Rozproszone systemy plików na przykładzie OpenGFS oraz Lustre

Przemysław Danilewicz

16 czerwca 2005

Spis treści

1 OpenGFS

1.1 Wstęp

OpenGFS czyli The Open Global File System jest systemem plików z właściwościami dziennika (journaling), który umożliwia jednoczesny dostęp do wspólnej przestrzeni dyskowej przez wiele węzłów. OpenGFS koordynuje dostępem do urządzeń dyskowych tak aby węzły nie mogły zapisywać jednocześnie w tych samych obszarach urządzenia, jednocześnie zapewniając możliwość równoczesnego odczytu. Węzły mają bezpośredni dostęp do dysków co umożliwia zmniejszenie przeciążenia sieci. W ogólnym schemacie OpenGFS należy zwrócić uwagę na moduł blokujący (lock module), który połączony jest z OGFS przy pomocy interfejsu typu plug-in.
OpenGFS zapewnia:

1. Spójny widok nazw woluminów (/dev) oraz rozmiarów i atrybutów z wszystkich węzłów. Możliwość połączenia wielu urządzeń dyskowych w jeden wolumin logiczny, co zapewniał wcześniej zarządca puli dyskowej (OpenGFS pool volume manager), teraz może być zapewnione przez inny menedżer woluminów.

2. Usługa blokowania (locking service), aby dwa węzły nie próbowały zapisać tej samej przestrzeni dyskowej w tym samym czasie, lub aby jeden węzeł nie czytał gdy drugi zapisuje ten sam obszar. OpenGFS posiada wewnętrzny system blokowania (locking system) umożliwiając użycie różnych protokoły blokowania (lock protocols) (np. protokołów OpenGFS „memexp" i „nolock"). Protokół memexp do komunikacji między węzłami używa sieci lokalnej (LAN). Dodany protokół OpenDLM również wykorzystuje LAN i jest bardziej wydajny niż memexp. Jest on też pozbawiony wady memexp single point of failure.

3. Niezależne dzienniki (journals) w każdym węźle.
   Dzienniki mogą być 3 rodzajów:

   - wewnętrzne–składowane razem z systemem plików
   - zewnętrzne–składowany w współdzielonym woluminie

   Dziennik dostarczony przez OpenGFS chroni tylko metadane (nie dane w plikach). Oznacza to że dane zawarte w plikach mogą być uszkodzone w przypadku awarii, lecz spójność systemu plików zostanie zachowana.

4. Usługa przynależności do klastra (Cluster membership services), służąca do przypisania dziennika dla węzła włączającego się do klastra, aby usługa blokowania został poinformowany o przyłączeniu węzła oraz aby oznaczyć nieoczekiwane opuszczenie klastra przez węzeł.
   Usługa ta jest związana z OpenGFS przez interfejs usługi blokowania (locking service interface). Usługa przynależności (membership service) może być różnie zaimplementowana dla różnych protokołów blokowania.

5. Odcięcie lub „Shoot The Other Mashine In The Head"(STOMITH), usługa umożliwiająca izolację węzła (a nawet wyłączenie go przy pomocy STOMITH), aby zapobiec zniszczeniu danych w przypadku awarii węzła. Usługa ta jest tylko częściowo związana z kodem OGFS.
   Usługa zarządzająca klastrem jest odpowiedzialna za użycie STOMITH, gdyż OGFS nie wykonuje tej operacji automatycznie.

6. Zmiana rozmiaru woluminu. Narzędzia administracyjne dostarczają możliwość zwiększania rozmiaru woluminu oraz dodawania dzienników. OpenGFS nie zapewnia w tej chwili zmniejszania rozmiaru woluminu ani usuwania dzienników.

1.2 Terminologia

Sektorem nazywamy 512 bloków tak jak jest to w jądrze Linuksa,natomiast*blok* odnosi się do pojęcia bloku w OpenGFS, rozmiar jest potęgą dwójki, od 512 do 65536.

1.3 Kolejność bajtów

Wszystkie dane na dysku w strukturze OpenGFS wykorzystują układ big endian. Kolejność bajtów na dysku zdefiniowana jest przez makro OGFS_ENDIAN_BIG w pliku fs/ogfs_ondisk.h. Nie ma żadnych gwarancji na to że bieżący kod będzie działał w układzie little endian.

1.4 Pula dyskowa

W przypadku współdzielenia wielu dysków na wielu komputerach w klastrze, istnieje duże prawdopodobieństwo że poszczególne węzły posiadają różne nazwy dysków. Moduł jądra puli dyskowej w połączeniu z narzędziami rodziny ptool tworzy warstwę pośredniczącą (pool), która zapewnia jednorodny dostęp do zasobów.
System plików OpenGFS zwykle nie jest instalowany bezpośrednio na partycji dyskowej, zamiast tego używa się urządzenia puli dyskowej (pool).
W każdym węźle wykonywane są operacje:

1. Przeglądanie wszystkich partycji w poszukiwaniu informacji o puli dyskowej. Odczytywany jest pierwszy sektor każdej partycji z listy w /proc/partitions, w poszukiwaniu nagłówków/etykiet OpenGFS. Te nagłówki/etykiety muszą być wpierw zapisane przy pomocy narzędzi *ptool* lub *pgrow* w czasie konfiguracji puli dyskowej. Kilka partycji może należeć do tej samej puli dyskowej, mogą występować dyski lub partycje nie należące do OpenGFS.

2. Tworzony jest plik specjalny urządzenia /dev/pool/xxxx, jeden dla każdej dyskowej odnalezionej w kroku 1, gdzie xxxx jest nazwą puli z kroku 1.

3. Konfiguracja modułu jądra puli aby zapewnić odwzorowanie abstrakcyjnych urządzeń puli na fizyczne dyski/partycje.

Pula dyskowa jest widziana przez system jako pojedyncze urządzenie z ciągłą, liniową przestrzenią sektorów. Pojedyncza pula może obejmować małe partycje, jak również całe grupy dysków (jak np. matryce dyskowe). Odwzorowanie może łączyć woluminy lub dzielić dane pomiędzy kilka woluminów (paskowanie *striping*) w celu zwiększenia wydajności.
Pod–pula dyskowa to logiczna grupa dysków lub partycji wewnątrz puli dyskowej. Każda pod–pula jest widziana jako ciągła grupa sektorów (wewnątrz puli), od określonego sektora. Jako logiczna całość nie dozwolone są przerwy pomiędzy nimi (ostatni sektor poprzedniej pod–puli znajduje się bezpośrednio przed pierwszym sektorem następnej), jak również nie dozwolone jest dzielenie pod–puli na logiczne części. Fizycznie

może występować podział pomiędzy dyskami, z odwzorowaniem łączącym dyski lub dzieleniem danych pomiędzy woluminy (paskowanie).
Jedyną metodą metodą dostępu do pod–puli jest posłużenie się adresem sektora. Moduł puli dyskowej używa adresu sektora do odwzorowania I/O na odpowiednią partycją zawierającą pod–pulę.
Pod–pule mogą być użyte do:

- Podziału dużej liczby woluminów na mniejsze grupy wykorzystywane do paskowania (stripingu).
- Izolacji dzienników od danych przez przypisanie typu ogfs_journal pod–puli do oddzielnej partycji, a typu ogfs_data do innych woluminów.
  Pod–pule typu ogfs_data zawierają tylko dane.
  Pod–pule typu ogfs_journal zawierają tylko dziennik, system plików wymaga utworzenia co najmniej jednego dziennika dla każdego węzła.

1.4.1 Etykiety puli dyskowych

Jeśli wolumin jest częścią puli, to musi zawierać etykietę w pierwszym sektorze (512 bajtów) partycji. Ta etykieta identyfikuje pulę i pod–pulę dyskową do których należy ta partycja. Pojedyncza partycja (i musi być to cała partycja) może należeć tylko do jednej puli i jednej pod–puli.
Etykiety puli są zapisywane w woluminie przy pomocy narzędzia *ptool* przy konfiguracji wstępnej, lub przy pomocy narzędzia *pgrow* podczas dodawania pod–puli do istniejącej puli.
Narzędzia te posługują się plikiem konfiguracyjnym puli. Plik ten dla każdej puli zawiera:

- nazwę puli dyskowej,
- liczbę pod–puli wewnątrz puli.

Oraz dla każdej pod–puli:

- identyfikator pod–puli (ID, początkowo zero, zwiększany o jeden),
- rozmiar paskowania (stripe size) (liczba 512-bajtowych sektorów, zero w przypadku braku paskowania),
- typ (ogfs_journal lub ogfs_data),
- liczba woluminów w pod–puli.

Oraz dla każdego wolumin/partycji:

- Identyfikator woluminu (ID, początkowo zero, zwiększany o jeden),
- pod–pulę do którego przypisywany jest ten wolumin,
- nazwa woluminu widziana przez system operacyjny (np. /dev/hda1),
- waga DMEP dla tego woluminu.

Identyfikatory pod–puli i woluminów determinują układ puli dyskowej widziany przez system plików.
Etykiety są odczytywane z poszczególnych węzłów (podczas startu węzła, lub przez narzędzie pgrow podczas powiększania systemu plików), po zgromadzeniu etykiet z wszystkich węzłów tworzony jest całościowy obraz puli dyskowej jako ciągła liniowa przestrzeń odwzorowująca sektory partycji na sektory puli.
Wyszukiwane są również brakujące partycje przez sprawdzenie ciągłości identyfikatorów pod–puli i partycji oraz przez sprawdzenie czy jest odpowiednia liczba pod–pili i urządzeń.
Należy zauważyć że odwzorowanie nazwy systemowej woluminu (np./dev/hda1) odbywa się w każdym węźle niezależnie.
W przypadku zwiększania rozmiarów puli istniejące pod–pule muszą być pozostawione bez zmian, powiększenie jest realizowane przez dodanie pod–puli na końcu puli dyskowej.

1.4.2 Moduł jądra puli dyskowej

Przeznaczeniem modułu jest jest:

- Odwzorowanie bloków z urządzenia /dev/pool na urządzenia fizyczne. Pula dyskowa jest zaczepiona w podsystemie obsługi bloków jądra i tłumaczy żądania bloków podczas ich tworzenia.
- Implementacja ioctl do kontroli, monitorowania, statystyk itp. pula odpowiada na wywołania ioctl od narzędzi ptool i innych.
- Wysyłanie komend DMEP (*hardware-based file locking*) do urządzeń puli.

