Rozproszone systemy plików

Piotr Broda, Andrzej Pańkowski, Jakub Waszczuk, Bartosz Zaborowski

18 stycznia 2008

Rozproszone systemy plików

- Współdzielone systemy plików*
  np. GlobalFS
- Rozproszone systemy plików
  np. AndrewFS
- Rozproszone systemy plików z tolerancją błędów
  np. Coda
- Rozproszone równoległe systemy plików
  np. Lustre
- Rozproszone równoległe systemy plików z tolerancją blędów
  np. GoogleFS

Wymagania stawiane przed systemami plików

- Przezroczystość albo niezależnośc położenia
  przezroczystość - może ujawniać zależności między składowymi nazwy a komputerami, nazwa nie daje informacji o fizycznym położeniu pliku, nie jest możliwa automatyczna zmiana położenia pliku. niezależność - nazwy pliku nie trzeba zmieniać wtedy, gdy plik zmienia swoje fizyczne położenie, lepsza abstrakcja pliku (nazwa określa zawartość, nie położenie), oddziela hierarchię nazw od hierarchii urządzeń pamięci
- Przezroczystość dostępu
  istniejące programy lokalne mogą żadnych zmian w kodzie działać na zdalnych plikach.

Wymagania stawiane przed systemami plików cd.

- Przezroczystość awarii
  awarie zdalnego serwera plików są postrzegane jako awarie niektórych lokalnych plików w VFS
- Przezroczystość wydajności
  konieczność przesyłania plików przez sieć nie ma dużego wpływu na wydajność
- Przezroczystość wędrówki
  przenoszenie plików między serwerami nie wpływa na sposób dostępu do nich

Wymagania stawiane przed systemami plików cd.

- Przezroczystość zwielokrotniania
  zwielokrotnianie plików pozwalające na przyspieszenie dostępu i zwiększenie bezpieczeństwa niewidoczne z punktu widzenia użytkownika
- Przezroczystość współbieżności
  możliwość sterowania współbieżnością przez zakładanie blokad na pliki
- Skalowalność
- Bezpieczeństwo przechowywanych danych
- ...

Spis treści

Wstęp

zFS to rozproszony system plików rozwijany przez IBM. W porównaniu do innych rozproszonych systemów plików, zFS ma charakteryzować się bardzo dużą skalowalnością oraz łatwością budowy. zFS rozszerza pracę wykonaną w innym, wcześniejszym projekcie IBM-a - DFS.

Cechy zFS

- Duża skalowalność
  zFS ma działać zarówno na kilku połączonych siecią komputerach, jak i na kilkudziesięciu tysiącach klientów.
- Budowa
  zFS można zbudować z łatwo dostępnych części (np. komputery osobiste) połączonych za pomocą szybkiej sieci.

Cechy zFS (2)

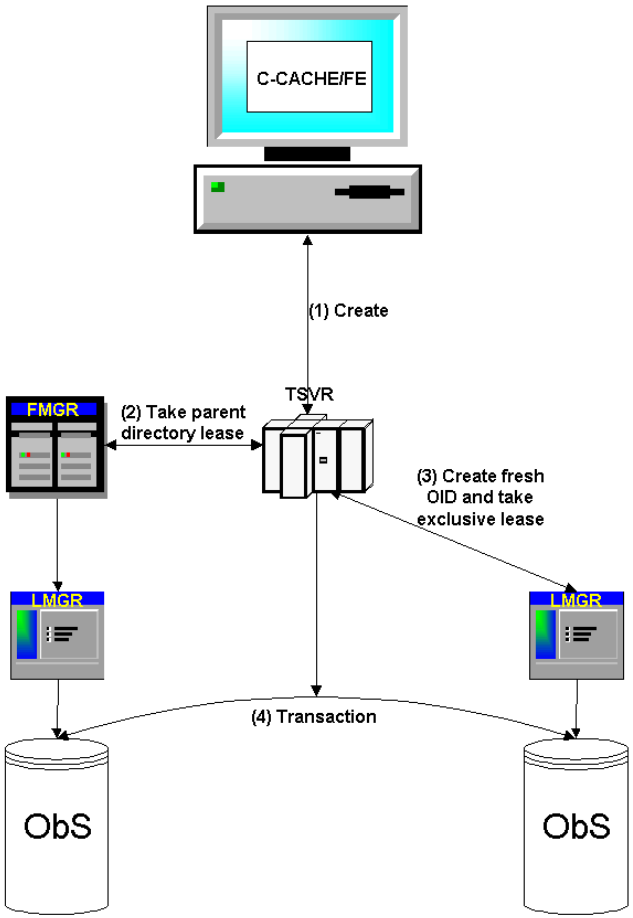
- Wspólna pamięć podręczna
  zFS używa pamięci połączonych komponentów jako wspólnej, globalnej pamięci podręcznej.
- Dodanie dodatkowych maszyn do sieci ma prowadzić do nieomal liniowego wzrostu wydajności całego systemu.

