

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marcin Szlufik

Nr albumu: 174762

**Implementacja macierzy RAID z
konfigurowalnym stopniem
nadmiarowości dla systemu Linux**

Praca magisterska
na kierunku **INFORMATYKA**

Praca wykonana pod kierunkiem
dr Janina Mincer-Daszkiewicz
Instytut Informatyki

Lipiec 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

Streszczenie

W pracy przedstawiono implementację sterownika macierzy dyskowej RAID z konfigurowalną liczbą dysków nadmiarowych. Jest to sterownik urządzenia blokowego dla systemu Linux. Pozwala na konstrukcję macierzy odpornych na awarię części dysków. Dopuszczalna liczba dysków które mogą ulec uszkodzeniu ustalana jest przy tworzeniu macierzy. Obliczenia związane z obsługą nadmiarowości oparto na teorii kodów korygujących wymazania.

Słowa kluczowe

RAID, macierz dyskowa, kody korygujące błędy, kody korygujące wymazania

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.4 Operating Systems
D.4.2 Storage Management

Tytuł pracy w języku angielskim

Configurable redundancy RAID implementation for Linux

Spis treści

1. Wstęp	5
1.1. Wprowadzenie	5
1.2. Postawienie problemu	5
1.3. Temat pracy	6
1.4. Struktura pracy	6
2. Macierze RAID geneza i klasyfikacja	7
2.1. Wprowadzenie	7
2.2. Dyski magnetyczne	7
2.2.1. Wstęp	7
2.2.2. Budowa i parametry	7
2.2.3. Uszkodzenia	8
2.3. Macierze dyskowe	9
2.3.1. Wstęp	9
2.3.2. Konfiguracja a parametry	10
2.3.3. Macierze RAID	11
2.4. Klasyfikacja RAID	11
2.4.1. Wstęp	11
2.4.2. RAID 0	12
2.4.3. RAID 1	12
2.4.4. RAID 2	12
2.4.5. RAID 3	13
2.4.6. RAID 4	14
2.4.7. RAID 5	14
2.4.8. RAID 6	14
2.5. Niezawodność	15
2.6. Metody realizacji macierzy RAID	16
3. Kody korygujące wymazania a macierze RAID	17
3.1. Wprowadzenie	17
3.2. Terminologia	17
3.3. Użycie kodu w macierzy RAID	18
3.4. Kody oparte na kodach Reeda-Solomona	19
3.4.1. Operacje na elementach ciał skończonych \mathbb{F}_{2^t}	19
3.4.2. Macierz kontroli parzystości	21
3.5. Kody \mathbb{F}_q -liniowe	22
3.5.1. Kod EVENODD	22

4. Projekt i implementacja	23
4.1. Założenia projektowe	23
4.1.1. Wstęp	23
4.1.2. Podsystem MD	23
4.1.3. Obliczanie informacji nadmiarowych	23
4.1.4. Rozdział danych nadmiarowych	24
4.2. Interfejs podsystemu MD	24
4.3. Struktury danych	27
4.4. Operacje kodowe	29
4.4.1. Obliczanie danych nadmiarowych w schemacie odczyt-modyfikacja-zapis.	31
4.4.2. Obliczanie danych nadmiarowych w schemacie rekonstrukcja-zapis	31
4.4.3. Sprawdzanie spójności danych nadmiarowych.	31
4.4.4. Odtwarzanie danych.	31
4.5. Synchronizacja i rekonstrukcja	32
4.6. Główny algorytm	33
4.6.1. Funkcja raidn_make_request	33
4.6.2. Funkcja raidn_sync_request	34
4.6.3. Funkcja handle_stripe	34
4.7. Narzędzia administracyjne	35
5. Testy	37
5.1. Wprowadzenie	37
5.2. Metody testów	37
5.3. Urządzenia pomocnicze	37
5.4. Funkcjonalność	37
5.5. Wydajność	39
5.5.1. Liczba dysków nadmiarowych a czas odczytu	40
5.5.2. Liczba dysków nadmiarowych a czas zapisu	40
5.5.3. Liczba dysków uszkodzonych a czas odczytu	40
5.5.4. Liczba dysków uszkodzonych a czas zapisu	41
5.5.5. Porównanie z innymi sterownikami RAID	41
6. Podsumowanie	45
6.1. Możliwe rozszerzenia	45
A. Zawartość płyty dołączonej do pracy	47
Bibliografia	49

Rozdział 1

Wstęp

1.1. Wprowadzenie

Różne dziedziny zastosowań pamięci masowych narzucają różne wymagania co do ich parametrów. W większości przypadków nacisk kładziony jest na czas dostępu, szybkość przesyłu informacji, pojemność i niezawodność. Przy obecnym poziomie rozwoju technologicznego wymagania te dość dobrze spełniają dyski magnetyczne stanowiąc główny budulec pamięci masowych. Mają one jednak szereg ograniczeń, z których najistotniejszym jest duży czas dostępu. Ujemne skutki dużego czasu dostępu mogą być częściowo wyeliminowane dzięki łączeniu dysków w grupy zwane macierzami dyskowymi. Macierze dyskowe pozwalają także uzyskać większą pojemność i szybkość przesyłu w stosunku do pojedynczych dysków. Podstawową wadą takich konstrukcji jest obniżona niezawodność. Można to skompensować przez dodanie dysków z danymi nadmiarowymi. Macierze takie określono terminem **RAID** będącym skrótem nazwy **Redundant Array of Inexpensive/Independent Disks**, co można tłumaczyć jako *nadmiarową macierz tanich/niezależnych dysków*.

1.2. Postawienie problemu

Jedną z popularniejszych konstrukcji macierzy dyskowych RAID jest RAID 5 posiadająca jeden dysk z danymi nadmiarowymi. Niestety stopień niezawodności tej macierzy obniża się przy skalowaniu liczby dysków, nie jest też możliwe jego podwyższenie przy ustalonej liczbie dysków.

Metodą pozwalającą wyeliminować te ograniczenia jest zwiększenie liczby dysków zawierających dane nadmiarowe. Problemem jest jednak znalezienie wydajnego sposobu obliczania danych nadmiarowych, który pozwoliłby równie wydajnie obliczać dane z uszkodzonych dysków w razie awarii. Szybka metoda obliczania brakujących danych pozwala również na zmniejszenie liczby operacji dyskowych, ponieważ opłacalne staje się wtedy zastąpienie części operacji dyskowych obliczeniami. Rozwiązania tej trudności można szukać w teorii kodów poprawiających błędy, i skorzystać z kodów do naprawy wymazań. Okazuje się, że uszkodzenia danych na dyskach odpowiadają właśnie wymazaniom, to znaczy wiemy, która część danych została utracona. Funkcjonalność umożliwiającą lokalizację uszkodzeń zapewniają same dyski, które używają kodów kontrolnych do testowania spójności odczytywanej informacji.

1.3. Temat pracy

W swojej pracy chciałbym przedstawić programową realizację sterownika macierzy dyskowej z konfigurowalną liczbą nadmiarowych dysków. Jako platformę implementacji wybrałem system operacyjny Linux, sama zaś macierz ma być dostępna poprzez interfejs urządzeń blokowych tego systemu. W pracy zarysuję także teoretyczne podstawy użytej metody kodowania danych nadmiarowych.

1.4. Struktura pracy

W rozdziale 2 przedstawiam klasyfikację obecnie stosowanych typów macierzy dyskowych. Rozdział 3 zawiera teoretyczną analizę opracowywanego zagadnienia. W rozdziale 4 opisuję zagadnienia implementacji. Rozdział 5 zawiera wyniki testów porównawczych. W rozdziale 6 podsumowuję temat. Dodatek A obejmuje opis zawartości płyty dołączonej do pracy.

Rozdział 2

Macierze RAID geneza i klasyfikacja

2.1. Wprowadzenie

Konieczność budowy macierzy dyskowych wynika z ograniczeń dysków magnetycznych, same zaś macierze korzystają z udostępnianej przez dyski funkcjonalności, dlatego najpierw przedstawione zostaną podstawowe właściwości dysków.

Oczekiwania względem pamięci zależą od korzystających z niej aplikacji. Macierze dyskowe pozwalają na przystosowanie swoich parametrów do konkretnego zastosowania poprzez zmiany w konfiguracji. Podstawowe konfiguracje macierzy zostały sklasyfikowane w publikacji z roku 1988 [Patterson88]. Wtedy wprowadzono również termin RAID. Zagadnienia te zostaną poruszone w dalszej części rozdziału.

2.2. Dyski magnetyczne

2.2.1. Wstęp

Dyski magnetyczne zwane też dyskami twardymi są podstawowym elementem używanym do budowy pamięci trwałych. Zakres ich obecnych zastosowań rozciąga się od wielkich systemów przetwarzania i gromadzenia danych po małe urządzenia przenośne. Przez kilkadziesiąt lat od momentu wprowadzenia zostały znacznie udoskonalone, jednak podstawy działania i związan z nimi charakterystyczne cechy są ciągle takie same.

2.2.2. Budowa i parametry

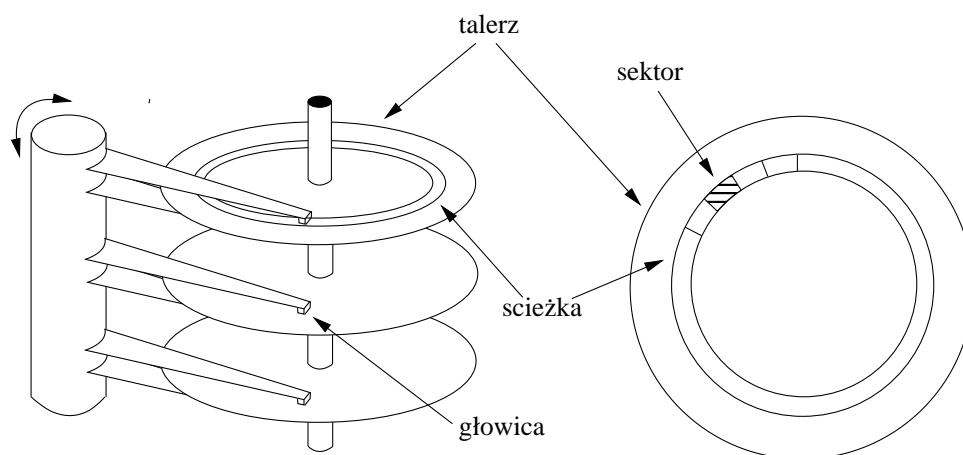
Na rysunku 2.1 przedstawiony jest schemat budowy typowego dysku magnetycznego.

Taki dysk posiada kilka talerzy o właściwościach magnetycznych. W czasie pracy wirują one z ustaloną prędkością, najczęściej 5.400, 7.200 lub 10.000 obrotów na minutę.

Odczyt i zapis danych odbywa się przy użyciu grupy głowic przesuwanych synchronicznie nad powierzchniami talerzy. Na talerz przypadają dwie głowice, po jednej na każdą z powierzchni talerza.

Dane na talerzach zapisywane są w postaci koncentrycznych okręgów zwanych ścieżkami. Zbiór ścieżek o tym samym promieniu ze wszystkich talerzy to cylinder.

Ścieżki podzielone są na sektory, z których każdy oprócz właściwych danych zawiera informacje kontrolne i synchronizacyjne. Typowy rozmiar danych zapisanych w jednym sektorze to 512 B. Podział ścieżki na sektory, nazywany też formatowaniem niskopoziomym, wykonywany jest zwykle przez producenta dysku.



Rysunek 2.1: Schemat budowy dysku magnetycznego

Dane na poziomie dysku adresowane są poprzez numer cylindra, numer głowicy oraz numer sektora. Dostęp do danych wymaga przesunięcia głowicy nad właściwą ścieżkę, a następnie odczekania aż żądany sektor przesunie się pod głowicę. Wtedy możliwe jest jednoczesne odczytywanie danych z kilku głowic.

Podstawowymi parametrami dysku są **czas dostępu** (ang. *access time*), **szybkość przesyłu informacji** (ang. *transfer rate*) i **pojemność** (ang. *capacity*).

Na czas dostępu, będący czasem między wysłaniem żądania do dysku a rozpoczęciem transmisji danych przez dysk, składają się:

- **czas szukania** (ang. *seek time*), czyli czas potrzebny na przesunięcie głowicy nad ścieżkę,
- **opóźnienie obrotowe** (ang. *rotational latency*), czyli czas oczekiwania na obrót talerza tak aby odpowiedni sektor znalazł się pod głowicą.

Czas szukania zależy od konstrukcji układu przesuwającego głowice i obecnie jest rzędu kilku ms.

Opóźnienie obrotowe zależy głównie od prędkości obrotowej talerza i średnio jest to połowa okresu obrotu talerza, czyli dla 10.000 obrotów na minutę wynosi $6 \text{ ms} / 2 = 3 \text{ ms}$.

Zamiast czasu dostępu używane jest też określenie **losowy czas dostępu** (ang. *random access time*) będący średnim czasem między wysłaniem żądania do dysku a rozpoczęciem transmisji. Czas dostępu przekłada się niemal bezpośrednio na **liczbę operacji wejścia-wyjścia na sekundę** nazywaną też **przepustowością** (ang. *throughput*).

Szybkość przesyłu informacji wyrażana w bajtach na sekundę, określa szybkość z jaką dane przesyłane są z/do dysku gdy głowica zostanie już ustawiona nad właściwym sektorem. Szybkość przesyłu informacji jest proporcjonalna do gęstości zapisu informacji na ścieżce, odległości ścieżki od środka dysku, prędkości obrotowej oraz liczby głowic. Dla współczesnych dysków szybkość przesyłu informacji jest rzędu kilkudziesięciu MB/s.

2.2.3. Uszkodzenia

Uszkodzenie pamięci jest znacznie groźniejsze, jeśli nie zostanie wykryte. Dyski magnetyczne posiadają mechanizmy weryfikacji poprawności odczytywanej informacji. Do sprawdzania czy dane na dysku nie zostały uszkodzone używa się **kodów korygujących błędy**

(ang. *error correction code*, ECC). Kody korygujące błędy stosowane są głównie w celu wykrywania, a w mniejszym stopniu do korekcji błędów. Wynika to stąd, że próbując naprawić błąd przy użyciu kodu korygującego zwiększamy prawdopodobieństwo niewykrycia błędu, który przekroczył możliwości korekcyjne kodu. Dlatego korygowane są tylko niewielkie błędy, dla reszty przekazywana jest informacja o błędzie.

W przypadku gdy sterownik sprzętowy dysku wykryje błąd przy odczycie, kilkakrotnie próbuje powtarzać odczyt i dopiero po kilku nieudanych próbach informacja o błędzie jest przekazywana użytkownikowi. Taka sytuacja dla przeciętnego dysku zdarza się średnio mniej niż raz na 10^{14} przesłanych bitów. Określa się ją terminem **błądu nienaprawialnego** (ang. *uncorrectable bit error*). Współczesne dyski zwykle oznaczają taki sektor jako uszkodzony i mapują go w wolny sektor z puli sektorów zapasowych. Prawdopodobieństwo niewykrycia błędu przez kod kontrolny zależy od zastosowanego kodu i dla typowego dysku jest bardzo małe, poniżej 1 sektora na 10^{21} przesłanych bitów.

Parametrem określającym **niezawodność** (ang. *reliability*) dysku jest MTTF (ang. *mean time to failure*) czyli **średni czas do uszkodzenia**. Dla typowego dysku wynosi on ok. kilkudziesięciu lat. Na jego podstawie możemy obliczyć prawdopodobieństwo uszkodzenia dysku w zadanym okresie.

2.3. Macierze dyskowe

2.3.1. Wstęp

Postęp technologiczny w różnym stopniu wpływa na poszczególne elementy systemów komputerowych. Od wielu lat parametry jednostek centralnych i pamięci operacyjnych polepszają się wielokrotnie szybciej niż parametry dysków magnetycznych. W przypadku dysków dość szybko rośnie pojemność, wolniej wzrasta szybkość transmisji (o ok. 20% rocznie), zaś najwolniej poprawiającym się parametrem jest czas dostępu. Zależy on od operacji mechanicznych i obniża się o mniej niż 10% w ciągu roku. Obecnie czas dostępu to ok. kilku ms, podczas gdy 20 lat temu wynosił kilkadziesiąt ms. W tym samym czasie szybkość procesorów rosła o ok. 50% rocznie zwiększając się w ciągu ostatnich 20 lat ok. 4000 razy. Biorąc pod uwagę to, że poza samym wzrostem szybkości procesorów udoskonalano ich architekturę oraz wprowadzono przetwarzanie równoległe, różnica ta jest jeszcze większa.

Ponieważ o wzroście wydajności systemu komputerowego decydują wszystkie jego elementy, te o najsłabszych parametrach stanowią jego największe ograniczenie. Wpływ wolnego elementu na pracę całego systemu określa równanie zwane Prawem Amdahla:

$$S = \frac{1}{(1 - f) + f/k} \quad (2.1)$$

gdzie:

- S – wypadkowe przyspieszenie,
- f – część pracy w trybie szybkim,
- k – przyspieszenie w trybie szybkim.

Z równania tego wynika, że nawet niewielka część pracy spędzana przez system w trybie wolnym niweluje znaczne przyspieszenie jakie można byłoby uzyskać przez zwiększenie szybkości reszty elementów. Dla przykładu jeśli system spędza 10% czasu w oczekiwaniu na operacje wejścia-wyjścia, to dowolne przyspieszanie procesora i pamięci nie zwiększy całkowitej szybkości systemu więcej niż 10 krotnie. Efekt ten ma szczególnie duże znaczenie w przypadku aplikacji nastawionych na wejście-wyjście.

Ograniczenia związane z dużym czasem dostępu można niwelować używając pamięci buforujących, pozwalających ograniczyć liczbę odwołań do dysku. Niestety metoda ta nadaje się tylko dla systemów charakteryzujących się lokalnością odwołań do pamięci.

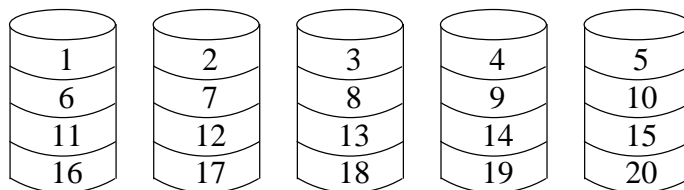
Innym sposobem jest wykorzystanie możliwości jakie daje równoległe wykonywanie operacji dyskowych. Samo zrównoleglenie operacji dyskowych nie zmniejsza czasu dostępu, ale pozwala podwyższyć liczbę operacji wejścia-wyjścia na sekundę (przepustowość), jak też zwiększyć szybkość przesyłu informacji. Aby móc równoległe wykonywać operacje dyskowe, należy użyć układu wielu dysków. Układy takie, nazwane macierzami dyskowymi, zostały zaproponowane w roku 1980.

2.3.2. Konfiguracja a parametry

Poprzez zmiany w konstrukcji macierzy można kształtować jej charakterystykę, dopasowując ją do możliwych zastosowań. Systemy obliczeniowe przetwarzające duże ilości danych najczęściej wymagają niewielkiej przepustowości, ale dużej szybkości przesyłu informacji. Przeciwnne wymagania mają systemy transakcyjne obsługujące jednocześnie wiele niezależnych żądań. Potrzebują one dużej przepustowości, natomiast mniej ważna jest szybkość przesyłu informacji. Istnieją też systemy pośrednie, dla których ważna jest zarówno przepustowość, jak i szybkość przesyłu informacji.

Podstawowym sposobem kształtowania charakterystyki wydajnościowej macierzy jest ustalenie wielkości bloku paskowania. **Paskowaniem** (ang. *striping*) nazywamy rozdział danych między dyski, w porcjach zwanych **blokami paskowania** (ang. *striping unit*).

Idea paskowania przedstawiona jest na rysunku 2.2.



Rysunek 2.2: Paskowanie. Bloki czytane w kolejności numeracji odpowiadają wyjściowym danym

Ustalając wielkość bloku paskowania na poziomie mniejszym niż przeciętny rozmiar żądania dla dysku, możemy zwiększyć szybkość przesyłu informacji, ponieważ równoległe odbywa się przesył danych do kilku dysków. Jest to pożądane w pierwszym schemacie korzystania z dysku, gdzie zależy nam na szybkości przesyłu informacji. Jeśli zrobimy tak dla drugiego schematu, to spowodujemy obniżenie wydajności, ponieważ jedna operacja będzie musiała odwołać się do kilku dysków. Dla aplikacji wymagającej dużej liczby operacji wejścia-wyjścia będzie to ograniczeniem.

Analogicznie ustalając wielkość bloku paskowania na poziomie większym niż przeciętny rozmiar żądania, poprawiamy charakterystykę macierzy dyskowej dla drugiego sposobu korzystania z dysku. W tym wypadku każde żądanie będzie obsługiwane przez jeden dysk, dzięki czemu kilka żądań może zostać obsłużonych równoległe. Zbytne zwiększenie wielkości bloku paskowania może obniżyć szybkość przesyłu informacji, ponieważ pojedyncze żądanie zostanie rozbite na mniejszą liczbę dysków. Może też ograniczyć równoległość, jeśli kilka żądań odwołuje się do tego samego bloku.

2.3.3. Macierze RAID

Użycie wielu dysków powoduje obniżenie niezawodności budowanych pamięci. Niezawodność pojedynczego dysku określona przez MTTF wynosi ok. kilkudziesięciu lat. Przy wzroście liczby dysków średni czas do awarii układu obliczany jest ze wzoru:

$$MTTF_N = \frac{MTTF}{N} \quad (2.2)$$

gdzie:

$MTTF_N$ – wypadkowy średni czas do awarii układu,

$MTTF$ – średni czas do awarii pojedynczego dysku,

N – liczba dysków w układzie.

Zależność ta zachodzi przy założeniu niezależności awarii poszczególnych dysków oraz stałej wartości rozkładu awarii w czasie. Wynika stąd, że dla 100 dysków średni czas do awarii wynosi kilka miesięcy, przy założeniu MTTF pojedynczego dysku na poziomie kilkudziesięciu lat. Jest to wartość nieakceptowalna w większości zastosowań.

Aby podwyższyć niezawodność, używa się dodatkowych dysków z danymi nadmiarowymi. W roku 1987 badacze z Uniwersytetu Kalifornijskiego w Berkley wprowadzili na określenie takich konstrukcji termin **RAID**, jednocześnie podając ich klasyfikację. Idea ta została opublikowana w roku 1988 [Patterson88]. Termin RAID początkowo będąc skrótem określenia **Redundant Array of Inexpensive Disks** czyli **nadmiarowa macierz tanich dysków**, obecnie traktowany jest raczej jako skrót nazwy **Redundant Array of Independent Disks**, to znaczy **nadmiarowa macierz niezależnych dysków**. Pierwsza nazwa związana jest z kontekstem w jakim zaproponowano użycie nadmiarowych macierzy dyskowych. Wtedy miały być alternatywą dla kosztownych dysków specjalizowanych, same budowane z tanich ogólnie dostępnych dysków powszechnego użytku. W związku z możliwością budowy macierzy nie tylko z tanich dysków, lepszym określeniem wydaje się drugie i dzisiaj to ono jest głównie używane.

Obliczanie informacji nadmiarowych jest ściśle związane ze strukturą danych narzuconą przez paskowanie. Informacje takie są zwykle obliczane na poziomie bloków paskowania.

Informacje nadmiarowe obliczane są dla pewnej grupy dysków, która niekoniecznie musi się pokrywać z całą macierzą. Grupa dysków, dla której obliczana jest informacja nadmiarowa nazywana jest **grupą parzystości** (ang. *parity group*). Cała macierz może się składać z kilku grup parzystości. Wraz ze wzrostem rozmiaru grupy parzystości pogarszają się parametry macierzy w przypadku awarii dysku, ponieważ rekonstrukcja uszkodzonego dysku wymaga odczytania wszystkich dysków z danej grupy. Może też obniżyć się niezawodność. Zmniejszanie grupy wiąże się ze wzrostem kosztu macierzy, ponieważ ta sama liczba dysków nadmiarowych przypada na mniejszą liczbę dysków z danymi.

2.4. Klasyfikacja RAID

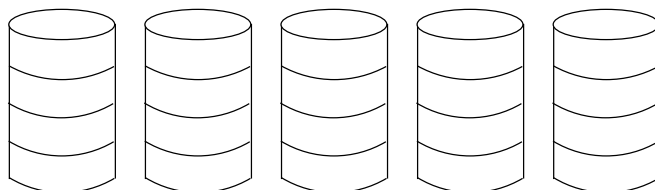
2.4.1. Wstęp

Od czasu wprowadzenia termin RAID i odpowiedniej klasyfikacji, część typów macierzy wtedy wyodrębnionych wyszła z użycia, pojawiło się też kilka nowych. W pracy [Patterson88] opisane zostały macierze stopni od 1 do 5. Obecnie klasyfikuje się także stopnie 0 i 6.

Przepustowość, szybkość przesyłu informacji, niezawodność oraz koszt są podstawowymi kryteriami oceny poszczególnych typów macierzy.

2.4.2. RAID 0

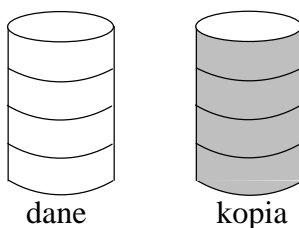
Nie posiada dysków nadmiarowych, wszystkie dyski służą do przechowywania danych, przez co ma najniższy koszt w przeliczeniu na uzyskiwaną pojemność. Zapewnia wszystkie korzyści jakie dają macierze dyskowe, lecz nie posiadając nadmiarowości jest podatna na awarię dowolnego z dysków. Przez to nie jest macierzą RAID w klasycznym rozumieniu. Ma najlepsze osiągi spośród macierzy RAID jeśli chodzi o operacje zapisu, bo nie musi aktualizować informacji nadmiarowych. Jeśli chodzi o operacje odczytu, to lepsze mogą być macierze nadmiarowe. Znajduje zastosowanie tam, gdzie liczą się głównie wydajność i pojemność, a mniejsze znaczenie ma niezawodność.



Rysunek 2.3: Raid 0

2.4.3. RAID 1

Na każdy dysk z danymi przypada dysk nadmiarowy zawierający kopię tych danych. Pozwala to zabezpieczyć się przed awarią jednego z dysków, bo dane zawsze można odczytać z drugiego. Przy zapisie informacje zapisywane są jednocześnie na obu dyskach, co powoduje że czas zapisu jest maksymalnym z czasów zapisu na pojedynczym dysku. Mimo tego ma najlepszy czas zapisu wśród macierzy nadmiarowych. Szybkość przesyłu danych i przepustowość mogą być zwiększone dzięki użyciu dwu dysków do odczytu informacji. Podstawową wadą jest konieczność przeznaczania połowy dysków na kopie danych. RAID 1 może też zostać rozszerzony do postaci, w której na jeden dysk z danymi przypada kilka dysków będących jego kopiami.

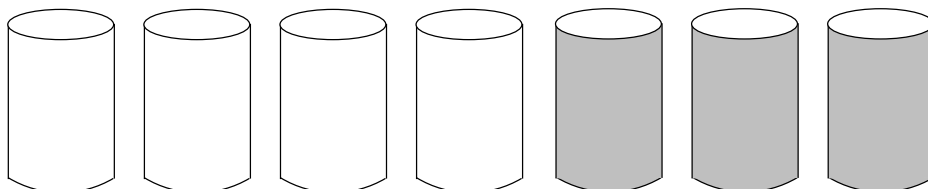


Rysunek 2.4: Raid 1

2.4.4. RAID 2

Z uwagi na znaczny koszt nadmiarowości RAID 1 poszukiwano lepszych rozwiązań. Tak powstał RAID 2, którego idea wywodzi się z kodów korygujących błędy. W tym przypadku użyto kodu Hamminga, który pozwala na określenie, na którym bicie nastąpił błąd i poprawienie go. Stosując kod Hamminga i używając n dysków, $\lceil \log_2(n + 1) \rceil$ spośród nich należy

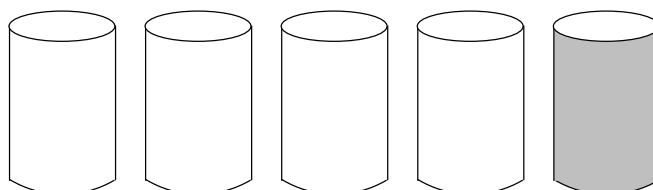
przeznaczyć na dyski z informacjami nadmiarowymi. Rozdzielanie danych na poszczególne dyski wykonuje się tutaj na poziomie bitu. Odpowiada to wielkości bloku paskowania równej 1 bit, przez co każda operacja wymaga dostępu do wszystkich dysków. Dla tak małych wielkości bloku paskowania opłacalne jest stosowanie dysków pozwalających na synchronizację obrotów. Uzyskano tu zwiększenie szybkości przesyłu, natomiast przepustowość nie ulega zmianie lub może się nawet pogorszyć. Operacje zapisu będące wielokrotnością rozmiaru sektora pomnożonego przez liczbę dysków z danymi wykonują się sprawnie, ponieważ na dyski trafiają całe sektory. Dla operacji zapisu, dla których to nie zachodzi, należy wykonać odczyt-modyfikację-zapis na każdym z dysków. Wynika to stąd, że dostęp do poszczególnych dysków odbywa się na poziomie sektorów, dlatego przy zapisie mniejszego kawałka danych należy odczytać sektor, na który ma on trafić, zmienić go i dopiero zapisać. Ten rodzaj macierzy wyszedł praktycznie z użycia, ponieważ współczesne dyski pozwalają na stwierdzenie czy dane na nich zapisane zostały uszkodzone. Macierz w takiej konfiguracji zbudowana z dysków wykrywających błędy mogłaby być odporna na awarię dwu dysków.



Rysunek 2.5: Raid 2

2.4.5. RAID 3

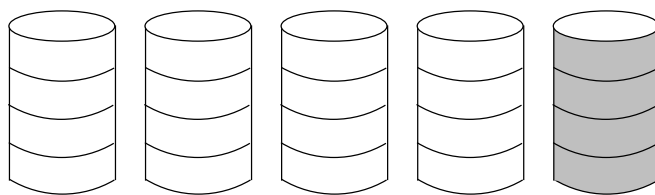
W RAID 2 część dysków nadmiarowych jest potrzebna do wykrycia, na którym dysku nastąpił błąd. Zastosowanie dysków samodzielnie sprawdzających błędy powoduje, że część dysków nadmiarowych staje się zbędna. Jeśli wiadomo, na którym dysku nastąpił błąd, to wystarczy tylko jeden dysk nadmiarowy, aby móc ten błąd naprawić. RAID 3 pozwala dla dowolnej liczby dysków z danymi użyć tylko jednego dysku nadmiarowego. Informacje na tym dysku obliczane są przy wykorzystaniu funkcji XOR po odpowiednich bitach wszystkich dysków z danymi. Dysk ten nazywany jest **dyskiem parzystości** (ang. *parity disk*). Także tutaj stosowany jest rozdział danych na dyski na poziomie 1 bitu. Każde żądanie odczytu używa wszystkich dysków z danymi, a każde żądanie zapisu zapisuje wszystkie dyski z danymi i dysk parzystości. Tak jak dla RAID 2 małe operacje zapisu wymagają sekwencji odczyt-modyfikacja-zapis przez co przepustowość jest nie lepsza niż dla pojedynczego dysku. Poprawę uzyskujemy w odniesieniu do szybkości przesyłu informacji. Macierz ta również wyszła z użycia.



Rysunek 2.6: Raid 3

2.4.6. RAID 4

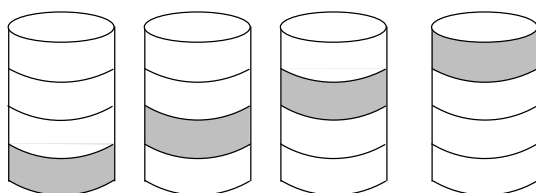
RAID 4 różni się od RAID 3 wielkością bloku paskowania. O ile w RAID 3 był on wielkości bitu, to tutaj może mieć dowolny rozmiar. Dzięki temu poprawiając przepustowość dla operacji odczytu, RAID 4 zachowuje zalety RAID 3. Przy większym rozmiarze bloku paskowania mała operacja odczytu nie używa wszystkich dysków, więc możliwe jest jednoczesne wykonywanie wielu takich operacji. Szybkość przesyłu informacji poprawia się dla dużych operacji, ponieważ informacje czytane są z kilku dysków jednocześnie. Gorzej wygląda sprawa zapisu. O ile dla dużych operacji zapisujących jednocześnie wszystkie dyski rośnie szybkość przesyłu informacji, o tyle małe operacje napotykają na wąskie gardło związane z koniecznością aktualizacji danych na dysku parzystości. Jest to powodem, dla którego przepustowość dla operacji zapisu takiej macierzy jest gorsza niż dla pojedynczego dysku.



Rysunek 2.7: Raid 4

2.4.7. RAID 5

Metodą pozwalającą zlikwidować wąskie gardło związane z aktualizacją informacji nadmiarowych w RAID 4 jest **rozzucanie** (ang. *distribute*) bloków parzystości na różne dyski. Dzięki temu RAID 5 pozwala na wykonywanie kilku operacji zapisu jednocześnie, o ile nie odwołują się one do tych samych dysków. RAID 5 zachowuje przy tym większość zalet macierzy poprzednich stopni. Lepsze parametry jeśli chodzi o czas zapisu spośród macierzy prawdziwie nadmiarowych ma tylko RAID 1. Związane jest to z koniecznością wykonywania przez RAID 5 operacji odczytu-modyfikacji-zapisu przy zapisach małych ilości danych.

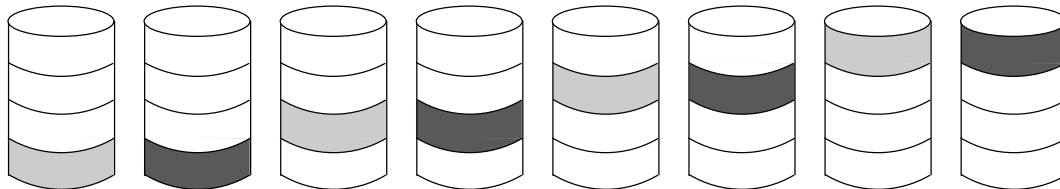


Rysunek 2.8: Raid 5

2.4.8. RAID 6

W celu podwyższenia niezawodności w RAID 6 dodano drugi dysk nadmiarowy. Pozwala to macierzy przetrwać awarię dwu dysków. Na każdą grupę bloków z danymi przypadają dwa bloki zawierające informacje nadmiarowe. Pierwszy z nich zawiera sumę XOR bloków z danymi. Drugi musi zawierać informacje obliczone inną metodą, tak aby można było obliczyć brakujące dane, w przypadku awarii dowolnej pary dysków. Sposób obliczenia informacji nadmiarowych dla drugiego dysku nie został sprecyzowany w klasyfikacji tego stopnia RAID.

Możliwe jest tu zastosowanie np. kodu EVENODD [BBBM95], lub kodów Reeda-Solomona [MS77]. Macierz tego typu nazywana jest też macierzą P+Q. Litery P i Q oznaczają dwa sposoby obliczania informacji nadmiarowych, P to parzystość, natomiast Q to pewna funkcja niezależna od P.



Rysunek 2.9: Raid 6

2.5. Niezawodność

Dodanie dysków nadmiarowych pozwala podwyższyć niezawodność określaną przez MTTF. Przy założeniu, że awarie dysków macierzy są od siebie niezależne, MTTF macierzy RAID z jednym dyskiem nadmiarowym wyraża się wzorem:

$$\frac{MTTF^2}{N * (G - 1) * MTTR}$$

gdzie:

MTTF – średni czas do awarii pojedynczego dysku,

MTTR – średni czas naprawy pojedynczego dysku,

N – całkowita liczba dysków w macierzy,

G – liczba dysków w grupie parzystości.

Wzór ten definiuje prawdopodobieństwo awarii macierzy przez prawdopodobieństwo awarii drugiego dysku w czasie między awarią pierwszego dysku a jego wymianą i rekonstrukcją zawartości.

Przy założeniu, że MTTF pojedynczego dysku wynosi 200.000 godzin, a MTTR 1 godzinę, dla 100 dysków w grupach parzystości po 16 otrzymujemy MTTF całego układu rzędu 3000 lat.

Dla macierzy posiadającej dwa dyski nadmiarowe analogiczny wzór na MTTF to:

$$\frac{MTTF^3}{N * (G - 1) * (G - 2) * MTTR^2}$$

Dla parametrów jak w poprzednim przykładzie obliczona wartość MTTF to 38 milionów lat [Chen93].

Obliczenia te pokazują jak dużą poprawę niezawodności można by było uzyskać stosując macierze RAID, przy założeniu niezależności awarii dysków i przyjmując, że jedynym źródłem awarii jest uszkodzenie dysku. Niestety w rzeczywistości należy uwzględnić takie czynniki jak awarie systemu oraz to, że uszkodzenia dysków nie muszą być niezależne.

Awarie systemu mogą powodować utratę synchronizacji macierzy doprowadzając do sytuacji, w której dane nadmiarowe nie są spójne z danymi właściwymi. W takiej sytuacji uszkodzenie choćby jednego dysku może spowodować utratę danych. Przykładem awarii systemu, w trakcie której dochodzi do rozsynchronizowania macierzy jest wyłączenie zasilania

w trakcie zapisu. Metodą pozwalającą się zabezpieczyć przed awarią systemu jest zastosowanie szybkich pamięci trwałych o czasach dostępu porównywalnych z pamięcią RAM. Są one stosowane głównie w macierzach zrealizowanych sprzętowo.

Przy uwzględnieniu wszystkich powyższych możliwych przyczyn awarii okazuje się, że dla jednego dysku nadmiarowego, przy typowych założeniach dotyczących parametrów dysków, w czasie dziesięciu lat prawdopodobieństwo awarii może osiągnąć wielkość 31% [Chen93]. Z tego względu ważne staje się podwyższenie niezawodności macierzy między innymi przez zwiększenie liczby dysków nadmiarowych.

2.6. Metody realizacji macierzy RAID

Macierze RAID mogą być realizowane jako kontrolery sprzętowe lub sterowniki programowe. Kontrolery sprzętowe pozwalają na korzystanie z macierzy w podobny sposób jak z zwykłego dysku nie stanowiąc obciążenia dla głównego systemu. Sam kontroler z reguły zawiera samodzielny procesor wykonujący odpowiedni program obsługi. Dodatkowo macierze sprzętowe korzystające z pamięci podtrzymywanej bateryjnie mogą osiągać lepszą wydajność przy dużej liczbie zapisów. Macierz zrealizowana programowo może być tańsza oraz pozwolić na większą elastyczność konfiguracji. Programowe realizacje macierzy RAID dołączone są do niektórych systemów operacyjnych. W systemie Linux dostępne są sterowniki macierzy RAID stopni 1, 4, 5 oraz 6. Sterowniki macierzy programowych zawiera także system FreeBSD. W systemie AIX możliwa jest budowa macierzy RAID1 przez właściwą konfigurację podsystemu urządzeń dyskowych.

Rozdział 3

Kody korygujące wymazania a macierze RAID

3.1. Wprowadzenie

Uszkodzenia danych na dyskach można traktować jak wymazania. Pozwala to stosować do ochrony danych w macierzach dyskowych kody korygujące wymazania. W tym rozdziale poruszam zagadnienia związane z zastosowaniem kodów korygujących wymazania w macierzach RAID. Rozważane będą kody liniowe i \mathbb{F}_q -liniowe.

3.2. Terminologia

Przez kod będzie określany podzbiór $C \subseteq \Sigma^n$, gdzie Σ jest alfabetem, a n to długość słowa kodowego. Dla kodu C definiujemy **minimalną odległość** d jako

$$d = \min_{x \neq y \in C} \Delta(x, y),$$

gdzie Δ jest odległością Hamminga. Błąd w słowie kodowym to przekłamanie symbolu alfabetu na jednej z pozycji słowa. Wymazanie w słowie kodowym to utrata informacji o symbolu alfabetu na ustalonej pozycji słowa. Kod o minimalnej odległości d jest w stanie naprawić $d - 1$ wymazań lub $\lfloor \frac{d-1}{2} \rfloor$ błędów [MS77]. Przy ustalonym alfabcie Σ typ kodu można opisać przez trójkę (n, k, d) , gdzie n to długość słowa kodowego, $k = \log_{|\Sigma|} |C|$, a d minimalna odległość kodu. Ograniczenie Singletona dla kodu (n, k, d) [Sin64] wyraża się nierównością

$$d \leq n + 1 - k.$$

Kody, dla których powyższa nierówność staje się równością, to kody **optymalne** (ang. *Maximum Distance Separable*, MDS). Dla kodów optymalnych typ można określić podając parę (n, k) .

Interesować nas będą **kody liniowe**. Kod C jest kodem liniowym gdy C jest podprzestrzenią liniową przestrzeni \mathbb{F}_q^n , gdzie \mathbb{F}_q jest ciałem skończonym o q elementach. Typ kodu liniowego oznaczany jest przez $[n, k, d]$, optymalne kody liniowe oznaczane są przez parę $[n, k]$.

Operacja kodowania dla kodu (n, k, d) nad alfabetem Σ polega na przypisaniu słowu $x \in \Sigma^k$ słowa kodowego $y \in \Sigma^n$. Dla kodów liniowych może być to zapisane jako

$$\vec{y}^T = \vec{x}^T * G,$$

gdzie G to **macierz tworząca** kodu, a \vec{x} i \vec{y} to wektory odpowiadające słowom x i y . Definiuje się także **macierz kontroli parzystości** H o własności $H * \vec{y} = 0$ wtedy i tylko wtedy, gdy $y \in C$. Macierz G jest w **postaci standardowej** gdy można ją przedstawić jako $[I_k|A]$. Odpowiadająca jej postać standardowa macierzy H to $[-A^T|I_{n-k}]$. Kod, którego macierz kontroli parzystości ma postać standardową nazywamy kodem **systematycznym**. Minimalna odległość d kodu zależy od macierzy kontroli parzystości. Jeśli dowolne r kolumn macierzy kontroli parzystości kodu liniowego jest liniowo niezależne oraz pewne $r+1$ kolumn jest liniowo zależne, to minimalna odległość d kodu wynosi $r+1$ [MS77].

3.3. Użycie kodu w macierzy RAID

Jednym ze sposobów wykorzystania kodu do zabezpieczenia danych w macierzy jest kodowanie tych danych połączone z rozmieszczeniem poszczególnych symboli słów kodowych na różnych dyskach. Uszkodzenie danych na jednym z dysków jest wymazaniem zapisanych na nim symboli alfabetu. Jeśli minimalna odległość kodu wynosi d , to możliwe jest odzyskanie słowa kodowego, mimo wymazania $d-1$ symboli. Zapisując na jednym dysku nie więcej niż jeden symbol z każdego słowa kodowego, możemy odzyskać dane nawet w przypadku uszkodzenia $d-1$ dysków.

Dostęp do danych macierzy odbywają się na poziomie bloków będących wielokrotnościami sektorów. Wybierając wielkość symbolu alfabetu dla kodu tak, by mieściła się całkowitą liczbę razy w bloku, możemy obliczać informacje nadmiarowe bez konieczności dostępu do bloków sąsiednich. Ponieważ wielkość bloku w bitach jest z reguły potęgą dwójki, rozmiar symbolu w bitach również powinien być potęgą dwójki.

Do zapisania informacji jakie można zmieścić na k dyskach bez kodowania, korzystając z kodu typu $[n, k, d]$, należy użyć n dysków. Chcąc przy ustalonej liczbie wymazań, które jest w stanie poprawić kod, zminimalizować liczbę dodatkowych dysków, należy zastosować kod optymalny. Dla takiego kodu zabezpieczenie przed uszkodzeniem danych na r dyskach wymaga dodania r dysków nadmiarowych. Zastosowanie kodu optymalnego obniża koszt macierzy, dlatego rozważane będą tylko kody optymalne.

Chęć uproszczenia kodowania oraz możliwość odczytu danych bez dekodowania są argumentami za użyciem kodu systematycznego. Dla kodu systematycznego $[n, k, d]$ kodowanie może być opisane jako dołączenie $n-k$ symboli dodatkowych do k -symboli słowa wejściowego. Oznaczmy przez $a_{i,j}$ i -ty symbol na j -tym dysku, wtedy kodowanie wymaga obliczenia symboli $a_{i,k}, \dots, a_{i,n-1}$ przy ustalonych $a_{i,0}, \dots, a_{i,k-1}$. Ponieważ sposób kodowania jest niezależny od i , dla uproszczenia notacji przyjmijmy $a_{i,j} = \alpha_j$. Przy takim oznaczeniu kodowanie opisuje równanie

$$\begin{pmatrix} \alpha_k \\ \vdots \\ \alpha_{n-1} \end{pmatrix} = A^T * \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}, \quad (3.1)$$

które jest formą zapisu równania

$$H * \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} = 0, \quad (3.2)$$

gdzie H jest macierzą kontroli parzystości w postaci standardowej $[-A^T|I_{n-k}]$.

Dla kodu systematycznego na dyski od 0 do $k-1$ dane wejściowe trafiają bezpośrednio, a dyski od k do $n-1$ są dyskami z danymi nadmiarowymi obliczonymi za pomocą kodu.

Podział ten jest pewnym uproszczeniem, ponieważ w rzeczywistych zastosowaniach położenie danych nadmiarowych nie musi być ograniczone do jednej grupy dysków. Dla uniknięcia efektu wąskiego gardła dyski, na które trafią dane nadmiarowe, mogą być określane na przykład w zależności od aktualnego paska.

Głównym zadaniem kodu korygującego wymazania jest odtwarzanie danych w przypadku ich uszkodzenia. Do obliczenia danych z uszkodzonych sektorów można wykorzystać równanie (3.2). Jeśli przez $\alpha_{j_0}, \dots, \alpha_{j_{r-1}}$ oznaczymy symbole, które chcemy obliczyć, to będzie to możliwe jeśli kolumny j_0, \dots, j_{r-1} macierzy H są liniowo niezależne. Jest to prawda, o ile $r < d$, gdzie d jest minimalną odległością kodu. Niech H' oznacza macierz powstałą przez wybranie z macierzy H tych wierszy, dla których kolumny j_0, \dots, j_{r-1} macierzy H' utworzą kwadratową macierz nieosobliwą. Wtedy

$$[H'_{j_0}, \dots, H'_{j_{r-1}}] * \begin{pmatrix} \alpha_{j_0} \\ \vdots \\ \alpha_{j_{r-1}} \end{pmatrix} + [H'_{j_r}, \dots, H'_{j_{n-1}}] * \begin{pmatrix} \alpha_{j_r} \\ \vdots \\ \alpha_{j_{n-1}} \end{pmatrix} = 0,$$

gdzie H'_j oznacza j -tą kolumnę macierzy H' , a $\{j_0, \dots, j_{r-1}\} \cup \{j_r, \dots, j_{n-1}\} = \{0, \dots, n-1\}$. Przekształcając mamy

$$\begin{pmatrix} \alpha_{j_0} \\ \vdots \\ \alpha_{j_{r-1}} \end{pmatrix} = -[H'_{j_0}, \dots, H'_{j_{r-1}}]^{-1} * [H'_{j_r}, \dots, H'_{j_{n-1}}] * \begin{pmatrix} \alpha_{j_r} \\ \vdots \\ \alpha_{j_{n-1}} \end{pmatrix}. \quad (3.3)$$

Równanie (3.1) może być wykorzystywane przy zapisie danych do macierzy natomiast (3.3) do odtwarzania danych.

3.4. Kody oparte na kodach Reeda-Solomona

Poszukując optymalnych kodów liniowych na ciałach skończonych należy rozważyć kody Reeda-Solomona. Głównymi zagadnieniami jakie wiążą się z ich użyciem w macierzach RAID jest wydajna realizacja operacji na elementach ciała, przekształcenie w kod systematyczny oraz skrócenie kodu. Mówiąc o ciałach skończonych będziemy mieć na myśli głównie te, które bezpośrednio odwzorowują się do postaci binarnej, czyli ciała \mathbb{F}_{2^t} .

3.4.1. Operacje na elementach ciał skończonych \mathbb{F}_{2^t}

Dowolne ciało skończone \mathbb{F}_{p^t} (p pierwsze) jest izomorficzne z ciałem reszt $\mathbb{F}_p[x]/P(x)$, gdzie $\mathbb{F}_p[x]$ jest zbiorem wielomianów o współczynnikach z \mathbb{F}_p , a $P(x) \in \mathbb{F}_p[x]$ jest wielomianem nierozkładalnym stopnia t [MS77, BKKKLZ95]. W przypadku $p = 2$ elementom \mathbb{F}_{2^t} można przypisać wielomiany nad \mathbb{F}_2 stopnia mniejszego od t . Wielomiany takie mogą być reprezentowane przez t -bitowe wektory. Jeśli elementowi a przypiszemy wielomian $a(x) = \sum_{i=0}^{t-1} a_i x^i$, to wektorem binarnym reprezentującym $a(x)$ jest $\vec{a} = (a_0, a_1, \dots, a_{t-1})^T$. Dla takiej reprezentacji operacja dodawania dwu elementów \mathbb{F}_{2^t} odpowiada operacji XOR na reprezentujących je wektorach bitów. Operacja mnożenia jest bardziej skomplikowana, można ją jednak sprowadzić do operacji XOR wykorzystując macierzową reprezentację elementów ciała. Innym sposobem jest stabicowanie samej operacji mnożenia lub wykorzystanie tablicy potęg i logarytmów elementu pierwotnego w ciele. Rozmiar tablicy mnożeń rośnie kwadratowo ze wzrostem liczby elementów w ciele, natomiast rozmiary tablic potęg i logarytmów rosną liniowo, przez co lepiej nadają się dla ciał o większej liczbie elementów. Obliczanie elementu odwrotnego również można stabicować.

Metoda mnożenia przez tablicowanie jest szybsza w przypadku operacji na pojedynczych elementach ciała. Jednak mnożenie wielu elementów przez pewien ustalony element, może być wykonane znacznie wydajniej z użyciem reprezentacji macierzowej tego elementu. Postać ta pozwala sprowadzić operację mnożenia do operacji XOR, która operując na słowach procesora może wykonać wiele jednoczesnych XOR na bitach. Przez właściwe rozmieszczenie bitów reprezentujących element ciała możliwe jest wykonanie wielu mnożeń jednocześnie. Sytuacja, w której należy wykonać wiele mnożeń przez ustalony element ma miejsce podczas kodowania z użyciem macierzy generującej kodu, gdy ustalone wartości elementów macierzy mnożone są przez symbole z bloków danych.

Macierzową reprezentację elementów ciała \mathbb{F}_{2^t} można wprowadzić korzystając z pojęcia **macierzy stowarzyszonej** [MS77, s. 106], [BKKKLZ95]. Niech $P(x) \in \mathbb{F}_2[x]$ będzie wielomianem nierozkładalnym stopnia t . Dla danego elementu $a \in \mathbb{F}_{2^t}$ niech $a(x) = \sum_{i=0}^{t-1} a_i x^i$ będzie wielomianem reprezentującym a . Wtedy macierz $M_a = (a_{i,j})$, gdzie i -tą kolumnę stanowi wektor $(a_{i,0}, a_{i,1}, \dots, a_{i,t-1})^T$ odpowiadający wielomianowi $a(x) * x^i \bmod P(x)$, nazywany macierzą stowarzyszoną z $a(x)$ nad \mathbb{F}_{2^t} . W ten sposób każdemu elementowi \mathbb{F}_{2^t} można przyporządkować macierz $t \times t$. Przykładowo dla $P(x) = x^4 + x^1 + 1$ oraz $a(x) = x^3 + x^2 + 1$ macierz stowarzyszona z $a(x)$ to

$$M_a = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

Jeśli macierz M_a jest macierzą reprezentującą a , to można znaleźć wektor binarny reprezentujący $a * b$ wykonując mnożenie wektora binarnego reprezentującego b przez M_a . Można to zapisać jako

$$\vec{ab} = M_a * \vec{b}.$$

W mnożeniu tym wykonujemy operacje XOR na pojedynczych bitach. Zapisując reprezentacje binarne elementów ciała \mathbb{F}_{2^t} tak, by i -te bity, $0 \leq i < t$, każdej z nich znalazły się we spójnych fragmentach pamięci, możemy w jednej instrukcji procesora wykonać do w operacji XOR na bitach, gdzie w jest liczbą bitów w słowie procesora. Pozwala to w -krotnie przyspieszyć mnożenie w stosunku do operacji na pojedynczych bitach. