieć. Unika się w ten sposób również przesyłania aktualizacji, które być może nigdy nie zostałyby wykorzystane (odczytane) przez innych użytkowników, co powodowałoby jedynie zwiększenie obciążenia komunikacyjnego w systemie. Wadą natomiast jest konieczność wysłania następnego komunikatu z aktualną kopią danych, jeżeli dane te faktycznie są jednak potrzebne.

Drugim podejściem stosowanym w protokołach koherencji jest **aktualizacja** (ang. *update protocol*). Polega ona na każdorazowym przesyłaniu komunikatu aktualizującego stan repliki. Komunikat taki może zawierać całość zaktualizowanego stanu obiektu lub jedynie zmianę względem poprzedniej wersji. W podejściu tym przesyłane komunikaty są większe, bo muszą zawierać również aktualizowane dane. Komunikaty muszą również być wysyłane praktycznie po każdej aktualizacji, chyba, że z własności modelu wynika, że dane te nie będą wykorzystywane, co pozwala na zgrupowanie kilku aktualizacji w jednym komunikacie (grupowanie takie może też dotyczyć kilku aktualizacji *różnych* obiektów). Zaletą tego rozwiązania jest natychmiastowa dostępność zaktualizowanych replik w poszczególnych węzłach. Może się jednak zdarzyć, że aktualizacje te nie będą w ogóle odczytywane.

IVY: Problem lokalizacji stron

1. **Statyczna scentralizowana** lokalizacja
   - dedykowany węzeł odwzorowujący strony
2. **Statyczna rozproszona** lokalizacja
   - identyfikacja zarządcy na podstawie numeru strony
3. **Dynamiczna** lokalizacja
   - wskaźniki naprowadzające
   - identyfikator węzła *prawdopodobnie* posiadającego stronę



System IVY był pierwszą realizacją koncepcji rozproszonej pamięci współdzielonej, zrealizowaną całkowicie programowo, bez wsparcia ze strony sprzętu. Realizacja ta pokazała, że idea DSM dla wielu aplikacji, szczególnie obliczeniowych, może umożliwiać proste i efektywne zrównoleglenie przetwarzania na wielu komputerach bez konieczności uciekania się do kłopotliwej wymiany komunikatów.

Jednym z istotnych problemów związanych z zarządzaniem rozproszoną pamięcią jest kwestia lokalizacji stron. W systemie IVY zaimplementowano i przetestowano trzy rozwiązania. Pierwsze, najprostsze, polega na zastosowaniu scentralizowanego zarządzania, w którym dedykowany węzeł przechowuje całość informacji o aktualnym położeniu poszczególnych stron pamięci. Rozwiązanie to sprawdza się dobrze, gdy w systemie działa taki dedykowany system i gdy liczba serwerów nie jest zbyt duża. Przy większej liczbie węzłów centralny serwer w sposób naturalny staje się wąskim gardłem.

Drugie rozwiązanie problemu lokalizacji polega na rozłożeniu zadania odwzorowywania stron na wiele serwerów. Rozkład ten jest statyczny: na podstawie numeru strony oblicza się identyfikator węzła, który jest zarządcą tej strony (np. modulo liczba węzłów). Dzięki temu rozwiązaniu każdy serwer wie do kogo wysyłać komunikat z zapytaniem o lokalizację strony. Statystycznie rzecz biorąc każdy węzeł powinien być podobnie obciążony zadaniem odwzorowywania stron. Lokalizacja wymaga w tym przypadku tyle samo komunikatów co w przypadku podejścia scentralizowanego. Wadą tego rozwiązania jest pewna jego statyczność: dodanie nowego serwera jest trudne do zrealizowania, bo wymagałoby zmiany przypisań stron do serwerów.

Trzecie rozwiązanie to zastosowanie dynamicznej lokalizacji. Jest to podejście oparte na wskaźnikach naprowadzających, opisywanych na wykładzie dotyczącym nazewnictwa. Każdy węzeł posiada tablicę odwzorowującą lokalizację poszczególnych stron. Tablica ta nie koniecznie zawiera aktualne i prawdziwe informacje. Jest to informacja o *prawdopodobnej* lokalizacji stron. Lokalizacja strony polega na wysłaniu zapytania do serwera, który jest podejrzewany o jej przechowywanie. Jeżeli jest to prawdą, odesłany zostanie komunikat z potwierdzeniem. Jeżeli nie, zapytanie zostanie przesłane do węzła, który jest podejrzewany przez węzeł właśnie odpytywany, itd. Na rysunku węzły oznaczone są etykietami w kwadratach. Węzeł S1 wysyła zapytanie do S2, bo podejrzewa właśnie S2 o posiadanie poszukiwanej strony. S2 jednak jej nie posiada i przekazuje zapytanie do S3, bo podejrzewa właśnie S3. S3 faktycznie stronę posiada i odpowiada węzłowi S1.

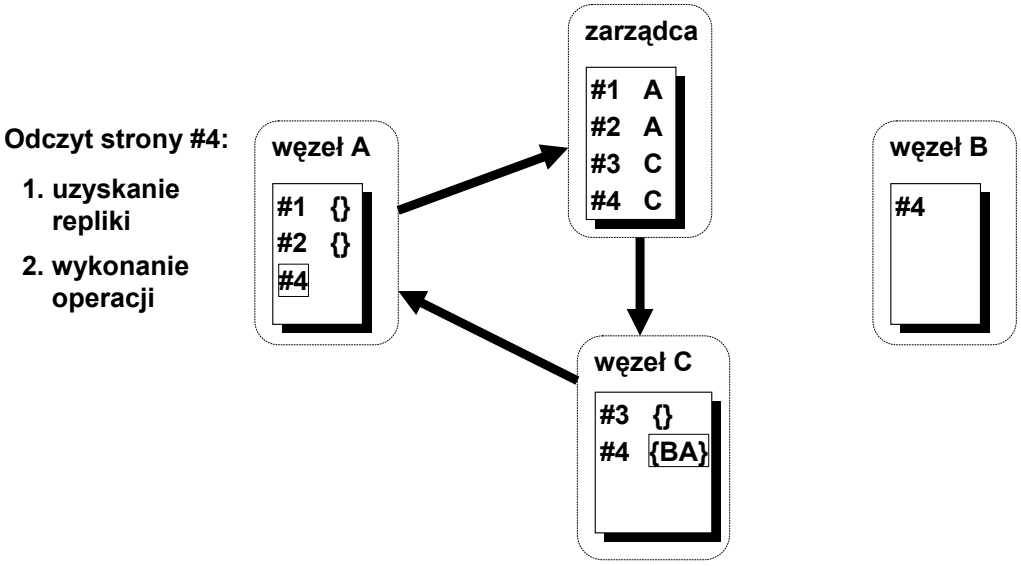
Wskaźniki naprowadzające sprawdzają się dobrze w przypadku niedużej liczby węzłów. Ponieważ ścieżki wskazujące ulegają skróceniu przy każdym odpytywaniu (S1 aktualizuje swoje wskazanie z S2 na S3), średnia długość ścieżki, którą w praktyce przebiegają zapytania pozostaje krótka. Wynika to również z częstego odwoływania się do stron.

IVY: pojęcia i struktury danych

- **Właściciel strony** – ostatni zapis strony
- **Zbiór kopii** (ang. *copyset*) – identyfikatory węzłów posiadających kopię strony
- **Zarządca** – informacje o właścicielach stron
- **Tablica właścicieli stron** – identyfikatory właścicieli dla każdej strony
- **Prawdopodobny właściciel** – identyfikator *prawdopodobnych* właścicieli dla każdej strony

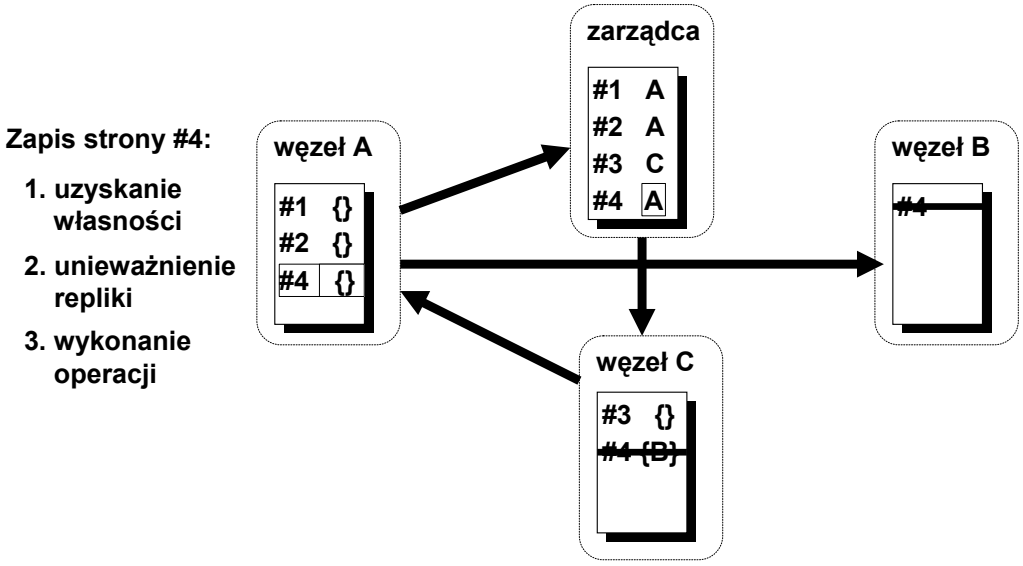
Protokół spójności zastosowany w systemie IVY utrzymuje pewne struktury danych dla potrzeb zarządzania spójnością. W systemie stosuje się relokację stron i zwielokrotnianie. Zwielokrotnianie jest stosowane w przypadku wielu odczytów zgłaszanych na różnych węzłach. Zapis powoduje unieważnienie wszystkich kopii danych i pozostawienie jednej. Węzeł, który jest uprawniony do wykonywania zapisu jest określany jako **właściciel strony**. Tworzenie kopii przeznaczonych do odczytu jest odnotowywane w **zbiorze kopii** (ang. *copyset*) przechowywanym przez właściciela strony. **Zarządca** stron to węzeł, który posiada informacje o aktualnej lokalizacji tych stron. Informacja o lokalizacji przechowywana jest w **tablicy właścicieli stron**, gdzie każda strona ma swoją pozycję zawierającą identyfikator węzła będącego aktualnie właścicielem strony. W przypadku dynamicznego rozproszonego lokalizowania stron węzły przechowują tablicę prawdopodobnych właścicieli stron.

