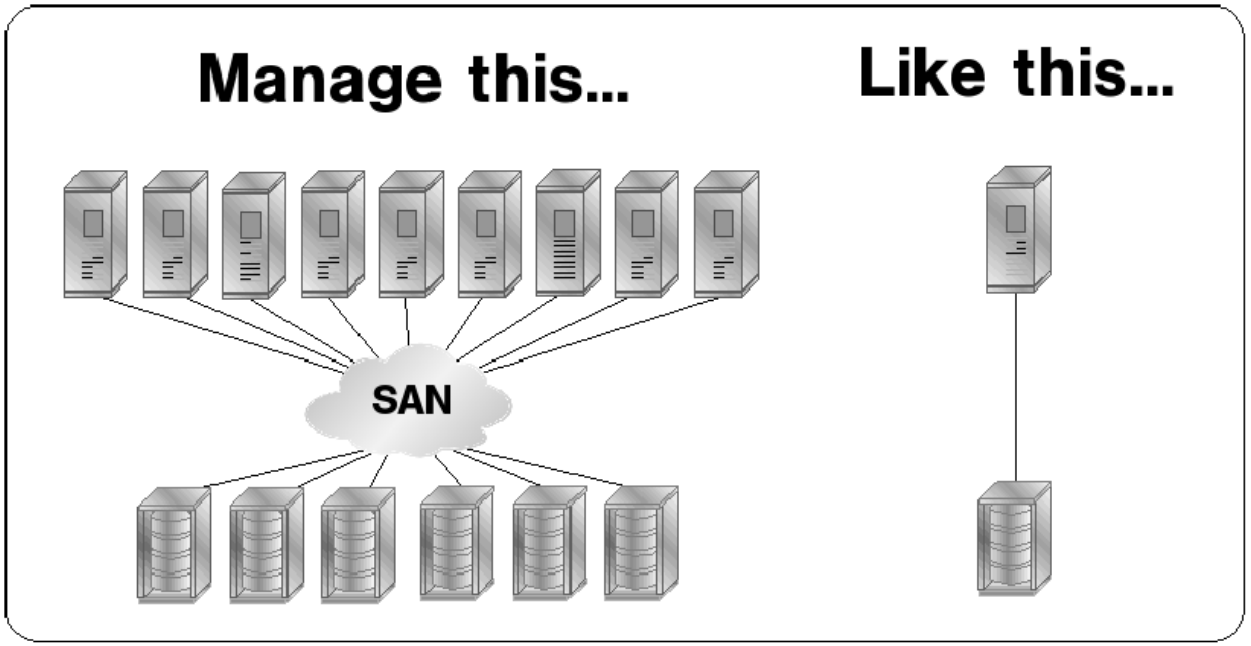
Spis treści

What if you could...



Motywacje użytkowników:

- Pieniądze ...

Motywacje twórców:

Motywacje użytkowników:

- Pieniądze ...
- Czasem jedyne wyjście ...

Motywacje twórców:

Motywacje użytkowników:

- Pieniądze ...
- Czasem jedyne wyjście ...

Motywacje twórców:

- Łatwo i wydajnie skalowalne (najlepiej liniowo)

Motywacje użytkowników:

- Pieniądze ...
- Czasem jedyne wyjście ...

Motywacje twórców:

- Łatwo i wydajnie skalowalne (najlepiej liniowo)
- Odporne na awarie

Motywacje użytkowników:

- Pieniądze ...
- Czasem jedyne wyjście ...

Motywacje twórców:

- Łatwo i wydajnie skalowalne (najlepiej liniowo)
- Odporne na awarie
- Szybkie

Motywacje użytkowników:

- Pieniądze ...
- Czasem jedyne wyjście ...

Motywacje twórców:

- Łatwo i wydajnie skalowalne (najlepiej liniowo)
- Odporne na awarie
- Szybkie
- Łatwe w użytkowaniu

Do czego to się może przydać

- Duże bazy danych

Do czego to się może przydać

- Duże bazy danych
- Systemy wymagające skalowalności

Do czego to się może przydać

- Duże bazy danych
- Systemy wymagające skalowalności
- Systemy wymagające dużej niezawodności

Do czego to się może przydać

- Duże bazy danych
- Systemy wymagające skalowalności
- Systemy wymagające dużej niezawodności
- Ułatwienie administracji

Do czego to się może przydać

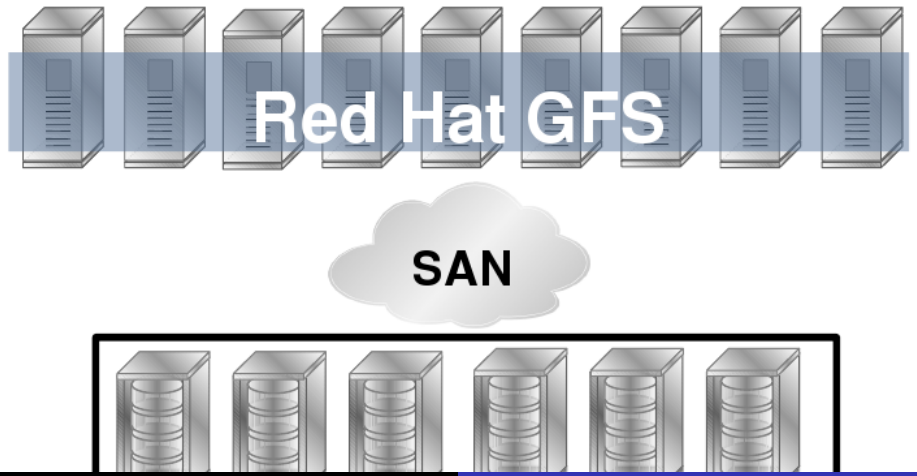
- Duże bazy danych
- Systemy wymagające skalowalności
- Systemy wymagające dużej niezawodności
- Ułatwienie administracji

Jest to obecnie rynek wart miliardy dolarów i ta liczba **bardzo** szybko rośnie

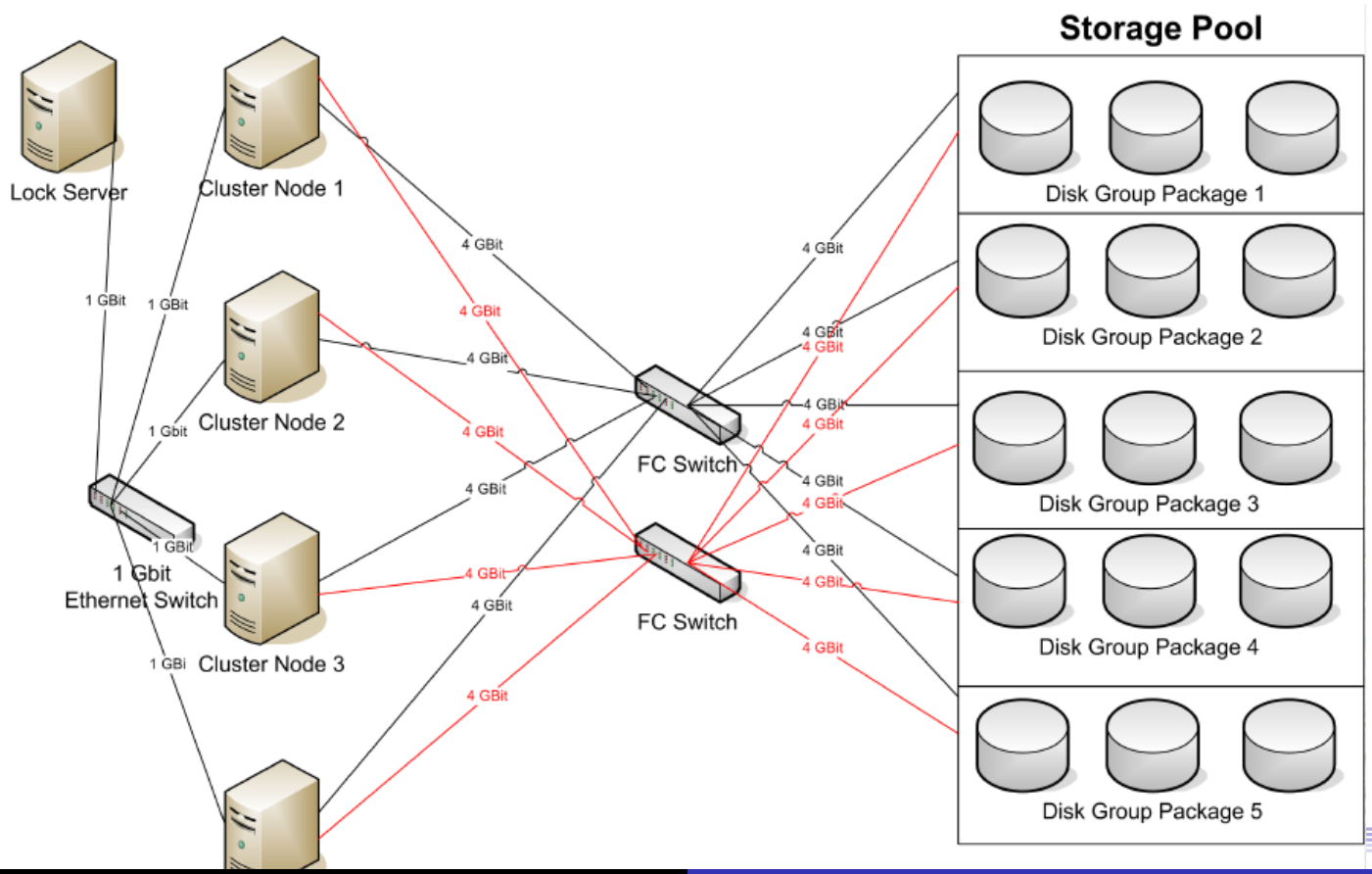
GFS - Idea

GlobalFS to:

- Klastrowy system plików
- Wspiera dzielenie wspólnych danych (znajdujących się na wspólnym urządzeniu/urządzeniach) przez wiele komputerów jednocześnie.



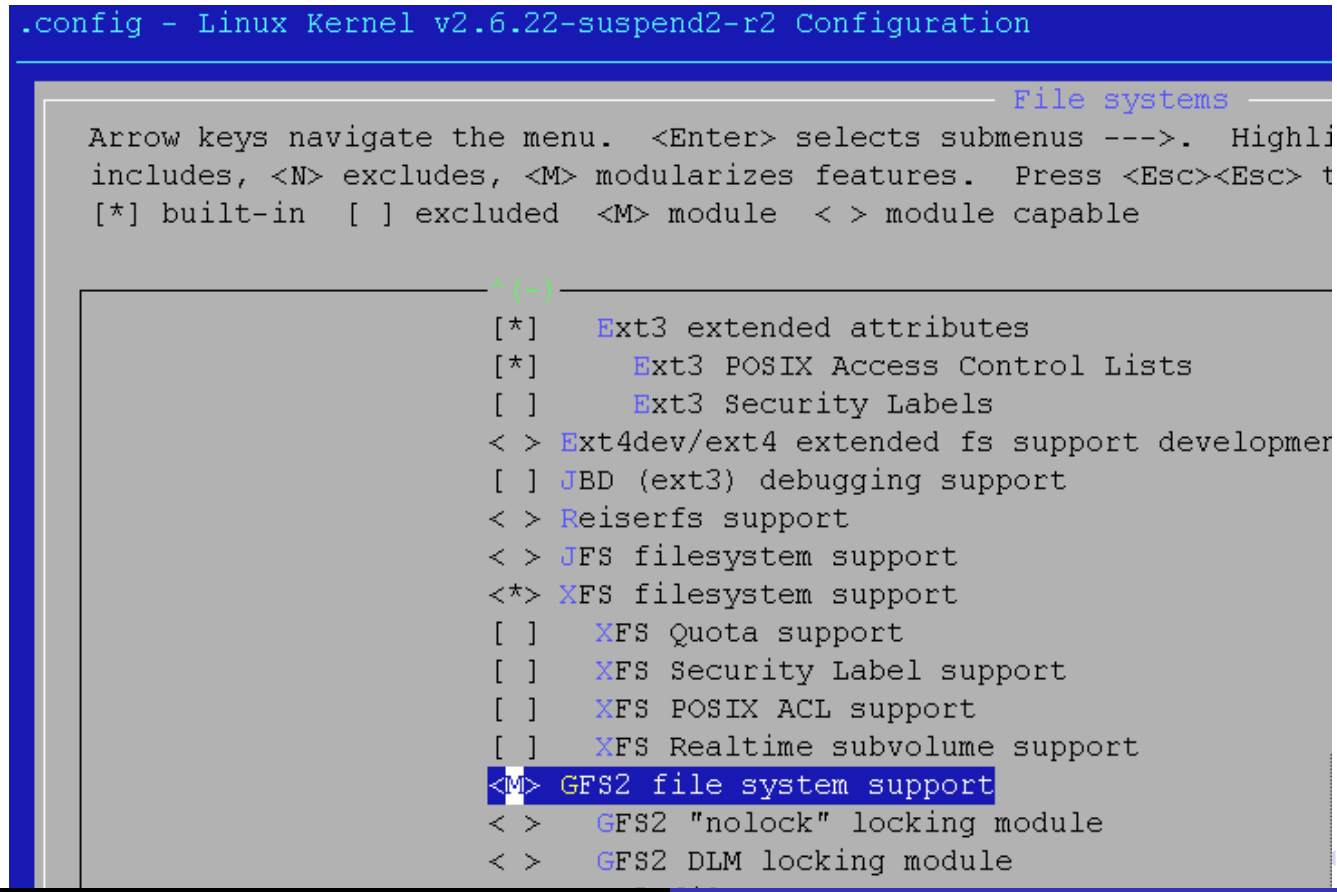
GFS - Idea cd


Storage Pool
Lock Server
Cluster Node 1
Cluster Node 2
Cluster Node 3
1 Gbit
Ethernet Switch
FC Switch
FC Switch
Disk Group Package 1
Disk Group Package 2
Disk Group Package 3
Disk Group Package 4
Disk Group Package 5
1 GBit
4 GBit

GlusterFS - Historia

- Oryginalny GFS był rozwijany na na University of Minnesota przez grupę Matta O'Keefe
- Późniejsze wersje zostały włączone do Linuxa
- Na przełomie 1990/2000 twórca założył firmę Sistina Software i GFS stał się komercyjny
- W roku 2003 firma została przejęta przez firmę Red Hat.
- Na podstawie doświadczeń związanych z tworzeniem GFS w roku 2005 został zaprojektowany GFS2.
- Po bardzo długim przeglądzie kodu i wielu modyfikacjach GFS2 został włączony do jądra 2.6.16.
- Podobnie jak ext2 można przerobić na ext3 tak też nie ma problemów ze zmianą GFS na GFS2

GFS2 - .config


.config - Linux Kernel v2.6.22-suspend2-r2 Configuration
File systems
Arrow keys navigate the menu. <Enter> selects submenus --->. Highli
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> t
[*] built-in [ ] excluded <M> module < > module capable
[*] Ext3 extended attributes
[*] Ext3 POSIX Access Control Lists
[ ] Ext3 Security Labels
< > Ext4dev/ext4 extended fs support developmer
[ ] JBD (ext3) debugging support
< > Reiserfs support
< > JFS filesystem support
<*> XFS filesystem support
[ ] XFS Quota support
[ ] XFS Security Label support
[ ] XFS POSIX ACL support
[ ] XFS Realtime subvolume support
<M> GFS2 file system support
< > GFS2 "nolock" locking module
< > GFS2 DLM locking module

GFS a NFS

GFS ma wiele podobieństw do NFS, ale:

- w NFS dostęp do danych odbywa się przez serwer NFS
- w NFS komputery-klienci nie mają bezpośredniego dostępu do urządzenia.
- rezultatem tego jest nadmiarowość w dostępie do danych związana z obsługą protokołu sieciowego.
- GFS wykorzystuje protokoły Fibre Channel oraz iSCSI
- iSCSI to technika umożliwiająca wykonywanie operacji wejścia-wyjścia na dysku twardym odległej maszyny za pomocą protokołu TCP/IP (tłumaczonego na polecenia w standardzie SCSI - potrzebne są dyski SCSI - droższe od dysków SATA/IDE).
- Fibre Channel podobnie, ale wykorzystuje specjalne kable (droższy, prawdopodobnie zostanie niedługo w całości zastąpiony przez iSCSI).

GFS - opis

W GlobalFS:

- Dyski sieciowe (network storage devices) obsługują bezpośrednio żądania klientów
- Rozproszony system plików jest widziany przez klienta jak “normalny” lokalny system, ale żaden klient nie “posiada”, ani nie jest odpowiedzialny za kontrolę żadnego dysku sieciowego
- Nie ma żadnej bezpośredniej komunikacji między klientami - dzięki temu każdy z każdy klientów jest niezależny od awarii i obciążeń innego klienta

GFS - jak to jest uzyskiwane

W GlobalFS mamy rozwiązanie podobne do tego jak rozwiązano dostęp do dzielonej pamięci w architekturach SMP. GlobalFS wykorzystuje mechanizm blokowania pamięci.

- Przed zmodyfikowaniem danych klient zakłada blokadę. Po zapisaniu modyfikacji zwalnia ją.
- Wspomniane dyski sieciowe wspierają sprzętowo ten mechanizm.

GFS a współbieżność

- system wtyczek - rożne algorytmy zapewniania współbieżności.
- Główny - DLM (Distributed lock manager).
- Red Hat poprawił oryginalnego DLM, został on włączony do jądra 2.6.19.

