Google File System

Janina Mincer-Daszkiewicz

Systemy rozproszone

MSUI, II rok

Materiały i rysunki zaczerpnięto z następujących źródeł

S. Ghemawat, H. Gobioff, Sun-Tak Leung, The Google File System, firma Google, SOSP 2003

http://students.mimuw.edu.pl/SR-MSUI/02-googlefs/gfs-sosp2003.pdf

R. Harris, Google File System Eval: Part I i Part II.

http://storagemojo.com/google-file-system-eval-part-i/ i

http://storagemojo.com/google-file-system-eval-part-ii/

Google FS z lotu ptaka ...

- Produkt firmy Google™
- Skalowalny, rozproszony system plików, dla dużych rozproszonych aplikacji, przetwarzających duże ilości danych (zgadnij jakich ;)
- Zapewnia odporność na błędy (ang. *fault tolerance*)
- Działa dobrze na tanim, standardowym sprzęcie
- Zapewnia dobrą wydajność dużej liczbie klientów
- Wykorzystywany przez firmę Google jako podstawowa platforma pamięci zewnętrznej w usługach i badaniach
- Największa instalacja: setki terabajtów pamięci, tysiące dysków, setki klientów

Cele projektowe

- Cele projektowe zbieżne z celami poprzednich projektów
  - Wydajność
  - Skalowalność
  - Niezawodność
  - Dostępność
- Nowe cele projektowe – wynikające ze specyficznych potrzeb aplikacji klienckich i specyficznego środowiska technicznego
  - Założenie o częstych awariach elementów składowych systemu

    Jest ich bardzo dużo, sprzęt jest średniej klasy
  - Musi zawierać rutynowe mechanizmy wspierające: monitorowanie, wykrywanie błędów, odporność na błędy, automatyczne odtwarzanie po awarii
  - Przeznaczony dla OLBRZYMICH plików

    System ma obsługiwać efektywnie miliony plików o typowym rozmiarze 100 MB, częstym przypadkiem będą też pliki wielkości wielu GB, należy wspierać małe pliki, ale bez szczególnych względów

Cele projektowe – cd

- Nowe cele projektowe – cd
  - Optymalizowany dla niestandardowych wzorców dostępu

    Odczyty: duże sekwencyjne odczyty tego samego pliku przez tego samego klienta (pojedyncza operacja rzędu setek KB, 1 MB lub więcej) i małe losowe odczyty (kilka KB w losowym miejscu)

    Zapisy: dopisywanie na końcu pliku, dużymi porcjami (jak przy odczytach), a nie nadpisywanie istniejących danych, po utworzeniu plik jest rzadko modyfikowany, małe losowe zapisy mogą występować, ale nie trzeba ich szczególnie wspierać

    (obsługa pamięci podręcznych po stronie klienta traci swoje znaczenie)
  - **API** systemu plików musi wspierać te cele projektowe i współgrać z potrzebami aplikacji

    wsparcie dla wielu klientów dopisujących atomowo na końcu pliku, z minimalnym narzutem na synchronizację
  - Ważniejsza je duża **szerokość pasma** (ang. *bandwidth*) niż **niskie opóźnienie** (ang. *latency*). Więcej aplikacji wymaga wysokiego tempa obsługi dużych ilości danych niż krótkich czasów reakcji przy indywidualnych odczytach i zapisach

API

- Nie implementuje interfejsu POSIX
- Pliki są zorganizowane hierarchicznie w katalogi i identyfikowane nazwami ścieżkowymi
- Standardowe operacje: *create*, *delete*, *open*, *close*, *read*, *write*
- Nowe operacje:
  - ***snapshot*** - tworzy (niskim kosztem) kopię pliku lub drzewa katalogu
  - ***record append*** – atomowe dopisanie na końcu pliku, dla współbieżnie działających procesów, bez dodatkowej synchronizacji (prościej dla aplikacji!)