Komponenty (budowa zFS)

zFS jest podzielony na 6 komponentów:

- Object Disk (inaczej: Object Store Device).
- Front End.
- Lease Manager.
- File Manager.
- Cooperative Cache.
- Transaction Server.

Przykładowy scenariusz - utworzenie nowego pliku



Object Disk (ObS)

- Przechowuje zawartość plików i folderów.
- Udostępnia API tworzenia, usuwania, pisania oraz czytania z plików.
- Zapewnia bezpieczeństwo, spójne zapisy do plików, etc.

Front End (FE)

FE jest uruchomiony na każdym stanowisku, na którym klient chce korzystać z zFS. FE udostępnia API systemu plików oraz dostęp do plików i folderów.

Lease Manager (LMGR)

- ”Dzierżawa” (lease) - blokada zasobu na pewien okres czasu.
- Dla każdego ObS-a przyporządkowany jest jeden LMGR.
- Po uzyskaniu ”major-lease”, LMGR zarządza przydzielaniem zasobów ObS-a klientom.
- LMGR pamięta listę klientów korzystających z ObS-a.

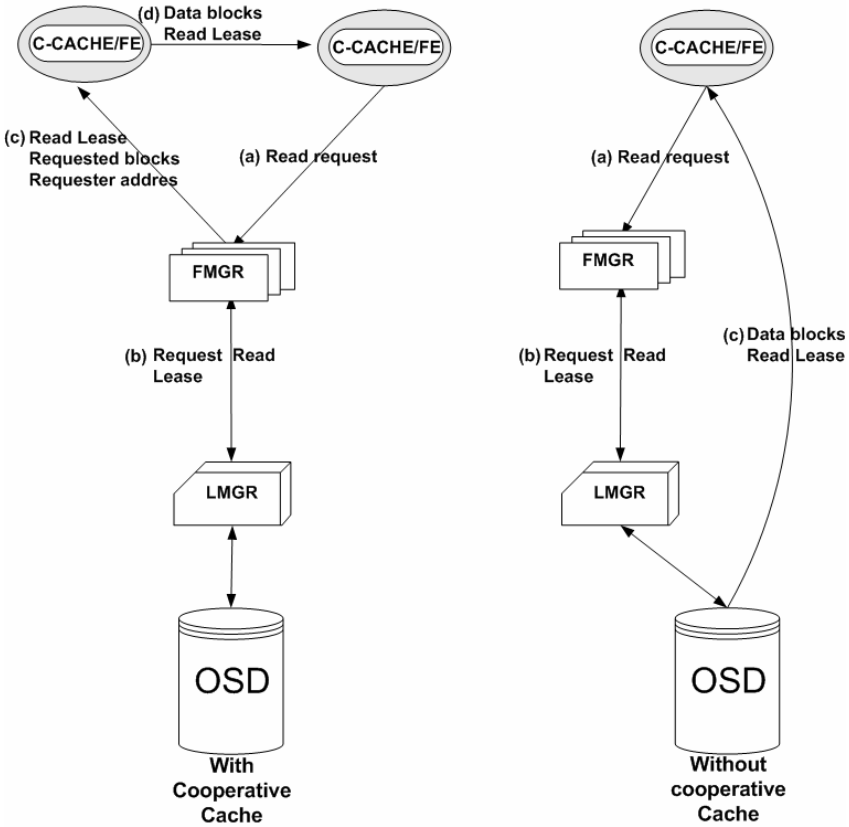
File Manager (FMGR)

- Każdemu otwartemu plikowi w zFS przyporządkowany jest jeden FMGR.
- FMGR pamięta wszystkie wykonane na pliku operacje - read, write, etc.
- FMGR przechowuje informacje o wewnętrznej strukturze pliku (czyli, w którym miejscu na ObS znajdują się części pliku).

Cooperative Cache (Cache)

Komponent zapewniający wysoką skalowalność systemu.
Wzrost szybkości sieci prowadzi do tego, że sprowadzenie danych z dysku lokalnego jest wolniejsze niż z pamięci innej maszyny.

Pobranie bloku danych



Transaction Server (TSVR)

W zFS, operacje na katalogach zrealizowane są jako rozproszone transakcje. W przedstawionym wcześniej przykładowym scenariuszu (tworzenie pliku), błąd (prowadzący do niespójności systemu) mógł wystąpić w 3 miejscach:

- Dodanie nowego obiektu pliku.
- Dodanie pliku do katalogu.
- Komputer inicjujący operację także mógł ulec awarii.

W przypadku błędu, system może być później odtworzony - albo przez ponowne wykonanie transakcji, albo przez jej cofnięcie.