DLM - Distributed Lock Manager

- DLM po pierwsze służy do blokowania plików
- po drugie do koordynacji dostępu do dysków

DLM - ogólna idea

Oryginalny DLM oferuje kilka trybów dostępu do zasobu:

- Null Lock.
- Concurrent Read. Nie zezwala jedynie na uzyskanie przez innych trybu Exclusive do obiektu.
- Concurrent Write. Zezwala innym na dostęp (typu Concurrent) do odczytu i zapisu. Wymusza to brak cachowania.
- Protected Read. Nie zezwala innym na modyfikowanie czytanego obiektu.
- Protected Write. Zablokowanie do zapisu. Pozwala jednakże na dostęp czytającym w trybie Concurrent Read.
- Exclusive. Nikt inny nie może mieć dostępu do urządzenia.

DLM - Distributed Lock Manager cd

DLM napisany przez Red Hata oferuje trzy z nich:

- Exclusive
- Shared - czyli Protected Read
- Deffered - czyli Concurrent Write

DLM - Distributed Lock Manager cd

- GFS pamięta żeton dla każdego i-węzła.
- Żądania zablokowania obiektu są kolejkowane. Komunikat o uzyskaniu dostępu do urządzenia może być wysłany synchronicznie lub asynchronicznie.

GFS - Przykład

GFS przykład użycia

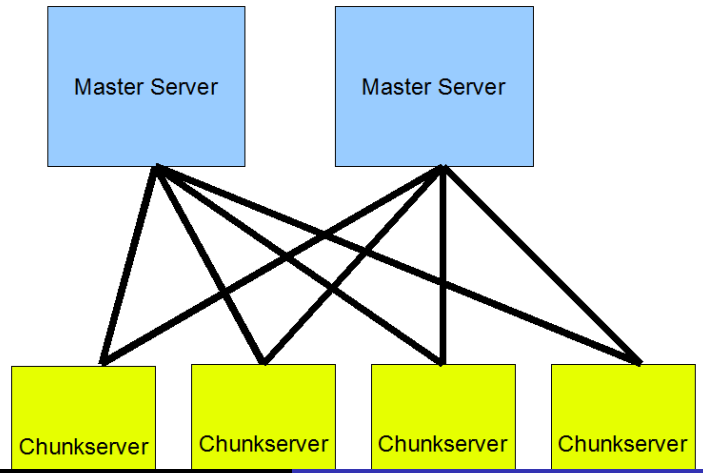
```
1 # gfs_mkfs -p lock_dlm -t alpha:gfs1 -j 8 /dev/vg01
    /lvol0
2 # mount -t gfs /dev/vg01/lvol0 /gfs1
```

GoogleFS - Idea

- Google File System (GoogleFS) jest rozproszonym systemem plików stworzonym do przetrzymywania ogromnych ilości danych.
- System powstał na bazie wcześniejszego projektu Googla - BigFiles - stworzonego przez założycieli: Larry Page oraz Sergey Brin (w połowie lat 90tych całość zajmowała 128 gb)
- System jest zbudowany z klastrów zwykłych komputerów. Zmniejsza to koszt oraz zwiększa to podatność na awarie.
- System zapewnia wysoki transfer danych, kosztem opóźnień.

GoogleFS - Realizacja

- Dane przetrzymywane są w dużych plikach, często kilka gigabajtów, które są żadko usuwane, nadpisywane, obcinane. Dane są z reguły dopisywane do końca istniejących plików.
- System składa się z master nodes oraz chunkservers.



GoogleFS - Podział komputerów

- Każdy chunkserver trzyma dane podzielone na 64 megabajtowe kawałki (chunks), do każdego z nich przydzielony jest 64-bitowy unikalny identyfikator.
- Master server utrzymuje wszystkie metadane związane z plikami.
  - Logiczna mapa plików do 64-bitowych identyfikatorów odpowiednich kawałków.
  - Mapa identyfikatorów do fizycznego położenia odpowiednich kawałków.
  - ...

GoogleFS - Przykład działania

- Master serwer daje procesowi uprawnienia do operowania na pliku przez określony czas (żaden inny proces nie ma prawa dostępu do pliku w tym czasie)
- Zmodyfikowany chunkserver, który jest właścicielem danego kawałka (chunk) rozsyła dane do chunkserwerów z kopiami zapasowymi.
- Zmiany nie są zachowywane do czasu ich potwierdzenia przez wszystkie chunkserwery, dzięki temu zostaje zachowana atomowość operacji.

GoogleFS - Usługi Googla

- Google Code
- Google Earth
- Google Print
- Google Reader
- Gmail
- Google Maps
- Orkut
- YouTube
- . . . (ponad 110 innych produktów)

GoogleFS - BigTable

- BigTable jest bazą danych stworzona na użytej Google. cechy:
- jest oparta na systemie plików GoogleFS
- posiada dużą wydajność, jest skompresowana
- korzystają z niej projekty - wymienione w poprzednim slajdzie
- zoptymalizowana pod szybki odczyt kolumn, zamiast wierszy
- jest zaprojektowana do operowania na petabajcie danych (10 do potęgi 15)
- zapewnia korzystniejsze koszty licencjonowania, skalowalność, lepsza kontrole nad wydajnością

GoogleFS - MapReduce

- MapReduce jest frameworkiem, napisanym w C++, który umożliwia zrównoleglenie obliczeń na danych wielkości do 100 terabajtów (10 do potęgi 12)
- Funkcje:
- Input reader - dzieli dane z wejścia na 16 - 128 mb bloki, które są kojarzone z funkcją map (czyta dane z GFS)
- Map function - pobiera zbiór par klucz / wartość i zwraca zbiór par klucz / wartość (jeśli aplikacja liczy ilość slow, to zwróciła by pary słowo -> ilość wystąpień)
- Partition function - funkcja alokuje dane z map do odpowiednich reduce (zna klucz reduce do którego dane maja zostać przekazane, ilość funkcji reduce - zwraca

GoogleFS - MapReduce 2

- Comparison function - wejście dla reduce jest brane z wyniku map, posortowane funkcja comparison
- Reduce function - jest przez framework wykonywane dla każdego klucza. Reduce może iterować po wszystkich wartościach związanych z kluczem i zwraca zbiór par klucz / wartość. (np. dla przypadku podanego w map, zwróciłaby pojedynczą wartość będącą sumą wystąpień wszystkich słów)
- Output writer - zapisuje wynik reduce to pamięci stałej (np. dysku twardego z użyciem GFS)

Historia

- ZFS został napisany przez zespół Suna pod przewodnictwem Jeffa Bonwicka.
- Przedstawiony światu 14 września 2004 roku. Później został wprowadzony do systemów Solaris i Open Solaris (kod źródłowy jest ogólnodostępny na licencji CDDL).
- Oryginalna nazwa pochodzi od Zettabyte File System.

Adaptacja do innych systemów

- Obecnie są prowadzone prace nad przeniesieniem ZFS do innych systemów operacyjnych.
- Apple ogłosił, że ZFS pojawi się w przyszłych wydaniach systemu Mac OS.
- ZFS działa w Linuksie tylko z FUSE (ograniczenia licencyjne nałożone na jądro przez licencję GNU).
- ZFS jest w większości we FreeBSD (dzięki Polakowi Pawłowi Jakubowi Dawidkowi).

Główne cele ZFS

- Wykrywanie i naprawianie błędów oraz cichych błędów (powstałych na skutek błędów w pamięci operacyjnej czy podczas przesyłania danych szyną) na nośnikach danych.
- Skalowalność (łatwa możliwość rozbudowy macierzy dyskowych).
- Łatwość administracji.
- Wydajność.

Ograniczenia obecnych systemów plików

- Brak wsparcia dla wykrywania cichych błędów.
- Skomplikowane w administracji (ciężko na przykład zmienić rozmiar partycji).
- Dużo limitów: limity wielkości pliku, ilości plików w katalogu, etc...
- Nie przenośne (metadane systemu plików zależne od endian procesora).
- Mało wydajne (liniowy czas tworzenia systemu plików - formatowanie, stała wielkość bloku danych).
- Ich podstawowe założenia sprawdzały się dobrze w poprzedniej epoce, w ZFS zostały odrzucone.

Odpowiedzą Suna jest ZFS!!!

- Sun twierdzi że ZFS jest bardzo rewolucyjnym rozwiązaniem w dziedzinie systemów plików.
- Inne systemy, obrastały w dodatkowe technologie mające rozwiązywać problemy, które ZFS rozwiązuje niejako naturalnie (np. ZFS nie potrzebuje journalingu).
- Tak przynajmniej twierdzi SUN...

Główne funkcjonalności ZFS

- Elastyczność zasobów (można szybko dodawać nowe dyski bez np. formatowania i zwiększania rozmiaru partycji).
- Transakcyjny system aktualizowania zawartości systemu plików (wszystko albo nic) oparty na zasadzie copy-on-write (wyjaśnię później).
- Snapshoty budowane niejako automatycznie dzięki copy-on-write (COW).
- Prostota administracji.
- Automatyczne wykrywanie i naprawianie błędów.
- Dynamiczne dzielenie danych pomiędzy wiele urządzeń blokowych w celu zrównoważenia zapełnienia i

Główne funkcjonalności ZFS

- 128 bitowy system plików (praktycznie brak ograniczeń wielkości).
- Dynamiczne metadane (tworzone tylko wtedy, kiedy potrzeba).