Architektura

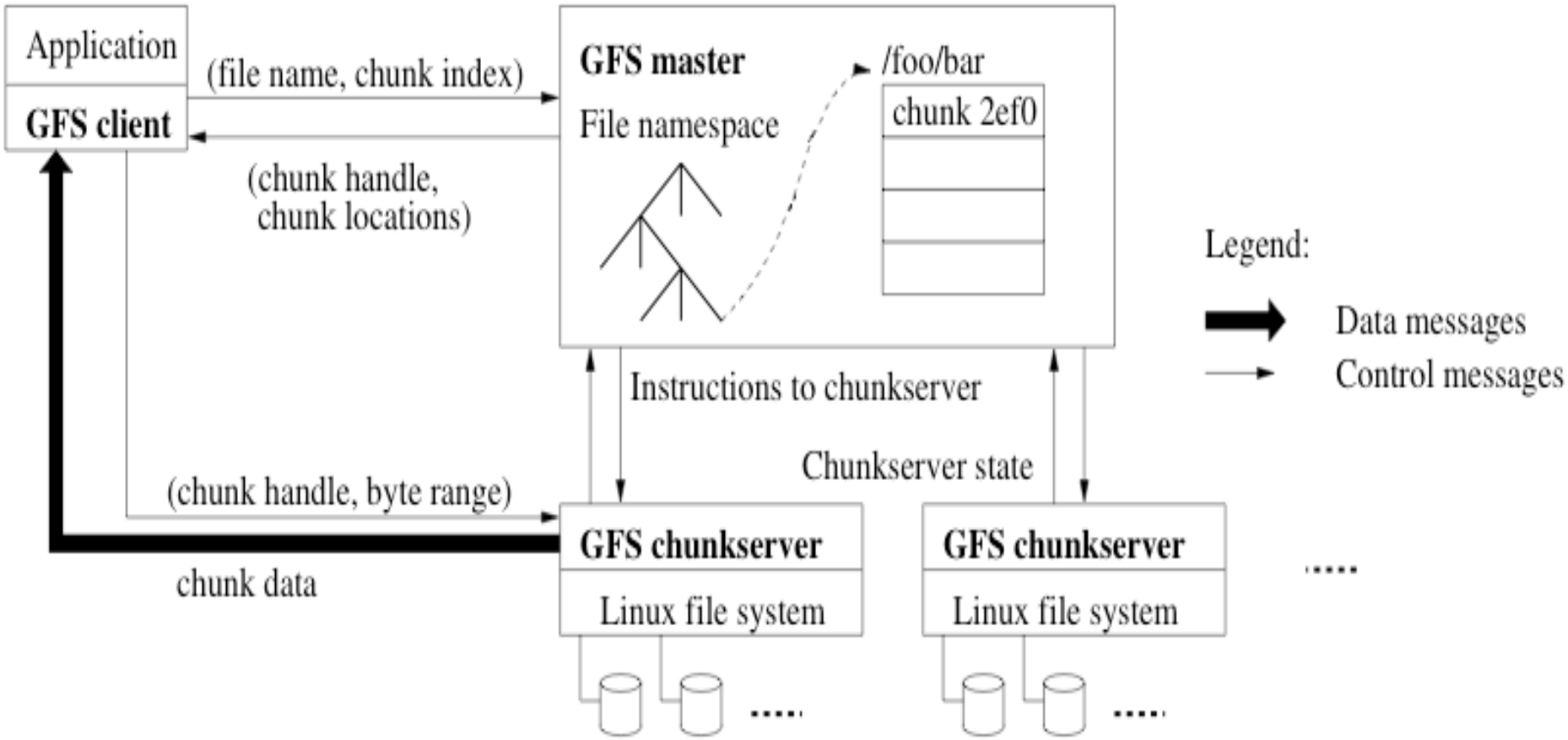


Figure 1: GFS Architecture

Architektura - cd

- Klaster GFS składa się z jednego **zarządcy** (ang. *master*) i wielu **serwerów kawałków plików** (ang. *chunkserver*) – zwykle to maszyna Linuksowa z procesem pełniącym rolę serwera, wykonywanym na poziomie użytkownika
- Zarządca i serwery plików mogą być usytuowane na tej samej maszynie fizycznej
- Pliku są dzielone na kawałki stałego rozmiaru, identyfikowane 64-bitowym uchwytem, który jest niezmienny i globalnie unikatowy
- Serwer plików zapisuje te kawałki na lokalnych dyskach jako linuksowe pliki