Scentralizowana lokalizacja – odczyt



Powyższy rysunek przedstawia realizację odczytu w systemie IVY w przypadku stosowania statycznego, scentralizowanego mechanizmu lokalizacji stron. W przykładowym systemie funkcjonują 4 węzły: 3 są przeznaczone do obsługi pamięci rozproszonej, a czwarty pełni rolę dedykowanego zarządcy stron. Żądanie odczytu pojawia się w węźle A i dotyczy strony #4. Ponieważ strony tej nie ma w tym węźle, należy ją zlokalizować. Wysyłane jest więc w pierwszym kroku zapytanie do zarządcy. Tam uzyskujemy informację o bieżącym właścicielu strony #4, którym jest węzeł C. Żądanie jest przezywane do węzła C. Właściciel przechowuje zbiór identyfikatorów węzłów, które przechowują kopię posiadanych stron. W tym przypadku stronę #4 posiada jeszcze węzeł B. Przekazane żądanie dostępu do strony #4 powoduje utworzenie nowej kopii tej strony na węźle A, co zostaje odnotowane w zbiorze kopii u właściciela (w węźle C). Na końcu następuje wykonanie lokalnej operacji odczytu w węźle A.

Scentralizowana lokalizacja – zapis



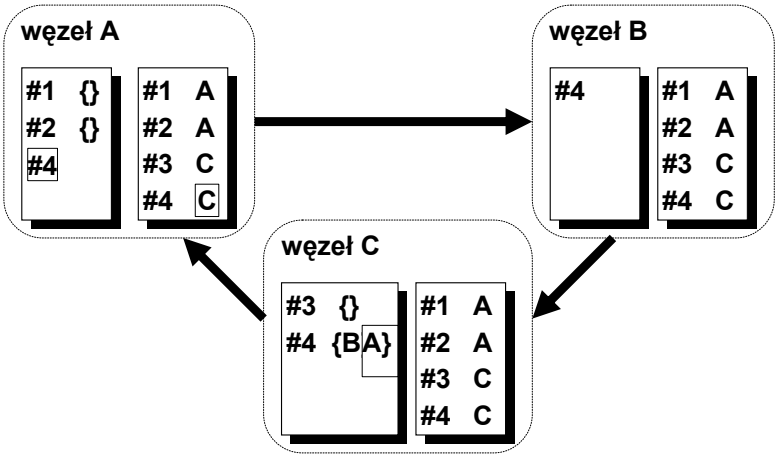
Realizacja zapisu w systemie IVY może być nieco bardziej skomplikowana, jeżeli węzeł, na którym będzie wykonywany zapis nie jest właścicielem tej strony. W takiej sytuacji przed wykonaniem zapisu musi się najpierw stać właścicielem. Rysunek przedstawia identyczną sytuację początkową jak w poprzednim przykładzie. W pierwszym kroku zlecenie wędruje do zarządcy stron w celu zlokalizowania strony przeznaczonej do zapisu. Początkowo jest nim węzeł C i do niego jest przekazywane żądanie. Jednocześnie jednak zarządca odnotowuje już fakt zmiany właściciela strony #4 na węzeł A. Zmiana właściciela oznacza, że informacje przechowywane na węźle C są przekazywane do nowego węzła A (m.in. zbiór kopii). Węzeł A po otrzymaniu strony #4 dokonuje unieważnienia wszystkich kopii tej strony, ponieważ będzie ona modyfikowana. W ten sposób nie dopuszcza się do powstawania problemu niespójności poszczególnych replik. Komunikat unieważniający wysyłany jest w tym przypadku do węzła B. Po otrzymaniu potwierdzeń o poprawnym unieważnieniu replik, węzeł A może zrealizować lokalnie operację zapisu.