Podsumowanie

Projektanci zFS oczekują, że budowa systemu z powyższych komponentów zapewni dużą wydajność i skalowalność, w oparciu o następujące cechy systemu:

- Rozdzielenie przechowywania danych od zarządzania plikami. Buforowanie i zarządzanie metadanymi odbywa się na innej maszynie niż ObS, w którym zapisana jest właściwa zawartość plików.
- Wspólna pamięć podręczna.
  Klienci otrzymują (szybszy) dostęp do danych poprzez pamięć podręczną innych klientów, tym samym redukując pracę wykonywaną przez ObS-y.

Podsumowanie (2)

- Brak maszyn dedykowanych.
  Dowolna z maszyn używających systemu, może uruchamiać FMGR ("file-manager"), lub LMGR ("lease-manager"). Dzięki temu, dowolna maszyna może:
  - Automatycznie otrzymać wyłączny dostęp do pliku/katalogu.
  - Przejąć rolę maszyny, która uległa awari.

RedHat Global File System

System plików przygotowywany przez Red Hat na potrzeby flagowej wersji dystrybucji - Enterprise Linux, skonstruowany dla potrzeb wysokodostępnych i wysokoskalowalnych serwerów biznesowych.

Główne cechy

- Klastrowy system plików (lub też współdzielony system plików)
- Pozwala wielu węzłom klastra używać współbieżnie jednego współdzielonego urządzenia pamięci masowej (zazwyczaj SAN - sieci pamięci masowej - Storage Area Network)
- Zapewnia pełną spójność danych
- Dosyć dobrze skalowalny - daje się stosować przy ponad 100 węzłach
- Dostarczany z RedHat Enterprise Linux, ale sam projekt powstaje na licencji GPL

