# Rozproszone zarządzanie blokadami

Projekt z przedmiotu RSO

Piotr Trojanek
P.Trojanek@elka.pw.edu.pl

7 czerwca 2005

## Spis treści

**1 Wprowadzenie** **1**
1.1 DLM jako element systemu klastrowego . . . . . 2

**2 Elementy porównania** **2**

**3 Przykładowe imlpementacje** **3**
3.1 OpenDLM . . . . . 3
3.2 Redhat Cluster DLM . . . . . 4
3.3 Lustre DLM . . . . . 4

**4 Dalsza część projektu – Lustre DLM** **4**
4.1 Otwartość kodu . . . . . 4
4.2 Dokumentacja . . . . . 5
4.3 Konfiguracja . . . . . 5
4.4 Implementowany algorytm . . . . . 5
4.5 Dostępne API . . . . . 7
4.6 Funkcjonalność . . . . . 8
4.7 Skalowalność . . . . . 8
4.8 Odporność . . . . . 9
4.9 Wydajność . . . . . 9

**5 Wnioski** **9**

## 1 Wprowadzenie

Podstawą działania rozproszonego systemu komputerowego jest dzielenie dostępu do zasobów. Dopiero dzięki temu możliwe staje się efektywne zarządzanie mocą obliczeniową wszystkich wezłów systemu.

W niniejszym opracowaniu przedstawię aspekty charakteryzujące zarządzanie blokadami w rozproszonym systemie komputerowym na przykładzie oprogramowania *OpenDLM* jako rozwiązania najbardziej zaawansowanego w porównaniu z innymi usługami. Już na wstępie jednak chciałbym podkreślić, że jest to oprogramowanie dedykowane dla zagadnienia poruszanego w opracowaniu i porównanie ma przede

wszystkim na celu wykazanie specyfiki tego kompleksowego rozwiązania. Ograniczę się przy tym do analizy związanej ze środowiskiem *Linux* z tej prostej przyczyny, że chociaż zarządzanie blokadami nie jest specyfiką tego systemu, to dostawcy komercyjni ukrywają zazwyczaj szczegóły swoich produktów, przez co niemożliwa staje się ich analiza.

### 1.1 DLM jako element systemu klastrowego

Poniższa ilustracja przedstawia umiejscowienie DLM w architekturze rozwiązania klastrowego [1]. Chociaż teoretycznie możliwe jest korzystanie z blokad na poziomie gniazd TCP/IP, w praktyce użytkownicy oczekują bardziej wygodnego interfejsu programistycznego. Podobnie do implementowania blokad wykorzystuje się już istniejące prymitywy komunikacyjne systemu rozproszonego (zazwyczaj jest to przesyłanie komunikatów z blokowaniem, (ang. *message passing*)). Do poprawnego działania potrzebny jest także mechanizm umożliwiający badanie stanu (aktywności) węzłów systemu wykrywający uszkodzenia pojedyńczych elementów.

## 2 Elementy porównania

Na podstawie wstępnego zapoznania z dostępnymi rozwiązaniami wyróżniłem następujące elementy porównawcze:

**otwartość kodu** – rozwiązanie DLM wywodzi się z zaawansowanych środowisk rozproszonych które do niedawna były dostępne tylko jako gotowe komercyjne produkty, stąd analiza części z nich oparta może być jedynie na udostępnionej dokumentacji – są to jednak sprawdzone rozwiązania o wysokiej jakości;
po drugiej stronie znajdują się rozwiązania *open–source* – czy ich jakość jest porównywalna z komercyjnymi?