Realizacja operacji dostępu do danych w przypadku stosowania rozproszonej statycznej lokalizacji wygląda bardzo podobnie. Jedyna różnica polega na wyborze węzła, który pełni rolę zarządcy. Komunikat lokalizujący stronę wysyłany jest do węzła, któremu przypisano funkcję zarządzania odpowiednią stroną, a nie do centralnego zarządcy.

Dynamiczna lokalizacja – odczyt

Odczyt strony #4:

1. **uzyskanie repliki**
2. **wykonanie operacji**

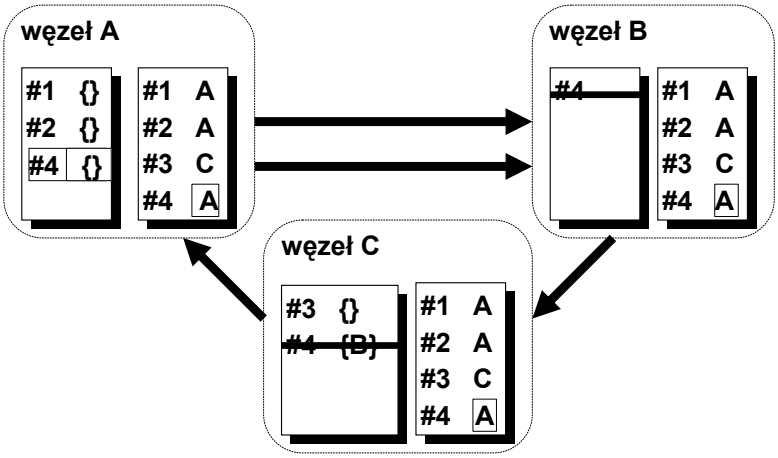


Rysunek przedstawia realizację odczytu w systemie IVY w przypadku stosowania dynamicznego, rozproszonego mechanizmu lokalizacji stron. W tym podejściu każdy z węzłów posiada tablicę prawdopodobnych właścicieli stron. Sytuacja początkowa jest identyczna z poprzednimi przykładami. Zlecenie odczytu ze strony #4 pojawia się w węźle A, który nie posiada tej strony. Z lokalnej tablicy właścicieli stron wynika, że prawdopodobnym właścicielem strony #4 jest węzeł B. W pierwszym kroku zlecenie pobrania strony kierowane jest więc do węzła B. Ten, pomimo że posiada kopię strony #4 nie odsyła jej a jedynie uczestniczy w przekazywaniu zlecenia do właściciela strony, czyli węzła C. Węzeł C odnotowuje w swoim zbiorze kopii, że węzeł A będzie posiadał replikę strony #4 i wysyła tą stronę do węzła A. Węzeł A aktualizuje swoją tablicę właścicieli stron na pozycji 4, wpisując tam węzeł C. W końcu następuje udostępnienie strony #4 do odczytu.

Dynamiczna lokalizacja – zapis

Zapis strony #4:

1. **uzyskanie własności**
2. **unieważnienie repliki**
3. **wykonanie operacji**



Zapis w przypadku systemu stosującego dynamiczną lokalizację wymaga podobnie jak w wersji scentralizowanej znalezienia właściciela strony i następnie unieważnienia wszystkich dotychczasowych replik. W pierwszym kroku zlecenie jest wysyłane do węzła B, który jest uznawany przez węzeł A za prawdopodobnego właściciela strony #4. Węzeł B nie jest jednak właścicielem i przekazuje żądanie dalej, do węzła A. Wcześniej jednak aktualizuje swoją tablicę właścicieli stron wpisując na pozycji 4 identyfikator węzła A, ponieważ wiadome już jest, że to on stanie się za chwilę właścicielem. Węzeł C, który jest właścicielem strony #4 przekazuje ją wraz ze wszystkimi danymi do węzła A. Jednocześnie również aktualizuje wpis w tablicy właścicieli stron na pozycji 4. Ostatecznie strona dociera do węzła A, który dokonuje w drugim kroku unieważnienia wszystkich replik strony, wysyłając komunikaty unieważniające. Po odebraniu potwierdzeń strona #4 jest udostępniana w węźle A do zapisu.