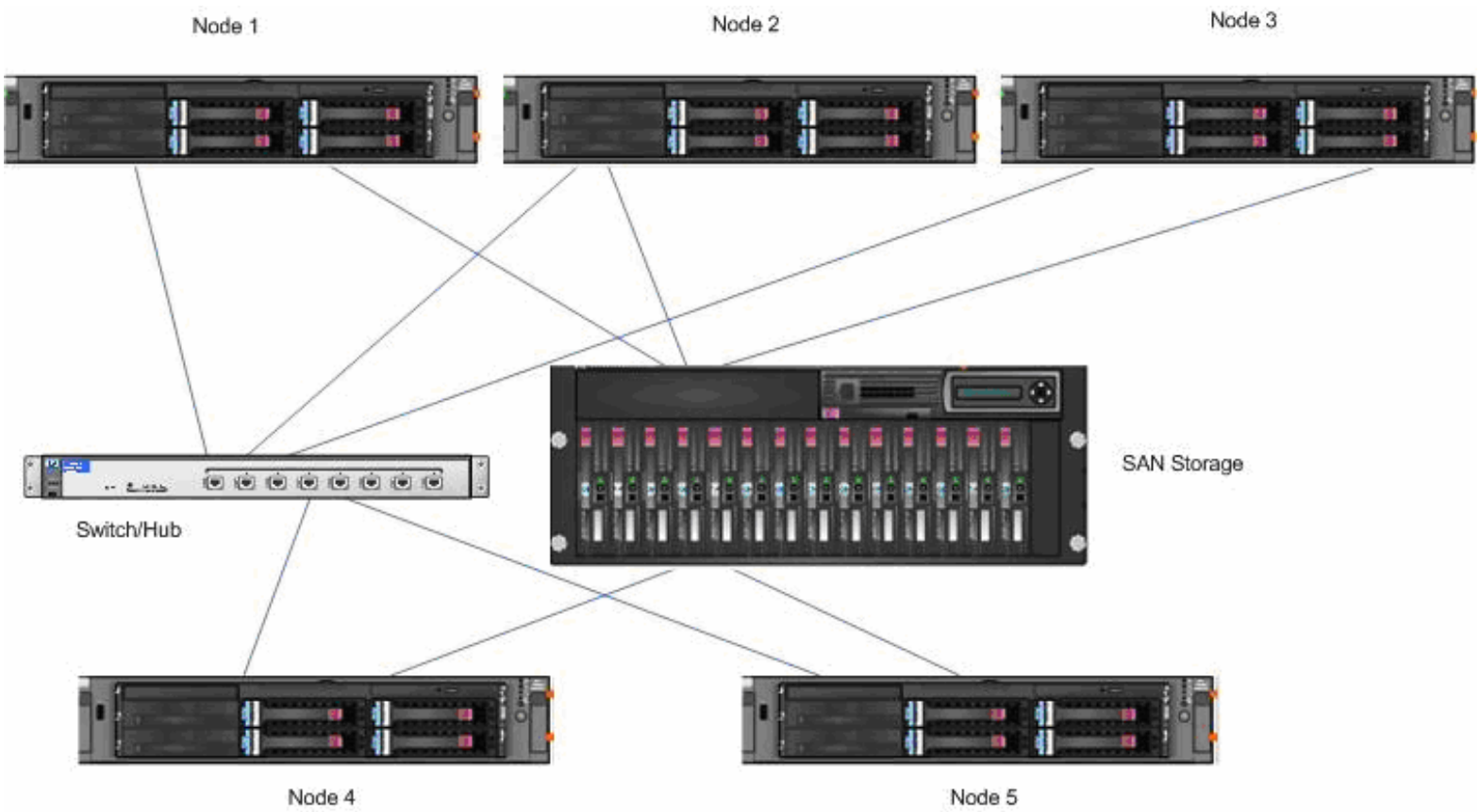
Budowa i działanie

- GFS używa urządzenia blokowego jak zwykły lokalny system plików
- Dodatkowo dostarcza opcjonalne moduły blokujące, pozwalające sterować współbieżnością wykonywanych operacji:
  - moduł GULM - za blokady odpowiada jeden lub wiecej dedykowanych węzłów lub serwerów zewnetrznych
  - moduł DLM - za blokady odpowiadają wszystkie węzły w klastrze

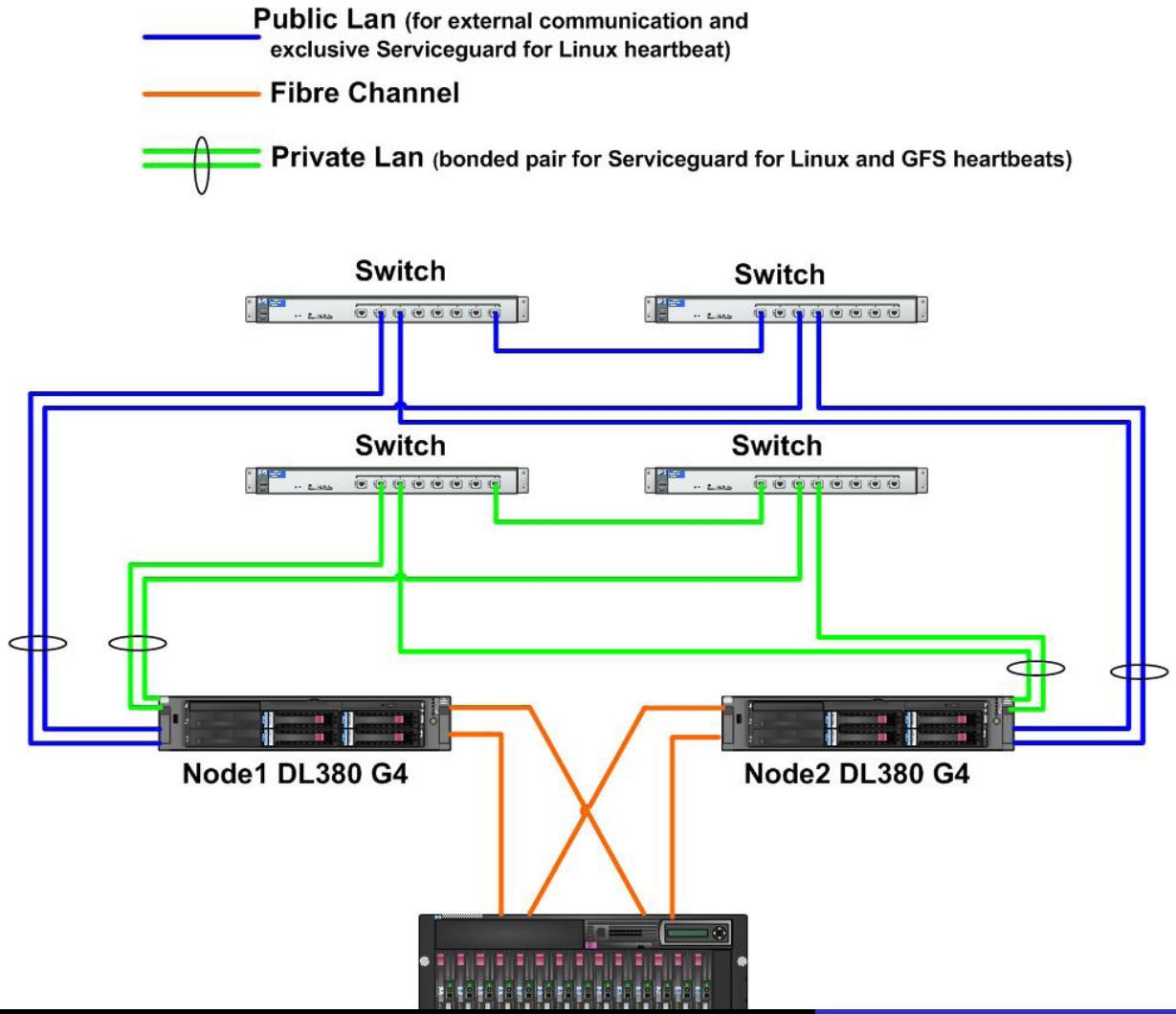
Budowa i działanie cd.