**dokumentacja** – bardzo ważnym aspektem poprawnego działania systemu DLM jest jego konfiguracja, a ta nie jest możliwa bez wyczerpującej dokumenacji,

**konfiguracja** – pod tym pojęciem rozumiem także uruchomienie usług wymaganych dla pracy DLM,

**implementowany algorytm blokad** – podstawowy element systemu DLM, od niego zależy zarówno funkcjonalność jak i wydajność,

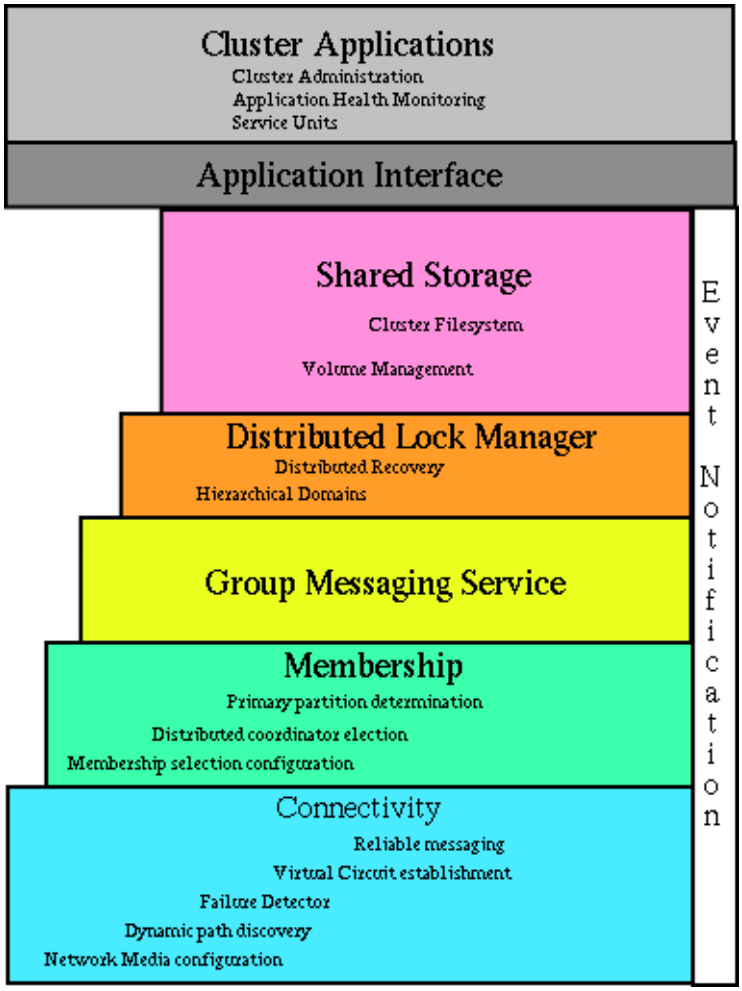
**dostępne API** – czy dane rozwiązanie jest uniwersalne, tzn. czy udostępnia API tylko dla określonego języka programowania oraz czy jest dostępne zarówno dla aplikacji użytkownika jak i jądra systemu operacyjnego,

**funkcjonalność** – w rozumieniu dostarczanego poziomu abstrakcji, co obejmuje np. blokady hierarchiczne, blokowanie do zapisu/odczutu, poziomy blokowania,

**skalowalość** – jak dobrze dane rozwiązanie działa wraz ze wzrastającą liczbą węzłów systemu komputerowego,

**odporność** – w rozumieniu braku tzw. SPOF (ang. *Single Point of Failure*) oraz zdolności do odbudowania systemu w przypadku awarii węzłów,

**wydajność** – efektywność działania pod względem wykorzystywanych zasobów CPU, pamięci oraz obciążenia łącz komunikacyjnych.




Rysunek 1: DLM jako warstwa rozwiązania systemu klastrowego

# 3 Przykładowe imlpementacje

Poniżej przedstawię krótkie charakterystyki dostępnych rozwiązań DLM

## 3.1 OpenDLM

Oprogramowanie *OpenDLM* [2] wywodzi się z rozwiązania *DLM* firmy *IBM*, wykorzystywanego w systemie *AIX*. W ramach wspierania wolnego oprogramowania *IBM* udostępnił część kodów źródłowych dla wykorzystania w środowisku systemu *Linux*, od tego czasu projekt wykorzystywany jest m.in. w ramach rozproszonego systemu plików *OpenGFS* (chociaż w zubożonej wersji).