1.5 Ogólny układ systemu plików

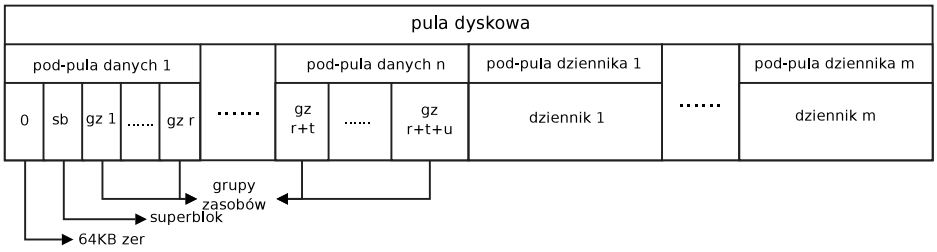
OpenGFS składa się z czterech składników najwyższego poziomu:

- super-bloku,
- grup zasobów,
- dzienników,
- 64KB nieużywanej przestrzeni na początku woluminu.

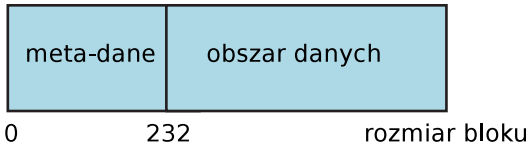
Dostępne bloki w każdej pod–puli są dzielone na grupy zasobów (resource groups). Pierwsza pod–pula zawiera pewne dodatkowe informacje (rys.1): super-blok zaczynający się z przesunięciem 64KB od początku pod–puli. Pozostała przestrzeń dzielona jest na grupy zasobów jak opisane jest niżej. Pierwsza grupa zasobów zawiera indeks dzienników i indeks grup zasobów(indeksy te mogą być rozłożone w wielu grupach zasobów przez użycie narzędzia *ogfs_expand*).

1.5.1 Rozproszone i-węzły (di-węzły)

OpenGFS przechowuje meta dane w i-węzłach, w terminologii OpenGFS nazywane one są di-węzłami (distributed inodes). Di-węzły są podobne do i-węzłów z innych systemów operacyjnych.



Rysunek 1: Budowa puli dyskowej.



Rysunek 2: Budowa di-węzła.

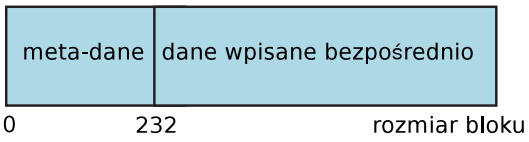
Każdy di-węzeł zajmuje cały blok, ma to swoje zalety po pierwsze prawdopodobieństwo iż wielu klientów będzie próbowało blokować ten sam blok jest mniejsze. Kolejną zaletą jest to że adres di-węzeł staje się jego unikalnym identyfikatorem. Z drugiej strony w pewnych przypadkach tracone jest dużo miejsca w porównaniu z innymi systemami plików.
Gdy tworzony jest system plików przy użyciu narzędzia mkfs.ogfs, generowane są tylko trzy di-węzły:

- di-węzeł indeksu dzienników — pierwszy blok w pierwszej grupie zasobów,
- di-węzeł indeksu grup zasobów — drugi blok w pierwszej grupie zasobów,
- di-węzeł katalogu głównego (root directory) — trzeci blok w pierwszej grupie zasobów.

Ponieważ rozmiar bloku jest ustalany w momencie tworzenia systemu plików, rozmiar di-węzłów może być różny. Stała cześć dostępnej przestrzeni jest zajmowana przez meta–dane, a pozostałą przestrzeń zajmują dane lub przez adres bloku. Meta–dane zajmują 232 bajty (rys.2). Ponieważ adres bloku zajmuje 64 bity (8 bajtów), oznacza to że liczbę adresów które można przechowywać w di-węźle określamy jako:
l.adresów = (rozmiar bloku di-węzą - 232):8
Czyli np dla bloku o rozmiarze 65536 bajtów, można przechowywać 8163 adresy.
Uwaga: Odpowiedni dobór rozmiaru bloków ma bardzo duży wpływ na ogólną wydajność systemu plików.

1.5.2 Zwalnianie di-węzłów

W przypadku zwolnienia di-węzła, blok trafia na początek listy wolnych di-węzłów w grupie zasobów i ustawiana jest flaga OGFS_DIF_UNUSED.
Uwaga: Gdy blok jest konwertowany na di-węzeł nie może on już przechowywać danych np gdy rozmiar bloku to 8192 i w katalog zawierający milion plików przechowywany jest w jednej grupie zasobów, di-węzeł zajmą 8GB. Przestrzeń ta nie



Rysunek 3: Budowa wypchanego (stuffed) di-węzła.

zostanie zwolniona nawet gdy pliki zostaną usunięte. Jedyną metodą na odzyskanie przestrzeni dyskowej jest użycie narzędzia *ogfs_tool*.

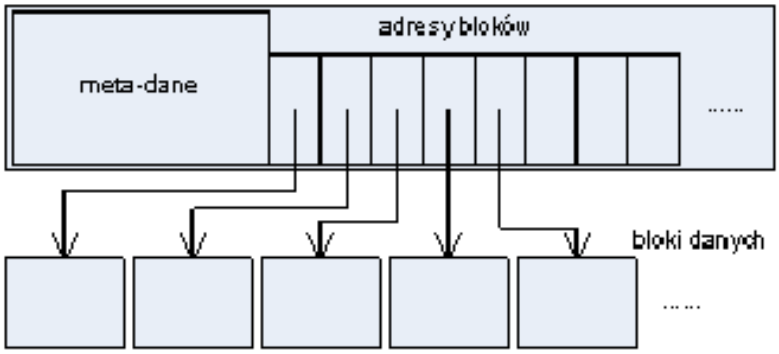
1.5.3 Tworzenie di-węzłów

W przypadku gdy di-węzeł jest potrzebny, wybierany jest pierwszy di-węzeł z listy wolnych di-węzłów z docelowej grupie zasobów. Jeśli nie ma wolnych di-węzłów, mapa bitowa grup zasobów (resource group bitmap) jest przeglądana w poszukiwaniu wolnego bloku meta–danych, w celu konwersji na di-węzeł. Gdy nie zostaną odnalezione wolne bloki, pierwsze 64 wolne bloki typu (OGFS_META_CLUMP) nie będące blokami meta–danych w grupie zasobów są konwertowane na typ OGFS_BLKST_FREEMETA i nowe di-węzły są tworzone w powstałej przestrzeni.
Decyzja o wyborze w której grupie zasobów będzie tworzony di-węzeł, podejmowana jest przed sprawdzeniem listy wolnych di-węzłów.

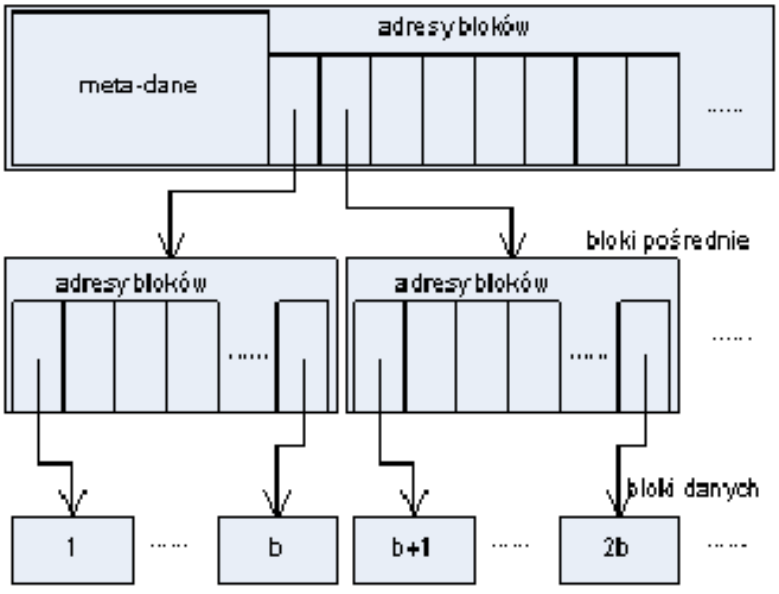
1.5.4 Pliki

Gdzie zachowywany jest plik zależy od tego czy jego rozmiar jest większy od (rozmiar bloku – 232) bajtów czy nie. Jeśli jest on wystarczająco mały, dane przechowywane są bezpośrednio w di-węźle (wypchany *„stuffed”* rys.3). Powoduje to zmniejszenie ilości czytanych bloków dla małych plików a szczególnie katalogów. Uwaga: Gdy plik zwiększy rozmiar przerastając wolne miejsce w di-węźle, dane są kopiowane do oddzielnych bloków (*„unstuffed”*). Jeśli plik raz został skopiowany z wnętrza di-węzła, nie zostanie umieszczony ponownie wewnątrz w przypadku gdy jego rozmiar jest wystarczająco mały. Jest to możliwe tylko jeśli zostanie on skasowany i zapisany ponownie.
Dla plików większych przestrzeń danych di-węzła zawiera bezpośrednie lub pośrednie odwołania (rys.4). Głębokość tych odwołań jest ograniczona przez teoretyczną maksymalną liczbę 2^{64} bloków. Adres bloku zajmuje 8 bajtów, oznacza to że w jednym di-węźle można zapisać do (rozmiar bloku - 232):8 adresów. Dodatkowe poziomy pośrednie są dodawane w miarę potrzeb (rys.5), ale dane są zawsze składowane na tym samym poziomie odwołania. Poniżej znajdują się tabele zawierające maksymalne rozmiary plików, dla podanych rozmiarów bloków (bs) i poziomu odwołania:

bs	stuffed	bezpośredni	pośredni	2x pośr.	3x pośr.	4x pośr.
512	280	17920	1120 KB	70 MB	4480 MB	280 GB
1024	792	101376	6336 KB	792 MB	99 GB	12672 GB
2048	1816	464896	29056 KB	7264 MB	1816 GB	454 TB
4096	3864	1978368	123648 KB	61824 MB	30912 GB	15456 TB
8192	7960	8151040	509440 KB	509440 MB	509440 GB	509440 TB
16384	16152	33079296	2019 MB	4038 GB	8076 TB	16152 PB
32768	32536	133267456	8134 MB	32536 GB	130144 TB	16 EB[1]
65536	65304	534970368	32652 MB	261216 GB	2089728 TB	16 EB[2]



Rysunek 4: Di-węzeł z bezpośrednim wskazaniem na bloki danych.