Przykładowe ograniczenia ZFS

- Maksymalna ilość plików w katalogu to: 2^{48}.
- Maksymalna wielkość pliku to: 16 EiB (2^{64}).
- Maksymalna ilość systemów plików w puli to: 2^{64}.
- Maksymalna długość nazwy pliku to: 255.
- Długości bloków od 512 bajtów do 128KB.

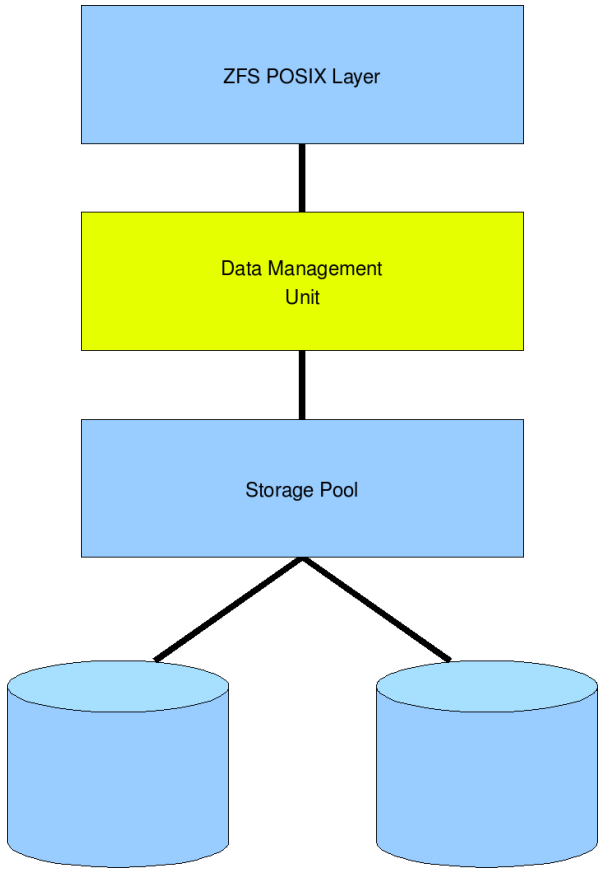
Prawo Moora odnosi się także do pojemności dysków (podwajanie się pojemności dysków co 9-12 miesięcy).
Stąd w ZFS obsługa tak dużych danych.
Ciekawostka: 128-bitowy system plików przekracza kwantowe możliwości zapisu tak dużej ilości danych na Ziemi... Energia użyta do zapisania 128 bitowego sytemu plików wystarczyłaby do zagotowania oceanów...

Istnieją jednak inne systemy plików oferujące podobną funkcjonalność

- System WAFL (Write Anywhere File Layout) 1994: snapshoty umożliwiające szybki powrót do stanu sprzed awarii (również zastosowany mechanizm copy-on-write).
- AdvFS (1990): wprowadzono journaling, tworzenie snapshotów w trakcie działania systemu, również oparty jak ZFS na koncepcji puli (opiszę je zaraz), tutaj nazywających się domenami plików.

Zatem Sun wcale nie był taki pierwszy...

Warstwy ZFS



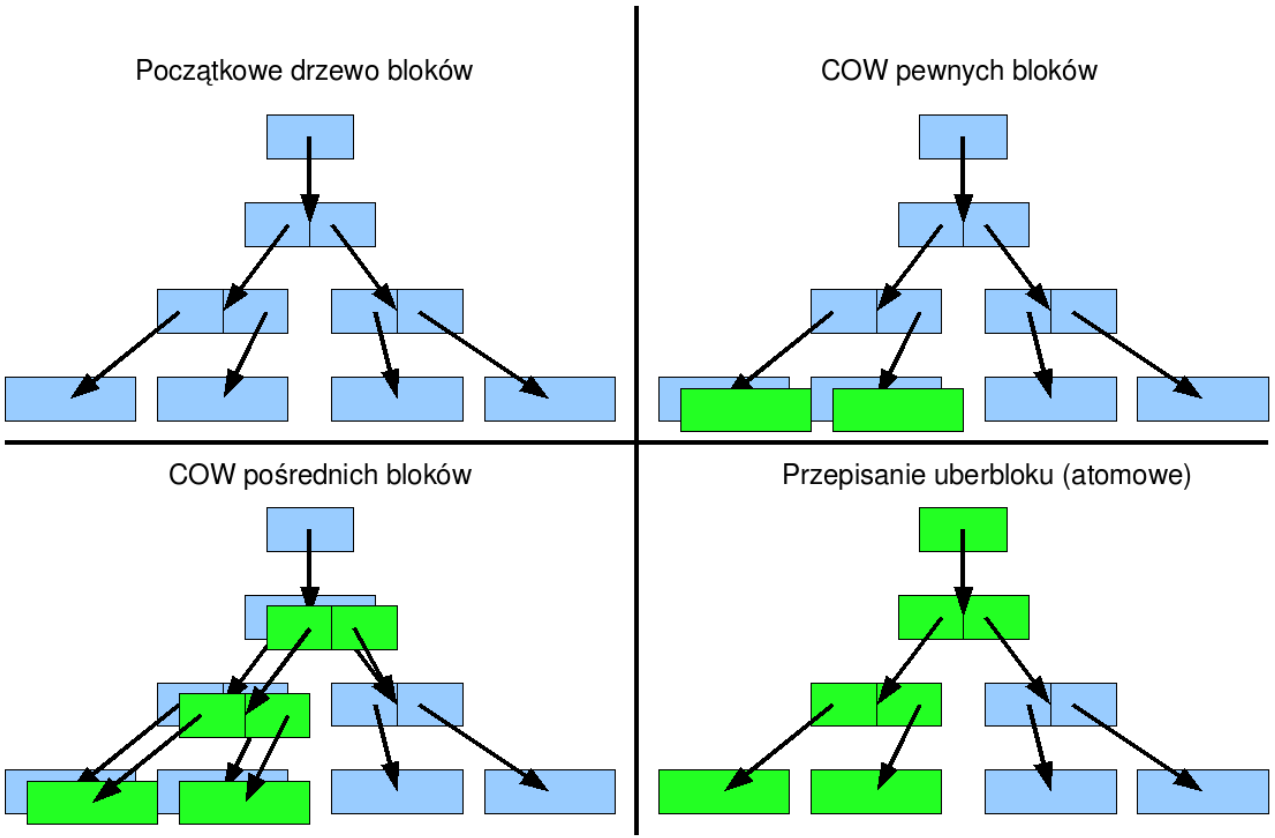
Cechy budowy ZFS

- Odsunięcie interfejsu blokowego jak najdalej na dół.
- Warstwa ZFS POSIX Layer bierze polecenia systemowe i przetwarza je na transakcje.
- DMU bierze transakcje i je grupuje (przy okazji układając je tak, aby ich realizacja była sprawniejsza i szybsza).
- Bogaty interfejs semantyczny pomiędzy ZFS POSIX Layer i DMU.
- DMU jest odpowiednikiem Volume Managerów (czyli urządzeń wspierających np. RAID-y).

Cechy budowy ZFS

W innych systemach plików warstwa programowa komunikuje się z Volume Managerami za pomocą operacji blokowych, a Volume Managery już nie mogą zmieniać kolejności odwołań (muszą je wykonywać tak jak je dostały). W razie braku prądu tworzą się problemy ze spójnością danych (np. udało się zapisać tylko na jeden dysk z RAID-1 (mirror) lub tylko jeden blok danych). W ZFS właśnie dzięki bogatemu interfejsowi DMU takie problemy nie występują (DMU ma większą wiedzę o tym co się dzieje niż Volume Managery, więc może rozsądnie operować na dyskach).

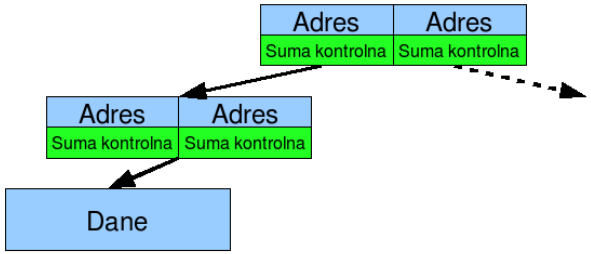
Mechanizm COW



Cechy mechanizmu COW

- Data Managment Unit wspiera (głównie grupuje transakcje) copy-on-write dzięki czemu COW jest wydajne.
- Dostajemy darmowe snapshoty.
- Nigdy aktualne dane nie są nadpisywane. W innych systemach plików można nadpisywać aktualne dane.

Sprawdzanie poprawności danych



- W ZFS suma kontrolna jest trzymana w metadanych poziom wyżej.
- Zapobiega to błędom przypadkowego zapisu, złego odczytu, etc..., ponieważ odczytane bloki danych są sprawdzane ze swoją sumą kontrolną umieszczoną gdzie indziej.