Spośród darmowych rozwiązań dedykowanych dla systemu *Linux* wydaje się to być najbardziej zaawansowany projekt. Do cech wyróżniających należą:

- API dostępne zarówno z poziomu aplikacji jak i jądra systemu,

- hierarchizowanie blokad,
- domeny blokad (dla niezwiązanych ze sobą aplikacji),
- oparcie o system badania aktywnością węzłów *HeartBeat*,
- brak *SPOF*.

### 3.2 Redhat Cluster DLM

Projekt działający w środowisku *Linux*, będacy częścią systemu klastrowego firmy *RedHat* wzorowany na rozwiazaniu z komercyjnego systemu *VMS*. Podobnie jak *OpenDLM* udostepnia API zarówno dla programów użytkownika jak i elementów jądra [3].

Celem tego projektu wydaje się chęć firmy *RedHat* do posiadania pod własną kontrolą oprogramowania typu DLM umożliwiającego działanie rozwijanego przez tą firmę rozproszonego systemu plików *GFS* (odpowiednik niekomercyjnego *OpenGFS*).

### 3.3 Lustre DLM

Autorzy projektu uznali, że dostępny już *OpenDLM* jest zbyt obszerny jak na potrzeby ich systemu plików twierdząc, że wymaganą funkcjonalność potrawią uzyskać przy znacząco mniejszych rozmiarach kodu źródłowego. Nie bez znaczenia jest zapewne fakt, że *Lustre* jest produktem komercyjnym, zaś udostępniane darmowo są jedynie jego starsze wersje.

Na potrzeby projektu został stworzony nowy system rozproszonego zarzadania blokadami określany mianem “innowacyjnego”.

## 4 Dalsza część projektu – Lustre DLM

Jako zadanie na dalszą część projektu zamierzam poddać weryfikacji stwierdzenia autorów projektu *Lustre* poprzez porównanie z dwoma wcześniej przedstawionymi rozwiązaniami.

Będzie się to wiązało z uruchomieniem i konfiguracją rozproszonego systemu plików oraz przeprowadzeniem testów wydajności oraz niezawodności.

Poniżej przedstawię charakterystekę systemu *Lustre* na podstawie wcześniej wprowadzonych punktów.

### 4.1 Otwartość kodu

System *Lustre*, wraz z wchodzącym w jego skład managerem blokad jest produktem komercyjnym, jednak właściciel (firma *Cluster File Systems, Inc.*) stosuje politykę udostepniania wcześniejszych wersji z pełnym kodem źródłowym. Aktualnie sprzedawana jest wersja oznaczona numerem serii *1.4.x*, której największymi atutami w porównaniu z udostępnianymi publicznie jest:

- wsparcie dla kolejnych rodzin procesorów (Intel “Nocona”, PowerPC),
- zarządzanie poprzez SNMP,
- zoptymalizowany re-export NFS/CIFS (dla obsługi stacji klienckich UNIX/Windows poprzez klastrowy serwer plików),

- implementacja mechanizmów blokowania plików *flock/lockf*,
- podwojenie liczby obsługiwanych węzłów klienckich.

## 4.2 Dokumentacja

Pod względem dokumentacji system *Lustre* pozostawia sporo do życzenia. Pierwsze wrażenie jakie wywołuje jej objętość jest bardzo zachecające, jednak już po zapoznaniu się z fragmentem zawartości czytelnik zdaje sobie sprawę, że jest to raczej luźny zbiór zapisków programistów niż dokumentacja poważnego komercyjnego projektu.

Już samo uspójnienie informacji zawartych w dokumentacjach z rzeczywisym zachowaniem posiadanej wersji oprogramowania nastręcza sporych trudności. Jako przykład niech posłuży chociażby konfiguracja backend'u systemu organizacji plików – właściwie nie wiadomo, czy dana wersja obsługuje *ext3*, tajemnicze *extN*, czy może mamy do czynienia ze wsparciem *reiserfs*.