- Działa w oparciu o specyficzny sprzęt - wielowejściowe urządzenie blokowe połączone ze wszystkimi węzłami klastra szybkimi łączami, zazwyczaj Fibre Channel
- To sprzęt realizuje szeregowanie pojedyńczych operacji zapisu/odczytu

Przykładowa konfiguracja sprzętowa



Schemat połączeń sieciowych



Zalety

- Bezpośredni dostęp każdego węzła do urządzenia blokowego zapewnia dużą wydajność
- Łatwa skalowalność (dodawanie i usuwanie węzłów oraz przestrzeni dyskowej w locie)
- Realizacja całkowicie na poziomie jądra linuksa - większa wydajność
- Natychmiastowe uaktualnianie zawartości dysku - pełna spójność danych
- Pełna zgodność z POSIX

Zalety cd.

- Współbieżny dostęp do danych
- Łatwe zapewnianie bezpieczeństwa danych - wystarczy odpowiednio skonfigurowac sprzęt aby zapewniał redundancję danych
- Łatwość backupowania - jest tylko jedna wersja danych i znajduje sie w całości w jednym miejscu
- Duża tolerancja na awarie węzłów

Osiągnięte cele

- Przezroczystość dostępu
- Niezależność położenia
- Przezroczystość wydajności
- Przezroczystość współbieżności (z użyciem opcjonalnych modułów)
- Przezroczystość wędrówki
- Skalowalność - spora ale da się lepiej

Wady

- Ze względu na sposób budowy wymaga drogiego specjalistycznego sprzętu i umieszczenia go fizycznie w niewielkiej przestrzeni
- Dane wszystkich węzłów znajdują się w jednym miejscu - brak tolerancji awarii sieci pamięci masowej
  - można się przed tym ustrzec duplikując SAN (duży koszt)

Niezrealizowane postulaty

- Przezroczystość zwielokrotniania
- Przezroczystość awarii (częściowo)

Historia Google FS

System plików został zaprojektowany przez Google dla specyficznych potrzeb firmy związanych z przechowywaniem danych będących wynikiem przeszukiwania Internetu.
Google FS wyrósł z systemu “BigFiles”, który powstał jeszcze na uniwersytecie *Stanford* na początku istnienia Google.
Google File System nie jest ogólnodostępny.

Założenia Google FS

Projektując Google FS jego autorzy poczynili kilka założeń:

1. System działa na ogólnodostępnych komponentach, które ulegają awariom
2. W systemie przechowywane jest głównie mała ilość dużych plików (min. 100 MB)
3. Obciążenie to głównie:
   - duże odczyty strumieniowe (min. kilkaset KB, zwykle więcej niż 1 MB)
   - małe losowe odczyty (kilka KB)
   - duże, sekwencyjne zapisy dopisujące dane do plików (dane są rzadko modyfikowane)
4. System dobrze implementuje dopisywanie do jednego pliku przez kilku klientów
5. Duża przepustowość jest ważniejsza niż opóźnienia.