Sprawdzanie poprawności danych

- Obecnie suma kontrolna bloku danych często jest umieszczona razem z blokiem danych (oracle tak robi, podobno Atachi sprzętowo w swoich dyskach też).
- Ponadto ZFS w tle sprawdza poprawność danych na dysku (scrubbing), szuka błędów, naprawia je itp... Analogia do pamięci ECC.

Sprawdzanie poprawności danych

W ZFS sprawdzaniem poprawności danych na dysku zajmuje się CPU. Kiedyś to było zbyt kosztowne. W obecnych CPU sprawdzanie zabiera tylko część jego mocy. W serwerach można na to zadanie przeznaczyć jeden CPU. Zazwyczaj jednak przepustowość CPU jest większa od przepustowości dysków (dla przykładu algorytmy sprawdzania ZFS mają przepustowość na nowoczesnych CPU rzędu 2-8GB/s, natomiast dyski tylko kilkaset MB/s).

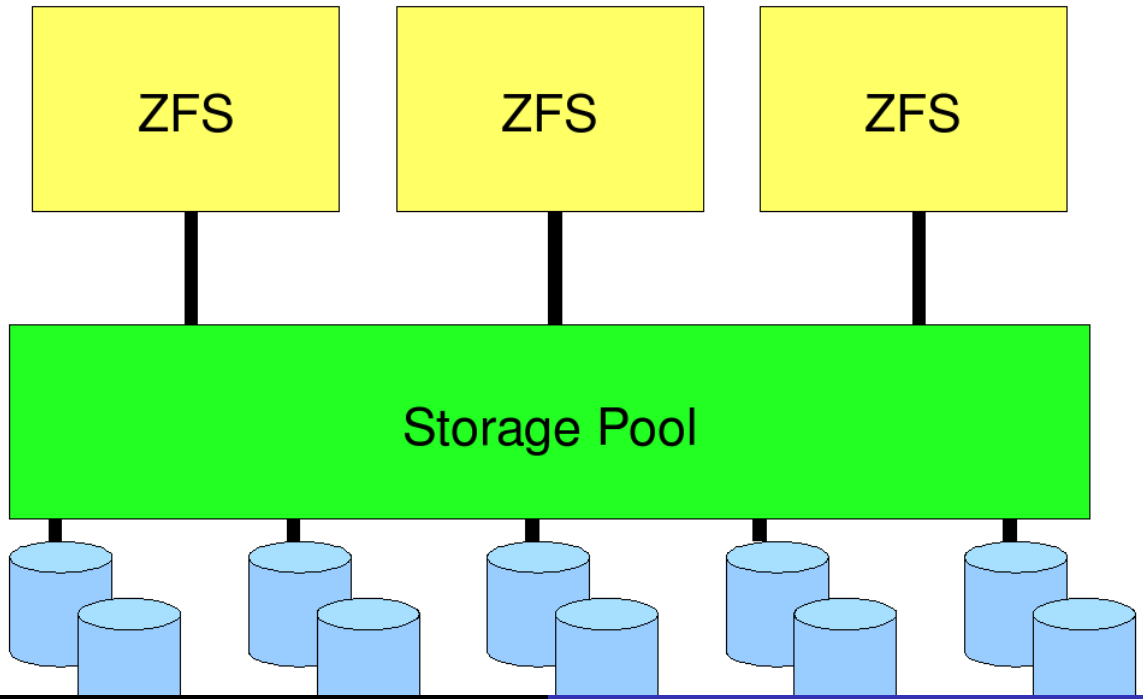
Pojęcie puli

- Głównym pojęciem ZFS jest pula (zpool).
- Pula jest zbudowana z wirtualnych napędów, które reprezentują rzeczywiste blokowe urządzenia. Mogą nimi być dyski, partycje na dysku, pliki oraz ich różne kombinacje (ale o tym później).
- Na puli tworzymy wirtualne systemy plików, mające do dyspozycji zasoby puli (zatem pamięć puli jest dzielona pomiędzy te wirtualne systemy plików). ZFS dba o to, aby dane w wirtualnych systemach plików były spójne oraz rozmieszczone tak, aby operowanie na nich było wydajne i bezpieczne.

Pojęcie puli

- Ponadto w wirtualnych systemach plików można tworzyć kolejne systemy plików. Atrybuty systemów plików są dziedziczone.
- Pula stanowi abstrakcję pamięci masowej dla wirtualnych systemów plików taką jaką dla aplikacji stanowi pamięć wirtualna.

Pojęcie puli



Co jeszcze może być wirtualnym napędem?

- Lustra (mirrors) - czyli dwa dyski, które zawierają kopie swoich danych.
- RAID-Z - układ dysków analogiczny do RAID-5.

Lustra - Mirrors

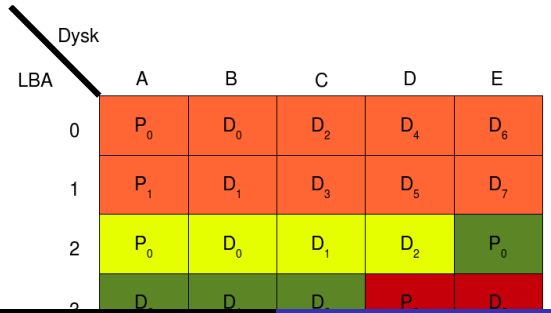
Lustra w ZFS są w stanie przetrwać uszkodzenie jednego dysku. Są ponadto odporne na ciche błędy. W przypadku wykrycia błędnych danych, ZFS automatycznie odczytuje dane z drugiego dysku i jeżeli są one prawidłowe przekazuje je aplikacjom. Ponadto w takiej sytuacji błędne dane są nadpisywane poprawnymi danymi oraz odnotowywane jest ostrzeżenie dla administratorów.

RAID-Z

- W RAID-5 występuje problem z „dziurą zapisu”. W przypadku gdy zabraknie prądu i byliśmy w trakcie zapisywania danych na jeden dysk oraz parzystości na drugi może zdarzyć się tak, że po włączeniu prądu parzystość będzie niezgodna z danymi. Obecnie radzimy sobie z tym oznaczając „brudne sektory” (kolejna metoda obejścia problemu) lub stosując drogie NVRAM (wąskie gardło).
- RAID-Z rozwiązuje problem „dziury zapisu”, gdyż aktualne dane nigdy nie są nadpisywane.

RAID-Z

- W ZFS rozmiar bloku danych jest zmienny, zatem dla każdego bloku danych ilość danych tego bloku na każdym z dysków jest zmienna (dynamiczny striping).
- W razie dodania nowego dysku nowo zapisywanie dane będą z niego korzystać. Ponadto operacje zapisu starych danych również będą powodowały dyskretne przepisywanie danych na nowy dysk (dzięki COW).



RAID-Z

- W razie wykrycia błędu chcę sprawdzić, na którym dysku jest błąd i go naprawić:
  - Mam sumę kontrolną oraz blok danych z błędem i parzystością.
  - Testuję po kolei dyski odtwarzając dane na nich na podstawie parzystości.
  - Jeżeli blok danych wraz z odtworzonymi danymi pasuje do sumy kontrolnej znaczy, że wykryłem na którym dysku jest błąd.
  - Przekazuję dane aplikacji i naprawiam błąd.
- Wniosek: ZFS lubi tanie dyski (kontrolą błędów zajmuje się ZFS, a nie specjalizowane urządzenia).

Inteligenty prefetching w ZFS

- Jest w stanie wykryć różne schematy dostępu do pliku (strumieniowe, liniowe itp...) Dla przykładu: jeżeli program odwoła się do bloków danych 10, 20, 30, to prefetching wczyta blok 40 (działa to również wstecz). Ponadto ZFS jest w stanie wykryć dziwniejsze schematy dostępu do pliku: program wczytał bloki 1, 2, 3, 11, 12, 13. Prefetching spowoduje wczytanie bloków 21, 22, 23...
- Taki prefetching optymalizuje dostęp do np. dużych macierzy zapisanych w plikach.

Nazwy urządzeń w Solarisie

„/**dev**/**dsk**/**c***n***t***n***d***n***s***n*”

- **c*n*** — numer kontrolera (np. **c0**)
- **t*n*** — ID urządzenia (np. **t0**)
- **d*n*** — numer urządzenia
- **s*n*** — numer części dysku (zazwyczaj reprezentuje to partycję). Poprawne numery od 0 do 7 (np. **s0**)

Tworzenie nowej puli

„zpool create <*nazwa_puli*> [<*opcjonalna konfiguracja*>] <*pliki urządzeń*>”

- Np. Polecenie **„zpool create moja_pula mirror c0t0d0 c1t0d0”** tworzy pulę **„moja_pula”** zbudowaną z dwóch lustrzanych dysków. Pula ta zostanie zamontowana w katalogu **„moja_pula”**.
- Zamiast **„mirror x y”** można również napisać **„raidz x y z”** lub bardziej złożone kombinacje typu: **„mirror x y mirror u v”**.

Tworzenie nowej puli

- Nazwa puli jest ciągiem zawierającym litery, cyfry, kropki, podkreślenia, myślniki. Musi zaczynać się od litery. Nazwą nie mogą być mirror, raidz, spare, oraz ciągi zaczynające się od „**c*n***”, gdzie ***n*** to cyfry od **0** do **9**.