Architektura - cd

- Kawałki są replikowane na wielu serwerach (domyślnie trzech)
- Zarządca zarządza metadanymi plików: przestrzenią nazw, prawami dostępu, odwzorowaniem plików w kawałki, położeniem kawałków na serwerach
- Ponadto zarządca steruje innymi zadaniami centralnymi, jak migrowanie kawałków między serwerami, odśmiecaniem. Cyklicznie wysyła do serwerów komunikaty kontrolne typu ***heartbeat***, przekazując instrukcje i zbierając informacje o stanie

Architektura - cd

- Kod klienta GFS, dołączany do aplikacji, implementuje GFS API i w imieniu aplikacji komunikuje się z zarządcą i serwerami plików
- Klient komunikuje się z zarządcą w celu wykonania operacji na metadanych, wszelkie transmisje danych są wykonywane bezpośrednio między klientem i serwerami plików
- Ani klient, ani serwery plików nie buforują danych. W przypadku klienta obszary robocze plików są zbyt duże by pomieściły się w pamięci podręcznej. Klient buforuje natomiast metadane. Serwery nie buforują danych, ponieważ kawałki są plikami w lokalnym systemie plików, które są już buforowane przez Linux

Projekt - zarządca

- Posiadanie pojedynczego zarządcy ułatwia podejmowanie globalnych decyzji, ale nie może on stać się wąskim gardłem
- Przykład – obsługa pojedynczego odczytu:
  - znając stały rozmiar kawałka, klient tłumaczy nazwę pliku i przesunięcie w indeks kawałka
  - wysyła do zarządcy żądanie podając nazwę i ten indeks
  - zarządca odpowiada podając uchwyt i miejsce położenia kawałków
  - klient buforuje tę informację
  - klient wysyła żądanie do jednej z replik podając uchwyt i zakres danych

Projekt – zarządca cd

- Późniejsze odczyty tego samego kawałka nie wymagają interakcji z zarządcą dopóki nie wygaśnie ważność buforowanych metadanych lub plik nie zostanie ponownie otwarty
- Klient zwykle pyta o wiele kawałków, a zarządca zwykle dołącza informacje o kolejnych (w sensie położenia w pliku) kawałkach

Projekt – rozmiar kawałka

- Rozmiar kawałka to **64 MB** (dużo więcej niż typowy rozmiar bloku w systemie plików)
- Korzyści
  - rzadsza komunikacja klienta z zarządcą (odczuwalna właśnie przy dużych sekwencyjnych odczytach i zapisach)
  - większa szansa, że klient wykona wiele operacji na tym kawałku poprzez jedno połączenie TCP
  - mniej miejsca zajmują metadane u zarządcy (można je przechowywać w pamięci)
- Wady
  - mały plik składa się z jednego kawałka, przechowujące go serwery mogą być mocno obciążone gdy korzysta z niego wiele klientów. Rozwiązanie: stworzyć więcej replik

Projekt – metadane

- Struktury danych w pamięci
  - wszystkie trzy rodzaje: nazwy plików i kawałków, odwzorowanie plików w kawałki, położenie kawałków
  - szybki dostęp
  - skanowanie w tle w celu implementacji odśmiecania, ponownego tworzenia replik w przypadki awarii serwerów, równoważenie obciążenia poprzez migrację replik
  - ile pamięci potrzeba: **64 bajty metadanych** na każdy kawałek, tyle samo na nazwy plików
- Położenie kawałków
  - Zarządca nie trzyma tej informacji w pamięci trwałej, tylko odpytuje serwery przy starcie oraz kontroluje migracje i cyklicznie monitoruje serwery
  - Nie próbuje się za wszelką cenę utrzymywać spójnego obrazu stanu zawartości serwerów

