Strona tytułowa
Rozproszone systemy plików
Autorzy:

- Bartosz Bekier
- Karol Kurach
- Krzysztof Pawlowski

Plan Prezentacji

- wstęp - co to jest DFS, zalety, wady
- omówienie wybranych systemów plików
  - ZFS
  - googleFS
  - globalFS
- porównanie
- inne rozproszone systemy plików
- podsumowanie

Definicja

- system plików, który pozwala na rozproszenie danych po wielu lokalizacjach fizycznych (serwerach) przy zintegrowaniu lokalizacji logicznej.
- mimo, że pliki są rozrzucone po wielu serwerach plików dla użytkownika mogą być one widoczne na jednym serwerze plików tyle że np. w wielu katalogach.

Zalety

- rozproszenie zasobów po wielu komputerach
- niezauważalne dla klienta - operuje jak na zwykłym systemie plików
- większe bezpieczeństwo danych (np. trzymanie kopii tego samego pliku na kilku serwerach)
- skalowalność - aby zwiększyć pojemność systemu , wystarczy dołączyć nowe serwery
- dzielenie obciążenia pomiędzy rożne serwery

Wady

- trudniejsze w administracji
- większy narzut czasowy przy modyfikacjach pliku - trzeba zmienić wszystkie kopie
- narzut czasowy wynikający z przesyłania danych przez sieć (do 3 razy wolniej niż zwykły odczyt z dysku)

Czym jest zFS?

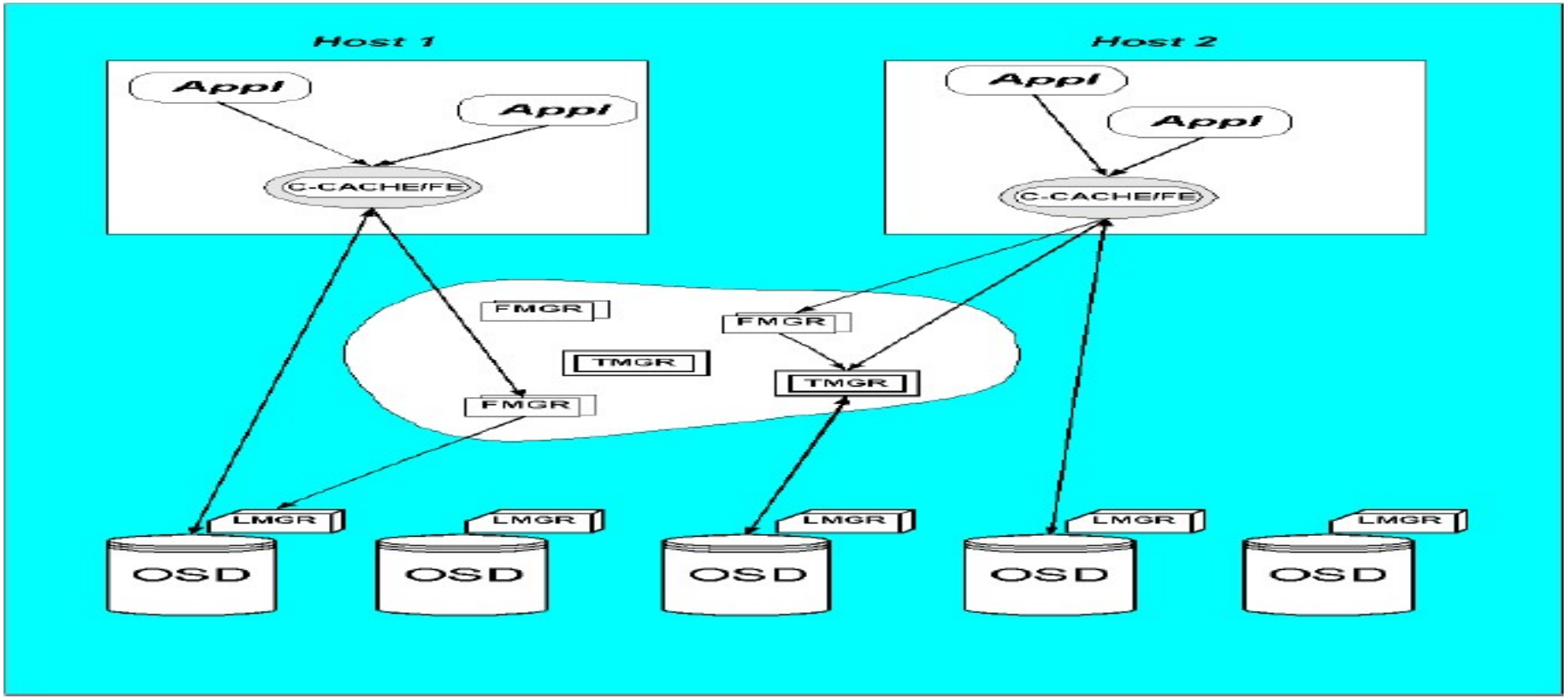
- rozproszony system plików rozwijany w IBM
- działa w środowisku wielu komputerów połączonych szybką siecią (np. klastry)
- zFS = zSeries File System
- działa na róznych systemach operacyjnych IBM-a

Historia

- dalszy ciąg prowadzonych badań, wcześniej:
  - Distributed Sharing Facility
  - Antara Object Store Design

Założenia projektowe

- system plików, który działa dobrze zarówno na niewielu jak i wielu maszynach (rzędu tysięcy)
- zbudowany z taniego, ogólnie dostępnego sprzętu
- bazuje na Object Store Devices (OSD) jako urządzeniach składowania danych
- rozproszenie zarządzania plikami i zarządzania składowaniem danych
- używa pamięci wszystkich komputerów jako globalnej pamięci podręcznej
- osiąga prawie liniową skalowalność