Tworzenie systemów plików

„**zfs create <*nazwa_systemu_piku*>**”

- Np. „**zfs create moja_pula/moj_system**” stworzy system plików „**moj_system**” w puli „**moja_pula**”. Standardowym punktem montowania będzie „**/moja_pula/moj_system**”.
- Aby zamontować nowy system plików gdzie indziej, piszemy: „**zfs set mountpoint=/moj_katalog/moj_system moja_pula/moj_system**”.

Dodawanie nowych dysków do puli

„zfs add moja_pula mirror c2t0d0 c3t0d0”

- ZFS dba o to, aby przy dodawaniu nowych urządzeń pula nie traciła właściwości bezpieczeństwa (np. ostrzega jeżeli próbujemy dodać do puli składającej się z dwóch dysków lustrzanych pojedynczy dysk).

Dodatkowe polecenia

- **„zfs set sharenfs=rw moja_pula/moj_system”**
  Dostęp poprzez NFS.
- **„zfs set compression=on moja_pula”**
  Włączenie kompresji. Można też **„compression=lzjb | gzip | gzip-N”**.
- **„zfs set quota=10g moja_pula/systemy_plikow/karol”**
  Ograniczenie pojemności.
- **„zfs set reservation=20g moja_pula/systemy_plikow/greg”**
  Zarezerwowanie pamięci.

Snapshoty

- Stworzenie snapshot-a systemu plików „**moj_system**”: **„zfs snapshot moja_pula/moj_system@nazwa_snapshota”**
- Cofnięcie się do poprzedniego snapshota: **„zfs rollback moja_pula/moj_system@nazwa_snapshota”**
- Snapshoty mogą tworzyć zwykli użytkownicy (jeżeli mają do tego prawa) oraz cofać się do nich bez konieczności ingerencji administratora.
- Tworzenie snapshotów jest atomowe.

Snapshoty

- Snapshoty dostępne poprzez katalog **„.zfs/snapshots/nazwa_snapshota”** znajdującym się w katalogu głównym systemu plików snapshota (w moim przykładzie **„moja_pula/moj_system”**).
- Można podglądać pliki: **„cat pula/moj_system/.zfs/nazwa_shapshota/test.txt”**.

Klony

- Klony tworzy się ze snapshotów.
- Do klonów można zapisywać dane.
- Przy czym dzięki COW jeśli nie zapisujemy danych w klonach, to nie zajmują one dodatkowej pamięci.
- Klony tworzymy:
  „zfs clone moja_pula/moj_system@nazwa_snapshota moja_pula/nazwa_systemu_plikow_klona”

Migracja danych

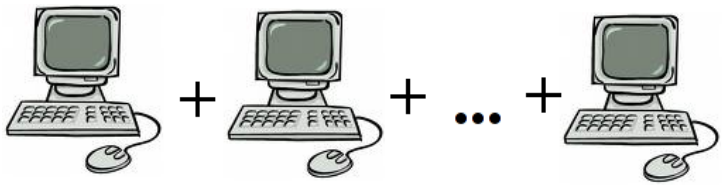
- Przejście w stan pozwalający na przeniesienie puli do innego systemu:
  „zfs export moja_pula”
- Zaimportowanie puli z innego systemu:
  „zfs import moja_pula”
- Wszystkie dane konfiguracyjne są trzymane w strukturach, które opisują.
- ZFS odporny na endian procesorów. W matadanych trzyma flagę, jakim endianem zapisane są tam dane i albo obraca bity, albo nie. Przy czym zapisuje metadane zawsze endianem bieżącego procesora.

Bezpieczeństwo

- Znany z Microsoftu i NFSv4 mechanizm ACL, czyli allow/deny dla użytkowników z dziedziczeniem.
- Przykładowe dodawanie praw użytkownikom w Solarisie: **„zfs allow karol create,mount zpool/moja_pula” „zfs allow maciek snapshot,rollback,mount zpool/moja_pula”**

GlusterFS - Idea

- GlusterFS to klastrowy system plików skalowalny do kilku peta bajtów.
- Umożliwia on połączenie różnych “cegiełek” (storage bricks) przez Infiniband RDMA lub protokół TCP/IP w jeden duży sieciowy system plików.
- Storage bricks mogą być właściwie dowolnym komputerem np. x86-64.