Rysunek 5: Di-węzeł z pośrednim wskazaniem na bloki danych.

1.5.5 Indeks grup zasobów

Indeks grup zasobów (Resource group index) jest specjalnym plikiem; di-węzeł nie znajduje się w żadnym katalogu lecz bezpośrednio w super-bloku. Istnieje jeden wpis w indeksie na każdą grupę zasobów w systemie plików. Indeks opisuje położenie i rozmiar grup zasobów, pozycję i numery bloków danych i rozmiar mapy bitowej bloków. W zależności od liczby grup zasobów i dostępnego miejsca w di-węźle, indeks jest przechowywany bezpośrednio w di-węźle lub w oddzielnych blokach.

1.5.6 Grupy zasobów

Informacje ogólne

Pod–pule dyskowe zawierające dane są podzielone na grupy zasobów (resource groups). Rdzeń systemu plików stara się trzymać bloki należące do tego samego pliku w jednej grupie, aby zmniejszyć potrzebną liczbę odwołań do dysku, oraz aby zmniejszyć liczbę blokad (locks) potrzebnych do ochrony danych.Grupa zasobów składa się z trzech komponentów: nagłówka, mapy bitowej bloków (block bitmap) i bloków danych.

Liczba grup zasobów w każdej pod–puli dyskowej danych zależy od liczby f dostępnych bloków pod–puli. Jeśli f jest równe lub większe od 573440 to liczba grup zasobów w tej pod–puli wynosi (f : 57344), zaokrąglając w dół. Jeśli f jest mniejsze to liczba ta wynosi 10. Pozostałe (f % 57344) bloki zostają przydzielone do pierwszej grupy zasobów danej pod–puli. Numeracja wszystkich grup zasobów jest ciągła zaczynając od 1 dla grupy zasobów znajdującej się bezpośrednio za superblokiem.

Nagłówek grup zasobów zawiera informacje o liczbie wolnych i zajętych bloków na dane, di-węzłów, meta–danych oraz listę nieużywanych di-węzłów.

Każdy z bloków jest reprezentowany przez parę bitów w mapie bitowej bloków. Niższy bit określa czy dany blok jest wolny (0) czy używany (1). Wyższy bit wskazuje czy blok (potencjalnie) zawiera meta–dane (1) czy normalne dane (0). Normalne dane czyli dane w di-węźle (*„stuffed”*) oraz wskazania pośrednie. Wszystkie inne bloki, włączając di-węzły, mają ustawioną flagę meta–dane.

Odwzorowanie pomiędzy bajtami mapy bitowej, a blokami jest liniowe tzn pierwszy bajt odnosi się do pierwszych czterech bloków, drugi bajt do drugiej czwórki itd. Niższe bity odpowiadają niższym numerom bloków: Mapa bitowa zaczyna się w pierwszym bloku grupy zasobów, bezpośrednio za nagłówkiem grup zasobów.

Możliwe kombinacje bitów:

bity	makro w C	znaczenie
00	OGFS_BLKST_FREE	wolny blok
01	OGFS_BLKST_USED	blok z danymi
10	OGFS_BLKST_FREEMETA	wolny blok meta–danych
11	OGFS_BLKST_USEDMETA	blok zajmowany przez meta–dane

Bloki danych

Bloki nie użyte przez nagłówek grup zasobów i mapę bitową są blokami danych. W przypadku gdy ilość bloków danych nie jest wielokrotnością czwórki, nadmierne bloki są nie używane.

Indeks dziennika

Indeks dziennika jest bardzo podobny do indeksu grup zasobów, zawiera informacje o dziennikach zamiast o grupach zasobów. Układ jest taki sam, różni się tylko formatem wpisu indeksu dzienników.

1.5.7 Dzienniki

Segmentacja dzienników

Dziennik każdego węzła jest podzielony na sekwencje bloków nazywanych segmentami. Liczba bloków w każdym segmencie jest określana podczas tworzenia systemu plików i musi wynosić co najmniej dwa. Domyślna wartość to 16. Wszystkie dzienniki mają ten sam rozmiar segmentu. W przypadku gdy liczba bloków dziennika nie jest wielokrotnością rozmiaru segmentu, nadmiarowe bloki przed pierwszym segmentem są obcinane i nie używane. Podobnie mogą być obcięta część bloków na końcu dziennika.
Segmenty są używane do zapisywania informacji o niedokończonych transakcjach zapisu meta–danych. Zapis danych nie jest logowany.

Segmenty

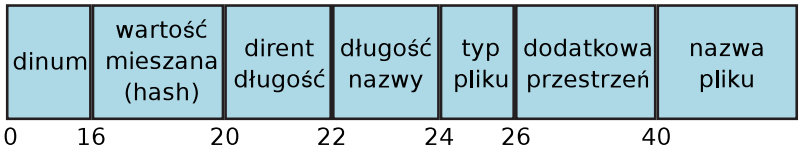
Segment jest sekwencją bloków z pewnego zakresu. Pierwszy blok jest blokiem nagłówkowym, a pozostałe bloki zawierają dane dziennika. Blok nagłówkowy zawiera strukturę ogfs_log_header_t na początku bloku i jej kopię na końcu bloku. Gdy system plików jest montowany, obie kopie są porównywane ze sobą, w przypadku gdy nie są identyczne, segment jest błędny. Poza meta–danymi opisującymi typ bloku, nagłówek zawiera numer sekwencji transakcji do której należy ten segment oraz flagę oznaczająca czy dziennik był poprawnie od-montowany.

Transakcje i wpisy w logach

Dla każdej operacji zapisu w systemie plików, tworzona jest w pamięci struktura transakcji oraz zapisywany jest do dziennika log ją opisujący. Dziennik jest zapisywany jako kolejka FIFO.
Dla każdej transakcji, wykonywane są akcje:

1. rezerwowane jest możliwie maksymalny obszar przestrzeni dysku dla transakcji,
2. blokady i bufory dla transakcji są zbierane w pamięci,
3. informacje o transakcji są zapisywane do logu,
4. znacznik końca jest zapisywany do logu,
5. operacje zapisu związane z transakcją są wykonywane,
6. wpis do logu jest zamykany.

Jeśli węzeł opuszcza klaster przed wykonaniem kroku (4), żadne dane nie zostały zapisane na dysk i nie potrzebne jest przywracanie dla tej transakcji. Chociaż może być konieczne zidentyfikowanie niekompletnego wpisu w logu. Za tym punktem, wpis w logu jest wykonywany ponownie w przypadku awarii węzła. o kroku (6) log może być ponownie zapisany (następuje nadpisanie) jeśli jest to konieczne.
Wpis w logu może się składać z jednego lub więcej segmentów. Nagłówek każdego segmentu zawiera numer, który jest zwiększany o jeden dla kolejnego wpisu. Początek logu jest odnajdywany podczas montowania dziennika, przez znalezienie wpisu z najwyższym numerem, takiego który jest kompletnie zapisany. Gdy wpis do logu jest zakończony, kolejny numer sekwencji jest wpisywany do następnego segmentu, jest to oznaczenie że wpis jest kompletny.



Rysunek 6: Struktura wpisu katalogu.

1.5.8 Katalogi

Każdy katalog w OpenGFS posiada di-węzeł, w którym zapisane są uprawnienia i właściciel katalogu oraz inne meta–dane. Jeśli katalog jest mały to umieszczany jest bezpośrednio w di-węźle, jeśli jest to niemożliwe nazwy plików są mieszane (hash) przy pomocy 32–bitowego algorytmu CRC. Tablica z mieszanymi nazwami jest przechowywana w di-węźle tak jakby była normalnym plikiem.
Katalog jest zapisany bezpośrednio w di-węźle jeśli rozmiar wszystkich wpisów (ogfs_dirent_t) nie przekracza (rozmiar bloku - 232). Jeśli katalog potrzebuje tabeli z mieszanymi nazwami, ale ma ona rozmiar mniejszy niż (rozmiar bloku - 232), to tablica jest zapisywana w di-węźle. Gdy tablica jest zbyt duża, zostaje zapisana w bezpośrednio lub pośrednio wskazywanych blokach danych.

Liście katalogów

Poza przypadkiem gdy katalogi są bezpośrednio zapisane w di–węzłach, wpisy katalogów są zapisywane w liściach. Meta–dane zawarte w liściu zawierają liczbę wpisów znajdujących się w liściu oraz inne informacje.

Wpisy katalogów

Zawartość wpisu katalogu pokazana jest na rys.6. Rozmiar wpisu katalogu (ogfs_dirent_t) jest rozmiarem nazwy pliku plus 40 bajtów nagłówka, zaokrąglone w górę tak aby suma była wielokrotnością 8 (makro OGFS_DIR_REC_LEN). Nazwa pliku nie jest zakończona zerem. Gdy wpis katalogu jest zwalniany, miejsce które zajmował jest dołączane do poprzedzającego go wpisu katalogu w tym samym liściu. Pierwszy wpis w liściu nie ma poprzednika. Zamiast tego numer di–węzła ustawiany jest na zero (ponieważ pierwszym wpisem w katalogu jest „.”, jest mało prawdopodobne że do tego dojdzie). Gdy wpis jest dodawany do wybranego łańcucha liści, łańcuch ten jest przeglądany w poszukiwaniu przydzielonego wpisu (lub wpisu którego di–węzeł ma numer zero), który ma wystarczająco dużo miejsca aby przechować nowy wpis. Jeśli taki wpis istnieje, nowy wpis jest umieszczany na początku wolnej przestrzeni w starym wpisie. Pozostała wolna przestrzeń należy do nowego wpisu.
Algorytm ten może prowadzić do fragmentacji przestrzeni w łańcuchu liści, a nawet prawie dwukrotnie zwiększyć przestrzeń zajmowaną przez katalog. Przestrzeń zajmowana przez łańcuch liści może tylko rosnąć, aby zwolnić zajmowane miejsce należy skasować katalog.
Jeśli nie znaleziono odpowiedniego liścia, tworzony jest nowy liść i cała wolna przestrzeń jest przydzielana dla nowego wpisu.

1.5.9 Specjalna budowa pierwszej grupy zasobów

W momencie tworzenia nowego systemu plików przy pomocy narzędzia *mkfs.ogfs*, pierwsza grupa zasobów ma specjalną budowę. Poza normalnymi danymi zawiera również specjalne di–węzły dla indeksu dziennika, indeksu grup zasobów i katalogu głównego (root directory). indeksy dziennika i grup zasobów są początkowo tworzone w pierwszej grupie zasobów, ale mogą być później rozszerzane na inne grupy zasobów przy użyciu *ogfs_expand*.
Pierwsze bloki danych są zarezerwowane dla trzech specjalnych di–węzłów i dwóch indeksów:

1.5.10 Generacja meta–danych

Wszystkie nagłówki meta–danych posiadają numer generacji (generation number). Numer generacji jest ustawiany na zero gdy do bloku zapisywane są meta–dane po raz pierwszy. Za każdym razem gdy (logowana) transakcja jest zapisywana na dysku, numer generacji w każdym bloku którego dotyczy transakcja jest zwiększany o jeden. Ponieważ numer generacji jest liczbą 64–bitową, przepełnienie jest mało prawdopodobne.
Generacje są używane tylko w czasie procesu odzyskiwania. Generacja transakcji w logu, dla węzła który uległ awarii, jest porównywana z aktualną generacją danych na dysku aby ustalić które transakcje muszą być powtórzone.

1.6 OpenDLM

Aktualnie OpenGFS używa OpenDLM jako rozproszonego zarządcy blokad

OpenDLM jest rozproszonym zarządcą blokad. Wiedza o blokadach jest rozproszona wśród zainteresowanych węzłów w klastrze i każdy węzeł posiada i wykonuje operacje żądania i przydzielania blokad.
W OpenDLM nie ma centralnego serwera blokad i nie ma centralnego urządzenia składującego dane blokad. Dlatego nie ma SPOF (Single Point Of Failure).
Jest on wydajny w pracy pomiędzy węzłami. Opiera się na schemacie zapytaj – zażądaj – przydziel. Nie ma tu ciągłego pobierania z serwera jak w memexp, co generuje duży ruch w sieci w przypadku gdy blokada nie jest dostępna.

1.6.1 Żądanie i przyznawanie blokad

Wiedza o blokadzie jest podzielona na trzy części:

- katalog – węzeł pytany,
- władca (master) – węzeł który posiada informacje o stanie blokady,
- żądający/Przyznający – węzeł który posiada lub chce posiadać blokadę.

Gdy węzeł potrzebuje blokady najpierw pyta węzła katalogowego. Każdy węzeł w klastrze służy jako węzeł katalogowy dla pewnego zakresu nazw blokad. Sposób mapowania nazw jest dobrze znany w klastrze więc węzeł wie który katalog należy odpytać. Algorytm mapowania opiera się na prostej funkcji przetwarzającej nazwę blokady (np. pierwsza litera nazwy blokady).
Katalog wie czy było wcześniej żądanie dotyczące tej blokady i który węzeł jest

władcą tej blokady. Jeśli nie było wcześniej żądania dotyczącego danej blokady katalog przyporządkowuje pytającemu węzłowi własność władcy blokady i tworzy nowy wpis w katalogu dla nowej blokady. Wpis w katalogu zawiera tylko określenie władcy dla blokady. Normalnie, władca pozostanie nim na zawsze i wpis nie ulegnie zmianie. Nowy władca przydziela sobie właściwość posiadania blokady.
Jeśli żądanie dotyczące danej blokady wystąpiło wcześniej katalog wskazuje pytającemu kto jest władcą blokady. Następnie pytający kieruje zapytanie do władcy.
Władca przechowuje stan blokady (który węzeł posiada blokadę i w jakim trybie) i jest zarządcą tej blokady.
Należy zauważyć że ten sam węzeł może być katalogiem, władcą i posiadaczem blokady.

1.6.2 Tryby blokowania

W OpenDLM istnieje 6 trybów blokad. Są one używane w strukturze hierarchicznej rodzic-dziecko:

- EX, tryb wyłączności, blokada zapisu,
- PW, chroniony zapis, blokada aktualizacji,
- PR, chroniony odczyt, blokada współdzielonego odczytu,
- CW, jednoczesny zapis,
- CR, jednoczesny odczyt,
- NL, null, bez blokady.

Najbardziej restrykcyjny jest tryb EX, najmniej – NL. Jeśli żądana blokada ma bardziej restrykcyjny tryb (lub równy w niektórych przypadkach), nie zostanie przyznana dopóki nie zostaną zwolnione wszystkie blokady mniej restrykcyjne.
Tryb NL oznacza brak blokady, ale niektóre węzły przetrzymują blokadę, aby jej władca i katalog wiedziały o jej istnieniu.
Tryby CR i CW są używane z hierarchią rodzic-dziecko, w przypadku których duże obszary pamięci, katalogi lub inne duże zasoby są chronione. Ponieważ blokada CW powinna zezwalać na jednoczesny dostęp do dużych obszarów, nie powinna bezpośrednio blokować zapisu. Zamiast tego powinny być tworzone blokady (PW lub EX) o mniejszej granulacji i określane jako dzieci blokady na dużym obszarze. Te mniejsze blokady są żądane jako dzieci. Podczas zwalniania blokady rodzica, zwalniane są również wszystkie blokady dzieci. Zapewnia to możliwość łatwego zarządzania blokadami na dużych obszarach i blokad na obszarach zawartych wewnątrz nich.

1.6.3 Stany blokad

Zasób blokady (lock resource) jest obiektem, który zawiera informacje o stanie blokady w całym klastrze i jest związany z konkretną nazwą blokady. Dany zasób blokady chroni jeden obiekt (plik, strukturę danych) i jest unikalnie identyfikowany przez nazwę.
Blokada może być w jednym z trzech stanów:

- przydzielona, blokada jest w węźle który jej zażądał,
- konwersja, oczekiwanie na przyznanie bardziej restrykcyjnego trybu,
- zablokowana, oczekiwanie na przydział.

Dla każdego stanu istnieje kolejka związana z zasobem blokady zawierająca identyfikator procesu/węzła, który posiada blokadę w danym stanie (oczywiście tylko dla tego zasobu).

1.6.4 Awaria i odzyskiwanie (recovery)

OpenDLM korzysta z zewnętrznej usługi aby określić czy węzeł jest aktywny. Gdy zarządca klastra wykrywa wyłączenie węzła reszta klastra musi odzyskać blokady które były: w posiadaniu, władzy lub katalogu wyłączonego węzła. Wiele zależy od tego czy w momencie wyłączenia węzła istnieją zainteresowane blokadami inne węzły. Jeśli nie blokady przepadają, a pojawienie się żądania traktowane jest jak nowe.

Odzyskiwanie katalogu

Gdy OpenDLM wykrywa odłączenie/dołączenie nowego węzła, na nowo rozdziela katalogi. Może to oznaczać kopiowanie informacji lub przydzielenie odpowiedzialności za pewien zakres nazw. Wpisy katalogów muszą być odpowiednio przebudowane.

Odzyskiwanie władcy

O zasobie blokady może wiedzieć więcej niż jeden węzeł. Węzeł, który posiada blokadę lub czeka na jej przydział, wie który węzeł jest jego władcą i jest informowany który węzeł został wyłączony. Gdy węzeł-władca jest wyłączany posiadacz/żądający blokady węzeł kieruje zapytanie do katalogu. Katalog przydzieli właściwość władcy pierwszemu węzłowi pytającemu tak jakby było to pierwsze żądanie.

Posiadacz blokady

Jeśli wyłączany jest posiadacz blokady, władca zakłada zwolnienie blokady.

1.7 Mechanizmy blokowania

Głównymi mechanizmami, które mają zapewnić spójność danych i wyłączny dostęp do nich są:

- Moduły blokujące (lock modules) — odpowiadają za zachowanie spójności danych i systemu plików przy operacjach pomiędzy węzłami, gdy różne węzły odwołują się do rozproszonego systemu plików.
- Warstwa „G-Lock” — odpowiada za zarządzanie dostępem do systemu plików oraz struktur danych w pamięci węzła przez wiele procesów (zapobiega przed równoległym dostępem do nich).

Dodatkowym mechanizmem jest uprząż (lock harness). Celem istnienia tego mechanizmu jest:

- przechowywanie listy modułów, które można zamontować,
- dołączanie wybranych modułów do systemu plików w etapie montowania w protokole blokowania (lock protcol).

Po etapie montowania w protokole blokowania, praca modułu uprzęży (harness module) jest zakończona.
Moduły blokujące i centrum blokad (lock storage facility) odpowiadają za:

- zarządzanie i przechowywanie blokad oraz LVBs (lock value blocks) międzywęzłowych,
- wygasanie blokad (lock request timeout) oraz wykrywanie zakleszczeń,
- nadzór nad aktywnością węzłów (Czy węzeł działa, czy nie jest uszkodzony ?),
- separuje węzły, odtwarza blokady i powtarza operacje dzienników w przypadku awarii węzłów.

Warstwa „G-Lock" jest częścią kodu systemu plików, odpowiada za:

- koordynację i kieszeniowanie (cacheing) blokad i LVBs pomiędzy procesami jednego węzła,
- komunikację z modułami blokującymi w celu zapewnienie blokad pomiędzy węzłowych,
- wykonywanie glops (patrz niżej) gdy są potrzebne,
- powtórzenia operacji dziennika w przypadku awarii.

Warstwa glops (G–Lock Operations — operacje warstwy G-Lock) jest również częścią kodu systemu plików. Implementuje ona szereg działań specyficznych dla: architektury, systemu plików i chronionych elementów, które muszą być podjęte przed lub po użyciu blokad takie jak:

- odczyt danych z dysku, lub z innego węzła poprzez LVB (Lock Value Block), po założeniu blokady,
- zapisywanie na dysku lub innym węźle, przed zdjęciem blokady,
- unieważnianie danych w buforach jądra po zapisie, tak aby proces nie odczytywał ich zawartości w chwili gdy inny proces zmienia ich zawartość.

Każda blokada posiada dołączone glops (G – Lock Operations) odpowiedniego typu. Punkt dołączania jest kluczem do przeniesienia systemu blokad do innych środowisk lub tworzenia blokad innego typu z operacjami specyficznymi dla nich.

1.7.1 Moduły blokujące i uprząż

Uprząż

W momencie montowania system plików używa uprzęży do rejestracji protokołu blokowania. Powoduje to udostępnienie usługi modułu blokowania dla systemu plików. Możliwe jest użycie wielu protokołów blokowania oraz wielu przestrzeni blokowania (lockspaces) dla jednego protokołu przy montowaniu wielu systemów plików OpenGFS na jednym komputerze. Przestrzeń blokowania jest kombinacją urządzenia informacji o klastrze (Cluster Information Device (cidev)) i instancji protokołu. Węzeł montujący trzy oddzielne systemy OpenGFS używające jednego protokołu *memexp* , będzie potrzebował trzech różnych atrybutów cidev.

Protokół jest usuwany przy od montowaniu systemu plików. Do tej operacji nie jest potrzebna uprząż, ponieważ system OpenGFS ma bezpośredni dostęp do zarejestrowanego modułu.

Moduły blokujące

Moduły blokujące są implementacją G-Locks i jądrze. Każdy zapewnia inny protokół blokowania, który może być użyty w systemie OpenGFS:

- locking/modules/memexp — zapewnia blokady międzywęzłowe, do użycia w klastrze,
- locking/modules/nolock — zapewnia „udawanie" blokad, nie należy używać w klastrze,
- locking/modules/stats — zapewnia statystykę, umieszczany nad innym protokołem.

1.7.2 Protokół memexp

Protokół memexp zapewnia operacje klastrowe i jest dość złożonym protokołem. Moduł memexp w każdym węźle śledzi przynależność do klastra oraz informacje który klaster posiada poszczególne blokady. Jest to możliwe przy współpracy z centrum blokad (central lock storage facility) oraz sieci LAN.
Za każdym razem gdy węzeł potrzebuje blokady musi wykonać kroki:

1. załadować blokadę z centrum blokad,
2. sprawdzić status blokady,
3. zmienić status blokady tak by wskazywała że dany węzeł jest w jej posiadaniu,
4. zapisać blokadę w centrum blokad.

Jeśli w kroku 2) blokada nie jest odpowiednia, nie jest nieużywaną blokadą, to węzeł musi potarzać kroki 1) i 2) do momentu otrzymania blokady o właściwym stanie. Aktualna implementacja to ciasna pętla, może to powodować duży ruch w sieci.
Krok 4) zapewnia również wykonanie operacji wczytania i zapisu jako jednej. Możliwe jest to dzięki użyciu numeru sekwencyjnego nadawanego parze operacji. Jeśli w centrum blokad numer ten jest inny, niż wykonywana operacja zapisu, to znaczy że inny węzeł odwołał się do tej blokady w międzyczasie i należy powtórzyć całą sekwencję operacji od początku.
Pierwszym działaniem podejmowanym przez moduł memexp jest jest odczytanie informacji o klastrze z urządzenia cidev (Cluster Information Device). Cidev jest urządzeniem składującym (lub dedykowaną partycję), oddzielony od systemu plików, dzienników oraz centrum blokad, posiadającym informacje o wszystkich węzłach klastra.
Przy tworzeniu systemu plików OpenGFS, tworzone są (na podstawie konfiguracji) i zapisywane do cidev informacje o węzłach przy pomocy narzędzia *ogfsconf*.
Każdy węzeł odczytuje informacje o wszystkich węzłach klastra z urządzenia cidev, i przechowuje je w statycznej liście.

1.7.3 Centrum blokad

Protokół memexp wymaga centralnego repozytorium danych, jest to urządzenie DMEP (Device Memory Export Protocol) lub serwer „udający" DMEP (memexpd). Serwer memexpd emuluje operacje DMEP składując dane w pamięci lub na własnym dysku. Jest on wydzielony od systemu plików, dzienników oraz urządzenia cidev.
Serwer memexpd jest zorganizowany jako zbiór buforów o równych rozmiarach. Domyślny rozmiar to 32 bajty (może być zmieniony przy pomocy opcji -b). Bufory danych stanowią 98% pamięci zajmowanej przez memexpd (domyślnie zajmuje on 32 MB), wszystkie elementy to:

- tablica mieszająca (hash table) — odwzorowuje ID buforów (bid, aktualnie kombinacja 8–bitowego typu blokady i 64–bitowego numeru blokady) na fizyczny numer bufora (pbn, lokalizacja w tablicy danych memexpd),
- tablica elementów (element table) — status, dowiązań do tablicy najczęściej używanych (MRU, Most-Recently-Used) oraz dowiązań do tablicy mieszającej dla każdego bufora danych,
- tablica danych (data table) — 32–bitowe bufory składujące blokady,
- tablica IP — adresy IP węzłów w klastrze,
- tablica usuniętych — lista („martwych węzłów") węzłów odizolowanych od memexpd.

Jeden element (35 bajtów) oraz jeden bufor danych (32 bajty) jest potrzebny na blokadę lub LVB. Oznacza to że w 32MB można przechować prawie pół miliona blokad.

Protokół Nolock Protokół nolock zapewnia lokalne operacje w systemie plików na tym samym węźle. Nie ma tu centralnego urządzenia składującego, dane blokad są przechowywane w pamięci w tablicy mieszającej.

Protokół Stats
Protokół stats zapewnia operacje statystyczne (np. liczbę odwołań do niektórych funkcji, liczbę blokad w systemie) o protokole znajdującym się „poniżej" na stosie. Statystyka jest wyświetlana gdy moduł jest odłączany.

1.7.4 G–Lock blokady globalne

Warstwa G–Lock jest abstrakcyjną warstwą mechanizmu blokowania używaną przez system plików. Dostarcza ona interfejsów usług które wspierają:

- użycie jednego z wielu protokołów blokowania (modułów blokujących),
- wykonanie specyficznych akcji dla systemu plików lub chronionych elementów (glops),
- zależności i relacje dziecko–rodzic, które może wprowadzić system plików.

interfejsy warstwy G–Lock zawierają:

- usługi G–Lock, dołączane do kodu systemu plików,
- interfejsy modułów blokujących, pomiędzy warstwą G–Lock i modułami blokującymi,
  - komendy blokad oraz LVBs do modułów blokujących,
  - funkcje zwrotne (callback) z modułów blokujących z rozdaniami zwolnienia blokad, powtórzenia operacji dzienników,
- uchwytów glops do instalowania akcji specyficznych dla systemu plików i typu blokady.

Warstwa G–Lock pośredniczy pomiędzy warstwą systemu plików a częścią wspomagającą (backend). Warstwa G-Lock decyduje czy blokada jest już w węźle czy musi zostać pobrana przy użyciu protokołu blokad.
Warstwa G-Lock zawiera wsparcie dla zależności wewnętrznych, relacji dziecko–rodzic, własności procesów i innych.
G-Lock posiada również, bardzo ważną ze względu na wydajność, zdolność przetrzymywania (caching) blokad. Blokada trafia do procesu, a następnie jest zwalniana przez ten proces, ale nie jest ona od razu zwracana. Zamiast tego jest ona przez pewien określony czas przetrzymywana w kieszeniach warstwy G–Lock (chyba że inny proces zażądał tej blokady).

Wygasanie i odzyskiwanie blokad

Moduł blokad jest odpowiedzialny za wykrywanie nieaktywnych węzłów jest to zapewnione przez protokół memexp. Nie istnieje oznaczanie blokad czasem i nie ma ograniczeń czasowych co do długości wykonywanych operacji, dla których pobrano blokadę.
Gdy węzeł jest nieaktywny wszystkie blokady tego węzła, które były w trybie dzielonym (odczyt) są zwalniane, a te które były w trybie wyłączności (zapis) są oznaczane jako wygasłe. Jest to wykonywane przez inny węzeł, który wykrył nieaktywność. Gdy wszystkie blokady zostały zwolnione lub oznaczone możliwe jest powtórzenie operacji dziennika nieaktywnego węzła.

1.7.5 Użycie G–Lock w OpenGFS

System plików używa mechanizmów G–Lock do ochrony różnego rodzaju struktur przed jednoczesnym dostępem z wielu węzłów. Chronionymi obiektami mogą być: di–węzły (włącznie z di-węzłami indeksów dziennika i grup zasobów), nagłówki grup zasobów, super-blok, bloki i dzienniki. Dodatkowo G–Lock używany jest do blokad związanych z koordynacją klastrem (non-disk locks).
Poniżej przedstawione są strategie ochrony dla różnych komponentów:

1. Specjalne blokady (non-disk locks) wszystkie typu LM_TYPE_NONDISK :
   - OGFS_SB_LOCK – chroni dostęp do odczytu super-bloku przed jego zapisaniem.

- OGFS_MOUNT_LOCK – umożliwia zamontowanie systemu plików tylko przez jeden węzeł. Zakładana w trybie wyłączności, z nie związanymi z dyskiem (non-disk) glops, podczas montowania. Zdejmowana gdy montowanie jest zakończone umożliwiając montowanie kolejnego węzła.
- OGFS_LIVE_LOCK – Zakładana w trybie współdzielonego dostępu, z nie związanymi z dyskiem (non-disk) glops, podczas montowania. Zdejmowana podczas od montowania. Oznacza że co najmniej jeden węzeł ma zamontowany system plików.
- OGFS_TRANS_LOCK – chroni operacje odtwarzania dziennika przed nowymi transakcjami. Używany w trybie współdzielonym przez transakcje, dlatego może być tworzonych wiele transakcji równolegle.
  Używany w trybie wyłączności przez *ogfs_recover_journal()* do poinformowania wszystkich węzłów o konieczności zakończenia transakcji, przed rozpoczęciem odzyskiwania (journal recovery) i powstrzymująca przed rozpoczęciem nowych transakcji. Daje to wyłączny dostęp do systemu plików dla procesu odzyskiwania.
- OGFS_RENAME_LOCK – ochrona przed wzajemnym wpływem w operacjach zmiany nazwy (przeniesienia) plików (katalogów), zawsze w trybie wyłączności.

2. Unikalne blokady:

   - Indeks grup zasobów (Resource Group Index) — chroni odczyty indeksu grup zasobów przed zwiększeniem (zmniejszeniem) rozmiaru systemu plików. Nie można zwiększyć rozmiaru systemu plików dopóki wszystkie węzły nie zdejmą tej blokady, dlatego wszystkie muszą być tymczasowe. Każdy dostęp do indeksu grup zasobów, przy normalnej pracy sytemu plików, jest dostępem do odczytu w trybie współdzielonym (blokada jest założona na di-węźle indeksu grup zasobów).
   - Indeks dzienników (Journal Index) — chroni odczyty indeksu dzienników przed dodawaniem (usuwaniem) dzienników. Nie można dodać (usunąć) dziennika dopóki wszystkie węzły nie zdejmą tej blokady, dlatego wszystkie muszą być tymczasowe.