Projekt – metadane

- Dziennik operacji
  - zawiera zapis krytycznych zmian w metadanych (tylko przestrzenie nazw i odwzorowanie plików w kawałki)
  - stanowi jedyny trwały zapis metadanych
  - definiuje porządek operacji współbieżnych
  - jest replikowany na wielu zdalnych maszynach
  - najpierw wrzuca się nowy rekord dziennika na dysk, lokalnie i zdalnie, a dopiero potem odpowiada klientowi (zarządca skleja kilka rekordów w jeden przed zrzuceniem na dysk)
  - zarządca odtwarza swój stan wykonując po kolei operacje z dziennika. Aby zmniejszyć rozmiar dziennika, po przekroczeniu ustalonego rozmiaru jest robiony (b. efektywnie, ok. 1 min dla segmentu z kilkoma milionami plików) zrzut aktualnego stanu (ang. ***checkpoint***). Zrzut ma postać zwartego B-drzewa, które można bezpośrednio odwzorować do pamięci

Projekt – model spójności

- GFS gwarantuje, że:
  - Zmiany w przestrzeni nazw plików (np. tworzenie pliku) są atomowe (tylko zarządca)
  - Stan fragmentu pliku:
    - ***spójny*** – jeśli wszyscy klienci zawsze widzą te same dane (niezależnie od wyboru repliki), lecz niekoniecznie pochodzące z jednej operacji zapisu (zwykle będą to pomieszane części różnych zapisów)
    - ***zdefiniowany*** – jeśli spójny i klienci widzą całość zapisanych zmian

Projekt – model spójności cd

Stan po zmianie zależy od rodzaju operacji, jej wyniku (sukces czy porażka) oraz przebiegu (czy współbieżne zapisy)

<table>
<tr><th></th><th>Write</th><th>Record Append</th></tr>
<tr><td>Szeregowy
Sukces</td><td>zdefiniowany</td><td rowspan="2">zdefiniowany
pomieszany z
niespójny</td></tr>
<tr><td>Współbieżny
Sukcesy</td><td>spójny ale
niezdefiniowany</td></tr>
<tr><td>Porażka</td><td colspan="2">niespójny</td></tr>
</table>

Projekt – model spójności cd

- **Write** – dane są zapisywane od podanego przez aplikację przesunięcia
- **Record append** – rekord jest dopisywany atomowo **co najmniej raz**, nawet przy współbieżnych zapisach (GFS nie gwarantuje, że kopie będą identyczne). GFS ustala przesunięcie i przekazuje klientowi. Jest to początek zdefiniowanego obszaru zawierającego rekord. GFS może pomiędzy wstawić dopełnienie (ang. ***padding***) lub duplikaty rekordu. Zajmują one obszary uważane za niespójne i zwykle są niewielkie w porównaniu z dużą ilością danych użytkownika.

Projekt – model spójności

- Spójność zapewniona dzięki: wykonywaniu zmian w tej samej kolejności na wszystkich replikach, stosowaniu numeru wersji kawałka do wykrywania nieaktualnych replik, regularne wykrywanie nieaktywnych serwerów
- Aplikacje mogą przystosować się do rozluźnionego modelu spójności przy użyciu kilku prostych technik:
  - polegając raczej na dopisywaniu (append) niż nadpisywaniu (write)
  - robiąc cykliczne zrzuty (checkpointing)
  - stosując sumy kontrolne (samo-walidujące się rekordy). Każdy rekord przygotowany przez pisarza zawiera sumę kontrolną. Czytelnik może dzięki niej rozpoznać i usunąć dopełnienie i duplikaty

Interakcje w systemie – dzierżawa i kolejność zapisów

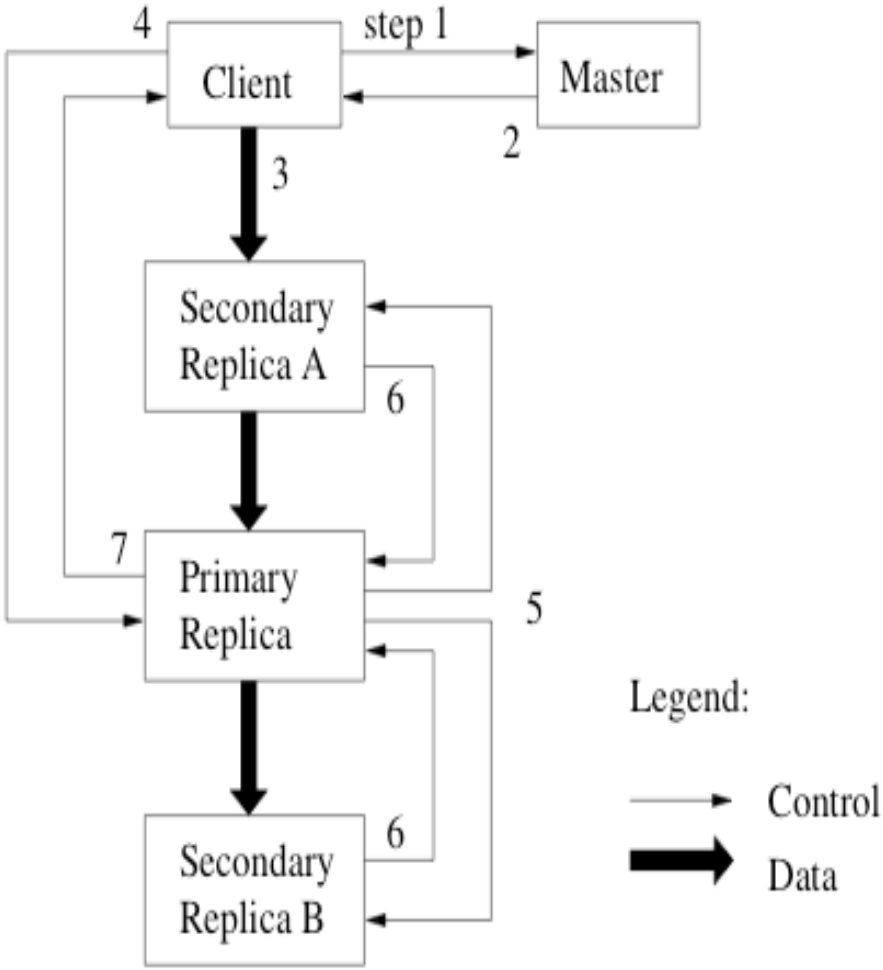


Figure 2: Write Control and Data Flow

1. klient pyta który serwer dzierżawi kawałek X i które serwery mają repliki X
2. serwer przesyła id'y repliki głównej i podrzędnych
3. wysłanie danych do wszystkich replik, w dowolnym porządku
4. wysłanie żądania zapisu do głównego serwera
5. żądanie zapisu przekazane do podrzędnych serwerów
6. podrzędne odpowiadają po zakończeniu zapisu
7. główny odpowiada klientowi

Interakcje – dzierżawa i kolejność zapisów

- **ad. 1** Zarządca przydziela dzierżawę kawałka pliku serwerowi głównemu na pewien okres (zwykle 60 sec); dzierżawę można przedłużyć; jeśli serwer przestanie działać, to dzierżawa sama wygaśnie
- **ad. 2** Klient zapamiętuje przekazane mu dane na potrzeby przyszłych zapisów. Będzie musiał ponownie skontaktować się z zarządcą wtedy, gdy serwer główny stanie się niedostępny lub wygaśnie dzierżawa
- **ad. 3** Przesyłanie danych potokowo, klient wysyła do najbliższego serwera, ten natychmiast do kolejnego najbliższego serwera itp., zgodnie z topologią sieci (pełne wykorzystanie przepustowości pasma w full-duplex Ethernet).

Interakcje – dzierżawa i kolejność zapisów

- **ad. 3 cd.** Serwery podrzędne zapisują dane w lokalnym buforze LRU.
- **ad. 4** Gdy wszystkie repliki potwierdzą otrzymanie danych, klient wysyła do głównego serwera żądanie zapisu. Ten przydziela wszystkim otrzymanym żądaniom zapisu danego kawałka (potencjalnie od różnych klientów) numery seryjne. Wykonuje operacje lokalnie u siebie w tej kolejności.
- **ad. 5** Serwer główny przekazuje podrzędnym żądanie zapisu z numerem seryjnym, wszystkie podrzędne u siebie wykonują operacje w tej zadanej kolejności.
- **ad. 6** Po zakończeniu zapisu podrzędne wysyłają do głównego potwierdzenie.

Interakcje – dzierżawa i kolejność zapisów

- **ad. 7** Serwer główny odpowiada klientowi. Wszelkie błędy w dowolnej replice także są raportowane klientowi. W przypadku błędu operacja mogła się powieść tylko w serwerze głównym i dowolnym podzbiorze pozostałych. W takim przypadku żądanie klienta kończy się porażką i modyfikowany obszar pozostaje w niespójnym stanie. Klient radzi sobie z tym powtarzając kroki 3-7, a w ostateczności powtarzając zapis.

  Jeśli zapis jest duży lub przekracza granice kawałków, to kod klienta GFS dzieli go na kilka operacji zapisu, które mogą zostać przeplecione z operacjami innych klientów (wynikowy obszar będzie w stanie spójnym, ale niezdefiniowanym)

Interakcje – atomowe record append

- Dołączanie na końcu pliku przebiega w krokach 1-7 z niewielką dodatkową logiką w serwerze głównym. Sprawdza on, czy dopisywane bajty zmieszczą się w całości w bieżącym kawałku. Jeśli tak, to działa jak poprzednio, wpp. dopełnia kawałek do maksimum, zleca serwerom podrzędnym zrobienie tego samemu i odpowiada klientowi, że musi on przekazać żądanie do następnego kawałka. Operacja dopisywania może obejmować obszar nie większy niż ¼ kawałka.
- Jeśli operacja kończy się porażką w dowolnej replice, klient powtarza ją. W rezultacie repliki tego samego kawałka mogą zawierać różne dane (duplikaty całości lub części tego samego rekordu).

Zarządca – obsługa przestrzeni nazw

- GFS nie utrzymuje standardowej informacji o katalogu i jego zawartości ani nie wspiera aliasów. Przestrzeń nazw jest logicznie reprezentowana jako tabela odwzorowująca pełne nazwy ścieżkowe w metadane. Dzięki kompresji prefiksowej tabela mieści się w pamięci. Z każdym węzłem w drzewie przestrzeni nazw jest związany **klucz** (ang. *lock*) **odczytu-zapisu**
- Każda operacja wymaga zbioru kluczy, np. podczas operacji na pliku/katalogu /d1/d2 .../dn/liść wymagane są klucze do odczytu dla /d1, /d1/d2, ..., /d1/d2.../dn i klucz do odczytu lub zapisu do /d1/d2 .../dn/liść
- Mechanizm kluczy dopuszcza współbieżne zapisy w tym samym katalogu

Zarządca – obsługa przestrzeni nazw

- **Przykład**: mechanizm kluczy zapobiegnie utworzeniu /home/user/foo w trakcie tworzenia migawki z /home/user w /save/user.
- migawka pobiera klucze do odczytu do /home i /save oraz klucze do zapisu do /home/user i /save/user
- operacja **create** pobiera klucze do odczytu do /home i /home/user oraz klucz do zapisu do /home/user/foo
- operacje są poprawnie serializowane, gdyż ubiegają się o pozostające w konflikcie klucze do /home/user
- **create** NIE WYMAGA klucza do zapisu do katalogu macierzystego (nie ma niczego takiego co by wymagało ochrony)

Zarządca – rozmieszczanie replik

- GFS klaster obejmuje setki serwerów rozmieszczonych pomiędzy różnymi półkami w szafach w serwerowni. Komunikacja pomiędzy dwoma maszynami na różnych półkach może wymagać pokonania jednego lub więcej przełączników (ang. *switch*)
- Strategia rozmieszczania ma na celu: maksymalizację niezawodności i dostępności danych oraz wykorzystania pasma sieci. Rozmieszczając dobiera się zarówno maszyny, jak i półki
- Powinny istnieć repliki kawałka, które przeżyły awarię całej półki, ruch związany z transmisją danych powinien wykorzystywać zagregowane pasmo wielu półek

Zarządca – tworzenie, re-replikacja, równoważenie obciążenia

- Nowe repliki są umieszczane na serwerach ze współczynnikiem wykorzystania dysków poniżej średniej
- Dąży się do ograniczenia liczby ostatnio tworzonych replik na każdym serwerze
- Rozrzuca się repliki pomiędzy półkami w szafie
- Zarządca tworzy nowe repliki, gdy liczba dostępnych replik spada poniżej minimum
- Zarządca cyklicznie bada rozproszenie istniejących replik i zmienia ich położenie realizując cele takie jak podczas tworzenia replik

Zarządca – odśmiecanie

- Gdy plik jest usuwany, GFS zapisuje zmiany w dzienniku, ale nie usuwa od razu pliku, tylko zmienia mu nazwę (na nazwę ukrytego pliku) i pozostawia ze stemplem czasowym z chwili usunięcia.
- Zarządca regularnie przegląda przestrzeń nazw systemu plików i usuwa ukryte pliki starsze niż trzydniowe
- Podczas takiego przeglądu identyfikuje także osierocone kawałki i wymazuje ich metadane
- Każdy serwer w cyklicznie wysyłanych komunikatach **heartbeat** przekazuje listę posiadanych kawałków, a zarządca odpowiada przesyłając dane kawałków, które nie są już dostępne w metadanych zarządcy

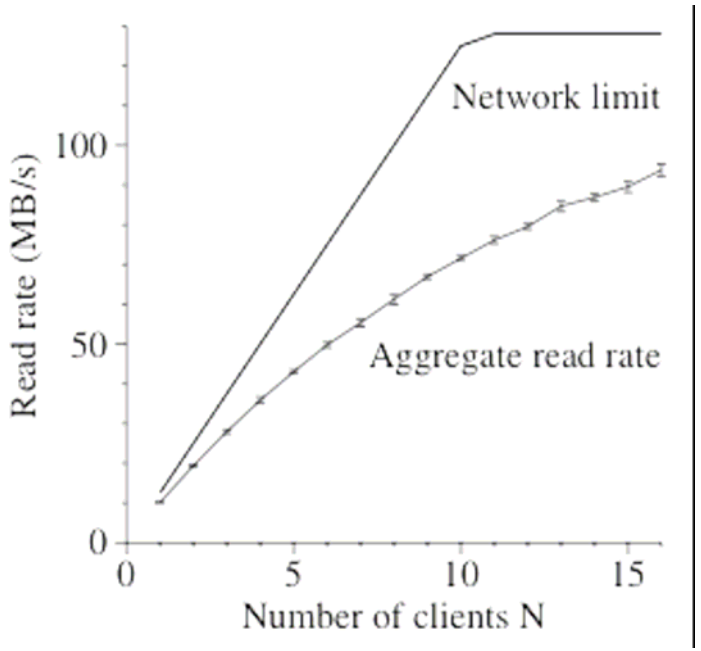
Zarządca – wykrywanie nieaktualnych replik

- Replika staje się nieaktualna, gdy zawierający ją serwer przeoczy jakieś modyfikacje
- Zarządca utrzymuje dla każdego kawałka **numer wersji**
- Kiedy zarządca przydziela nową dzierżawę kawałka, zwiększa numer wersji i przekazuje go aktualnym replikom
- Gdy serwer dołącza do klastra po okresie nieaktywności, przekazuje zarządcy listę swoich kawałków i ich numery wersji
- Zarządca usuwa nieaktualne repliki podczas rutynowego odśmiecania (w międzyczasie traktuje je jakby nie istniały)
- Zarządca przesyłając klientowi listę replik dołącza numer wersji – klient weryfikuje numer komunikując się z repliką

Benchmarki

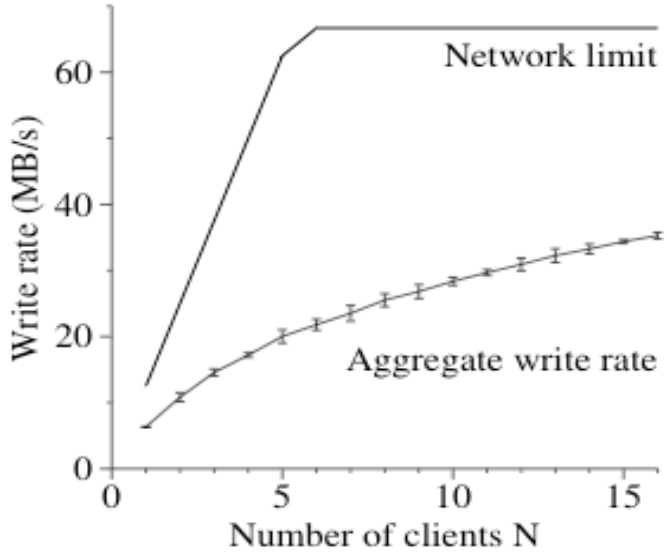
- Testy na małym klastrze: jeden zarządca, dwie kopie zarządcy, 16 serwerów, 16 klientów
- Każda maszyny – dobre PC
- 19 serwerów na jednym przełączniku, 16 klientów na drugim
- 1Gb połączenia między serwerami
- 320 GB danych

- **Test odczytu**
  - Każdy klient czyta losowo 4MB, 256 razy
  - 32 GB RAM, 10% cache hit ratio

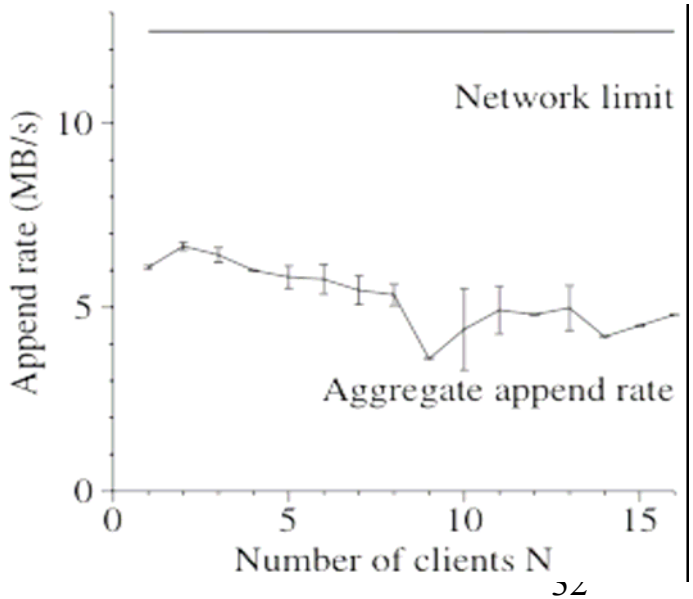


Benchmarki - cd

- **Test zapisu**
  - N klientów pisze do N plików, 1 GB w 1 MB fragmentach
  - Ograniczony przez konieczność zapisywania wszystkiego w 3 kopiach



- **Test dołączania rekordu**
  - N klientów dołącza rekordy do jednego pliku
  - Wydajność ograniczona przez przepustowość sieci w drodze do serwera zawierającego plik



Wyniki testów

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s

GFS - Podsumowanie

- Zalety
  - **Dostępność**. Potrójna redundancja, potokowa replikacja kawałków, szybkie przejmowanie zadania zarządcy przez inną maszynę, inteligentne rozmieszczanie replik, automatyczna re-replikacja, tanie migawki (snapshot). Tę zaletę użytkownicy Google'a obserwują na co dzień.
  - **Wydajność**. Ponad 90% operacji to odczyty. GFS zapewnia b. dobrą wydajność przy dużych, sekwencyjnych odczytach. Google mógł sobie pozwolić na udostępnianie użytkownikom plików video na swoich serwerach. Także: rozdzielenie przepływu danych i sterowania.
  - **Zarządzanie**. System sam radzi sobie z awariami, oferuje automatyczne równoważenie obciążenia, dodatkowo udostępnia migawki, trzyma do 3 dni usunięte pliki.
  - **Koszt**. Nie wymaga żadnych kosztownych elementów sprzętowych.

GFS - Podsumowanie

- Wady
  - Wydajność przy małych losowych odczytach i zapisach nie jest wystarczająca dla aplikacji obsługiwanych w standardowych centrach danych. Brak buforowania danych nie wszędzie się sprawdzi.
  - Operacja **record append** i rozluźniony model spójności nie sprawdzałyby się w standardowych zastosowaniach.
  - GFS nie jest na sprzedaż, nie jest systemem o ogólnie dostępnych źródłach.

Podsumowując: nawet gdyby był na sprzedaż, jest to niszowy produkt, który nie odniósłby sukcesu na otwartym rynku. Jest natomiast znakomitych przykładem tego co MOŻNA osiągnąć dostosowując oprogramowanie (nie sprzęt) do specyficznych potrzeb.