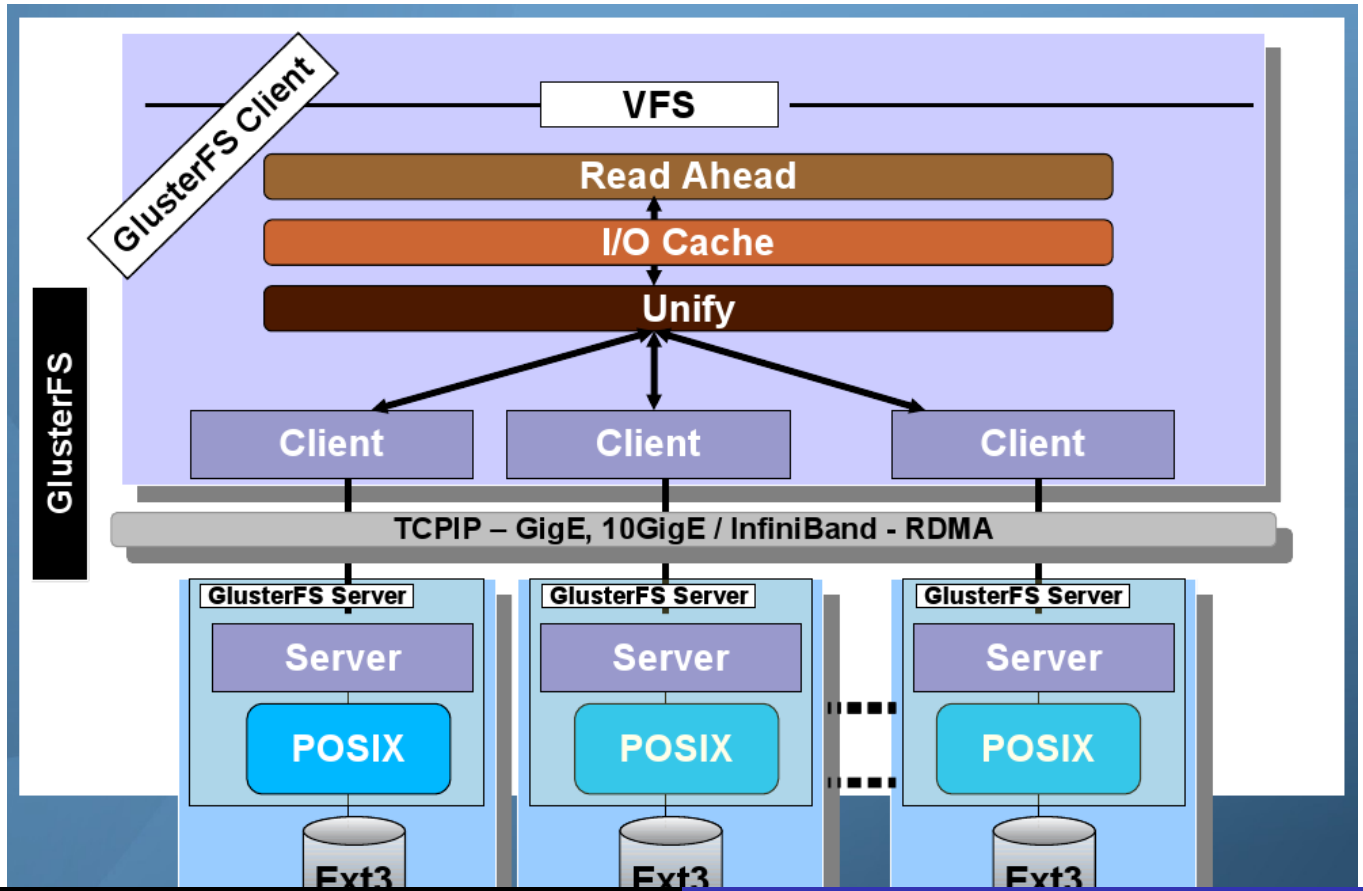
=

N x pojemność

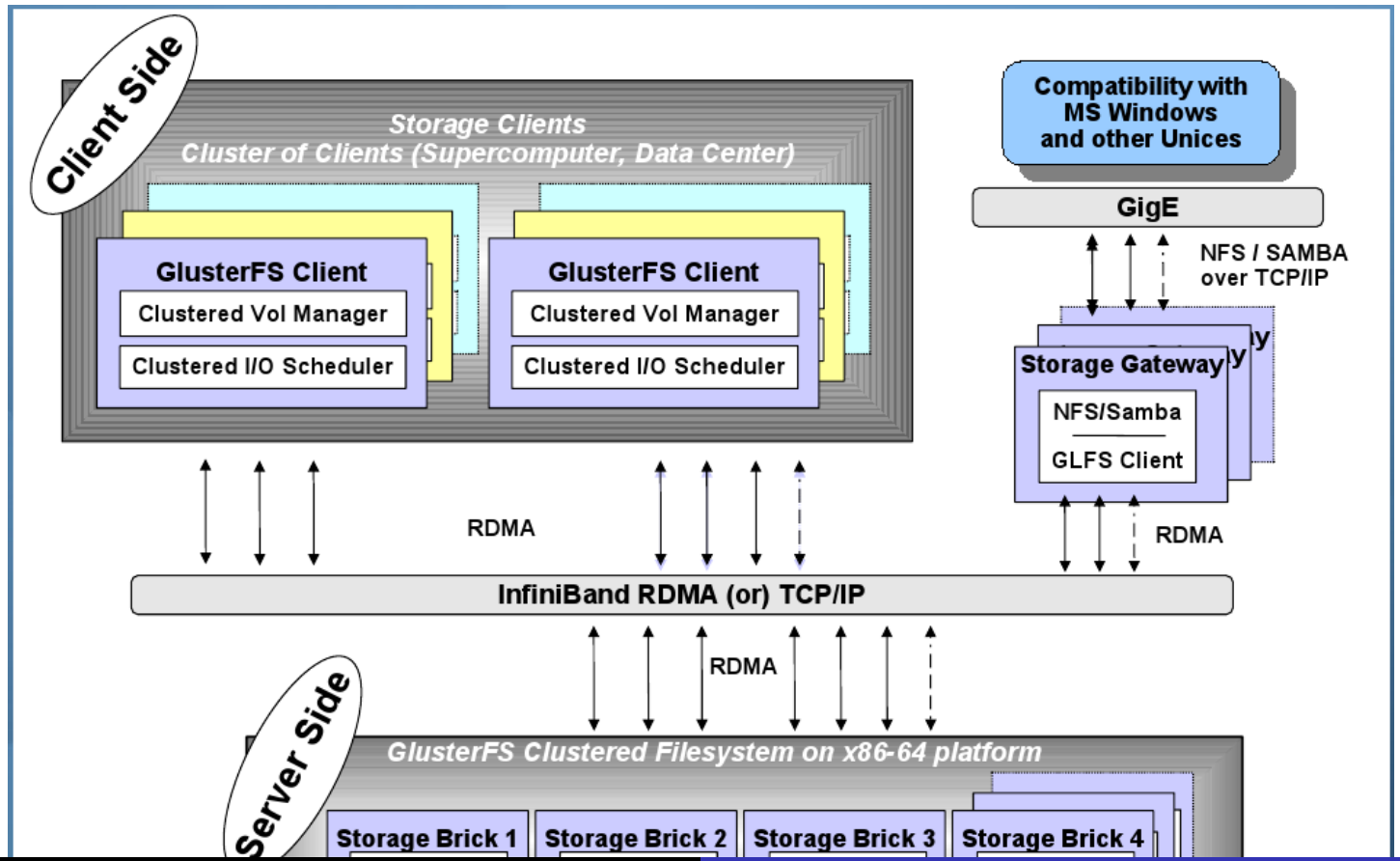
GlusterFS - Założenia

- Skalowalność pojemności
- Możliwość podłączenia klastrowych schedulerów wejścia-wyjścia
- Wykorzystanie zalet RDMA
- Niezawodność (brak meta-danych, brak przerw w działaniu)
- Łatwy w zarządaniu (NFS-like, samonaprawiający)
- Elegancka implementacja (wysoki poziom abstrakcji - brak uzależnienia od konkretnej architektury czy systemu operacyjnego)

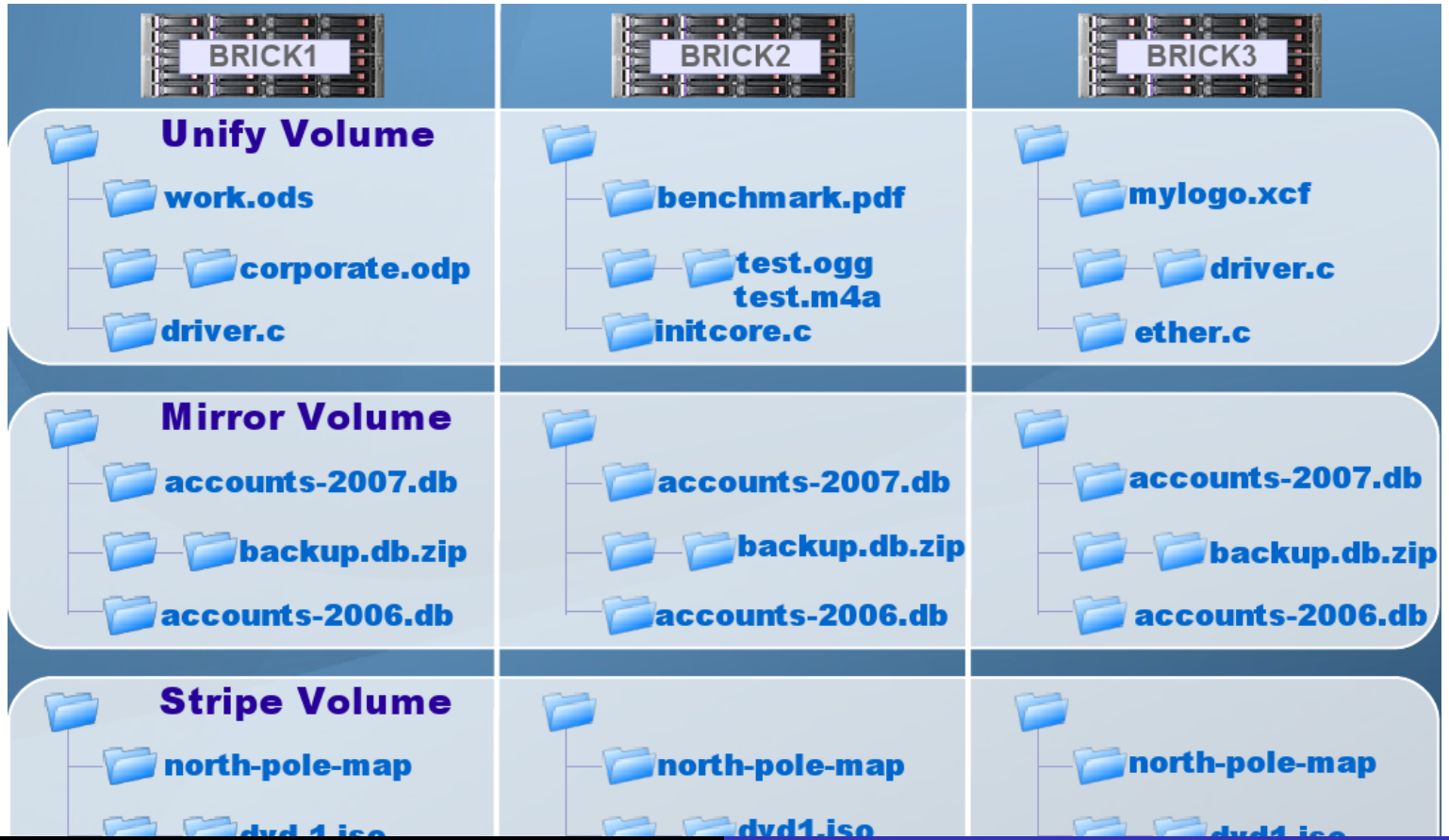
GlusterFS


GlusterFS Client
VFS
Read Ahead
I/O Cache
Unify
GlusterFS
Client
Client
Client
TCPIP – GigE, 10GigE / InfiniBand - RDMA
GlusterFS Server
Server
POSIX
Ext3
GlusterFS Server
Server
POSIX
Ext3
GlusterFS Server
Server
POSIX
Ext3

GlusterFS


Client Side
Storage Clients
Cluster of Clients (Supercomputer, Data Center)
GlusterFS Client
Clustered Vol Manager
Clustered I/O Scheduler
GlusterFS Client
Clustered Vol Manager
Clustered I/O Scheduler
Compatibility with MS Windows and other Unices
GigE
NFS / SAMBA over TCP/IP
Storage Gateway
NFS/Samba
GLFS Client
RDMA
RDMA
InfiniBand RDMA (or) TCP/IP
RDMA
Server Side
GlusterFS Clustered Filesystem on x86-64 platform
Storage Brick 1
Storage Brick 2
Storage Brick 3
Storage Brick 4

GlusterFS


BRICK1
BRICK2
BRICK3
Unify Volume
work.ods
corporate.odp
driver.c
benchmark.pdf
test.ogg
test.m4a
initcore.c
mylogo.xcf
driver.c
ether.c
Mirror Volume
accounts-2007.db
backup.db.zip
accounts-2006.db
accounts-2007.db
backup.db.zip
accounts-2006.db
accounts-2007.db
backup.db.zip
accounts-2006.db
Stripe Volume
north-pole-map
north-pole-map
north-pole-map