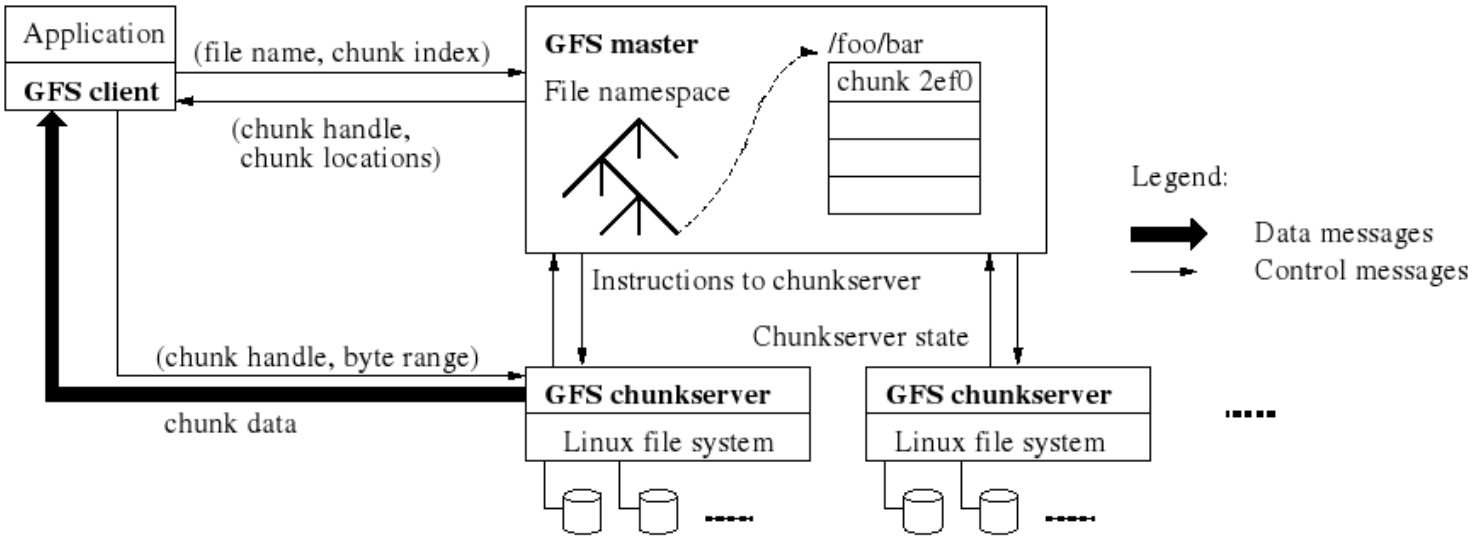
Operacje na plikach

Pliki w systemie Google FS są zorganizowane w katalogi. Google FS obsługuje następujące operacje na plikach:

- create
- delete
- open
- close
- read
- write
- snapshot
- record append

Architektura

Architektura Google FS opiera się na podziale jednego klastra Google FS na jednen komputer *master* i wiele *chunkservers*. Do takiego klastra odwołuje się jednocześnie wielu klientów.



Architektura Google FS

Architektura
Master server

Master server przechowuje wszystkie metadane:

- namespaces
- access control information
- mapping from files to chunks
- locations of chunks

Master server okresowo komunikuje się z *chunkservers* przy pomocy komunikatów *HeartBeat* w celu przekazania im instrukcji i sprawdzenia ich statusu.

Architektura
Chunkservers

Chunkservers to komputery PC działające pod systemem Linux. Przechowują one pliki w postaci fragmentów danych (ang. *chunks*) o stałej wielkości 64 MB, które identyfikowane są przez niezmienne i globalnie unikalne 64 bitowy *chunk handles* nadawane przez *master server* podczas tworzenia każdego fragmentu.
Dla zapewnienia niezawodności każdy *chunk* istnieje na różnych *chunkserverach*.

Uwaga

Klienci nie wykorzystują *master servera* do pisania lub czytania danych aby nie wyczerpać jego przepustowości.

Przechowywane dane

Chunks

Zalety 64 MB:

- zmniejsza ilość komunikacji między klientem a masterem
- umożliwia zmniejszenie narzutu (*overhead*) na sieciowy transfer danych
- zmiejsza ilość metadanych przechowywanych na masterze

Wady:

- w przypadku gdy mały plik jest odczytywany przez wielu klientów obciążenie dysku może być znaczne — np. gdy plik jest uruchamialny (taki problem częściowo rozwiązano replikując taki plik na większej ilości serwerów)

Architektura
Metadane I

Metadane

1. In-Memory Data Structures
2. Chunk Locations
3. Operation Log

Wszystkie metadane są przechowywane w pamięci RAM. Daje to szybkie operacje na masterze oraz możliwość regularnego skanowania danych w pamięci i aktualizowania ich stanu.

In-Memory Data Structures

Master server przechowuje mniej niż 64 bajty metadanych na każde 64 MB danych. Również ścieżki plików zajmują zwykle mniej niż 64 bajty dzięki kompresji prefiksów.

Architektura
Metadane II

Chunk Locations

Master kontroluje położenie poszczególnych fragmentów danych przy pomocy regularnie wysyłanych komunikatów *HeartBeat*. *Chunkserver* rozstrzyga jakie posiada fragmenty a jakie nie (np. bo awaria dysku zniszczyła dane).

Operation Log

Krytyczne metadane posiadające kopie zapasowe na zdalnych serwerach. Operacje klientów potwierdzane są tylko po zsynchronizowaniu logu (lokalnym i zdalnym). Master odtwarza swój stan na podstawie tego logu. Aby zapobiec rośnięciu logu co pewien czas tworzony jest *checkpoint*.

