Rozproszone systemy plików

Bartosz Grabiec
Jerzy Brzeziński
Cezary Sobaniec

Wykład ten ma na celu prezentację zagadnień związanych z tematyką rozproszonych systemów plików.

Rozproszone systemy plików ze względu na swoją budowę i działanie obejmują bardzo szerokie spektrum zagadnień związanych z ogólnie rozumianymi systemami rozproszonymi. Problemy i kwestie, które zostały poruszone na poprzednich wykładach w dużej mierze dotyczą również systemów rozproszonych, jak choćby nazewnictwo, problem spójności, mechanizmy synchronizacji.

Na początku wykładu zajmiemy się omówieniem ogólnej charakterystyki rozproszonych systemów plików. Następnie przedstawimy m.in.. modele plików, modele dostępów do pliku, mechanizmy pamięci podręcznej. Opiszemy także kwestię zwielokrotniania plików, serwery pełnostanowe i bezstanowe oraz problem niezawodności. Na koniec przedstawimy wybrane rozwiązania stosowane w rzeczywistych systemach plików (NFS, Coda).

Tytuł przedmiotu

Wprowadzenie

- Rozproszone systemy plików wprowadzają abstrakcję zarządzania danymi w środowiskach rozproszonych
- Główne zadania to:
  - Utrwalanie informacji
  - Ułatwienie współdzielenia danych
- Rozproszone systemy plików sprawia, że zdalny dostęp do danych staje się podobny do lokalnego dostępu

Tytuł modułu (2)

Tak jak dzieje się to w scentralizowanych systemach operacyjnych, rozproszone systemy plików wprowadzają pewną abstrakcję zarządzania danymi. Użycie plików ma na celu m.in. utrwalenie informacji poprzez przechowywanie ich na odpowiednich nośnikach danych oraz ułatwienie współdzielenia informacji.

Stosowanie systemu plików uwalnia użytkowników, programistów od wielu czynności związanych z zarządzaniem danymi. W przypadku scentralizowanych systemów plików można powiedzieć w skrócie, że użytkownicy zostają zwolnieni z bezpośredniego manipulowania urządzeniami przechowującymi dane. Natomiast rozproszone systemy plików pozwalają użytkownikom na wygodne wykonywanie operacji plikowych nie tylko na lokalnym komputerze, ale także w środowisku rozproszonym, gdzie dane przechowywane są na wielu rozproszonych urządzeniach.

Aby pokrótce zaprezentować możliwości rozproszonych systemów plików, wymienimy kilka ich wybranych cech. Umożliwiają np. zdalne współdzielenie informacji. Oznacza to, że plik, który został utworzony na jednym komputerze może być dostępny z innego komputera w sieci. Rozproszony system plików powinien być również być możliwie przeźroczysty dla użytkownika. Np. przeźroczystość lokalizacji sprawia, że użytkownik nie musi znać fizycznego miejsca, w którym znajduje się plik, tym powinien zająć się system plików. Kolejną cechą rozproszonych systemów plików, która staje się w dzisiejszych czasach coraz bardziej pożądana, jest umożliwienie użytkownikom przemieszczania się między różnymi miejscami w sieci oraz tymczasowe odłączanie się od niej. Innymi słowy użytkownik nie powinien być zmuszony do pracy na jednym ściśle określonym komputerze, ale powinien mieć możliwość przełączenia się pomiędzy różnymi komputami, które być może są w innym geograficznie odległym miejscu. Z mobilnością użytkownika wiąże się również pojęcie **dostępności** (ang. *availability*). W środowisku rozproszonym jeden z serwerów, który przechowuje dane użytkownika może ulec awarii. W tym celu rozproszony system plików przechowuje wiele kopii danych na różnych węzłach systemu, czyli tzw. repliki plików. W momencie awarii system może połączyć użytkownika z innym serwerem, który zawiera kopie jego danych. W idealnym przypadku klient rozproszonego systemu plików nie musi być świadomy awarii i może dalej działać na swoich danych.

Rozproszone systemy plików pozwalają także na przechowywanie i organizacje danych, które nie byłyby możliwe do przechowania na pojedynczych stanowiskach.

Tytuł przedmiotu

Usługi związane z rozproszonymi systemami plików

- *Bezpośrednie zarządzanie danymi na nośnikach danych*
  - Tworzy logiczną warstwę zarządzania danymi na urządzeniach fizycznych
- *Właściwa usługa plikowa*
  - Zarządzanie plikami i ich zawartością
- *Usługa nazewnicza*
  - Odwzorowanie nazw na identyfikatory systemowe

Tytuł modułu (3)

Z rozproszonymi systemami plików wiążą się różne rodzaje usług. Są one niezbędne do osiągnięcia pożądanej funkcjonalności.

Wśród nich znajduje się m.in. usługa odpowiedzialna za bezpośrednie zarządzanie danymi na urządzeniach przechowujących dane (ang. *storage service*). W skład tej usługi wchodzą np. zadania związane z alokacją i zarządzaniem przestrzenią dyskową, w której są przechowywane pliki. Usługa ta tworzy pewną logiczną warstwę zarządzania danymi. Architektura tej usługi jest bardzo podobna dla scentralizowanych i rozproszonych systemów plików.

Kolejną usługą we wspomnianej grupie usług jest **właściwa usługa plikowa** (ang. *true file service*). Dostarcza ona odpowiednich operacji do zarządzania poszczególnymi plikami danych i ich zawartością. Pojawiają się tu operacje tworzenia i kasowania plików, operacje czytania i pisania do plików itp. Z usługą tą powiązanych jest wiele dodatkowych kwestii takich jak: mechanizm dostępu do plików, semantyka współdzielenia plików, mechanizmy pamięci podręcznej, zwielokrotnianie plików, sterowanie współbieżnością, mechanizm zachowania spójności danych, protokoły aktualizacji danych, mechanizm kontroli dostępu i wiele innych. Wyodrębnienie tej usługi od poprzedniej daje możliwość odseparowania warstwy zarządzania danymi od warstwy nośników danych.

Trzecią i ostatnią prezentowaną tu usługą jest **usługa nazewnicza** (ang. *name service*). Zajmuje się ona odwzorowaniem nazw plików na specjalne identyfikatory. Identyfikatory są rodzajem uchwytów do poszczególnych plików i ułatwiają zarządzanie plikami przez rozproszony system plików. Z usługa nazewniczą związana jest **usługa katalogowa** (ang, *direktory service*), która wprowadza hierarchię do struktury plików i pozwala nią zarządzać.

Tytuł przedmiotu

Cechy rozproszonych systemów plików (1)

- Przeźroczystość
  - Struktury
  - Dostępu
  - Nazewnictwa
  - Zwielokrotniania
- Mobilność
- Wydajność (komunikacja, przetwarzanie żądań itp.)
- Prosta obsługa

Tytuł modułu (4)

Dobry rozproszony system plików powinien posiadać zestaw pewnych właściwości.

Jedną z takich właściwości jest przeźroczystość (ang. *transparency*). W systemach rozproszonych można wyróżnić wiele typów przeźroczystości.

Przeźroczystość struktury pozwala na ukrycie przed klientami faktu korzystania z wielu serwerów. Klient nie musi znać liczby serwerów, na której przechowuje swoje dane. Najlepiej gdyby widział rozproszony system plików jako jeden scentralizowany system.

Przeźroczystość dostępu sprawia, że zdalny i lokalny dostęp do plików wygląda identycznie. W idealnym wypadku zdalne i lokalne operacje na plikach powinny być takie same.

Przeźroczystość nazewnictwa wprowadza niezależność nazw plików od ich lokalizacji. Na podstawie nazwy użytkownik nie jest w stanie określić miejsca fizycznego przechowania pliku. Zmiana miejsca przechowywania pliku nie wpływa tym samym na nazwę pliku.

Przeźroczystość zwielokrotniania.ukrywa fakt istnienia wielu kopii tego samego piku na różnych urządzeniach.

Kolejną, wspomnianą już wcześniej właściwością, która staję się coraz bardziej pożądana (m.in. jako skutek rozwoju sieci bezprzewodowych), jest mobilność (ang. *mobility*) użytkowników. Mobilność w idealnym przypadku daje użytkownikowi dostęp do jego danych bez względu na to gdzie się znajduje. W praktyce osiągnięcie pełnej mobilności jest często niemożliwe. Pojawia się tu m.in. problem operacji wykonywanych przez klienta w odłączeniu od serwera plików.

Istotnym czynnikiem, który określa m.in. przydatność rozproszonego systemu plików jest czas wykonania żądań klienta. Na wydajność działania systemu plików składa się wiele czynników. Są to np. czas dostępu do informacji na nośniku fizycznym, czas przetwarzania żądania przez procesor oraz czas komunikacji. Dobry rozproszony system plików powinien mieć wydajność podobną do scentralizowanego systemu.

Z punktu widzenia użytkownika system plików musi być prosty w obsłudze. Semantyka systemu powinna być możliwie prosta. Ma to swoje odzwierciedlenie w małej liczbie łatwych do zrozumienia operacji. Interfejs rozproszonego systemu plików znowu powinien być maksymalnie podobny do tego w scentralizowanych systemach.

Tytuł przedmiotu

Cechy rozproszonych systemów plików (2)

- Skalowalność
- Wysoka dostępność
- Niezawodność
- Spójność danych
- Bezpieczeństwo
- Heterogeniczność

Tytuł modułu (5)

Wraz z rozwojem sieci komputerowych i ich rozrastaniem się do coraz większych rozmiarów pojawił się problem **skalowalności** (ang. *scalability*) systemów, które działają w środowiskach rozproszonych. Istotne jest, aby rozproszone systemy plików, które działają w rozległych sieciach były skalowalne. Skalowalność oznacza możliwość łatwego podłączania nowych serwerów i klientów do systemów. Najlepiej gdyby fizyczne rozrastanie się systemu nie miało negatywnego wpływu na wydajność dotychczas działających serwerów i klientów.

Wysoka dostępność (ang. *high availibilty*) charakteryzuje miedzy innymi odporność systemu na awarię. System, który uległ częściowej awarii powinien być nadal dostępny i świadczyć usługi. Oczywiście nie jest wykluczone, że będą one w okrojonym zakresie. Jednym ze środków utrzymywania wysokiej dostępności jest użycie zwielokrotniania.

Dobry rozproszony system plików musi być **niezawodny**. Prawdopodobieństwo utraty danych powinno być zminimalizowane do granic praktycznych możliwości. Tworzenie kopii zapasowych, przywracanie systemu do stanu sprzed awarii to zadania, które powinien wykonywać automatycznie system plików.

Ponieważ w rozproszonym systemie plików, operacje na plikach mogą być wykonywane współbieżnie przez wielu użytkowników, często istnieje potrzeba zagwarantowania odpowiedniego poziomu **spójności danych**. Stosowane są to np. mechanizmy transakcyjne, protokoły spójności nastawione na dane oraz na klienta.

Bardzo istotnym elementem systemu plików jest bezpieczeństwo. Wraz z rosnącymi rozmiarami systemu, rośnie problem **bezpieczeństwa** systemu. Bezpieczny system musi zapewnić użytkownikom zachowanie ich prywatności. Informacje przechowywane w plikach powinny być odpowiednio chronione przed niepożądanym dostępem z zewnątrz. Również prawa dostępu do pliku muszą być należycie zabezpieczone.

Heterogeniczność jest na ogół cechą bardzo pożądaną w środowiskach rozproszonych. Im system większy i rozleglejszy tym przeważnie jest on bardziej zróżnicowany. Heterogeniczność jest cechą, która wiąże się poniekąd z poprzednimi cechami tj. mobilnością, skalowalnością itp. Użytkownik, który może się przemieszczać powinien mieć swobodę wyboru platformy systemowej, a nawet sprzętowej, na której chce pracować. Pożądana jest sytuacja, gdy użytkownik ma dostęp do swoich danych na różnych komputerach niezależnie od np. systemu operacyjnego. Heterogeniczność oznacza również zdolność do podłączania do systemu wielu *różnych* nośników danych. Z pewnością zwiększa to użyteczność takiego rozproszonego systemu.

Tytuł przedmiotu

Modele plików (1)

- Pliki o nieznanej strukturze zawartości – model charakterystyczny dla większości systemów operacyjnych
- Pliki ze znaną strukturą zawartości – dane ułożone w ciąg rekordów. Model ten jest charakterystycznych dla baz danych. W ramach tego modelu można dodatkowo wyróżnić:
  - pliki, których rekordy posiadają indeksy oraz
  - pliki bez indeksów

Tytuł modułu (6)

Ponieważ pliki różnią się zawartością i sposobem użycia, rozróżniamy wiele modeli plików. Ze względu na zawartość plików rozróżniamy pliki, które posiadają pewną strukturę wewnętrzną oraz pliki, których struktura jest nieznana dla systemu plików. W przypadku sposobu użycia plików ogólnym kryterium według, którego możemy rozróżnić pliki jest to, czy mogą być one modyfikowane czy też nie.

Najprostszy model plików zakłada, że struktura danych jest nieznana tzn. serwer plików nie ma możliwości interpretacji danych. Z punktu widzenia systemu operacyjnego nie jest interesujące co zawiera dany plik. Tym zajmują się aplikacje, które będą go używać. Rzadziej używanym modelem pliku jest plik o znanej strukturze. Dla tego rodzaju plików zakłada się, że ciąg danych jest uszeregowany. Taki poszeregowany ciąg danych tworzą rekordy, które w przypadku różnych plików mogą być różnych rozmiarów. W wypadku strukturalizowanych plików rekord jest najmniejszą jednostką danych, do której można uzyskać dostęp. Operacje odczytu i zapisu wykonywane są wtedy na zbiorach rekordów.

W dalszej kolejności strukturalizowane pliki dzieli się na dwa typy. Pliki, których rekordy posiadają indeksy i pliki, których rekordy takich indeksów nie posiadają. W tym momencie wkraczamy jednak w zagadnienia związane z bazami danych i nie będziemy ich tutaj dalej omawiać. Należy jednak pamiętać, że oba zagadnienia są ze sobą ściśle związane.

Wróćmy z powrotem do modeli plików, których struktura jest nieznana. Taki model używany jest przez większość dzisiejszych systemów operacyjnych. Jest to spowodowane między innymi tym, że zarządzanie takimi plikami i ich zawartością jest zdecydowanie prostsze nie tylko dla systemu plików, ale także dla różnych aplikacji.

Tytuł przedmiotu

Modele plików (2)

1) *Pliki modyfikowalne* – model plików, który pozwala na nadpisywanie zawartości plików i dodawanie do pliku nowych danych
2) *Pliki niemodyfikowalne* – pliki nie mogą być modyfikowane. W zamian tworzone są kolejne wersje plików dla każdej operacji modyfikującej zawartość pliku

- W przypadku plików niemodyfikowalnych znika częściowo problem z utrzymaniem spójności wielu kopii. Wadą jest tu duże zużycie przestrzeni dyskowych

Tytuł modułu (7)

Kolejnym kryterium rozróżniania modeli plików jest możliwość ich modyfikacji. W ogólności mamy tu do czynienia z dwoma rodzajami plików. Mianowicie pliki modyfikowalne oraz pliki, które mają tylko możliwość odczytu. Najbardziej naturalnym rozwiązaniem jest model pliku, który zezwala na modyfikację plików poprzez nadpisywanie starych danych lub dodawanie nowych. Taki plik może być praktycznie zmieniany w dowolny sposób. Istnieją również systemy, które używają plików niemodyfikowalnych. W przypadku takich systemów pliki mogą być np. utworzone, ale potem można je już jedynie skasować. Dla takich plików stosuje się m.in wersjonowanie, które jest pewnym sposobem na przechowywanie informacji o ich aktualizacjach. Charakteryzuje się to np. tym, że plik reprezentowany jest przez pewien ciąg jego wersji. Zamiast modyfikowania tworzone są nowe wersje tego pliku. W celu optymalizacji zapamiętywane są często tylko różnice między kolejnymi wersjami plików.

Istotną zaletą używania modeli plików niemodyfikowalnych jest łatwość utrzymania spójności w sytuacji, gdy dane są współdzielone. Łatwiej wtedy utrzymać integralność replik plików w rozproszonym środowisku, ponieważ system zostaje zwolniony z problemu związanego z utrzymaniem spójności wielu różnych kopii. Wersjonowanie wprowadza jednakże kilka problemów, takich jak zwiększone zużycie przestrzeni dyskowych.

Atrybuty plikowe

- Atrybuty plikowe, które mogą modyfikować tylko usługi związane z systemem plików. Np.:
  - data utworzenia pliku
  - właściciel pliku – ze względów bezpieczeństwa
- Atrybuty plikowe, które mogą być modyfikowane przez użytkownika:
  - niektóre prawa dostępu do plików

Jednostki danych jakimi są pliki posiadają pewne atrybuty. Pozwalają one na opisanie pewnych własności plików. Każdy atrybut posiada swoją nazwę oraz wartość. Korzystając z atrybutów możemy uzyskać informacje między innymi o właścicielu, rozmiarze, prawach, dacie utworzenia, dacie modyfikacji itp. Niektóre z atrybutów mogą być modyfikowane przez użytkowników, ale niekoniecznie wszystkie. Jednym z atrybutów, których użytkownik nie modyfikuje bezpośrednio może być np. rozmiar pliku. Zmianą takich atrybutów zajmuję się już system plików.

Tytuł przedmiotu

Model dostępu do plików

1) Zdalny dostęp do pliku – operacje zlecane są serwerowi, który odsyła ewentualne wyniki
   - Problem z narzutem komunikacyjnym
2) Użycie pamięci podręcznej – dane są przesyłane do klienta i dopiero na lokalnych danych wykonywane są operacje
   - Problem z zachowaniem spójności danych

Tytuł modułu (9)

Sposób w jaki klienci dostają się do plików zależy od modelu dostępu do plików używanego przez system plików. W tym przypadku modele możemy rozróżniać ze względu na dwa główne czynniki tzn. sposób dostępu do zdalnych plików oraz jednostki danych dostępu do plików.

Na początku zajmiemy się zdalnym dostępem do plików. Zasadniczo rozróżniamy dwie metody zdalnego dostępu do plików. Pierwsza metoda, to model bazujący na zdalnej usłudze. W tym przypadku wszystkie operacje, których żąda klient wykonywane są na serwerze plików tzn. klient wysyła pewne żądanie wykonania operacji na plikach do serwera, serwer wykonuje żądanie klienta, a następnie ewentualnie wysyła wyniki z powrotem do klienta. Można powiedzieć, że model ten bazuje na komunikatach przesyłanych przez sieć w celu dostępu do plików i odesłania wyników przez serwer. W niektórych przypadkach nakład spowodowany przez komunikację może być znaczący.

W przypadku projektowania protokołów bazujących na dostępie do plików w postaci zdalnych usług należy zwrócić uwagę na ilość przesłanych komunikatów oraz nakład przy generowaniu wiadomości.

Kolejnym podejściem do zdalnego dostępu do plików jest użycie pamięci podręcznej. W przypadku poprzedniego modelu zdalnego dostępu do plików z każdym żądaniem związany był ruch w sieci. Podejście stosujące pamięć podręczną stara się zminimalizować ten ruch poprzez wykonywanie niektórych dostępów do plików lokalnie. Zasada działania takiego modelu jest następująca. Gdy klient żąda jakichś danych z pliku, a nie ma ich lokalnie stara się je uzyskać od serwera. Dopiero na lokalnych danych klient wykonuje operacje dostępu. Z użyciem pamięci podręcznej wiąże się m.in. polityka usuwania zbędnych kopii. Używana jest tu np. metoda *LRU* (ang. *least recently used*), która może określać kiedy i które kopie danych mają być usunięte.

Zastosowanie pamięci podręcznej redukuje ruch w sieci. Problemem jest natomiast zachowanie spójności kopii podręcznej z oryginałem.

Tytuł przedmiotu

Jednostka transferu danych

- *Przesyłanie całych plików*

- *Przesyłanie fragmentów/bloków plików*

- *Przesyłanie bajtów*

- *Przesyłanie rekordów*

Tytuł modułu (10)

W systemach plików, które korzystają z pamięci podręcznej ważną kwestią jest rozmiar podstawowej jednostki danych, która jest przesyłana np. podczas operacji zapisu lub odczytu. W zależności od jej rozmiaru można wyróżnić kilka najczęściej używanych modeli.

Pierwszy model bazuje na przesyłaniu całych plików (ang. *file-level transfer model*). Gdy tylko potrzebne są dane z pewnego pliku, przesyłany jest cały plik. Plik może być przesyłany od serwera do klienta, jak również odwrotnie. Takie podejście do przesyłania ma kilka zasadniczych zalet. Liczba komunikatów przy przesyłaniu całego pliku jest mniejsza niż przy przesyłaniu tego samego pliku w kawałkach w odpowiedzi na kilka żądań. Przesłanie całości pliku naraz do klienta ogranicza liczbę dostępów do serwera, co zwiększa potencjalnie skalowalność systemu plików i odporność na awarię serwera plików lub sieci. Minusem tego podejścia jest przesyłanie zbędnych danych. Czasami klient potrzebuje tylko części danych z pliku. W gorszym przypadku klient może po prostu nie mieć miejsca na przechowanie całego pliku.

Zamiast pliku w całości można przesyłać tylko jego fragmenty – bloki (ang. *block-level transfer model*). Jako bloki pliku należy tu rozumieć fragmenty pliku, których rozmiar jest zazwyczaj stały. Zaletą tego modelu jest na pewno zmniejszenie wymagań co do przestrzeni dyskowej klienta ze względu na rozmiar bloków. Wadą jest zwiększony ruch sieciowy, szczególnie w przypadku przesyłania plików złożonych z wielu małych bloków.

Jeszcze mniejszą jednostką przesyłania danych plików są bajty (ang. *byte-level transfer model*). Główną zaletą tego modelu jest jego elastyczność. Niestety pojawia się tu problem z zarządzaniem pamięcią podręczną.

Powyższe trzy modele charakteryzują się tym, że mogą być używane do plików o znanej strukturze i do plików bez struktury. Gdy weźmiemy pod rozwagę tylko pliki o znanej strukturze, automatycznie nasuwa się pomysł, aby tam gdzie jest to możliwe, użyć rekordów jako jednostek transferu danych (ang. *record-level transfer model*).

Tytuł przedmiotu

Semantyka współdzielenia plików

- Semantyka systemu UNIX
  - wymaga uporządkowania operacji w czasie
- Semantyka sesji
  - pomiędzy otwarciem a zamknięciem pliku wyniki operacji są widoczne tylko dla użytkownika pliku
- Semantyka współdzielonych plików niemodyfikowalnych
  - każda aktualizacja to nowa pełna wersja pliku
- Semantyka transakcji
  - zachowanie własności transakcji dla operacji na plikach; transakcja wyróżniona jest specjalnymi instrukcjami: *początek_transakcji* i *koniec_transakcji*

Tytuł modułu (11)

W systemach plików, które umożliwiają korzystanie przez wielu użytkowników z tego samego pliku w tym samym czasie, pojawia się problem jak ma wyglądać efekt takich jednoczesnych operacji. Problem związany jest głównie z operacjami modyfikującymi pliki, dlatego niezbędne jest określenie m.in. sposobu propagacji zmian wśród klientów plików.

Jedno z rozwiązań tego problemu bazuje na semantyce systemu UNIX. Użycie tej semantyki wymaga całkowitego uporządkowania operacji w czasie. W ten sposób każda operacja odczytu widzi wszystkie operacje zapisu, które nastąpiły przed nią. W wypadku gdy plik jest współdzielony przez kilku użytkowników, modyfikacja jednego użytkownika będzie natychmiast widoczna przez operacje odczytu innego użytkownika. Semantyka ta wydaję się dosyć naturalna, z tego też powodu często jest implementowana w systemach plików, szczególnie tych nierozproszonych. Implementacja takiej semantyki współdzielenia w systemach rozproszonych nie jest jednak prosta. Jednym ze sposobów osiągnięcia takiej semantyki jest użycie jednego wybranego serwera do koordynacji operacji zapisów i odczytów na plikach. W rozwiązaniu tym zakłada się również brak pamięci podręcznej po stronie klientów. Podejście takie nie gwarantuje jednakże zachowania takiej semantyki. Problemem może być np. kolejności wykonania żądań z powodu opóźnień komunikacyjnych. Co więcej wydajność, niezawodność i skalowalność takiego rozwiązania pozostawiają wiele do życzenia.

Wspomniana semantyka systemu UNIX w praktyce jest znacznie rozluźniona dla systemów rozproszonych, a dla aplikacji, które mimo wszystko wymagają tej semantyki proponuje się specjalne mechanizmy, niezależne od systemu plików np. blokady.

Semantyka sesji jest kolejnym modelem współdzielenia plików. Użycie plików przy zastosowaniu semantyki sesji wygląda następująco: otwarcie pliku, wykonywanie operacji zapisu lub odczytu, zamknięcie pliku po skończeniu wszystkich operacji. Ciąg operacji pomiędzy operacjami zamknięcia i otwarcia pliku określany jest mianem sesji. Sesja charakteryzuje się tym, że wszystkie operacje wykonywane przez danego klienta w ramach jednej sesji są widoczne tylko dla niego. Dopiero po zakończeniu sesji zmiany dokonane na pliku są widoczne przez procesy innych klientów. Ponieważ ten sam plik może być modyfikowany równocześnie przez kilku klientów, pojawia się pytanie jak wygląda plik wynikowy po takich operacjach. W praktyce najczęściej zakłada się, że wynik takich równoczesnych sesji działających na tym samym pliku daje w wyniku plik, który jest wynikiem działania jednej z nich. Innymi słowy nie wiadomo, która z aktywnych sesji utworzy zawartość pliku, na którym operują. Semantyka pliku ze względu na swój charakter i wymagania używana jest najczęściej stosowana z polityką przesyłania całego pliku.

Kolejna semantyka określa pliki jako niezmienne (ang. *immutable*). Każda aktualizacja pliku oznacza utworzenie kolejnej pełnej wersji pliku. Semantyka ta zezwala na współdzieleni plików tylko w trybie do odczytu. Znika tu problem, kiedy modyfikacje danego pliku maja być widoczne dla innych użytkowników pliku.

Ostatnią prezentowaną semantyką współdzielenia plików jest semantyka transakcji. Transakcja podobnie jak sesja obejmuje ciąg operacji zapisu i odczytu zawartych pomiędzy specjalnymi instrukcjami *początek_transkacji* i *koniec_transakcji*. Różnica w stosunku do semantyki sesji polega na tym, że transakcja może obejmować operacje na wielu plikach. Wykonanie kilku współbieżnych transakcji na danym pliku daje nam gwarancję, że zawartość tego pliku będzie taka, jak gdyby transakcje te były wykonane sekwencyjnie w pewnym porządku.

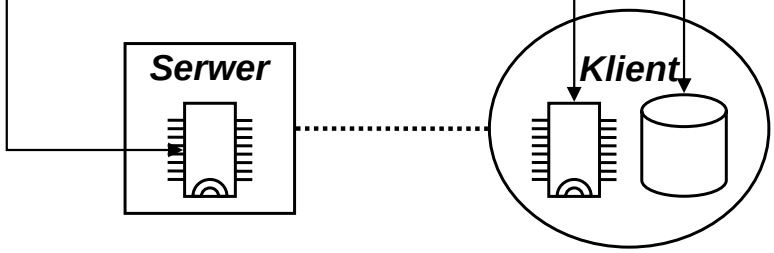
Pamięć podręczna

- Pamięć podręczna służy do przechowywania *podręcznej* kopii danych. Mogą to być np. całe pliki lub tylko wybrane ich fragmenty.
- Użycie pamięci podręcznej poprawia znacząco działanie rozproszonych systemów plików. Poprawiają się m.in. skalowalność oraz niezawodność.
- Czynniki mające wpływ na działanie pamięci podręcznej:
  - rozmiar minimalnej jednostki danych
  - sposób obsługi pamięci podręcznej

Stosowanie pamięci podręcznej służy przechowywaniu pewnych danych lokalnie w celu ograniczenia liczby zdalnych odwołań, które są znacznie bardziej czasochłonne w przypadku rozległych sieci komputerowych. Również systemu scentralizowane korzystają z dobrodziejstwa pamięci podręcznej np. poprzez zapamiętywanie niektórych danych w pamięci głównej, żeby później zapobiec odwołaniom do dysku. Ogólne założenie jest takie, że w przypadku stosowania pamięci podręcznej lokalne operacje na danych są szybsze, a przez to bardziej wydajne. Jak udowodniono schemat użycia pamięci podręcznej w rozproszonych systemach plików ma znaczący wpływ na jego skalowalność i niezawodność. Znaczącymi czynnikami przy doborze odpowiedniego schematu stosowania pamięci podręcznej są m.in. rozmiar minimalnej jednostki danych, która jest przechowywana w pamięci podręcznej oraz to w jaki sposób jest ona obsługiwana. W dalszej części zajmiemy się takimi cechami pamięci podręcznej jak lokalizacja pamięci podręcznej, propagowanie modyfikacji i walidacja pamięci podręcznej.

Lokalizacja pamięci podręcznej

- Miejsce przechowywania kopii podręcznej:
  - *pamięć główna serwera*
  - *dysk z danymi po stronie klienta*
  - *pamięć główna klienta*



Ważnym czynnikiem dla efektywności działania pamięci podręcznej jest jej umiejscowienie. Przy założeniu, że oryginał pliku znajduje się na serwerze, mamy trzy możliwe miejsca przechowania podręcznej kopii pliku.

W celu uniknięcia dostępu do dysku serwera określone pliki można przechowywać w pamięci głównej serwera. Klient, który żąda pliku oszczędza w ten sposób na operacjach dyskowych po stronie serwera. Mechanizm ten jest stosunkowo prosty do implementacji, a ponadto może on być całkowicie przeźroczysty dla klienta. Kopia pliku w pamięci podręcznej jest fizycznie blisko oryginału, dlatego zapewnienie spójności obu nie powinno Stanowic większego problemu. Zaletą tego podejścia jest także wsparcie semantyki systemu UNIX dla współdzielenia plików. Należy jednak zauważyć, iż zastosowanie pamięci podręcznej na serwerze nie zmniejsza wcale kosztów komunikacji i jest praktycznie obojętne, jeżeli chodzi o skalowalność i niezawodność systemu rozproszonego.

Pamięć podręczna może być zastosowana również po stronie klienta. Mamy tu do wyboru dysk lub pamięć główną. Użycie dysku eliminuje ruch w sieci, ale pozostaje kwestia operacji dyskowych. Jedną z głównych zalet przechowywania podręcznej kopii pliku na dysku klienta jest niezawodność. Pamięć podręczna tego rodzaju może być również zazwyczaj bardzo duża, czego rezultatem jest m.in. trafność odwołań do danych w takiej pamięci. Gdy pamięć główna klienta jest za mała, użycie takiej pamięci podręcznej okazuje się wręcz niezbędne. W przypadku tego podejście istotną cechą jest ograniczenie komunikacji pomiędzy serwerem plików a klientem co procentuje w postaci skalowalności systemu rozproszonego. Niedogodnościami pamięci podręcznej po stronie klienta jest konieczność kontaktowania się z serwerem dla każdej operacji dostępu do pliku mimo, że dane są dostępne lokalnie.

Przechowywanie kopii podręcznej w pamięci głównej klienta ma dwie zasadnicze cechy: zmniejsza konieczność komunikacji klienta z serwerem oraz ogranicza liczbę operacji dyskowych. Użycie takiej pamięci podręcznej wpływa znacząco na skalowalność systemu, a także na jego niezawodność.

Podsumowując zastosowanie dysku po stronie klienta, jako miejsca pamięci podręcznej zwiększa znacząco niezawodność systemu, jednakże to pamięć główna klienta gwarantuje szybszy dostęp do danych.

Tytuł przedmiotu

Propagowanie modyfikacji

- Od sposobu propagacji modyfikacji ściśle zależy semantyka współdzielenia danych
- Sposoby propagowania zmian:
  1) Natychmiastowe przepisywanie zmodyfikowanych danych (ang. *write-through*)
  2) Opóźnione zapisy

Tytuł modułu (14)

Użycie pamięci podręcznej w systemach rozproszonych wprowadza istotny problem aktualizacji kopii. Określony plik może posiadać w jednym czasie wiele podręcznych kopii na wielu stanowiskach komputerowych. Kopie, które są jednocześnie modyfikowane w różnych miejscach, po pewnym czasie mogą stać się niespójne. Konieczne jest utrzymanie spójności tych kopii. Problem utrzymania spójności jest szczególnie widoczny dla pamięci podręcznej po stronie klienta. Kluczowe są tu dwie kwestie: kiedy przesłać zmodyfikowane dane w pamięci podręcznej do serwera oraz jak zweryfikować poprawność danych w pamięci podręcznej. Od sposobu propagowania modyfikacji ściśle zależy semantyka współdzielenia plików w rozproszonych systemach plików. Ponadto ma ona bardzo duży wpływ na wydajność i niezawodność systemu.

Podejście polegające na natychmiastowym przepisywaniu (ang. *write-through*) zmodyfikowanych danych do serwera jest jednym ze schematów propagowania modyfikacji. Takie rozwiązanie oferuje zalety w postaci wysokiej niezawodności oraz dogodności dla zastosowania semantyki systemu UNIX. Pierwsza zaleta wynika z tego, że dane są przesyłane natychmiast do serwera i ewentualna awaria klienta nie powinna spowodować utraty większej ilości aktualnych danych. Wadą takiego podejścia jest natomiast konieczność komunikowania się klienta z serwerem podczas każdej operacji zapisu. Efektem tego jest znacznie obniżona wydajność. Z tego względu rozwiązanie to zalecane jest szczególnie dla systemów, gdzie liczba operacji odczytu znacznie przewyższa liczbę operacji zapisu.

Opóźnione przepisywanie

- Minimalizuje narzut komunikacyjny poprzez nagromadzenie pewnej liczby modyfikacji i wysłanie jej do serwera w postaci jednego pakietu – komunikatu
- Schemat opóźnionego zapisu można rozróżnić w zależności od tego kiedy są aktualizowane dane na serwerze
  - Zastosowanie pewnej szczególnej polityki aktualizacji
  - Okresowa aktualizacja
  - Przy zamykaniu pliku
- Problemy z niezawodnością

Poprzednio prezentowany schemat natychmiastowego przepisywania był skuteczny w przypadku stosunkowo małej liczby zapisów. W celu zminimalizowania narzutu komunikacyjnego przy aktualizowaniu pamięci podręcznej wprowadzono opóźnione zapisy zmian na serwerze. Gdy klient wykonuje operacje zapisu w pamięci podręcznej, jej wyniki nie są od razu przekazywane do serwera. Klient jedynie może powiadomić serwer, że zaszła jakaś zmiana. Dopiero po pewnym czasie i być może zebraniu wielu operacji modyfikacji serwer jest powiadamiany o wszystkich zmianach.

Można wyróżnić kilka odmian schematów z opóźnionym zapisem w zależności od tego kiedy aktualizowane są dane na serwerze. Pierwsza metoda zakłada istnienie pewnej polityki, która decyduje o tym kiedy dane z kopii podręcznej są przesyłane do serwera. Zastosowanie odpowiedniej polityki może znacznie poprawić efektywność systemu. Minusem takiego rozwiązania może być zbyt długie przechowywanie aktualnych danych tylko w pamięci podręcznej, co wiąże się z kolejnymi problemami np. niezawodnością.

Kolejna metoda zakłada okresowe przesyłanie danych od klienta do serwera. Co pewien okres czasu wszystkie modyfikacje, które nastąpiły od momentu ostatniej synchronizacji pamięci podręcznej klienta z serwerem, są przesyłane do serwera.

W trzeciej metodzie opóźnionego zapisu dane z pamięci podręcznej przesyłane są do serwera, gdy odpowiadający im plik zostanie zamknięty. Taki schemat przesyłania nadaje się szczególnie dla plików, które otwierane są na długie okresy i są często modyfikowane. W przypadku plików, które otwierane są na krótko i są rzadko modyfikowane, zmniejszenie narzutu komunikacyjnego może okazać się niewielkie.

Stosowanie opóźnionego przepisywania zmian z pamięci podręcznej na serwer ma kilka ważnych cech. Operacje zapisu wykonywane są stosunkowo szybko, ponieważ zmiany dokonywane są tylko na kopii podręcznej. Zmodyfikowane dane mogą być skasowane zanim trafią na serwer. W ten sposób redukowane są zbędne komunikaty. Poza tym przesyłanie kilku modyfikacji pliku jednocześnie jest bardziej wydajne niż przesyłanie ich jedna za drugą. Opóźnione zapisywanie nie jest pozbawione jednak wad. Pojawiają się tu problemy z niezawodnością (np. awaria klienta i utrata zawartości pamięci podręcznej). Poza tym semantyka współdzielenia nie jest tu ściśle określona i zależy od czasu propagacji danych.

Walidacja pamięci podręcznej

- W rozproszonym systemie plików wielu klientów może mieć kopie tych samych danych i mogą na nich równocześnie operować
- Walidacja jest mechanizmem, który pozwala na
  - sprawdzenie czy zawartość pamięci podręcznej jest aktualna i
  - zaktualizowanie w razie potrzeby kopii podręcznej
- Podejścia do walidacji pamięci podręcznej:
  - Walidacja inicjowana przez klienta
  - Walidacja inicjowana przez serwer

W rozproszonym systemie kopie tych samych plików mogą istnieć jednocześnie na wielu komputerach. Co więcej mogą być one jednocześnie modyfikowane. To z kolei wiąże się z modyfikacja pamięci podręcznej wielu klientów. Poprzednio omawialiśmy sposób propagowania danych od pewnego klienta do serwera. Powstaje jednak problem, co zrobić z pamięcią podręczną innego klienta, który korzysta z tego samego pliku co pierwszy. Gdy klient modyfikuje dane pewnego pliku, kopie podręczne innych klientów stają się przestarzałe. W tym momencie ważna staja się odpowiednia walidacja spójności pamięci podręcznej z innymi kopiami. Jeżeli dane okazują się niespójne, trzeba je unieważnić i aktualizować. Wyróżnia się dwa główne podejścia do sprawdzania i aktualizacji pamięci podręcznej: Walidacja inicjowana przez klienta oraz walidacja inicjowana przez serwer. Obie zostaną omówione w dalszej części wykładu.

Tytuł przedmiotu

Walidacja inicjowana przez klienta

- Klient kontaktuje się z serwerem w celu sprawdzenia poprawności danych przechowywanych w swojej pamięci podręcznej
- W zależności od częstotliwości walidacji różnie się semantyka współdzielenia
  - Przy każdej operacji dostępu
  - Okresowo
  - W momencie otwierania pliku
- Różnice pomiędzy danymi wykrywa się stosując np.: wektory wersji, sumy kontrolne lub porównując daty modyfikacji

Tytuł modułu (17)

W przypadku walidacji inicjowanej przez klienta (ang. *client-initiated validation*) to klient kontaktuje się z serwerem, aby sprawdzić czy jego kopia podręczna jest spójna z główną kopią. Semantyka współdzielenia różni się w zależności od częstotliwości walidacji pamięci podręcznej.

Klient może na przykład sprawdzać aktualność pamięci podręcznej przy każdej operacji dostępu do danych. Rozwiązanie takie nie jest jednak do końca pożądane ze względu na to, że podważa ono sens używania pamięci podręcznej.

Walidacja inicjowana przez klienta może być również przeprowadzana co pewien stały okres czasu.

Jeszcze inne podejście zakłada sprawdzanie pamięci podręcznej w momencie otwierania odpowiedniego pliku. Schemat ten często używany jest przy okazji semantyki sesji z propagacją zmian przy zamknięciu pliku.

To czy kopie danych pliku w pamięci podręcznej różnią się od głównej kopii pliku, określa się np. za pomocą różnic w datach modyfikacji. Innym sposobem porównania kopii plików jest użycie znaczników czasu, wektorów wersji, sum kontrolnych itp.

Tytuł przedmiotu

Walidacja inicjowana przez serwer

- Stosowana w celu zmniejszenia ruchu jaki generuje walidacja inicjowana przez klienta
- Klient powiadamia serwer, że otwiera plik do czytania lub pisania, a serwer śledzi stan plików, tak aby wykrywać ewentualne konflikty
- Implementacja bardziej złożona niż w przypadku walidacji inicjowanej przez klienta

Tytuł modułu (18)

W sytuacji gdy częstotliwość walidacji pamięci podręcznej inicjowanej przez klientów jest duża, ruch w sieci również jest zwiększony, a serwer może być znacznie obciążony. Aby rozwiązać ten problem wprowadzono walidację pamięci podręcznej inicjowaną przez serwer.

W tym podejściu klient powiadamia serwer, kiedy otwiera plik do czytania lub pisania. Serwer pamięta tryby w jakich są używane poszczególne pliki w taki sposób, że może śledzić możliwości powstania ewentualnych niespójności danych. Np. gdy serwer zauważy, że jeden plik został otwarty w trybie do zapisu, to wie że nie może otworzyć jednocześnie tego samego pliku drugi raz. Natomiast jeżeli plik został otwarty w trybie do odczytu, nic nie stoi na przeszkodzie, aby inni użytkownicy czytali go w tym samym czasie. Gdy użytkownik kończy korzystać z pliku, wysyła do serwera odpowiednie powiadomienie, żeby ten aktualizował swoje informacje.

Walidacja inicjowana przez serwer mimo swojej efektywności ma też kilka wad. Komplikuje kod klienta i serwera, co jest skutkiem odejścia od tradycyjnego modelu klient-serwer i żądanie-odpowiedź. Serwer musi być w tym przypadku pełnostanowy. Co więcej można powiedzieć, że w pewien sposób nadal używana jest walidacja pamięci podręcznej inicjowanej przez klienta, ponieważ sprawdzanie pamięci podręcznej przebiega podczas otwarcia pliku.

Walidacja pamięci podręcznej w systemie AFS

- AFS – Andrew File System – rozproszony system plików
- System AFS wykorzystuje mechanizm wywołań zwrotnych w celu walidacji pamięci podręcznej
- Serwer zanim dokona zmian na oryginale pliku, powiadamia wszystkich klientów danego pliku, o których wie, że posiadają jego kopię
- AFS używa semantyki sesji – powiadamianie odbywa się po zakończonej sesji

Opiszemy teraz krótko walidacje inicjowaną przez serwer w rozproszonym systemie plików AFS. AFS używa mechanizmu wywołań zwrotnych (ang. *callback*). W podejściu zakłada się, że pamięć podręczna jest prawidłowa do momentu, gdy serwer powiadomi, że jest inaczej. Dla każdego pliku, serwer przechowuje dane o klientach, którzy posiadają kopie pliku w pamięci podręcznej. W ten sposób serwer, który posiada listę klientów używających określonych plików, zanim dokona zmian na oryginale jakiegoś pliku powiadamia o tym swoich klientów. Ponieważ system AFS używa semantyki sesji, powiadamianie odbywa się po dokonanych operacjach na pliku a przy zamykaniu sesji. Wszyscy klienci, którzy posiadają kopie w pamięci podręcznej dezaktualizują je do momentu ponownej aktualizacji.

W porównaniu z wcześniej prezentowanym rozwiązaniem, w przypadku systemu AFS serwer nie musi być informowany o operacjach otwarcia pliku, a jedynie w momencie zakończenia sesji, które zawierały operacje zapisu.

Tytuł przedmiotu

Zwielokrotnianie plików

- Wpływa znacząco na dostępność plików
- Najczęstszą jednostką danych, która jest zwielokrotniana jest plik
- Zastosowanie zwielokrotniania wpływa na:
  - Niezawodność systemu
  - Zmniejszenie ruchu w sieci
  - Poprawę przepustowości systemu
  - Skalowalność
- Stosując zwielokrotnianie można wykorzystać równoważenie obciążenia przy odwołaniach do danych

Tytuł modułu (20)

Zwielokrotnianie jest jednym z głównych środków do osiągnięcia dużej dostępności danych w środowisku rozproszonym, dlatego używa się go również w rozproszonych systemach plików. Podstawową jednostką zwielokrotniania jest najczęściej plik. Kopie pliku zwane replikami są najczęściej umieszczane na wielu rozproszonych serwerach.

Poza wcześniej wymienioną zwiększoną dostępnością, zwielokrotnianie oferuje szereg innych korzyści. Utworzenie wielu kopii tych samych danych pozwala na zwiększenie niezawodności systemu plików. Jeżeli chodzi o korzyści komunikacyjne, zwielokrotnianie może poprawić czasy reakcji na żądania klientów, zmniejsza ruch w sieci, poprawia przepustowość systemu. Zwielokrotnianie plików daje również możliwość rozłożenia obciążenia związanego z żądaniami klientów na wiele serwerów. Jeżeli dwa lub więcej serwerów posiadają identyczne dane, żądania dostępu do tych danych można rozdzielić pomiędzy serwery, tak aby jeden serwer nie był nadmiernie obciążony. Wpływa to także na poprawę skalowalności takiego systemu.

Zwielokrotnianie po stronie klienta umożliwia również klientom przeprowadzanie operacji w odłączeniu od serwera, co jest dużą zaletą w przypadku, gdy używane są urządzenia przenośne.

Tytuł przedmiotu

Zwielokrotnianie a pamięć podręczna

- Przyjmuje się, że pamięć podręczna jest związana z kopią danych po stronie klienta
- Kopia danych po stronie serwera (replika) związana jest natomiast ze zwielokrotnianiem (replikacją)
- Repliki są określane zazwyczaj jako bardziej trwałe, bezpieczne, dokładne oraz bardziej dostępne

Zwielokrotnianie podobnie jak pamięć podręczna używa kopii oryginału pliku do wykonania pewnych operacji. Istnieją jednak pewne cechy tych mechanizmów, które pozwalają na ich rozróżnienie.

Kopie oryginału używane przy zwielokrotnianiu (repliki) są zazwyczaj powiązane z serwerem, natomiast kopie w pamięci podręcznej umiejscowione są zazwyczaj u klienta. Przyjmuje się ponadto że repliki danych są bardziej trwałe, dostępne, bezpieczne i dokładne od danych przechowywanych w pamięci podręcznej. Zazwyczaj kopie podręczne są zależne od replik i na nich bazują np. swoje aktualizacje. Pamięć podręczna stosowana jest również tam gdzie wymagana jest lokalność przy dostępie do danych. Repliki są stosowane z kolei w celu poprawienia dostępności i wydajności.

Przeźroczystość zwielokrotniania

- Użytkownik systemu plików nie musi być świadomy fizycznego rozproszenia swoich danych
- Sterowanie replikacją określa:
  - Ile powinno być kopii danych,
  - Na jakich serwerach powinny się one znajdować
- Replikacja może być:
  - Jawna dla użytkownika
  - Ukryta przed użytkownikiem

Zwielokrotnianie jest mechanizmem, który powinien być w praktyce niewidoczny dla użytkownika. Użytkownika często nie interesuje gdzie są jego dane i w ilu kopiach. Interesuje go raczej to, że dane są ciągle dostępne i że są bezpieczne. Dlatego system, który używa zwielokrotniania danych powinien w zasadzie zachowywać się, tak jakby go w praktyce nie było. Ze przeźroczystością zwielokrotniania wiążą się dwa podstawowe aspekty: przeźroczystość sterowania zwielokrotnianiem oraz nazewnictwo replik.

Sterowanie replikacją określa ile powinno być kopii danych i na jakich serwerach. Zwykle sterowanie zwielokrotnianiem odbywa się w pełni automatycznie, ale nie jest tak zawsze. Są sytuacje, w których użytkownik powinien być świadomy niektórych aspektów sterowania. Pozwala to na zwiększenie elastyczności systemu i daje większą swobodę użytkownikowi.

Wyróżnia się dwa odmienne podejścia do replikacji. Pierwsze podejście określa replikację jako zupełnie jawną dla użytkownika. Określa on np. gdzie mają być umieszczone repliki danego pliku, ile ma być tych replik, może kasować wybrane repliki. Drugie podejście reprezentuje zupełnie odmienny styl zwielokrotniania i ukrywa przed użytkownikiem cały proces replikacji.

Aktualizacja wielu kopii

- Podejścia do problemu aktualizacji wielu kopii
  - Repliki tylko do odczytu (ang. *read-only replication*)
  - Czytaj z dowolnej kopii, zapisuj do wszystkich (ang. *read-any – write-all protocol*)
  - Zapisuj w dostępnych replikach (ang. *available-copies protocol*)
  - Zastosowanie podstawowej repliki, na której dokonywane są zapisy propagowane później do innych serwerów (ang. *primary-copy protocol*)

Rozproszony system plików, który umożliwia współbieżną modyfikacje wielu plików prędzej czy później napotka na problem utrzymania spójności danych. Współbieżna aktualizacja wielu kopii jest jednym z ważniejszych zagadnień projektowych w przypadku rozproszonych systemów plików. Do rozwiązania tego problemu stosowane są w miarę potrzeb różne podejścia. Poniżej opiszemy niektóre z nich.

Repliki tylko do odczytu (ang. *read-only replication*) stosowane są tam gdzie tworzone są repliki plików tylko do odczytu. Ponieważ pliki, które mogą być modyfikowane nie są poddawane zwielokrotnianiu, to problem aktualizacji replik znika całkowicie. Takie podejście nadaje się szczególnie w systemach i aplikacjach, gdzie dane są bardzo często czytane, ale rzadko modyfikowane.

Systemy plików, które stosują inne mniej restrykcyjne od poprzedniego podejście do replikacji, pozwalają na odczyt danych z dowolnej repliki pliku, ale operacja zapisu musi być jednocześnie wykonana na wszystkich replikach pliku (ang. *read-any – write-all protocol*). Do implementacji takiego podejścia używane są m.in. blokady zakładane na plikach.

Kolejne podejście eliminuje zasadniczą wadę poprzedniego protokołu. Mianowicie, przedstawiony wcześniej protokół aktualizacji replik wymagał od systemu, aby zapis był wykonany na wszystkich replikach. Pojawiał się jednak problem, co robić jeżeli jedna lub więcej replik nie jest dostępnych w danej chwili. Z tego powodu rozluźniono jeszcze bardziej wymagania co do liczby aktualizowanych replik. W podejściu, które aktualizuje tylko dostępne repliki (ang. *available-copies protocol*) nie wymaga się, aby operacja zapisu musiała być wykonana jednocześnie na wszystkich replikach. Serwer, który uległ awarii, zanim zrealizuje jakiekolwiek żądanie klienta, odzyskuje dane kopiując je np. z innego serwera mającego aktualne dane.

Jeszcze inne podejście do aktualizacji wielu replik polega na zastosowaniu podstawowej kopii danych (ang. *primary-copy protocol*). Każdy plik posiada swoją podstawową kopię danych. Wszelkie operacje odczytu mogą być wykonywane przez klientów na kopii podstawowej lub dowolnej innej replice pliku. Natomiast operacje zapisu muszą być wykonywane tylko na tej jednej kopii podstawowej. Każdy serwer, który posiada replikę danego pliku aktualizuje swoje dane poprzez synchronizację z serwerem przechowującym podstawową replikę tego pliku. Może to zrobić czekając aż serwer z kopią podstawowa powiadomi go zaszłych modyfikacjach lub sam może żądać informacji o aktualizacjach.

Tytuł przedmiotu

Aktualizacja wielu kopii – protokół używający kworum (1)

- Operacja odczytu:
  1) Zdobądź odpowiednią liczbę *r* replik, które będą stanowiły kworum do odczytu
  2) Wybierz spośród wybranych kopii tę o najwyższym numerze wersji,
  3) Wykonaj operację odczytu na tej replice.

- Operacja zapisu:
  1) Zdobądź odpowiednią liczbę *w* replik (kworum do zapisu),
  2) Wybierz replikę z największym numerem wersji,
  3) Zwiększ numer wersji,
  4) Zapisz nową wartość do wszystkich replik wchodzących w skład kworum oraz przypisz im nowy numer wersji.

Tytuł modułu (24)

Stosując wcześniej przedstawione rozwiązania możemy zauważyć problem przy aktualizacji wielu replik w sieciach, które ulegają okresowemu podziałowi na mniejsze grupy. Z pomocą przychodzą protokoły, które używają kworum w celu rozstrzygnięcia czy dany zapis lub odczyt mogą być wykonane. Kluczowe są tu dwie liczby *r* i *w*. Pierwsza liczba *r* (kworum do odczytu), określa ile replik musi system skonsultować, aby wykonać operację odczytu. Druga liczba *w* (kworum do zapisu), oznacza liczbę replik, na których musi być dokonany zapis, aby mógł on być uznany za ważny. Ograniczeniem jakie się dodatkowo nakłada na liczby *r* i *w*, jest spełniony warunek ($r + w > n$), gdzie *n* jest liczbą wszystkich replik . Warunek ten ma zagwarantować, że istnieje co najmniej jedna wspólna replika pliku dla wszystkich zbiorów, które tworzą kworum do odczytu i zapisu. Innymi słowy w zbiorze replik do odczytu lub zapisu musi istnieć co najmniej jedna aktualna kopia.

Ponieważ operacje zapisu nie muszą być wykonywane na wszystkich replikach zaistniała potrzeba rozróżniania wersji replik. Robi się to np. poprzez przypisanie każdej replice numeru wersji, który jest zwiększany wraz z kolejnymi jej modyfikacjami.

Operacja odczytu dla pliku przebiega w tym protokole następująco: 1) Zdobądź odpowiednia liczbę *r* replik, które będą stanowić kworum do odczytu, 2) Wybierz spośród wybranych kopii tę o najwyższym numerze wersji, 3) Wykonaj operację odczytu na tej replice.

Operacja zapisu do pliku wykonuje się w następujących krokach: 1) Zdobądź odpowiednią liczbę *w* replik (kworum do zapisu), 2) Wybierz replikę z największym numerem wersji, 3) Zwiększ numer wersji, 4) Zapisz nową wartość do wszystkich replik wchodzących w skład kworum oraz przypisz im nowy numer wersji.

Tytuł przedmiotu

Aktualizacja wielu kopii – protokół używający kworum (2)

- Odmiany protokołu:
  - *r = 1, w = n* (ang. *read-any – write-all*)
  - *r = n, w = 1* (ang. *read-all – write-any*)
  - *r ≈ w* (ang. *majority-consensus protocol*)
  - protokół z wagami (ang. *consensus with weighted voting*)

r – kworum wymagane do odczytu

w – kworum wymagane do zapisu

n – liczba wszystkich replik

Tytuł modułu (25)

W ramach protokołów stosujących kworum do aktualizacji replik możemy wyodrębnić kilka ich odmian. Rozróżnienie wynika głównie z wielkości liczb *r* (kworum do odczytu) i *w* (kworum do zapisu).

Jeżeli *r=1*, a -*w=n* (gdzie *n* jest liczbą wszystkich replik), mamy do czynienia ze specjalnym przypadkiem protokołu, który został zaprezentowany wcześniej jako protokół, który czyta z dowolnej repliki a pisze do wszystkich (ang. *read-any – write-all*).

Gdy *r=n*, a *w=1* mamy do czynienia z protokołem, który czyta do czytania wymaga wszystkich replik, natomiast do zapisu tylko jednej (ang. *read-all – write-any*). Protokół ten może być wykorzystany szczególnie tam, gdzie liczba operacji zapisu jest znacznie większa od liczby operacji odczytu.

Trzecia odmiana protokołu bazującego na kworum (ang. *majority-consensus protocol*) zakłada, że liczba *r* i *w* są równe lub prawie równe. Protokołu tego używa się tam, gdzie liczba operacji zapisu i odczytu jest podobna.

Czasami zakłada się również, że niektóre repliki z pewnych względów (np. bezpieczeństwo, wydajność) są ważniejsze od innych. Mamy wtedy do czynienia z podejściem, w którym używa się wag replik do określenia kworum (ang. *consensus with weighted voting*). Liczby *r* i *w* określają odpowiednio liczbę (sumę wag) potrzebną do wykonania operacji odczytu i zapisu. Warunek, który muszą spełnić liczby *r* i *w* zmienia się na warunek (*r+w>v)*, gdzie *v* oznacza sumę wszystkich wag przypisanych do replik danego pliku.

Niezawodność

- Czynniki, które wpływają na niezawodności systemu plików:
  - Dostępność danych
  - Odporność na awarie nośników danych
  - Możliwość odzyskiwania danych
- Wpływ na niezawodność ma to, czy serwer plików jest pełnostanowy albo bezstanowy

Ważnym zagadnieniem dla rozproszonych systemów plików jest niezawodność. Ponieważ tematowi temu poświęcamy odrębny wykład, wspomnimy poniżej tylko o niektórych aspektach niezawodności, ważnych z punktu widzenia systemów plików.

W środowiskach rozproszonych często będziemy mieli do czynienia z mniejszymi lub większymi awariami. Ich występowanie może okazać się fatalne w skutkach, nie tylko dla spójności danych, ale i całego systemu.

W celu osiągnięcia jak najwyższej niezawodności systemu plików ważne są szczególnie takie cechy plików jak dostępność danych, odporność na wszelkiego rodzaju awarie nośników danych, możliwość odzyskiwania danych (np. po cofnięciu operacji przez klienta). Dostępność opisuje w tym wypadku m.in. czas przez jaki dany plik jest niedostępny.

Istotny wpływ na niezawodność ma architektura serwera plików, a dokładniej to czy serwer przechowuje historię interakcji z klientem, czy też nie. W zależności od tego wyróżniamy serwery pełnostanowe i bezstanowe. Oba typy serwerów zostaną przedstawione w dalszej części wykładu.

Tytuł przedmiotu

Pełnostanowe serwery plików

- Przechowuje informacje o stanie klienta
- Pozwala na efektywniejsze wykonywanie operacji wraz z przyrostem informacji o kliencie
- Pojawia się problem nadmiernej ilości informacji – rozwiązaniem jest np. okresowe kasowanie informacji o stanie klienta

Tytuł modułu (27)

Serwer pełnostanowy przechowuje i zbiera informacje o stanie klienta w trakcie otrzymywania kolejnych żądań. Zapamiętanie stanu przez serwer pozwala mu np. wykonać pewne operacje znacznie wydajniej, niż gdyby tych informacji nie posiadał. Z powodów ograniczonych zasobów serwer nie może często przechowywać całej historii interakcji z klientem, dlatego informacje te są przechowywane tylko przez pewien okres. Może to być np. okres trwania sesji zawarty pomiędzy operacjami otwarcia i zamknięcia pliku.

Jako przykład operacji, która wymaga zapamiętania stanu przez serwer posłużymy się operacją odczytu. Niech klient wyśle żądanie otwarcia pliku do serwera, a następnie operację odczytu pierwszych *n* bajtów pliku. Serwer w odpowiedzi na żądanie wykona operację otwarcia pliku i odczyta żądane dane, które następnie prześle do klienta. Klient zażąda odczytania kolejnych bajtów z otwartego pliku. W tym momencie serwer, który pamięta miejsce w pliku gdzie zakończył poprzednio czytać dane, może kontynuować odczyt następnych bajtów.

Tytuł przedmiotu

Bezstanowe serwery plików

- Nie przechowuje żadnych informacji o kliencie
- Odporny na awarie w przeciwieństwie do serwera pełnostanowego, który po awarii musi odzyskiwać dane
- Serwery bezstanowe są z reguły mniej wydajne od serwerów pełnostanowych
- Trzeba zadbać o idempotentność operacji, aby w razie wielokrotnego powtarzania nie powstały błędy

Tytuł modułu (28)

Bezstanowy serwer plików w przeciwieństwie do pełnostanowego nie przechowuje żadnych informacji o stanie klienta. Można powiedzieć, że każde żądanie klienta musi być w pewien sposób samowystarczalne, zawierać wszystkie niezbędne informacje do jego wykonania.

Istotną zaletą serwera bezstanowego jest jego odporność na ewentualne awarie. W przypadku awarii serwera pełnostanowego często tracone są wszystkie informacje o stanie i potrzebne są mechanizmy do przywracania stanu sprzed awarii. Ponieważ serwer bezstanowy nie przechowuje informacji, jak serwer pełnostanowy problem odzyskiwania stanu praktycznie znika.

Serwery bezstanowe mają jednak swoje słabości. Wykonanie operacji przez takie serwery trwa z reguły dłużej niż robią to serwery pełnostanowe. W przypadku powtarzania pewnego żądania trzeba uważać na to czy operacja, którą ma wykonać serwer bezstanowy jest idempotentna tzn. czy daje ten sam wynik mimo tego, że jest powtarzana wiele razy.

Tytuł przedmiotu

Transakcje w systemach plików

- Transakcyjność wprowadza do systemu plików m.in.:
  - Mechanizm odzyskiwania danych
  - Atomowość
- Transakcje ułatwiają współbieżne wykonywanie operacji na plikach poprzez utrzymanie odpowiedniego poziomu spójności danych

Tytuł modułu (29)

Zajmiemy się teraz przyczynami dla których systemy plików stosują mechanizmy transakcyjne.

Pierwszą przesłanką dla, których stosuje się transakcje w rozproszonych systemach plików jest mechanizm odzyskiwania danych, przywracania stanu. Istotną cecha transakcji jest tu atomowość. Przypuśćmy np., że któryś z serwerów plików uległ awarii podczas, gdy jakiś klient wykonywał na nim operacje. Atomowość zapewni nam, że stan po awarii będzie taki jak przed awarią. Natomiast gdy przyjrzymy się co by było gdyby nie było atomowości zobaczymy, że dane po takiej awarii mogłyby być zupełnie niespójne.

Transakcje ułatwiają również współbieżne współdzielenie plików, które zachowuje spójność modyfikowanych danych. Wykonywanie wielu operacji jednocześnie przez wielu klientów na tych samych danych często prowadzi do niepożądanej zawartości pliku wynikowego. Transakcje pozwalają rozwiązać ten problem szeregując odpowiednio operacje odczytu i zapisu. W ten sposób można przewidzieć wynik grupy operacji.

Problematyka transakcji została poruszona w odrębnym wykładzie i nie będziemy się dalej nią tutaj zajmować.

Tytuł przedmiotu

NFS

- NFS (Network File System) – sieciowy system plików stworzony przez firmę Sun Microsystems
- Pracuje w środowisku heterogenicznym
- Oferuje wiele poziomów przeźroczystości m.in.:
  - Położenia
  - Dostępu
  - Wędrówki
- Pozwala na importowanie i eksportowanie danych
- Każdy komputer może pełnić jednocześnie role klienta i serwera

Tytuł modułu (30)

NFS (Network File System) jest standardem sieciowego systemem plików stworzonym przez firmę Sun Microsystems. Istnieje wiele implementacji systemu NFS. NFS został stworzony z myślą o możliwości pracy na wielu platformach systemowych w środowiskach heterogenicznym. Wraz z rozwojem systemu NFS powstają kolejne jego wersje. Najnowsza wersja została opatrzona numerem 4.

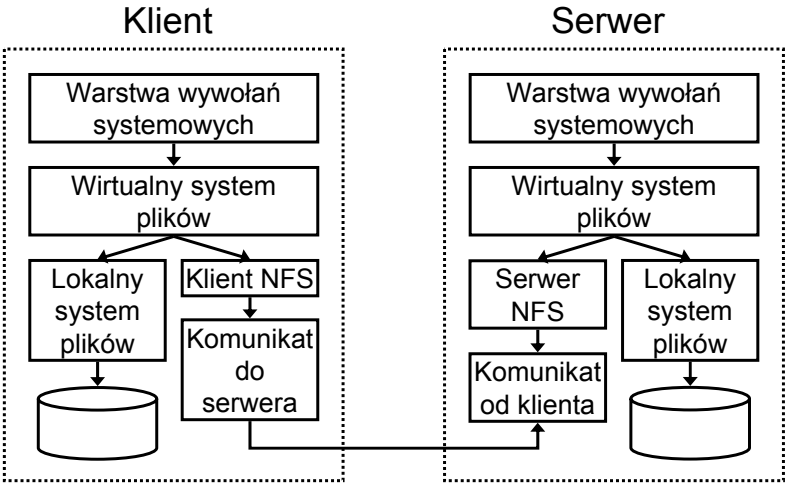
Spójrzmy teraz na główne cechy systemu NFS. NFS umożliwia wyeksportowanie części struktury katalogowej wraz z zawartością, czyli plikami na inny zdalny komputer. Na tym zdalnym komputerze wyeksportowana część struktury katalogowej jest zaszywana w lokalnej strukturze plików w taki sposób, aby dostęp do danych niczym nie różnił się od operacji na lokalnych plikach. Sam proces importowania i przyłączania zdalnej struktury katalogów określany jest jako montowanie. W systemie tym role klienta i serwera może praktycznie pełnić każdy komputer, który posiada odpowiednie oprogramowanie.

NFS jest jednym z najczęściej używanych systemów plików tego typu. Charakteryzuje się przeźroczystością na poziomie położenia, wędrówki, dostępu. W najnowszej wersji systemu NFS widać dążenie twórców do zwiększenia skalowalności systemu. Przejawem tego jest m.in. przygotowanie pewnych prostych mechanizmów do zastosowania replikacji, która nie była w ogóle wspierana w poprzednich wersjach tego systemu.

Tytuł przedmiotu

Architektura systemu NFS

- Ilustracja przykładowej implementacji systemu NFS:



Tytuł modułu (31)

NFS oferuje przeźroczysty dostęp do plików, a klienci tego systemu nie są w zasadzie świadomi gdzie znajduje się plik, na którym operują. NFS należy do systemów, które bazują na modelu zdalnego dostępu. Aby zrealizować taki model dostępu posłużono się warstwą wirtualnego systemu plików, który pośredniczy pomiędzy warstwą wywołań systemowych, a rzeczywistym systemem plików, zdalnym lub lokalnym. Jeżeli okazuje się, że klient żąda dostępu do lokalnego systemu plików żądanie przekazywane jest do lokalnego systemu plików. W przeciwnym razie, gdy żądanie odwołuje się do pliku na zdalnym komputerze, informacja o tym przekazywana jest do warstwy klient NFS, która komunikuje się ze zdalnym serwerem NFS. Serwer NFS poprzez warstwę wirtualnego systemu plików stara się zrealizować żądanie i odsyła ewentualny wynik do klienta.

Tytuł przedmiotu

Przykładowe usługi plikowe w systemie NFS

- Przykładowe operacje (NFS wersja 4)

Operacja	Opis
`Open`	Otwarcie pliku
`Close`	Zamknięcie pliku
`Lookup`	Szukanie pliku za pomocą nazwy pliku
`Readdir`	Czytanie zawartości katalogu
`Write`	Zapisywanie danych do pliku
`Read`	Odczytywanie danych z pliku

Tytuł modułu (32)

W skład usług systemu NFS wchodzi szereg operacji plikowych. Różnią się one w różnych wersjach tego systemu, co widać gdy prześledzimy np. listę operacji dla wersji 3 oraz 4.

Wśród operacji znajdziemy oczywiście operacje do odczytu (*read*) i zapisu (*write*) danych z i do pliku. Są także operacje do zarządzania atrybutami plików. Istotną operacją jest operacja *lookup*, która służy do poszukiwania uchwytu plikowego (identyfikatora) na podstawie nazwy pliku. Operacja *readdir* stosowana jest z kolei do czytania zawartości katalogów. W wyniku jej działania klient uzyskuje nazwy plików wraz z odpowiadającymi im identyfikatorami. Poza tym mamy także operacje do zmiany nazwy plików, usuwania plików itp.

Pewna nowością w wersji 4. systemu NFS jest wprowadzenie operacji otwarcia (ang. *open*) i zamknięcia (ang. *close*) pliku. Wcześniejsze wersje systemu NFS pozwalały na użycie bezstanowego serwera plików. W wersji 4. założono jednak, że serwer przechowuje stan pomiędzy operacjami.

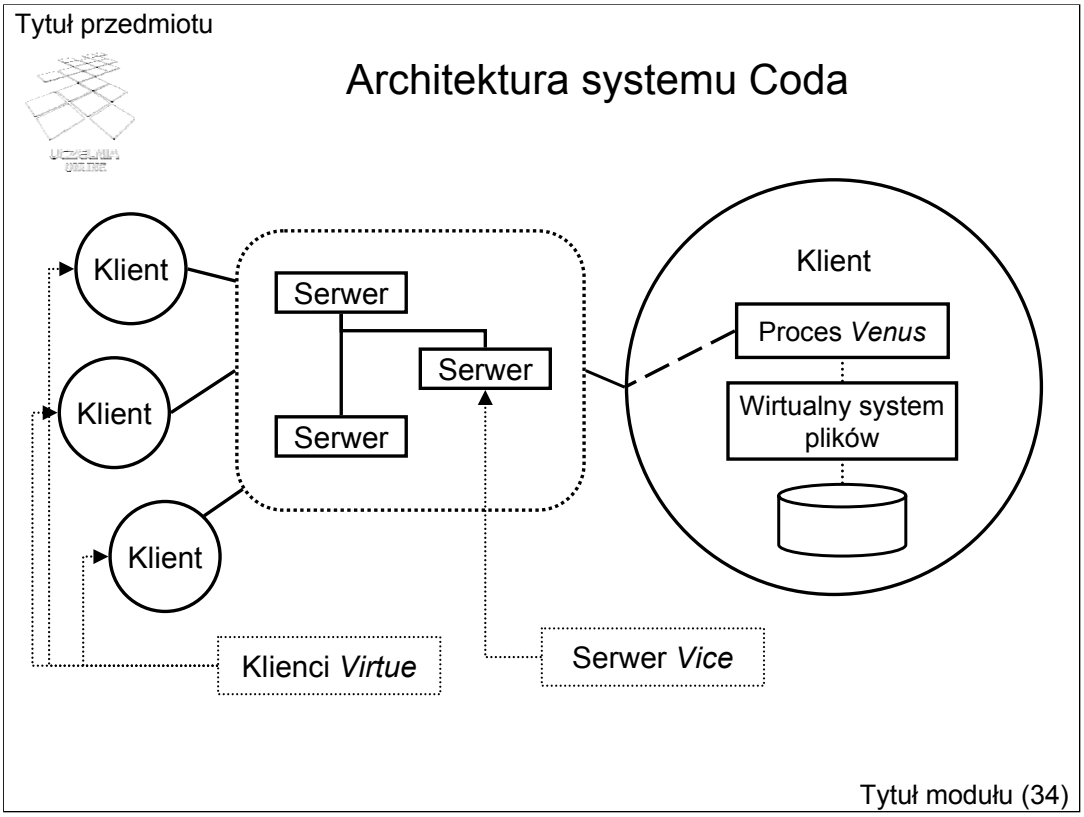
Coda

- Duży nacisk położono na wysoką dostępność danych
- Duży stopień przeźroczystości
- Zwielokrotnianie modyfikowalnych plików
- Coda daje możliwość przeprowadzania operacji na danych w odłączeniu od sieci – kluczowa cecha dla urządzeń przenośnych
- Globalna dzielona przestrzeń nazw

Coda jest rozproszonym systemem plików opracowanym w Carnagie Mellon University (CMU). Jednym z głównych celów systemu Coda jest uzyskanie wysokiej dostępności. Aby to osiągnąć Coda używa m.in. mechanizmu pamięci podręcznej. Poza dostępnością w systemie Coda duży nacisk położono również na skalowalność i bezpieczeństwo. System charakteryzuje się również przeźroczystością lokalizacji, nazewnictwa oraz przeźroczystością awarii. Poza tym Coda dostarcza globalnie dzielonej przestrzeni nazw.

Bardzo przydatną cechą systemu jest możliwość wykonywania operacji na danych, gdy klient jest czasowo odłączony od sieci lub odpowiednie serwery nie są dostępne.

Interfejs operacji plikowych w Coda podobny jest do systemu UNIX.

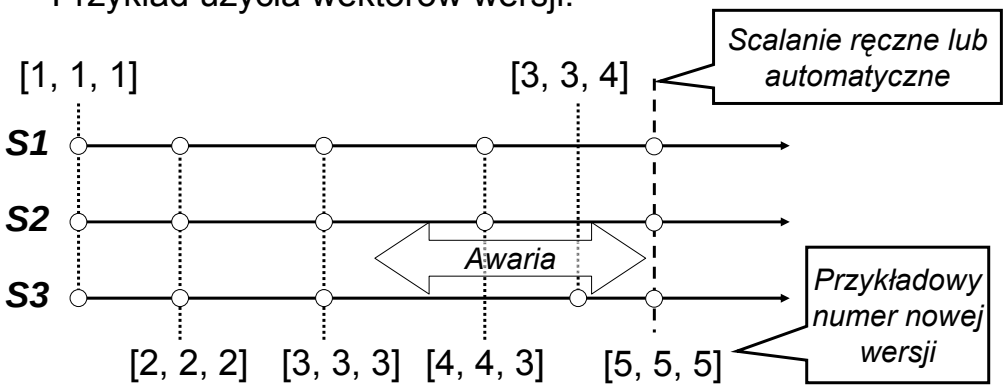


Coda wywodzi się z wcześniejszego systemu plików AFS. Architektura systemu Coda jest z tego względu, pod wieloma względami identyczna jak w systemie AFS. Komputery, które korzystają z systemu Coda podzielone są na dwie grupy. Pierwsza grupa nazwana *Vice* składa się z pewnej liczby serwerów plików. Druga grupa to komputery użytkowników określane jako *Virtue*. *Virtue* pozwalają użytkownikom na dostęp do systemu plików. W ramach każdej stacji *Virtue* znajduje się proces użytkownika, *Venus*. *Venus* oprócz funkcji udostępniania klientowi danych z serwera, odpowiedzialny jest za umożliwienie użytkownikowi kontynuowanie operacji na plikach nawet jeżeli niedostępne są chwilowo serwer plików.

Tytuł przedmiotu

Zwielokrotnianie w systemie Coda

- *Wektory wersji* przechowują numery wersji pliku na poszczególnych serwerach z grupy *AVSG*
- Przykład użycia wektorów wersji:



Tytuł modułu (35)

Coda jest systemem plików, który umożliwia zwielokrotnianie plików modyfikowalnych. Zanim krótko opiszemy mechanizm zwielokrotniania przedstawimy niezbędne pojęcia. Plik w systemie Coda przechowywane są w tzw. **tomach** (ang. *volumes*). Tom można porównać do fragmentu systemu plików, który udostępnia jakiś użytkownik. Zbiór serwerów, które maja kopię tomu nazywamy **grupą przechowywania tomu** (ang. *Volume Storage Group, VSG*). W wypadku gdy na skutek awarii klient będzie miał dostęp tylko do części do serwerów, grupę tych serwerów określamy jako **dostępna grupa przechowywania tomu** (ang. *Accessible Volume Storage Group, AVSG*). Jeżeli grupa *AVSG* jest pusta, mówimy że klient jest odłączony.

System Coda używa optymistycznej strategii zwielokrotniania. Klienci mogą jednocześnie modyfikować ten sam plik. W ten sposób powstaje wiele wersji pliku, które przesyłane są do odpowiednich grup *AVSG*. Kluczowe jest tu zagadnienie wykrywania niespójności i usuwanie ich. W tym celu Coda używa **wektorów wersji Coda** (ang. *Coda version vector*), *CVV*. Taki wektor wersji posiada każdy plik. Poszczególne pozycje wektora odpowiadają liczbie modyfikacji pliku wykonanych przez serwery z grupy *AVSG*.

Rozpatrzmy teraz przykład przedstawiony na diagramie. System składa się z trzech serwerów, z których każdy niech przechowuje replikę pewnego pliku *f*. Na początku działania systemu wektory wersji dla pliku *f*, na każdym serwerze są identyczne i wynoszą [1, 1, 1]. W skutek pierwszych dwóch, pomyślnie przeprowadzonych modyfikacji wektory wersji pliku *f* zwiększyły swoją wartość na wszystkich serwerach do wartości [3, 3, 3]. W tym momencie pojawiła się awaria, która spowodowała odcięcie trzeciego serwera od dwóch pozostałych. Kolejne modyfikacje na pliku *f* spowodowały powstanie dwóch różnych wersji tego samego pliku oznaczonych odpowiednio wektorami wersji [4, 4, 3] na serwerach 1 i 2 oraz [3, 3, 4] na serwerze 3. Kiedy nastąpi awaria zostanie usunięta, wektory wersji kopii pliku *f* zostaną porównane i nastąpi faza scalania. Scalanie w zależności od możliwości może być przeprowadzone automatycznie lub przy asyście użytkownika.