GlusterFS

Benchmark Environment

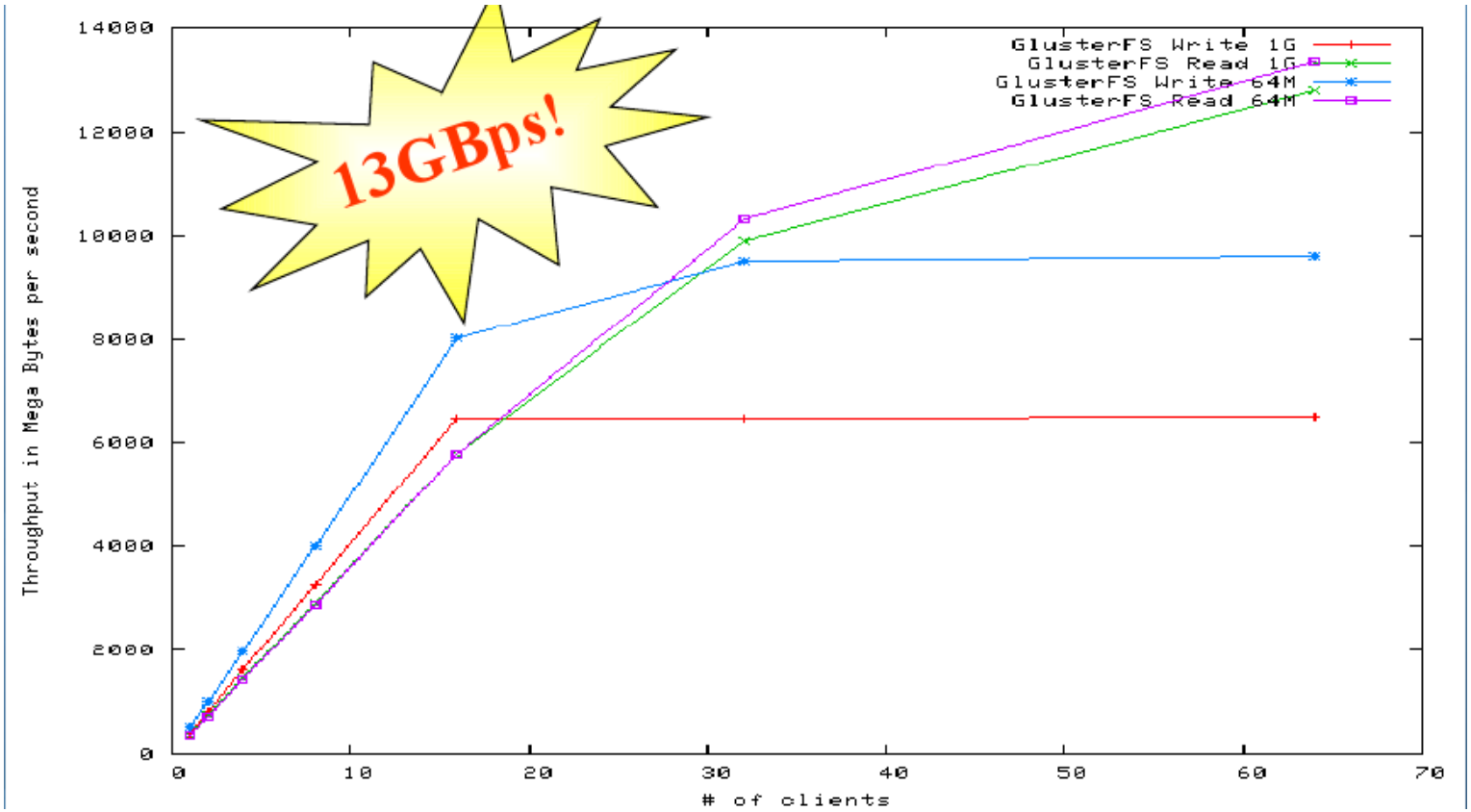
Method: Multiple 'dd' of varying blocks are read and written from multiple clients simultaneously.

GlusterFS Brick Configuration (16 bricks)
Processor - Dual Intel(R) Xeon(R) CPU 5160 @ 3.00GHz
RAM - 8GB FB-DIMM
Linux Kernel - 2.6.18-5+em64t+ofed111 (Debian)
Disk - SATA-II 500GB
HCA - Mellanox MHGS18-XT/S InfiniBand HCA

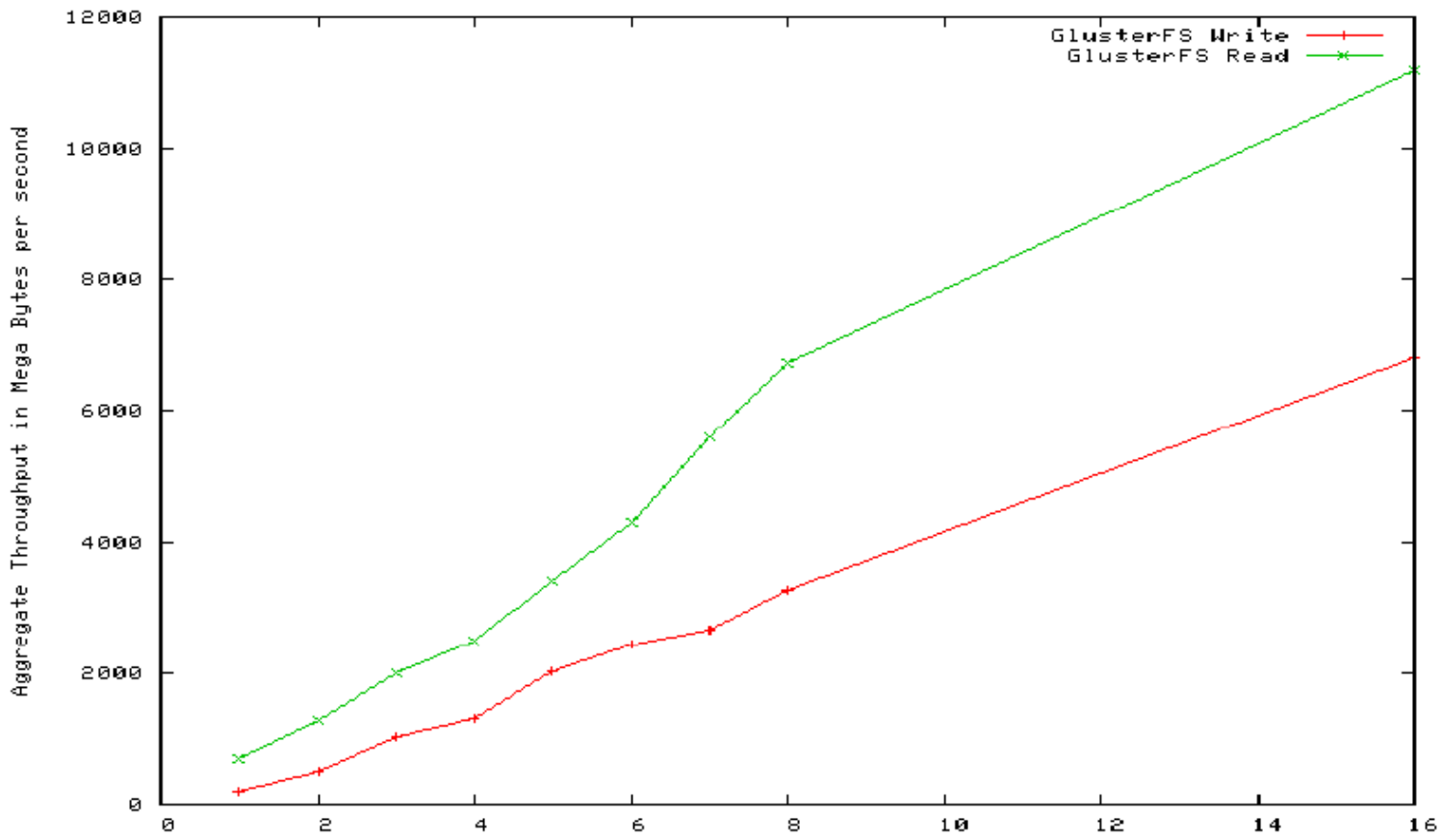
Client Configuration (64 clients)
RAM - 4GB DDR2 (533 Mhz)
Processor - Single Intel(R) Pentium(R) D CPU 3.40GHz
Linux Kernel - 2.6.18-5+em64t+ofed111 (Debian)
Disk - SATA-II 500GB
HCA - Mellanox MHGS18-XT/S InfiniBand HCA

Interconnect Switch: Voltaire port InfiniBand Switch (14U)

GlusterFS


13GBps!
GlusterFS Write 1G
GlusterFS Read 1G
GlusterFS Write 64M
GlusterFS Read 64M
Throughput in Mega Bytes per second
# of clients
0
2000
4000
6000
8000
10000
12000
14000
0
10
20
30
40
50
60
70

GlusterFS


GlusterFS Write
GlusterFS Read
Aggregate Throughput in Mega Bytes per second
0
2000
4000
6000
8000
10000
12000
0
2
4
6
8
10
12
14
16

Dziękujemy za uwagę.