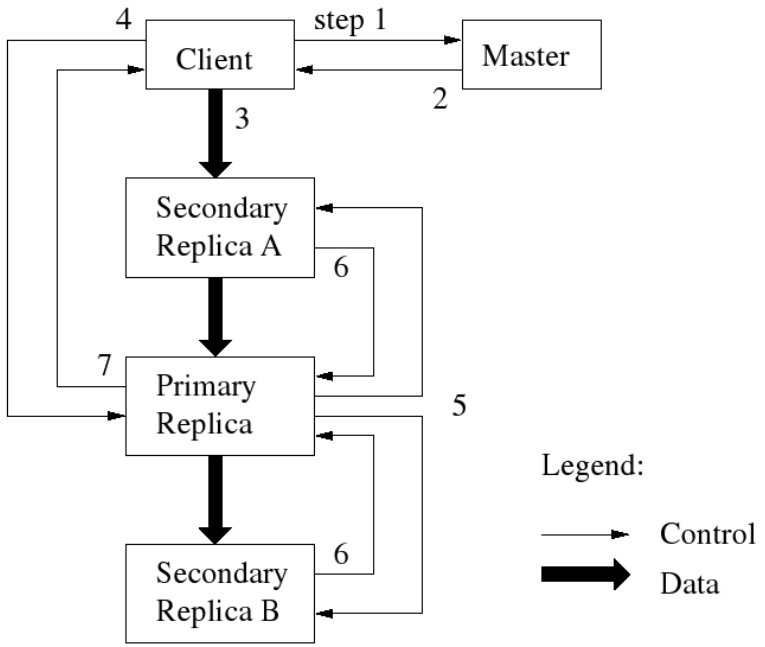
Spójność danych w Google FS

Dane w Google FS mogą być: "spójne" (jeśli wszyscy klienci widzą te same dane) oraz "zdefiniowane" (jeśli po zmianie danych w pliku, dane są spójne oraz wszyscy klienci widzą całą zmianę danych).
Google FS gwarantuje, że zmiany w przestrzeni nazw plików (np. tworzenie pliku) są atomowe.
Aby zmienić dane potrzebna jest "dzierżawa" (*lease*) *Chunkserver* mający dzierżawę na dany fragment danych to tzw. *primary* — określa w jakiej kolejności zajdą zmiany w danych.

Implementacja operacji Google FS

Kontrola zapisu i przepływ danych

1. Klient dowiaduje się od mastera jaki *chunkserver* jest *primary* dla danego fragmentu.
2. Dane po przekazaniu od klienta do wszystkich *chunkserverów* są modyfikowane na polecenie *primary*.
3. *Primary* odpowiada klientowi.



Implementacja operacji Google FS

Record Append

W przeciwieństwie do standardowej operacji zapisu wykonując *record append* klient określa jedynie dane, a nie ofset gdzie mają one być zapisane.
W *record append* dane są dopisywane jednorazowo w sposób atomowy pod ofsetem wybranym przez Google FS i który to ofset jest zwracany do klienta.

1. Dane są rozsyłane do *chunkserverów* mających ostatni fragment pliku.
2. *Primary* sprawdza czy dane mogą być dodane do ostatniego fragmentu.
3. *Primary* wydaje polecenia i odpowiada klientowi.

Implementacja operacji Google FS

Snapshot

Operacja *snapshot* tworzy kopię pliku lub drzewa katalogów prawie natychmiastowo bez przerywania trwających innych operacji w tym drzewie.
Do zaimplementowania *snapshot* użyta jest technika *copy-on-write*.

1. Po otrzymaniu informacji o operacji *snapshot*, *master server* odbiera wszystkie "dzierżawy" w danym drzewie katalogów.
2. Każdy nowy klient chcący tam pisać musi otrzymać "dzierżawę" co umożliwia masterowi wstrzymanie operacji.
3. Każdy *chunkserver* na polecenie mastera ma stworzyć nowy fragment i zrobić lokalną kopię z podanego przez mastera fragmentu.

Implementacja operacji Google FS

Usuwanie plików

Po skasowaniu pliku miejsce w klastrze nie jest zwalniane — zmieniana jest jedynie nazwa pliku na ukrytą wraz z datą jego ukrycia.
Po 3 dniach *master server* usuwa z metadanych wszelkie odniesienia do usuniętego pliku. Wraz z następnym komunikatem *HeartBeat* do *chunkserverów* są one informowane o możliwości usunięcia wszystkich fragmetów pliku.

Obsługa fragmentów

Tworzenie replik fragmentów danych

Creation

Podczas tworzenia nowego fragmentu *master server* decyduje gdzie nalezy umieścić repliki kierując się kilkoma wytycznymi dotyczącymi *chunkserverów*: wykorzystania przestrzeni na dyskach, ilości tworzonych replik/fragmentów, rozproszenie fragmentów.

Re-replication

Repliki są tworzone gdy ich ilość dla danego fragmentu spadnie poniżej ustalonego poziomu. Do klonowania wybierane są fragmenty o największym priorytecie.