3. Blokady di-węzłów. Blokowanie w trybie współdzielenia do odczytu oraz blokowanie w trybie wyłączności dla zapisu.

4. Blokowanie grup zasobów. Pliki i di-węzły należą do grup zasobów. Plik może należeć do wielu grup zasobów. Podczas wykonywania operacji na plikach lub di-węzłach wszystkie grupy zasobów, w których znajdują się bloki danych lub bloki meta–danych muszą być blokowane.
   Ponieważ grupy zasobów są rozproszone na dysku, wczytywanie ich do pamięci, za każdym razem gdy następuje do nich odwołanie, powodowałoby bardzo duży spadek wydajności. Jednakże tylko niewielka część nagłówków grup zasobów jest kiedykolwiek zmieniana są to dane statystyczne. Dlatego aby uniknąć odczytywania tych danych, w momencie montowania statystyki są składowane w LVB (lock value block). Gdy węzeł klastra zmienia grupy zasobów zapisuje statystyki go na dysk oraz do LVB warstwy G–Lock. Następny węzeł potrzebujący tej blokady może ją odczytać z G–Lock zamiast z dysku.

5. Blokowanie super-bloku. Super-block jest odczytywany i zapisywany tylko podczas montowania i od montowania systemu plików i tylko w tych momentach jest blokowany.

6. Dzienniki nie muszą być chronione ponieważ są one używane tylko przez jeden węzeł za każdym razem.

2 Lustre

2.1 Wstęp

Lustre jest rozproszonym skalowalnym systemem plików aktualnie dostępnym tylko na platformie Linux. Używa on zorientowanych obiektowo urządzeń dyskowych (Object-Based (Storage) Devices – OBD's, OSD's, OBSD's). System zapewnia brak pojedynczych punktów awarii (single points of failure), szybki start, dużą wydajność. Zawiera połączenie właściwości, które można zaobserwować w wielu rozproszonych systemach plików min. AFS, Coda, InterMezzo, Locus CFS.

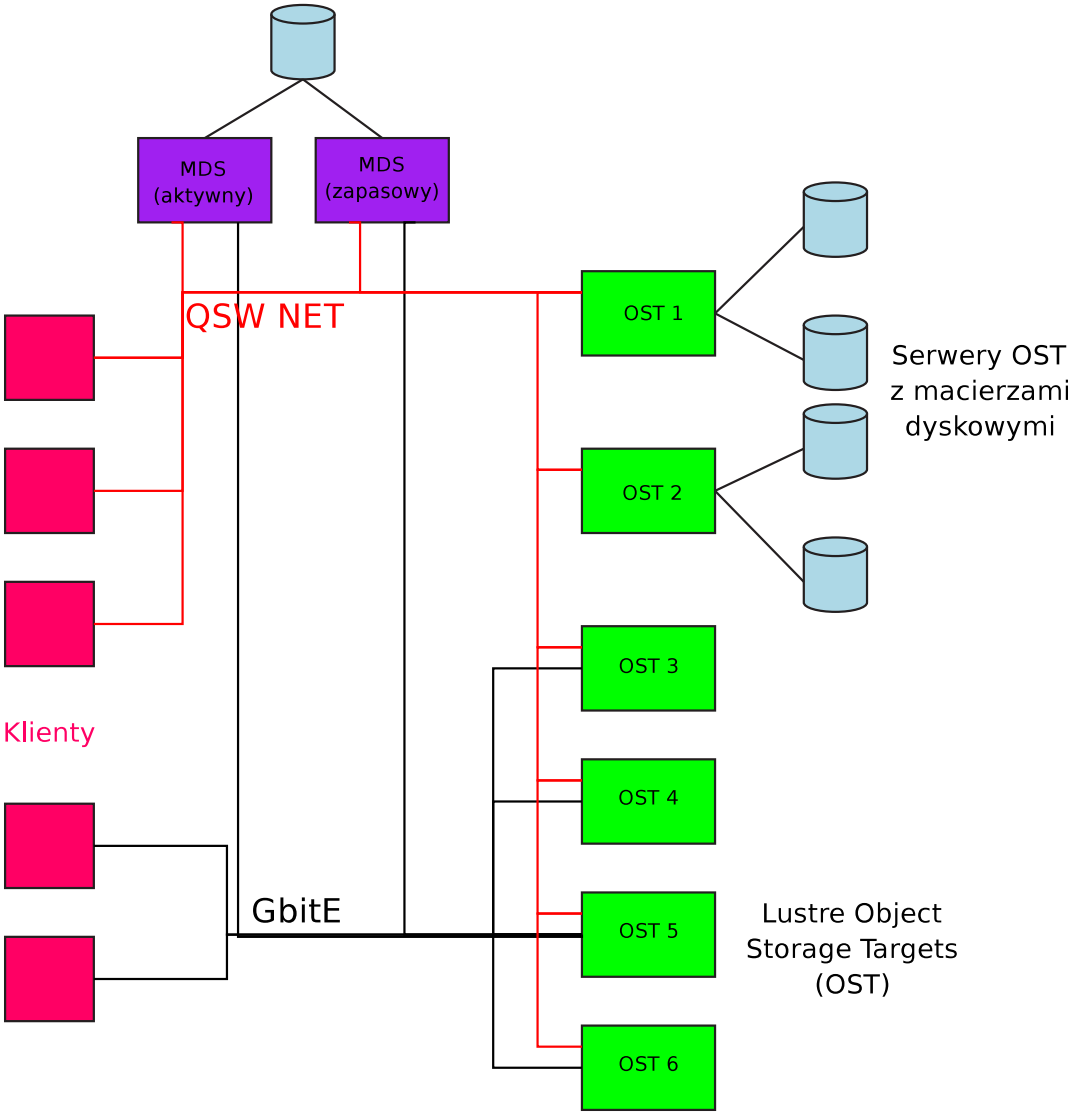
2.2 Podsystemy i protokoły

Lustre składa się z trzech głównych podsystemów i kilku systemów pobocznych (rys.7). Głównymi podsystemami są:

- system plików klienta (Client FileSystem CFS),
- serwer meta–danych (Meta-Data Servers MDS),
- składnice obiektów (Object Storage Targets OST).

Każdy z tych podsystemów może działać na osobnej maszynie, ale możliwe jest również uruchomienie kilku podsystemów na jednej maszynie (tak mówi teoria w dokumentacji technicznej, ma prezentacji wdrożenia w firmie ATM wersji 1.2 Lustre, było powiedziane że są problemy z uruchomieniem OST i CFS na jednej maszynie). Poszczególne podsystemy komunikują się za pomocą specjalnych protokołów:

1. **Klient – OST**: klienty komunikują się na zasadach klient-serwer z OST aby wykonać bezpośrednie operacje wejścia/wyjścia (I/O). Blokowaniem plików zarządzają usługi OST w połączeniu z usługą odwołania po stronie klienta. OST zapewnia również politykę bezpieczeństwa. Awaria jednej ze stron powoduje włączenie protokołów odzyskiwania, które obejmują użycie dzienników oraz przywrócenie lub odwołanie blokad.
2. **Klient – MDS**: żądania zmiany w przestrzeni nazw są kierowane do MDS. Protokół ten dynamicznie przystosowuje się do obciążenia systemu plików. Gdy klient ulega awarii włączany jest protokół odzyskiwania podobny jak w przypadku komunikacji klient – OST. Zmiany w przynależności do klastra powodują najpierw odzyskiwanie MDS, a dopiero później klientów i OST.
3. **OST – MDS**: protokół typu klient-serwer, umożliwiający OST (klient w tym protokole) aktualizację atrybutów oraz informacji o obciążeniu i dostępnych zasobach.
4. **MDS – MDS**: komunikacja jest podobna do używanej w klastrach VAX, włączając rozproszone zarządzanie blokadami, przemianami w klastrze.
5. **OST – OST**: komunikacja będąca częścią zarządzania składnicami, w celu migracji obiektów bezpośrednio pomiędzy OST.



Rysunek 7: Architektura systemu Lustre.

2.2.1 Składnice obiektów OST

Lustre wykorzystuje pojęcie składnic obiektów (object storage). Obiekt może być kojarzony jako i–węzeł i służy do przechowywania plików. Dostęp do tych obiektów jest możliwy przez OST, który zapewnia operacje wejścia/wyjścia dla plików w klastrze. OST wykonuje przydział bloków dla obiektów, co prowadzi do skalowalnego rozproszonego przydziału meta–danych. OST zapewnia również bezpieczny dostęp do obiektów przez klientów.
OST zapewnia interfejsy sieciowe do połączenia z innymi składnicami obiektów. Częścią składującą dane jest dysk obiektowy (Object–Based Disk OBD).
Na samym dole stosu znajduje się sterownik urządzenia obiektowego, bezpośrednio pośredniczący z urządzeniem fizycznym. Jest to bezpośredni sterownik OBD (direct OBD driver). bezpośredni sterownik może być częścią urządzenia (dysku) lub być symulowanym oprogramowaniem nad urządzeniem blokowym. Urządzenie kontrolowane przez sterownik to OBSD (Object-Based Storage Device). Na szczycie stosu znajduje się system plików lub specjalizowane aplikacje takie jak obiektowe bazy danych lub optymalizowane serwery plików. System plików i aplikacje będą nazywane SOA (Storage Object Applications).
Pomiędzy warstwą SOA a warstwą bezpośrednich sterowników można utworzyć różne pośrednie lub logiczne sterowniki. Przykładem może być agregacja obiektów na wielu urządzeniach składujących (np RAID).
Każde urządzenie ODB posiada tablicę metod, które umożliwiają interakcje z obiektami. Metody te służą do zainicjowania urządzenia i pobrania z niego obiektów. Struktury danych sterowników są zdefiniowane w *struct obddev*. Struktura ta jest analogiczna do super-bloku w innych systemach plików: odnosi się do OBD specyficznego typu. Zawiera dane generyczne dla urządzeń OBD, metody specyficzne dla danego OBD, i dane specyficzne dla tego OBD.
Urządzenia OBD są jednoznacznie identyfikowane przez UUID, któremu może towarzyszyć ciąg znaków rozpoznawalny przez użytkownika. Dyski logiczne oznaczają zależności używając UUID i są identyfikowane przez UUID. Aby przestrzeń nazw nie była jednolita, zaproponowano podział na grupy. Stąd UUID nie jest bezpośrednim odwzorowaniem, lecz umożliwia również rozpoznanie do jakiej grupy należy.
Bardzo ważnym aspektem przy architekturze obiektowych składnic jest poprawne posługiwanie się meta–danymi. W systemie, OBDO jest strukturą reprezentującą obiekt. W jądrze systemu operacyjnego i-węzły są samo opisującymi się strukturami. Identyfikacja pliku, możliwe na nim operacje i związane dane takie jak kieszenie, super-blok wszystkie są częścią struktury di-węzła. Struktura OBDO powinna mieć te same właściwości. Powinna zawierać przynajmniej:

- metody do zapisu/odczytu atrybutów,
- metody do zapisu/odczytu danych,
- metody do odłączenia (unlink) obiektu,
- połączenie do *obd_device*, który zawiera wskaźnik do struktury urządzenia.

Struktury OBDO posiadają typ, tak jak i-węzły. W i-węźle znajduje się obszar U (U area) zawierający informacje unikalne dla danego sytemu plików. Podobnie OBDO ma obszar U, którego zawartość jest inaczej interpretowana w zależności od rodzaju sterownika OBD.

2.2.2 Usługi meta–danych MDS

MDS przechowuje meta–dane oraz zapewnia ich odnawianie transakcyjne przez interfejsy sieciowe. MDS zawiera moduły blokujące (locking modules), i mocno wykorzystuje istniejące możliwości systemów plików z dziennikami. W Lustre Lite istnieje tylko jeden serwer meta–danych, mimo tego system unika pojedynczego punktu awarii, stosując istniejące rozwiązania np Kimberlite.
W pełnej implementacji Lustre obciążenie przetwarzaniem meta–danych będzie równoważone. API stworzone do obsługi meta–danych zapewnia jednolitą widoczność elementów w systemie plików. Wymaga to, aby wszystkie klienty miały jednolitą przestrzeń nazw drzewa katalogów systemu plików. I-węzły opisane przez przestrzeń nazw to obiekty jednoznacznie identyfikowane przez *identyfikatory plików fid* dostarczane przez MDS. Elementy budujące przestrzeń nazw są trójkami złożonymi z:

- katalog rodzica, który jest identyfikowany przez fid,
- nazwę,
- fid i-węzła wskazywanego przez nazwę.

Ważnymi wymaganiami dotyczącymi atrybutu fid jest:

- fid jest używany najwyżej raz (nie używa się go ponownie),
- i-węzeł jest jednoznacznie identyfikowany przez fid.

Jednoznaczne identyfikatory są przekazywane dla klientów ponieważ ma to ważne konsekwencje przy odzyskiwaniu.

Synchronizacja elementów przestrzeni nazw
Lustre zapewnia kilka mechanizmów zapewniających synchronizację przestrzeni nazw pomiędzy wielu klientów. W pierwszym modelu, każdy element jest zarządzany oddzielnie. Klient może użyć elementu przestrzeni nazw tylko gdy założył blokadę na nim i blokada gwarantuje w przypadku kieszeniowania element jest wciąż aktualny. Aby zapobiec wyścigom występującym przy odwołaniach blokad, Lustre wymusza uzyskanie blokady przed użyciem danych związanych z blokadą. Blokady są uzyskiwane przez pojedyncze wywołanie RPC.
Drugim modelem jest *write back model*. Pojedyncza blokada chroni całe poddrzewo. W celu stworzenia takiej blokady używana jest relacja rodzic/dziecko do odszukania poprzednika, który posiada blokadę. Aby powołać taką blokadę należy zdjąć blokady z wszystkich elementów poddrzewa.

Atrybuty i-węzła
Oprócz identyfikatora fid i-węzeł posiada również atrybuty:

- atrybuty ochrony (protection attributes) — właściciel, właściciel grupy, bity trybu oraz listy kontroli dostępu (access control lists) są atrybutami używanymi do autoryzacji,
- zawartość katalogu,
- rozmiar i czas modyfikacji,

- atrybuty lokalizacji — plik może być zapisany w wielu obiektach, i-węzeł meta–danych pliku przechowuje deskryptor lokalizacji,
- rozszerzone atrybuty (extended attributes) — Lustre umożliwia aplikacjom przechowywanie dodatkowych atrybutów w i-węzłach,
- cel linków symbolicznych.

Protokoły blokad MDS

MDS zapewnia dwa różne protokoły blokowania. W trybie *write-back* pojedyncza blokada chroni całe poddrzewo i jest przekazywana tylko jednemu klientowi. W trybie klient–serwer obiekty mogą być współdzielone przez wiele węzłów. Aby wybrać tryb pracy:

- Domyślną blokadą przydzielaną jest blokada zapisu lub odczytu poddrzew. Jeśli zasoby związane z obiektem wykryły duże wykorzystanie blokad w ostatnim czasie blokada w trybie klient-serwer jest przydzielana.
- Gdy ta blokada jest przydzielana inne blokady dotyczące tego samego obiektu są odwoływane.
- Jeśli klient przejdzie całe poddrzewo poniżej blokady *write-back* i wypełni jego kieszeń, inne blokady muszą być odwołane

Uaktualnienie rekordów związanych z blokadą poddrzewa może być przesłane jako „paczka”, nie ma potrzeby potwierdzać każdej operacji oddzielnie.

2.2.3 Klient

Podsystem klienta używa kiszeni stron (page cache) Linuxa 2.4/2.5. Kieszenie te co pewien czas lub na żądanie zapisują dane. Dane te są przesyłane do sterownika logicznego obiektowego woluminu (Logical Object Volume driver LOV). LOV przesyła dane do OSC (Object Storage Client jedna z warstw nad SOA). OSC jest maszyną stanową wejścia/wyjścia (I/O). Wykonuje ona:

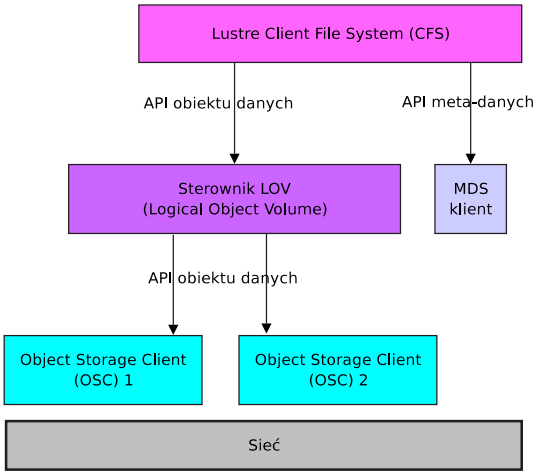
- włącza i wyłącza urządzenie, aby zatrzymać lub uruchomić operacje wejścia/wyjścia,
- zbiera wystarczająco dużo stron, aby wykonać wywołanie RPC do OST,
- zarządza wieloma wywołaniami RPC, które są aktywne.

Podsystem składa się z kilku części (rys.8), oprócz warstwy sieciowej zawiera on trzy elementy umożliwiające komunikację z pozostałymi częściami systemu: klienta ODB, klienta MDS oraz klienta blokad. Wszystkie części klienta mogą być przezroczyście przełączone do innych usług w razie awarii.

2.3 Stos sieciowy

Lustre posiada własny stos sieci, począwszy od żądań przetwarzania na górze a kończąc na sterownikach urządzeń na dole. Warstwy pośrednie to:

1. sterowniki urządzeń,
2. portale (Sandia Portals), warstwy przesyłające wiadomości,



Rysunek 8: Budowa klienta.

3. niobuff, warstwa przesyłająca dane,
4. warstwa żądań przetwarzania.

Portale wymagają dodatkowej warstwy NAL (Network Abstraction Layer). Na dole stosu konieczne jest wspomaganie urządzeń dla konkretnych interfejsów. W większości przypadków sterowniki urządzeń są dostępne, ale rozszerzenia interfejsów z niskopoziomowym przekazywaniem komunikatów są ciągle konieczne np dla zdanego DMA. Urządzenia będące punktem zainteresowania to:

- Ethernet,
- Myrinet,
- Quadrics,
- InfiniBand,
- Urządzenia przystosowane do pracy z VIA,
- Tradycyjne sieci SAN.

2.4 Odzyskiwanie (recovery)

Lustre jest złożone z trzech podsystemów MDS, OST i klientów. Każdy z nich potrzebuje innego mechanizmu odzyskiwania. Stan klienta nie jest zapisywany, jedynym wymaganiem jest, aby po ponownym podłączeniu mógł ponownie korzystać z systemu plików.W przypadku MDS i OST jest inaczej, muszą one również odzyskać spójność własnego stanu i danych.
Mechanizmem wykrywania awarii w Lustre jest przekroczenie czasu (timeout). Każde żądanie ma związaną wartość przekroczenia czasu, zegar startuje po wysłaniu żądania. Jeśli czas upłynie przed otrzymaniem odpowiedzi, przyjmowany jest jakiś rodzaj awarii i włączane odzyskiwanie. Sytuacjami, które spowodują włączenie odzyskiwania są:

- przekroczenie czasu żądania blokady,
- przekroczenie czasu odpowiedzi na żądanie,
- przekroczenie czasu masowych operacji wejścia/wyjścia,
- otrzymanie statusu -ENOTCONN w odpowiedzi po próbie połączenia zakończonej porażką (w przypadku gdy usunięto urządzenie ale w systemie ciągle znajdują się żądania jego dotyczące..

Odzyskiwanie w Lustre opiera się na podstawowej infrastrukturze, określającej kiedy odzyskiwanie jest potrzebne, które żądania kwalifikują się do odzyskiwania. Są to:

- Numer epoki (epoch number) — ma go każdy kontroler składnicy (storage controller) jest zwiększany przy każdym udanym odzyskiwaniu lub włączaniu. Serwer posiada pliki z informacjami o otrzymanych żądaniach dla każdego numeru epoki. Dzięki temu można określić które żądania kwalifikują się do odzyskiwania.
- Numer inkarnacji (Incarnation number) — MDS posiada numer inkarnacji zwiększany przy każdej zmianie konfiguracji (na początku zawiera czas włączenia lidera grupy). Numer ten jest używany przy odzyskiwaniu pomiędzy MDS.
- Numer generacji (Generation number) — połączenia pomiędzy klientami i innymi podsystemami (OST, MDS) mają te numery. Jeśli klient jest oznaczany jako odłączony zmienia się numer generacji.
- Identyfikator transakcji (Transaction ID(xid)) — unikalny w obrębie jednego klienta (dwóch klientów może mieć te same identyfikatory). Każde żądanie ma swój identyfikator i jest on śledzony przez serwer.
- Numer transakcji (Transaction number(transno)) — numer transakcji jest utrzymywany przez serwer i unikalny w obrębie serwera.

2.4.1 Awaria klienta

Klient tracący kontakt z OST lub MDS musi odzyskać stan. Jeśli klient stracił kontakt przez awarię i został ponownie włączony, oznacza to że nastąpi jego normalne przyłączanie do systemu plików. Jedyne wsparcie potrzebne to odwołanie blokad i innych zasobów trzymanych przez niego. Jeśli klient nie odpowiedział na odwołanie blokady lub nie kontynuuje przesyłania danych jest usuwany z klastra. Jeśli utrata połączenia wynika z awarii OST lub MDS klient musi powtórzyć żądania i inne operacje. Możliwe jest również przełączenie klienta do zapasowego serwera jeśli taki jest dostępny. Jeśli klient nie powtórzy operacji w określonym czasie jest usuwany z klastra.

2.4.2 Awaria MDS/OST

W przypadku MDS lub OST, dane na dysku są odzyskiwane przy użyciu dzienników systemów plików na których zainstalowane są podsystemy. Lustre wspiera przywrócenie MDS, zależy to od użycia wspólnego urządzenia składującego dla serwera głównego i zapasowego. W momencie awarii zapasowy serwer przejmuje funkcje serwera głównego. Inaczej jest z OST, podsystem główny i zapasowy działają tu

równocześnie i w przypadku awarii drugi kontynuuje normalne funkcje.
Odzyskiwanie MDS jest połączone z oprogramowaniem o wysokiej dostępności takim jak *kimberlite* lub *clumanager*. W momencie awarii, *clumanager* informuje o tym fakcie serwer zapasowy, który wykonuje sekwencję startową i uruchamia niezbędne usługi. Następnie informuje serwer LDAP że jest aktywnym serwerem i przyjmuje żądania związane z odzyskiwaniem klientów.

2.5 Rozproszone zarządzanie blokadami w Lustre

Każdy system w klastrze będzie zapewniał usługi blokowania (lock service), każdy węzeł klastra może zażądać blokady na zasobie, usunąć blokadę, lub skonwertować blokadę na inny typ blokady. System zarządzający zasobami będzie przechowywał dane blokady.
Zarządzanie danymi blokad obejmuje:

- możliwość odnalezienia danych blokady mając nazwę zasobu,
- utrzymywanie listy żądań przydzielonych blokad na zasób,
- utrzymywanie listy żądań konwersji blokady,
- utrzymywanie listy żądań oczekujących.

Typy blokad:

tryb	nazwa	dostęp	znaczenie
EX	Wyłączny	RW	Żaden inny proces nie może otrzymać dostępu R lub W.
PW	Chroniony zapis	Write	Żaden inny proces nie może otrzymać dostępu W.
PR	Chroniony odczyt	R	Żaden inny proces nie może otrzymać dostępu R.
CW	Konkurencyjny Zapis	W	Brak ograniczeń.
CR	Konkurencyjny Odczyt	R	Brak ograniczeń.
NL	NULL	brak	Do oznaczenia zainteresowania zasobem.

Kompatybilność blokad określa, które blokady mogą współistnieć. Konwersja (nowa blokada) jest możliwa tylko gdy nowy typ jest kompatybilny z wszystkimi przydzielonymi trybami.

	EX	PW	PR	CW	CR	NL
EX	0	0	0	0	0	1
PW	0	0	0	0	1	1
PR	0	0	1	0	1	1
CW	0	0	0	1	1	1
CR	0	1	1	1	1	1
NL	1	1	1	1	1	1

Możliwe żądania to:

- pobierz nową blokadę dla zasobu,
- konwertuj blokadę zasobu,
- odwołaj blokadę.

Gdy żądania jest nowa blokada zarządca przydzieli ją jeśli będą spełnione warunki:

- kolejka nieprzyznanych żądań konwersji jest pusta,

- kolejka nieprzyznanych żądań nowych blokad jest pusta,
- wszystkie przyznane blokady dla zasobu mają typ kompatybilny z typ który jest żądany.

Żądanie konwersji może być przydzielone w przypadku gdy typ żądanej blokady jest kompatybilny z wszystkimi przyznanymi blokadami.
Zarządca blokad wstawia żądania do jednej z kolejek — konwersji, przydzielonych, nieprzydzielonych.
Trzy zdarzenia mogą spowodować przeglądania kolejek:

- blokada z kolejki przydzielonych jest odwołana,
- żądanie konwersji zakończy się sukcesem,
- inne żądanie z nieprzydzielonych lub konwersji zostało przydzielone.

Usługi blokowania używają bloków statusu blokad do przekazywania żądań, odpowiedzi oraz blokad. Gdy potrzebna jest blokada lub konwersja, strona wywołująca przekazuje wskaźnik do bloku *lck_status*. Identyfikator blokady jest wpisywany przez usługę jako pierwszy i w każdym przypadku może służyć do przesyłania dalszych żądań.

3 Projekt

3.1 Zadanie projektowe

Napisać skrypt testujący oraz porównujący wydajność systemów plików *OpenGFS* oraz *exy3*.

3.2 Warunki

Klaster w laboratorium posiada specyficzną konfigurację to znaczy tylko jeden węzeł (węzeł numer 1) posiada dysk. Wszystkie operacje dyskowe trafiają do tego węzła. Połączenie sieciowe w laboratorium ma prędkość 100Mbit/s, co jest wartością niższą niż transfer danych na dysk lokalny w komputerach. Prowadzi to do konieczności wykonywania testu wydajności dla systemu plików *ext3* z węzła innego niż numer 1, aby zapewnić takie same warunki testowe zarówno *OpenGFS* jak i *ext3*. Drugim warunkiem przeprowadzenia testu jest konieczność zainstalowania systemu systemu plików *OpenGFS*.

3.3 Realizacja

3.3.1 Skrypt

Skrypt został napisany przy użyciu składni poleceń powłoki *bash*. Całość skałada się ze skryptu głównego (perf_test.sh)oraz szeregu skryptów w katalogu */functions* Test jest przeprowadzany w katalogach, które wcześniej musi przygotować użytkownik. Katalogi te nie mogą zawierać podkatalogów o nazwach : */test1 /test2*. Skrypt może działać w dwóch trybach, testując pojedyńczy system plików lub tworząc zestawienie dla dwóch, jest to zależne od tego czy jako parametry skryptu zostanie podany jeden czy dwa katalogi. Dodatkowymi parametrami mogą być:

- *onnode numer_węzła* — polecenia będą wykonywane na węźle numer_węzła (dotyczy to tylko poleceń, które służą testowaniu pierwszego systemu plików w przypadku testu porównawczego)
- *nc* — włącza użycie kolorów

Działanie skryptu polega na utworzeniu wewnątrz katalogów wyspecyfikowanych jako parametry katalogów testowych */test1* i */test2*. Następnie wewnątrz tworzone są pliki o zdefiniowanych rozmiarach (ustawienia te zależą od wartości w tablicach F_COUNT B_COUNT B_SIZE B_POSTFIX B_NAME) aktualnie tworzone są cztery kategorie plików różniące się ich rozmiarem oraz ich ilością. Tworzenie plików realizowane jest przy pomocy polecenia *dd if=/dev/zero ....* Następnie na plikach tych wykonywane są operacje :

- kopiowanie
- przenoszenie
- zmiana atrybutów (polecenie *chmod*)

dodatkową operacją jest tworzenie katalogów.Wyniki są zamieszczane w tabelce.

3.3.2 Instalacja OpenGFS

Instalacje OpenGFS zakończyła się niepowodzeniem. Aby zainstalować OpenGFS zdecydowałem o instalacji podsystemów :

- EVMS (Enterprise Volume Management System) wymaga zinstalowania:
  - Device-Mapper
  - Linux-HA pakiet heartbeat
- OpenDLM

Pierwszyszym krokiem było pobranie wszystkich koniecznych pakietów. Oprócz źródeł bądź pakietów instalacyjnych dla systemu Deabian (jeśli były one dostępne) konieczne było przygotowanie odpowiedniego jądra systemu. Klaster aktualnie jest oparty na jądrze 2.4.22-1.2199.nptl-ssi-686-smp oraz wersji instalacji OpenSSI 1.2.2. Pierwszysm krokiem konieczne było przygotowanie jądra z instalacją OpenSSI dlatego pobrałem źródła OpenSSI z repozytorium CVS projektu oraz jądro linuksa wersja 2.4.22 ze strony http://kernel.org. Przy próbie łatania (patch) jądra okazało się że występują błędy. Przeanalizowanie skryptów wykonujących łatanie umożliwiło zlokalizowanie plików *.diff służących jako źródła dla łatania. Próba wykonania sekwencji łatania bez użycia skryptu a tylko przy wykorzystaniu plików diff zakończyła się równierz niepowodzeniem, powodem był brak w źródłach jądra linuksa części plików które miały być łatane.
Kolejna próba, którą podjąłem była możliwa po odnalezieniu pakietu kernel-source-2.4.22-1.2199.nptl-ssi-686-smp zawierającego gotowe łatane źródła jądra linuksa. Pakiet został pobrany następnie przy pomocy polecenia make menuconfig skonfigurowany. Niestety przy próbie kompilacji jądra wystąpiły błedy. Próby wykorzystania konfiguracji zawartej w katalogu *openssi/kernel.configs/* (informacja o konfiguracji znajduje się na stronie http://openssi.org/cgi-bin/view?page=docs2/1.2/debian/devel/INSTALL.cvs) nie dały żadnego rezultatu, kompilacja równierz zakończyłe się błedami.

Aby potwierdzić że błedy kompilacji nie wynikają z mojego błedu lub braku uprawnień poprosiłem o powtórzenie operacji przez jednego z administratorów pana R.L.Jóźwiaka. W tym przypadku próba kompilacji równierz zakończyła się niepowodzeniem.
*ptrace.c:407: warning: implicit declaration of function 'VPOP_NOTIFY_PARENT' ptrace.c:407: error: structure has no member named 'p_vproc' make[2]: *** [ptrace.o] Błąd 1 make[2]: Leaving directory '/home/rso.b/pdanilew/install/usr/src/kernel-source-2.4.22-1.2199.nptl-ssi-686-smp/kernel' make[1]: *** [first_rule] Błąd 2 make[1]: Leaving directory '/home/rso.b/pdanilew/install/usr/src/kernel-source-2.4.22-1.2199.nptl-ssi-686-smp/kernel' make: *** [_dir_kernel] Błąd 2*
Z powodu braku możliwości utworzenia jądra linuksa OpenSSI uznałem iż instalacja systemu plików OpenGFS na klastrze jest niemożliwa.