Rysunek: Architektura zFS

Komponenty

- Object Store Device (OSD) - urządzenia składujące dane
- Lease Manager (LMGR) - zarządca dzierżaw (rodzaju blokad)
- File Manager (FMGR) - zarządca plików
- Transaction Manager (TSVR) - zarządca transakcji
- Front-End/Cache (FE/Cache) - front-end i pamięć podręczna

Object Store Device

- OSD Api:
  - tworzenie i usuwanie objektów
  - czytanie zakresów bajtów z obiektu
  - zapisywanie zakresów bajtów do obiektu
- OSD zapewnia:
  - abstrakcję pliku (na poziomie strumienia bajtów)
  - bezpieczeństwo - prawa dostępu na poziomie objektów
  - korzystanie z takiej abstrakcji umożliwia skupienie uwagi na zarządaniu plikami i skalowalności

Lease Manager

- Potrzebny był komponent, który zarządza blokadami na plikach, aby zapewnić spójność danych przy jednoczesnym dostępie
- Rozwiązuje dwie kwestie związane z blokowaniem:
  - kontrolowanie dostępu do dysków - potrzebne w każdym systemie plików
  - w środowisku działania systemu zFS, wiele maszyn może żądać jednocześnie dostępu do urządzeń składowania danych (OSD)

Dlaczego Lease Manager?

- Można było włączyć blokowanie do maszyn OSD, ale wyodrębnienie osobnego komponentu zapewnia lepszą skalowalność!
  - każdy OSD ma dokładnie jednego Lease Manager-a i utrzymuje jego adres
  - każdy OSD ma jedną główną dzierżawę (blokadę), którą przyznaje swojemu Lease Manager-owi
  - LMGR zarządza i przyznaje dzierżawy na konkretne obiekty (pliki, katalogi)
  - zarządca plików (FMGR) przyznaje dzierżawy zakresowe na żądanie Front-Endu

Dlaczego dzierżawy, a nie blokady?

- Lease, czyli dzierżawa = standartowa blokada + ograniczenie czasowe
- Zalety:
  - pozwala uniknąć skomplikowanego mechanizmu wykrywania niedziałających maszyn trzymających blokady
  - w razie awarii maszyny, odzyskujemy dostęp do zasobu po upływie określonego czasu
- Wady:
  - narzut czasowy na odnawianie dzierżaw, przy dłuższym dostępie do plików

File Manager

- przypisywany do pliku podczas jego pierwszego otwarcia
- każde żądanie dzierżawy na pliku jest obsługiwane przez jego zarządce
- zarządca plików współpracuje z odpowiednim Lease Manager-em w celu pozyskania dzierżawy pliku i przyznaje front-endom dzierżawy na konkretne fragmenty pliku
- utrzymuje informacje o położeniu bloków plików i ich dzierżawach
- przydział zarządców plików jest dynamiczny
- zarządza przekazywaniem danych miedzy pamięciami podręcznymi komputerów w razie potrzeby

Transaction Manager

- operacje na metadanych dotyczą kilku obiektów naraz
- aby zapewnić spójność systemu plików, zFS implementuje je jako transakcje rozproszone
- wszystkie operacje na metadanych są obsługiwane przez zarządce transakcji

Front End

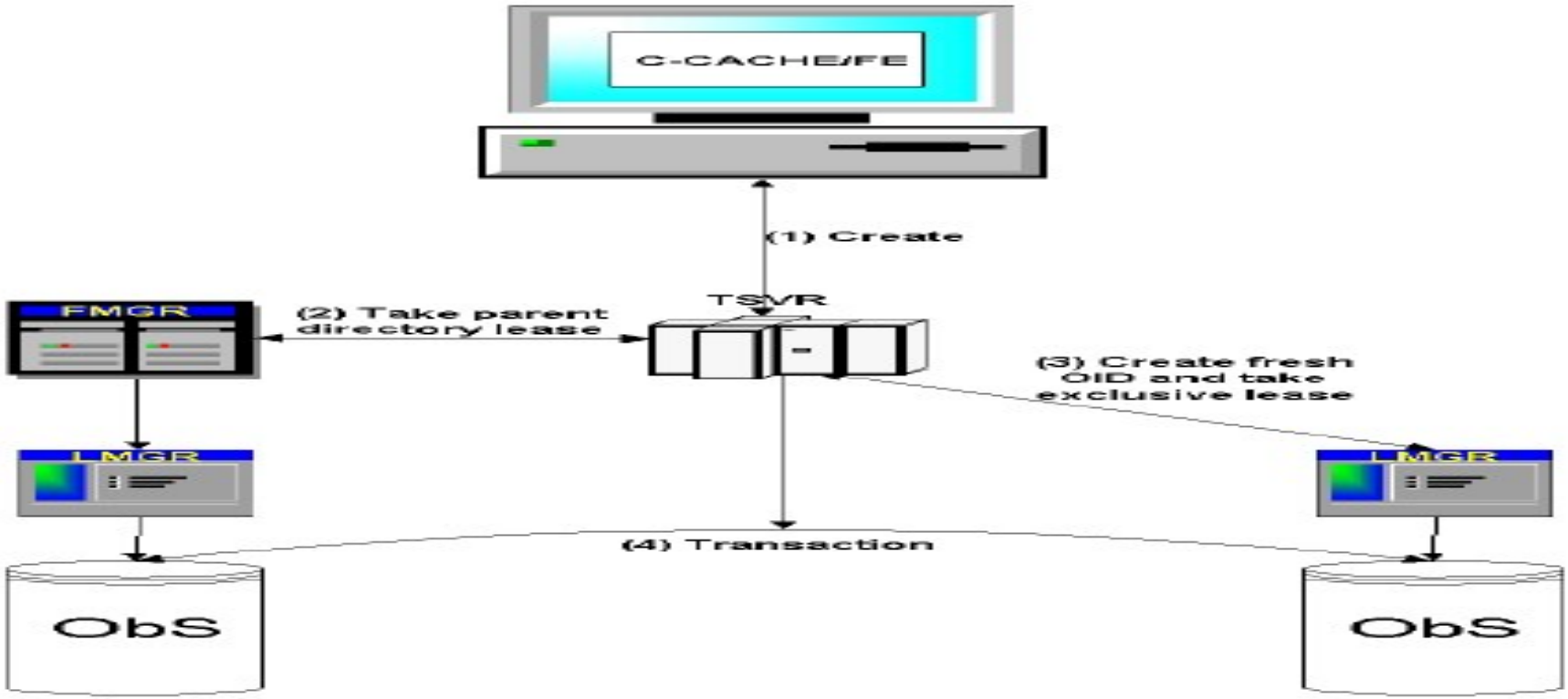
- uruchamiany na każdej maszynie chcącej korzystać z systemu zFS
- tworzy standardowy interfejs POSIX systemu plików
- na systemach Unix-owych integruje się z warstwą VFS