## 4.3 Konfiguracja

Za sprawą niekompletnej (a w wielu miejscach wręcz mylącej) dokumentacji konfiguracja pakietu nie jest rzeczą oczywistą. Oparcie łatania źródeł jądra systemu o program *quilt* wydaje się o tyle słuszne, co niedopracowane. Kolejne łatki, zamiast nakładane na siebie w formie stosu wymagają ręcznych interwencji w pliki źródłowe. Załączone przykładowe pliki konfiguracyjne klastra wymagają modyfikacji w celu uruchomienia.

Jeżeli chodzi o sam element DLM systemu, nie posiada on możliwości konfiguracji. Jest to rzecz niezrozumiała, gdyż w dokumenacji mowa jest o nierównomiernym przydziale roli *master* w zarządzaniu pojedyńczymi blokadami (np. bardziej wydajne maszyny, które powinny zarządzać większą ilością zasobów).

## 4.4 Implementowany algorytm

Jak podają autorzy algorytm zarządzania blokadami wzorowany jest na klasycznym (w tym sensie, że często kopiowanym) mechaniźmie systemu *VMS*.

Podstawowymi pojęciami DLM systemu *Lustre* są:

**przestrzeń blokad** – inaczej *domena*, grupuje blokady związane z określoną funkcjonalnością; dostęp do niej może byc chroniony,

**zasoby** – tworzące drzewiastą hierarchię w obrębie domeny; są identifikowane poprzez nazwę (obecnie 31 znaki), przy czym nazwy potomków tego samego rodzica w drzewie muszą być unikalne,

**nadzorca blokady** – węzeł, który zarządza przydziałem dostępu do zasobu,

**właściciel blokady** – węzeł (program/podsystem) posiadający wyłączność na dostęp do zasobu; zasób może być jednocześnie w posiadaniu wielu klientów, ale zawsze będzie miał co najwyżej jednego nadzorce.

Zarządznie blokadami realizowane jest poprzez następujące operacje:

- kojarzenie nazwy zasobu z opisującą ją strukturą danych,

- zarządzanie listą przydzielonych blokad zasobu,
- zarządzanie listą żądań zmiany poziomu blokady dla zasobu,
- zarządzanie listą oczekujących żądań dostępu do zasobu.

Podobnie jak w oryginalnym DLM systemu *VMS* określone są następujące poziomy blokad oraz tabela ich wzajemnej kompatybilności:

| MODE | NAME | ACCESS | MEANING |
|---|---|---|---|
| EX | Exclusive | RW | No other process can get R or W access. |
| PW | Protected Write | Write | No other process can get a W access. |
| PR | Prot. Read | R | No other process can get a W access. |
| CW | Concurrent Write | W | No restrictions on other processes. |
| CR | Concurrent Read | R | No restrictions on other processes. |
| NL | Null | none | To indicate an interest in a resource. |

Tablica 1: Lock Modes

| | 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 |

Tablica 2: Lock Mode Compatability

Możliwe są następujące operacje na blokadzie:

1. uzyskanie blokady dla operacji na zasobie,
2. konwersja poziomu blokady,
3. zrzeknięcie się blokady.

Z każdym zasobem związane są trzy kolejki:

- oczekujących żądań blokady,
- oczekujących żądań konwersji poziomu blokady,
- przydzielonych blokad.

Blokada przydzielana jest tylko wówczas, kiedy spełnione są poniższe warunki:

1. kolejka oczekiwania na konwersję blokady jest pusta,
2. kolejka oczekiwania na uzyskanie blokady jest pusta,
3. już przydzielone blokady dla zasobu są kompatybilne z żądanym poziomem wyłączności.

Operacja konwersji poziomu blokady realizowana jest natychmiastowo, jeżeli żądany poziom wyłączności jest kompatybilny z już założonymi blokadami. W przeciwnym wypadku, operacja zostaje przypisana do kolejki oczekiwania na konwersję.

Trzy zdarzenia powodują przegląd kolejek oczekujących związanych z zasobem:

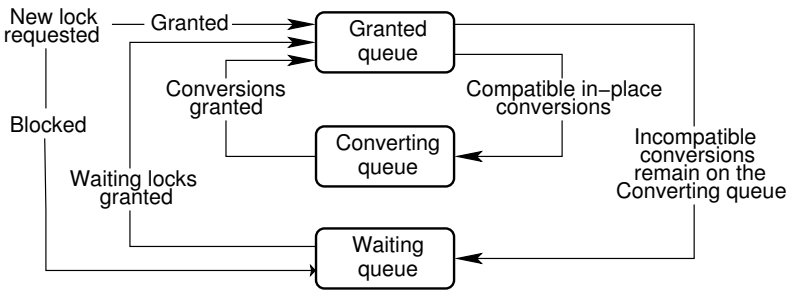
1. zrzeknięcie się przydzielonej blokady,
2. zakończona pomyślnie operacja konwersji poziomu blokady,
3. zakończona pomyślnie operacja przydziału lub konwersji innej blokady.

Autorzy systemu *Lustre* zwracają uwagę na następujące dwa problemy odziedziczone z DLM systemu *VMS*:

1. zakańczane natychmiastowo operacje konwersji poziomu blokad mogą wstrzymać w nieskończoność zakolejkowane operacje konwersji;
   *Rozwiązanie:* dodatkowa flaga *QUEUECONV* wymuszająca umieszczenie żądania konwersji w kolejce,
2. nowe żądania blokad na poziomie *NL* (*zaznacz zainsteresowanie*) są wstrzymywane przez operacje konwersji poziomu blokady.
   *Rozwiązanie:* dodatkowa flaga *EXPEDITE* wymuszająca przydzielenie blokady.

Dodatkowo dostępna jest flaga *NOQUEUE* – pozwala na odrzucenie żądania bez umieszczania go w kolejce jeśli nie może być zrealizowane natychmiastowo.

Poniżej znajduje się kompletny graf przejść pomiędzy stanami związanymi z umieszczeniem żądań w odpowiednich kolejkach.




Rysunek 2: Lock Queues

## 4.5 Dostępne API

Ponieważ DLM *Lustre* jest zintegrowaną częścią pakietu, nie udostępnia API ani dla innych podsystemów jądra ani dla programów użytkowanika. Jest to znacząca wada w porównaniu chociażby z *OpenDLM*.

Jednocześnie stoi to w sprzeczności z dokumentacją, według której funkcjonalność DLM "powinna" być dostępna w trzech formach:

1. biblioteki poziomu jądra,
2. demona poziomu jądra,

3. biblioteki poziomu użytkowanika.

Opis wywołań związanych z DLM daje tylko ogólny pogląd o ich funkcjonalności, nie pozwala jednak na ich przetestowanie ani porównanie z konkurencyjnym oprogramowaniem.

Dodatkowo *Lustre* obsługuje 3 typy blokad:

**(ang. *Plain*)** : standardowe pojedyńczne blokady zapewniające wyłączność korzystana z zasobu,

**(ang. *Extent*)** : blokady obszarów, stosowane do implementacji rezerwowania obszarów plików (optymalizowane pod kątem tego zastosowania – zmiennego zakresu),

**(ang. *Intent*)** : blokady z intencją, np. klient zamiast blokowania katalogu w celu utworzenia pliku uzyskuje blokadę na plik jako wynik pojedyńczej operacji (serwer zna cel zakładania blokady).

## 4.6 Funkcjonalność

Podobnie jak pierwowzór z systemu *VMS* DLM *Lustre* dostarcza:

- rozdzielnych domen dla blokad związanych z różnymi podsystemami,
- hierarchie blokad (np. katalog a dalej znajdujące się w nim pliki),
- powiadamianie o statusie operacji poprzez funkcje *callback*,
- zarówno synchroniczne (blokujące) jak i asynchroniczne (nieblokujące) wywołania.

## 4.7 Skalowalność

Teoretycznie skalowalność składnika DLM *Lustre* nie powinna stanowić problemu. Stosowanie funkcji skrótu (ang. *hash*) dla odnajdywania wezła odpowiedzialnego za blokadę pozwala na bardzo szybkie komunikowanie się bez zbędnego pośrednictwa maszyn trzecich.

Potencjanym zagrożeniem skalowalności jest obsługa zmian topologii klastra (ang. *cluster transitions*). Zgodnie z tym co podają autorzy taka sytuacja wymaga następujących operacji:

1. kiedy nowy węzeł dołącza do klastra, konieczne jest wyznaczenie nowej tablicy odwzorowującej zasoby (poprzez funkcje skrótu) na dostępne maszyny,
2. wezły nadzorujące zasoby (*master*) powinny zwolnić założone blokady lub ponownie zarejestrować sie jako ich właściciele (co wymaga powiadomienia wszystkich maszyn),
3. wezły posiadające blokady zasobów powinny powtórzyć producedurę ich pozyskiwania (ang. *reacquire*).

Autorzy zaznaczają, że w wielu przypadkach jest to postępowanie nadmiarowe i możliwe jest daleko idące uproszczenie tej procedury.

### 4.8 Odporność

Podstawowy mechanizm odporości na uszkodzenia pojedyńczych węzłów klastra jest analogiczny jak sposób postępowania w przypadku zmiany topologii klastra. Obydwa mechanizmy opierają się na usłudze zarządzania/kontroli przynależności do klastra (ang. *membership*).

### 4.9 Wydajność

Wydajność systemu DLM ograniczana jest przez narzut na komunikację międzywęzłową. W tej sytuacji rozsądne wydaje się przyjęcie przez autorów koncepcji (ang. *dynamic remastering*), która oznacza, że wezeł nadzorujący blokadę zasobu może zrzec sie tej roli na rzecz innego, który wykazuje się większą aktywnością jeżeli chodzi o korzystanie z zasobu.

Zasady przekazywania nadzoru nad zasobem zawarte są w poniższych punktach:

- początkowo zasób jest kontrolowany przez pierwszy wezeł, który chce z niego skorzystać,
- jeżeli kilka wezłów uzyskuje blokadę na zasób, zarządcą zostaje ten z najwyższą wagą,
- jeżeli kilka wezłów z tą samą wagą uzyskuje blokadę na zasób, zarządcą zostaje ten, który najbardziej aktywnie go wykorzystuje.

Aktywność mierzona jest jako średnia geometryczna z operacji blokowania, konwersji oraz opuszczenia kolejki związanej z zasobem.

## 5 Wnioski

Podsystem DLM oprogramowanie *Lustre* nie okazał się rozwiązaniem innowacyjnym – w dużej mierze jest to kopia funkcjonalności DLM z systemów *VMS*. Atut autorów w postaci małej ilości kodu potrzebnej do implementacji według mnie jest przyczyną utraty uniwersalności i przeszkodą w spełnieniu założeń projektowych o udostępnieniu API dla programów przestrzeniu użytkownika.

Trzy typy blokad, które mają być unikalne w stosunku do pierwowzoru (ang. *Plain, Extent, Intent*) według mnie istnieją obecnie jedynie w formie planów – a przynajmniej nie są zaimplementowane w udostępnianych wersjach, które nie implementują blokad *flock/lockf*. W praktyce brak tej funkcjonalności uniemożliwił przetestowanie DLM pod kątem wydajności, ponieważ nie był możliwy żaden inny dostęp do blokad z poziomu kodu użytkownika.

Biblioteka *liblustre* nie udostępnia w aktualnej wersji dostępu do systemu DLM, a jedynie interfejs wywołań systemowych *open,read,write,close,....*

Cały system *LustreFS* pozostawia sporo do życzenia, jeżeli chodzi o dokumentacje, która w wielu miejscach okazuje się bądź niekompletna, bądź myląca poprzez podawanie informacji o jeszcze niezaimplementowanych koncepcjach.

## Literatura

[1] *OSDL Cluster Architecture http://developer.osdl.org/dev/osdlclusters/OSDL-cluster.html*

[2] *The OpenDLM Project http://opendlm.sourceforge.net/*

[3] *DLM Project http://sources.redhat.com/cluster/dlm/*

[4] *Lustre Scalable Storage http://www.clusterfs.com/*