Rebalancing

Master server sprawdza jak wykorzystywane są zasoby i może

Obsługa błędów

Ominięcie zmiany

Jeśli *chunkserver* nie działał w momencie wykonywania zmiany w jednym z posiadanych przez niego framentów to taki fragment będzie nieaktualny. Aby zapobiec takim sytuacjom każdy fragment posiada numer wersji zwiększany przy każdej zmianie.

Metody stabilizowania systemu

1. Fast Recovery
2. Replication
   - Chunk Replication
   - Master Replication

Spis treści

Wstęp

- Implementacja rozproszonego sieciowego systemu plików *Andrew File System* stworzonego w Carnegie Mellon University.
- Miał wpływ na NFS 4 oraz wywodzi się z niego sieciowy DFS o nazwie Coda.
- Istnieją wdrożenia AFS z 50k klientów.

Logo OpenAFS

Cechy OpenAFS

- *AFS* = moduły jądra (*Cache Manager* i rozszerzenie *VFS*) + narzędzia
- serwery są stanowe
- klient może być jednocześnie serwerem
- zrzucenie możliwie dużej części obliczeń na klienta
- stworzony głównie z myślą o sieciach WAN
- do komunikacji wykorzystywany jest protokół *Rx* - odmiana *RPC* stworzona specjalnie dla *AFS*

Cechy OpenAFS - *komórka*

- pojęcie komórki (*cell*): kolekcja serwerów *AFS*, logiczne zgrupowanie
  - każda komórka prezentuje spójny system plików na podstawie systemów eksportowanych przez serwery wchodzące w jej skład
  - odpowiednik pojęcia domeny

Cechy OpenAFS, cd.

- przezroczystość położenia
  - schemat ścieżki: */afs/[nazwa komórki]/...*
- niezależność położenia

Cechy OpenAFS - bezpieczeństwo

- bezpieczeństwo: *Kerberos* do uwierzytelniania, *ACL* do praw dostępu
  - symetryczne uwierzytelnianie: tożsamość przedstawia zarówno klient jak i serwer usługi
  - każdy użytkownik ma oddzielne konto w *AFS*
  - *ACL* przydzielane tylko dla katalogów
    - dowiązania twarde (*hard links*) w *AFS* są złym pomysłem
    - przeniesienie pliku zmieni jego prawa dostępu
    - ale chmod działa poprawnie, bo jest wychwytywany z poziomu jądra

Cechy OpenAFS - *replikacja*

- możliwość replikacji do kopii tylko do odczytu
  - klient sam szuka dostępnej kopii nie znając jej położenia a jedynie ścieżkę
  - wybierana jest dostępna kopia położona najbliżej klienta

Cechy OpenAFS - semantyka operacji

- semantyka Uniksa - porządkowanie operacji
- semantyka sesji - modyfikujemy lokalną kopię po czym ew. uaktualniamy fizyczny plik na serwerze
  - *write-on-close-or-fsync*
  - *odwołania* (*callbacks* dla *Cache Managera*)
- transakcje - wszystko albo nic
- buforowanie po stronie klienta (*Cache Manager*)

Organizacja OpenAFS - *woluminy*

- organizacja za pomocą *woluminów*
  - tworzą je administratorzy
  - mają przydzielone ścieżki
  - mogą zawierać pliki, katalogi i odwołania do innych woluminów
  - mogą mieć limity, np. na maksymalną objętość, liczbę plików, itp.
  - po udostępnieniu są widoczne u klientów, można z nich korzystać jak z lokalnego systemu plików

Organizacja OpenAFS, cd.

- korzeniem *AFS* jest */afs*
  - to jest właśnie podział na wspólną (*/afs*) i lokalną (reszta) przestrzeń nazw
  - wspólna przestrzeń nazw dla wszystkich klientów
    - oczywiście z dokładnością do praw dostępu *ACL*
  - pod nim znajdują się drzewa katalogów eksportowane przez komórki

Organizacja OpenAFS, cd.

- każde poddrzewo w */afs* jest przypisane do jednego serwera - *nadzorcy*
- podczas lokalnego buforowania pliku, informowany jest o tym nadzorca
- przy modyfikacji współdzielonego pliku przez klienta, nadzorca informuje pozostałych klientów mających lokalną kopię pliku o jego zmianie

Organizacja OpenAFS - *blokady*

- można zakładać blokady (*locks*) jedynie na całe pliki
  - w lokalnym systemie semantyka pozostaje bez zmian
  - organizacja blokad odbywa się na podobnej zasadzie co organizacja lokalnych kopii plików
    - system jest scentralizowany względem *nadzorcy* żądanego zasobu
    - klient zdalny zgłasza się do *nadzorcy* z prośbą o blokadę
    - *nadzorca* jest stroną rozstrzygającą
    - odpowiada: *OK* lub *EWOULDBLOCK*