Cache

- pamięc podręczna przechowująca dane systemu zFS
- udostępnia przechowywane dane innym maszynom korzystającym z zFS-a
- oparta na założeniu, że szybciej można przesłać informacje przez sieć, niż odczytać je z dysku

Utworzenie nowego pliku

- front-end dostaje polecenie utworzenia nowego pliku
- zna położenie root-a („/”) i tam zaczyna poszukiwania
- o nieznane fragmenty ścieżki odpytuje zarządce transakcji
- otrzymujemy uchwyt do katalogu, w którym ma zostać utworzony nowy plik (uchwyt to para: (osd_id, oid))
- zarządca transakcji pobiera wszystkie potrzebne dzierżawy (na katalog oraz wyłączną dzierżawę na OSD) i decyduje,w którym OSD umieścić plik
- następuje wykonanie transakcji (przez zarządce transakcji) i zwrócenie uchwytu do pliku



Rysunek: Przykład działania zFS (tworzenie pliku)

Schemat postępowania w razie awarii

- awaria Front-Endu
  - wszystkie dzierżawy wygasną po określonym czasie i dostęp do plików/katalogów, które były używane przez Front-End zostanie przywrócony
- awaria zarządcy plików
  - klienci zapisują zmienione bloki do OSD
  - zauważana przez Front-Endy, gdy nie mogą już odnowić dzierżaw
  - po określonym czasie wygasa dzierżawa na pliki zarządzane przez zarządce plików, który uległ awarii
  - jeśli klienci chcą dalej korzystać z tych plików, tworzą nowego zarządce plików

Schemat postępowania w razie awarii c.d.

- awaria zarządcy dzierżaw
  - zauważana przez zarządców plików, gdy nie mogą już odnowić dzierżaw
  - FMGR-y informują klientów, żeby zapisali zmienione bloki do OSD i zwolnili pliki
  - po określonym czasie wygasa dzierżawa na OSD, utrzymywana przez zarządce dzierżaw
  - klienci tworzą nowego zarządce dzierżaw
- awaria OSD
  - oops... nic już nie da się zrobić, chyba że dane były zreplikowane gdzie indziej

Osiągnięte cele

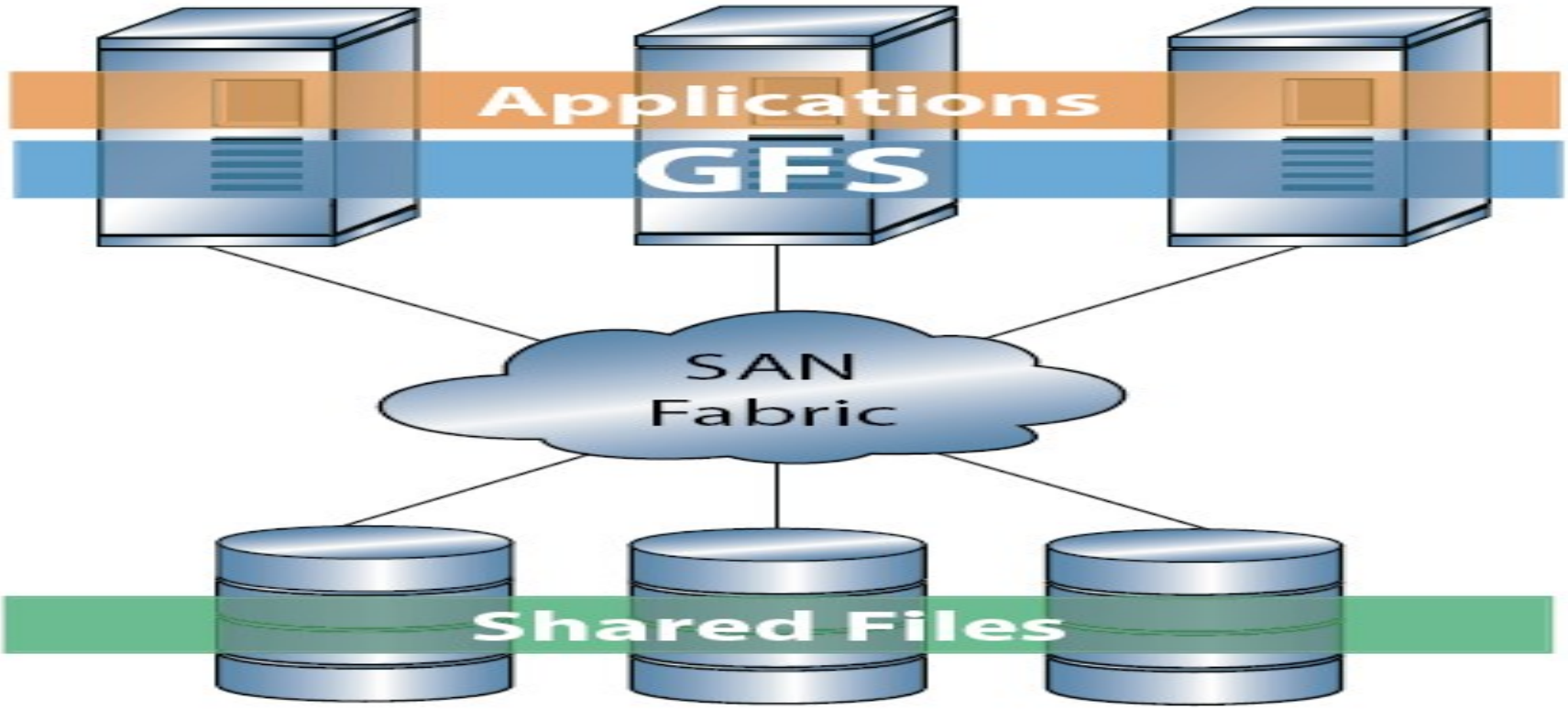
- wysoka skalowalność i wydajność, dzięki:
  - oddzielenie zarządzania składowaniem danych od zarządzania plikami (OSD)
  - globalna pamięć podręczna umożliwia pozyskiwanie danych od innych klientów, zamiast od maszyn OSD
  - brak centralnego serwera i maszyn dedykowanych - zwiększa niezawodność

Czym jest GFS?

- rozproszony system plików rozwijany przez Red Hat Inc.
- klastrowy system plików przeznaczony dla systemu Linux
- działa w oparciu o dzielone urządzenia blokowe - zapewnia węzłom bezpośredni dostęp do nich
  - dzielone urządzenia blokowe = specjalny sprzęt (Fibre Channel, iSCSI i inne)
- dostępny za darmo albo jako komercyjny produkt w ramach Red Hat Enterprise Linux

Rys historyczny

- badania prowadzone na Uniwersytecie Minnesota
- projekt podjęty przez Sistina Software (ok. roku 2000) jako open-source
- zmiana licencji GFS-a na komercyjną (2001), powstanie open-sourceowych forków
- Red Hat przejmuje Sistina Software (2003) i „uwalnia” GFS pod licencją GPL (2004)
- GFS2



Rysunek: Środowisko działania GFS

Najważniejsze cechy

- natywny system plików Linux-a - integruje się z wartstwą VFS
- spójność systemu plików - dzielony dostęp do urządzeń składowania danych jest realizowany poprzez zakładanie blokad
- perfekcyjna spójność danych - zmiany wprowadzone przez jednego klienta są natychmiast widoczne na pozostałych maszynach
- journaling, rozpraszanie metadanych, dinode-y (trochę inne inode-y)
- najbardziej skalowalny klastrowy system plików działający pod Linuxem - ponad 100 węzłów

Współbieżny dostęp do urządzeń składowania danych

- GFS korzysta z zewnętrznych zarządców blokad, są dostępne:
  - Distributed Lock Manager (DLM)
    - najnowsze i zalecane rozwiązanie
    - działa na każdej maszynie w klastrze
    - udostępnia blokowanie na różnych poziomach granularności
  - Grand Unified Locking Manager (GLUM)
    - starsze rozwiązanie, które wychodzi z użycia
    - wymaga przynajmniej 3 zewnętrznych komputerów do roli „serwerów blokad”
  - Nolock
    - jeśli tylko jeden węzeł korzysta z danego urządzenia składowania danych

Po co nam GFS?

- eliminuje potrzebę duplikowania danych na każdym węźle
- pozwala na współbieżny zapis/odczyt danych przez wielu klientów
  - powoduje, że węzły mają dostęp do systemu plików na takich zasadach jak procesy w architekturze SMP
- ułatwia zarządzanie danymi, tworzenie backup-ów i odtwarzania w przypadku awarii
- upraszcza dodawanie nowych serwerów

Wstęp i założenia projektowe

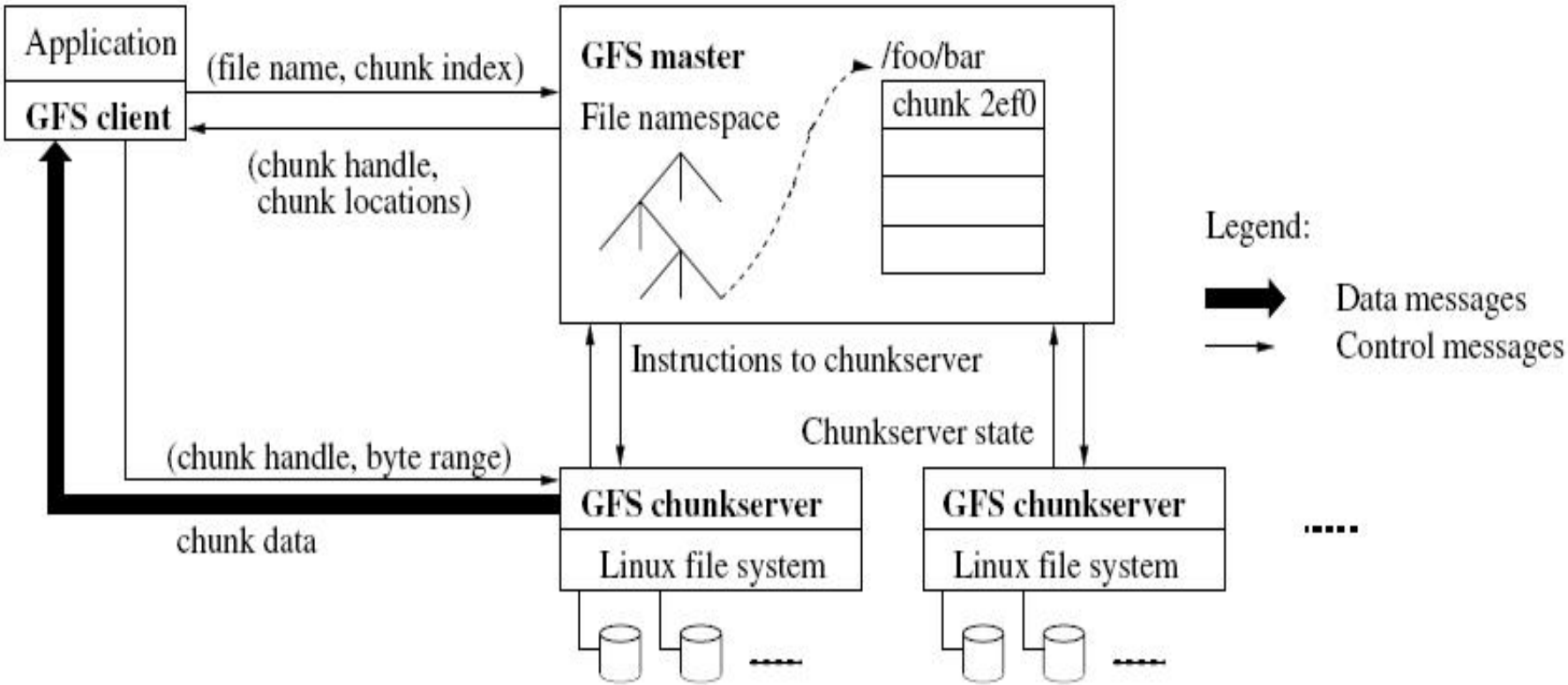
- System plików stworzony przed 2003 r przez Google dla własnych potrzeb
- Google posiada ponad 450,000 serwerów na świecie
- Zamknięty, niedostępny na zasadach komercyjnych ani open-source
- Wywodzi się z wcześniejszego systemu plików BigFiles
  - stworzonego przez Sergeya Brina i Larrego Page'a, założycieli Google'a
- Napędza takie serwisy jak google.com (wyszukiwarka), GMail, Google Maps, Google Video

Założenia

- niedrogi, łatwo dostępny sprzęt
- duże, wielogigabajtowe pliki
- typowe operacje:
  - małe, swobodne zapisy - są wykonywane rzadko i nie muszą być efektywne
  - dopisywanie (ang. append) dużej ilości danych
  - długie, sekwencyjne odczyty
- współbieżne dopisywanie do tego samego pliku
- duża przepustowość ważniejsza od małego opóźnienia

Interfejs

- Create
- Delete
- Open
- Close
- Read
- Write
- **Snapshot**
- **Record Append**


Application
GFS client
(file name, chunk index)
(chunk handle,
chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system
Legend:
Data messages
Control messages

Architektura

- jeden master-serwer, wiele chunk-serwerów, wiele klientów
  - proces użytkownika działający na zwykłej maszynie linuksowej
  - kod klienta GoogleFS dołączony (wlinkowany) do każdej aplikacji
- logiczne pliki podzielone na 64MB-owe ”kawałki”
  - 1 kawałek trzymany jako dokładnie 1 fizyczny plik na standardowym systemie plików
  - jednak w wielu kopiach, na różnych chunk-serwerach
- spamiętywanie wykonuje tylko klient, i dotyczy ono wyłącznie metadanych

Pojedynczy Master

- Czemu centralizacja? Prostota!
- Globalna wiedza jest potrzebna dla:
  - rozmieszczania ''kawałków''
  - decyzji dotyczących replikacji plików

Rozmiar kawałków

- 64 MB - znacznie więcej niż tradycyjny blok dyskowy - dlaczego ?
- Zalety:
  - zmniejszenie interakcji klient - master
  - zmniejszenie narzutu sieci
  - zmniejszenie rozmiaru metadanych
- Wady:
  - fragmentacja wewnętrzna (rozwiązanie: leniwa alokacja)
  - "Hot Spoty- wielu klientów próbujących zdobyć dostęp do danego kawałka. możliwe rozwiązania:
    - wyższy współczynnik replikacji pliku
    - komunikacja klient-klient

Metadane

- Przestrzeń nazw plików i kawałków
  - przechowywana w pamięci oraz w dzienniku master-serwera
- Odwzorowanie plik-kawałek
  - jw, przechowywana w pamięci oraz w dzienniku master-serwera
- Lokacje kopii kawałków
  - w pamięci master-serwera
  - uaktualniane, gdy:
    - master startuje
    - chunk-serwer dołącza do kolekcji

Metadane - przechowywanie w pamięci

- Dlaczego metadane są przechowywane w pamięci master serwera?
  - Szybkość!
- Czy pamięć master-serwera nie ograniczy pojemności całego dysku?
  - Nie, 64MB-owy kawałek potrzebuje 64B metadanych (więc na 640TB potrzeba 640MB pamięci na master-serwerze)
    - Większość plików jest duża
    - Kompresja prefiksowa na nazwach plików

Metadane - dziennik operacji

- Jedyny trwały zapis metadanych
- Określa kolejność współbieżnych operacji
- Krytyczny dla działania systemu
  - Powielany na wielu zdalnych maszynach
  - Klient dostaje odpowiedź, dopiero gdy udało się uaktualnić dziennik oraz jego kopie
- Regularne tworzenie punktów kontrolnych (checkpoint), aby ograniczyć rozmiar dziennika

Model spójności

- Spójność - wszyscy klienci otrzymają te same dane, niezależnie z której repliki je odczytają
- Atomowość operacji
- Po ciągu udanych zapisów, gwarancja że region jest w zdefiniowanym stanie
  - Taka sama kolejność danych na wszystkich kopiach
  - Z każdym kawałkiem trzymany numer wersji, co pozwala wykryć przestarzałe dane

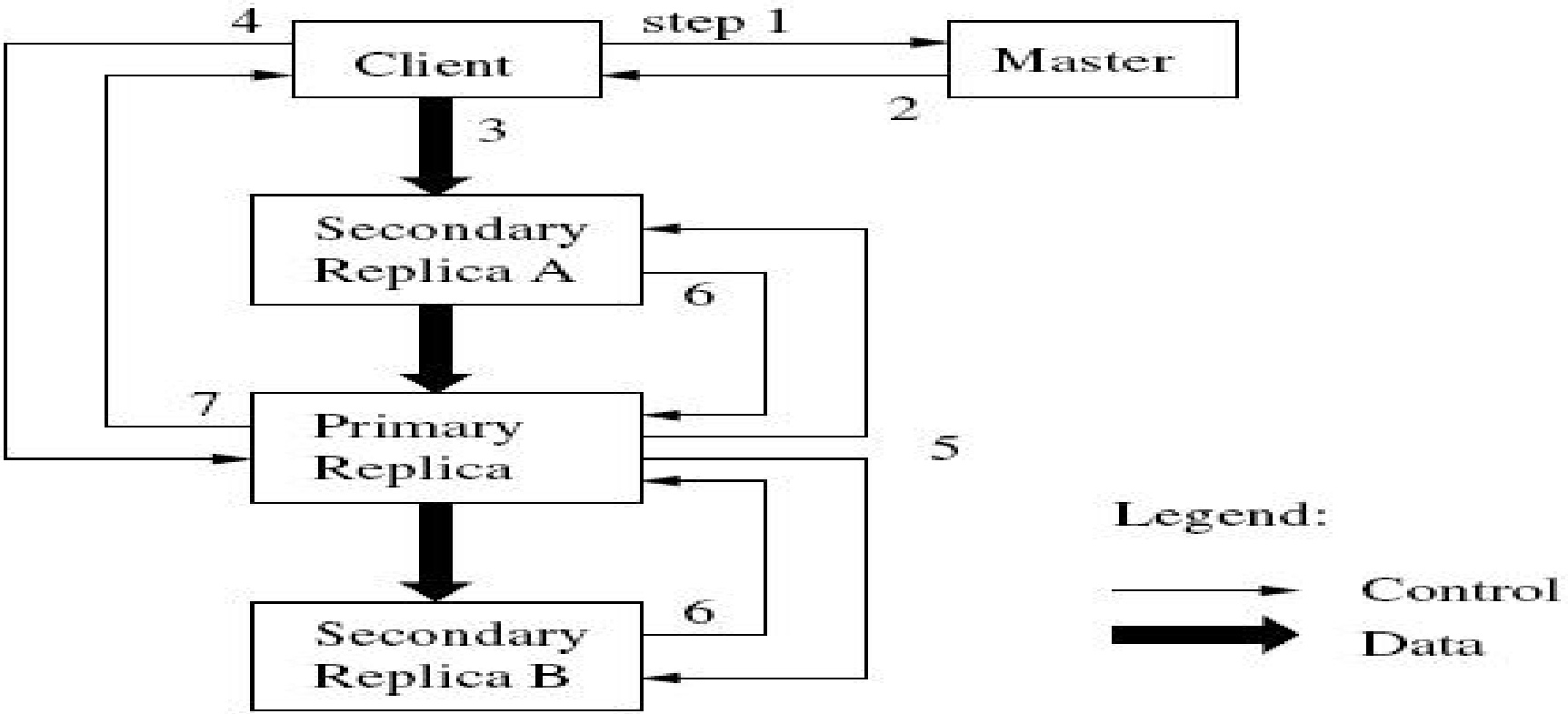
Interakcje z systemem - dzierżawa (lease)

- Dotyczy jednego kawałka
- Przydzielana przez master-serwer
- Dla ustalonego kawałka co najwyżej jeden chunk-serwer posiada dzierżawę, (tzw. główny serwer)
- Cele
  - Zapewnienie spójności wszystkich kopii plików
  - Odciążenie master-serwera, to wybrany chunk-serwer zarządza zmianami

Dzierżawa - dzierżawa, cd.

- Master przyznaje dzierżawę jednemu z chunk-serwerów, który posiada kawałek (główna replika)
- Klient komunikuje się z chunk-serwerem, który posiada dzierżawę
- Wybrany chunk-serwer komunikuje się z pozostałymi chunk serwerami, uaktualniając ich stan
- Domyślnie dzierżawa jest przyznawana na 60 sekund, lecz może być przedłużona bądź skrócona

Przepływ informacji - rysunek



Przepływ informacji

- Przepływ danych jest oddzielony od przepływu sterującego
- Przepływ sterujący (control flow)
  - Klient <-> **master**, główny chunk-serwer
  - Master <-> główny chunk-serwer
- Przepływ danych (data flow)
  - Klient <-> chunk-serwer
  - Chunk-serwer <-> chunk-server
- Dane propagowane do najbliższej repliki (odległość określana na podstawie adresów IP)
- Pipelining

Atomowe dopisywanie danych

- Współbieżne dopisywania są serializowane
- Klient określa tylko dane, które mają zostać dopisane do pliku
- GoogleFS wykonuje operację dopisania
- Zwraca klientowi pozycję zapisanych danych
- Mechanizm bardzo często wykorzystywany przez Google
  - Nie wymaga dodatkowej synchronizacji (jak Distributed Lock Manager)
  - Kolejki wielu producentów-jeden konsument
  - Scalanie wyników obliczeń wykonywanych jednocześnie przez wiele kopii aplikacji

Operacja snapshot

- Efektywnie kopiuje fragment struktury katalogowej, bądź pliku
- Użycie techniki copy-on-write (znane np z SVN)
- Implementacja
  - Modyfikacje metadanych
    - Anulowanie dzierżawy
    - Zapisanie informacji do dziennika
    - Skopiowanie i uaktualnienie metadanych
  - Kopiowanie przy zapisie
    - Może się odbywać lokalnie na chunk-serwerach bez użycia sieci
    - Operacje dyskowe są ok 3x szybsze niż przesyłanie przez sieć

Master serwer - zarządzanie przestrzenią nazw

- Brak struktury opisującej katalogi
- Brak obsługi dowiązań (zarówno twardych jak symbolicznych)
- Synchronizacja dostępu do poszczególnych składowych przestrzeni nazw zapewnia serializację
- Słownik odwzorowuje pełne ścieżki plików na metadane
  - Kompresja prefiksowa
- Pojedyncza metadana - 64B

Synchronizacja przestrzeni nazw

- Każdy węzeł (plik lub katalog) posiada blokadę typu czytalnicy-pisarze
- Przykład blokad podczas współbieżnych operacji SNAPSHOT i CREATE
  - SNAPSHOT "/home/user"to "/save/user"
    - blokada czytalnika /home i /save
    - blokada pisarza na /**home**/**user** i /save/user
  - CREATE "/home/user/foo"
    - blokada czytelnika na /home oraz /**home**/**user**
    - blokada pisarza na /home/user/foo
- Określona kolejność blokad, aby uniknąć zakleszczeń

Strategie

- Tworzenie nowego kawałka
  - Preferowane chunk-serwery z mniejszą zajętością dysku
  - Ograniczenie na ilość tworzonych kwałków dla danego chunk-serwera w jednostce czasu
  - kopie kawałków są rozrzcane po różnych szafach serwerowych (ang. rack)
    - Duże prawdopodobieństwo awarii całej szafy (wspólne zasilanie, sieć...)
    - Komunikacja między szafami jest wolniejsza niż w obrębie jednej szafy
- Replikowanie kawałków
  - Kawałki, które występuja w najmniejszej ilości kopii
  - Kawałki, blokujące żądania klienta

Usuwanie plików

- Leniwość
  - Natychmiastowe logowanie faktu usunięcia pliku
  - Oznaczenie pliku jako skasowanego poprzez zmianę nazwy
    - Faktycznie usunięcie danych z dysku 3 dni później
    - W międzyczasie możliwe odzyskanie pliku poprzez przywrócenie oryginalnej nazwy
- Regularne skanowanie w poszukiwaniu osieroconych kawałków
  - osierocone kawałki to takie, które nie należą do żadnego pliku
  - w przypadku wykrycia, master wysyła do chunk-serwera polecenia usunięcia

Usuwanie plików - c.d.

- Zalety
  - Proste i niezawodne, nawet gdy:
    - Tworzenie kawałka się nie powiodło
    - Polecenia usunięcia zostały zgubione
  - Jednolity sposób obsługi nieużywanych kawałków
  - Wykonywane w tle, amortyzacja kosztu
  - Ochrona przed przypadkowym nieodwracalnym usunięciem

Usuwanie plików - c.d.

- Wady
  - Marnowanie przestrzeni dyskowej, szczególnie w przypadku plików tymczasowych, ktore szybko zostają usuwane
- Rozwiązanie
  - ''Podwójne'' skasowanie zwalnia miejsce na dysku od razu
  - Różne strategie dla różnych fragmentów przetrzeni nazw
- Wykrywanie nieaktualnych kopii
  - Master pamięta najnowszą wersję każdego kawałka
  - W przypadku wykrycia starszej wersji, prosi chunk-serwer o usunięcie i zleca wykonanie repliki aktualnej wersji

Odporność na błędy

- Szybkie przywracanie
  - Automatycznie przewraca stan na podstawie dziennika i rozpocznij działanie w parę sekund
  - System nie odróżnia prawidłowego zamknięcia systemu od awarii
- Replikacja kawałków
  - Różne poziomy replikacji dla różnych części struktury katalogowej. Domyślnie 3 kopie.
  - W przypadku awarii któregoś z chunk-serwerów, brakujące kopie są tworzone na innych działających serwerach
  - Wykrywanie zepsutych kawałków za pomocą sum kontrolnych

Odporność na błędy master serwara

- Dziennik jest replikowany na innych maszynach
- W przypadku padu mastera...
  - Niezależne procesy monitorujące uruchamiają drugą kopię na innej maszynie
  - Aktualny master jest rozpoznawany na podstawie wpisu w dns.
    - wystarczy zmienić ten wpis, aby klienci i chunk-serwery zaczęły korzystać z nowej maszyny
- Zapasowe master-serwery (ang. "shadow" masters)
  - Zapewniają dostęp tylko do odczytu w przypadku, gdy główny master-serwer nie działa
  - Odwzorowują stan głównego master-serwera poprzez śledzenie logów

Spójność danych

- Sumy kontrolne do wykrycia błędnych danych
- Kawałek (64MB) jest podzielony na 64-kilobajtowe bloki, każdy sprawdzany sumą kontrolną
- Chunk-serwer weryfikuje sumę kontrolną. W przypadku wykrycia błędu dane nie są propagowane.
- Operacja dopisywania
  - Przyrostowe modyfikowanie sumy kontrolnej
- Operacja swobodnego zapisu
  - Trzeba wczytać w całości pierwszy i ostatni blok, zmodyfikować, policzyć nową sume kontrolną a następnie zapisać

Podsumowanie

- GoogleFS obsługuje przetwarzanie danych na dużą skalę, wykorzystując tanie komputery osobiste
- Weryfikacja podstawowych założeń tradycyjnego systemu plików
  - Uwzględnienie wymagań aplikacji
  - Traktowanie awarii różnych części systemu jako norma, a nie jako wyjątek
  - Optymalizacja dla ogromnych plików, do których dane są dopisywane, bądź czytane sekwencyjnie
  - Niepełna implementacja standardowego interfejsu systemu plików

Podsumowanie - c.d.

- Odporność na błędy
  - Stałe monitorowanie
  - Replikowanie istotnych danych
  - Szybkie i automatyczne przywracanie
  - Sumy kontrolne dla wykrycia błędów
- Utrzymanie wysokiej przepustowości
  - Oddzielenie informacji kontrolnej od przesyłanych danych
  - Minimalizacja narzutu poprzez duże rozmiary kawałków oraz dzierżawę chunk-serwerów do obsługi większości zadania

Andrew FS

- rozproszony sieciowy system plików, stworzony w Carnegie Mellon University jako część projektu Andrew
- ma przewagę nad tradycyjnymi sieciowymi systemami plików, głównie w kwestiach bezpieczeństwa i skalowalności
- stosuje uwierzytelnianie protokołem Kerberos i listy kontroli dostępu na katalogach dla użytkowników i grup

Wolumin

- wolumin = drzewo plików i podkatalogów
- woluminy tworzone są przez administratorów i podłączane do poszczególnych ścieżek w komórce AFS
- w stworzonym woluminie użytkownicy systemu plików mogą tworzyć katalogi i pliki w zwykły sposób, nie zajmując się jego fizyczną lokalizacją
- administrator może przenieść wolumin na inny serwer lub dysk bez potrzeby informowania o tym użytkowników: operacja może przebiegać w czasie, kiedy wolumin jest używany

Wolumin

- Woluminy mogą być też zreplikowane do jedenastu kopii do odczytu.
- Kiedy aplikacja kliencka korzysta z plików tylko do odczytu, dane pobiera się z jednej z tych kopii.
- użytkownicy tych danych nie muszą się zajmować fizyczną lokalizacją tych kopii, a administratorzy mogą je tworzyć i przemieszczać w razie potrzeby

Przestrzeń nazw

- Przestrzeń nazw na stacji roboczej używającej AFS jest podzielona na wspólną (ang. shared) i lokalną (ang. local)
- Wspólna przestrzeń nazw jest identyczna na wszystkich stacjach
- Lokalne są różne i zawierają pliki tymczasowe potrzebne podczas uruchamiania
- Każde poddrzewo w przestrzeni wspólnej jest przypisane do jednego serwera zwanego nadzorcą
- Pliki z przestrzeni wspólnej są na żądanie buforowane na stacji lokalnej
- Spójność bufora jest utrzymywana przez mechanizm odwołań (ang. callback)

Implementacje

- 3 głowne implementacje
  - Transarc (IBM)
  - OpenAFS
  - Arla
- AFS miał duży wpływ na wersję 4 popularnego systemu NFS
- AFS jest poprzednikiem systemu plików Coda.

Coda

- rozproszony, sieciowy system plików, wywodzący się z AFS2 (Andrew FS)
- Testowy serwer Cody ma adres "testserver.coda.cs.cmu.edu"
- Na bazie Cody powstał eksperymentalny system uruchamiania oprogramowania dla Linuksa, Konvalo (konvalo.org)

Ficzersy

- zupełnie za darmo
- prawie działa mimo braku połączenia z serwerem
- wysoka wydajność - cachowanie po stronie clienta
- działa nawet jak część sieci padnie
- dobra skalowalność

Ficzersy

- replikacje pliku po stronie serwera
- dobrze zdefniowana semantyka dzielenia zasobów (nawet w przypadku błędów w sieci)

NFS

- Network File System (NFS) to oparty o UDP lub TCP protokół zdalnego udostępniania systemu plików.
- opracowany przez Sun Microsystems (1984)
- NFS jest standardowym sieciowym systemem plików na Uniksach.
- każdy komputer w sieci może działać zarówno jako klient, jak i serwer NFS
- adaptowalny do różnych architektur

Założenia projektowe

- Przezroczystość dostępu
- Przezroczystość położenia
- Przezroczystość awarii
- Przezroczystość wydajności
- Przezroczystość wędrówki

A to sie nie udało :(

- Przezroczystość zwielokrotniania
- Przezroczystość współbieżności
- Skalowalność

Inne Problemy

- Trudno zapewnić, że dana operacja została wykonana
- Scenariusz działania:
  - żądanie zostaje wysłane przez klienta
  - żądanie zostaje odebrane przez serwer
  - operacja zostaje wykonana
  - potwierdzenie zostaje wysłane przez serwer
  - potwierdzenie zostaje odebrane przez klienta
- Jeśli miedzy odebraniem żądania a wysłaniem potwierdzenia wystąpi błąd, klient nie może się dowiedzieć, czy operacja została wykonana.

SSHFS

- SSHFS(Secure SHell FileSystem) jest linuksowym systemem plików
- implementacja wykorzystuje moduł kernela FUSE(Filesystem in Userspace)
- jest wersja dla Mac OS X (MacFuse)

Korzyści

- operowanie jak na normalnym systemie plików
- każdy użytkownik może sobie podmontować zdalną partycję
- połączenie szyfrowane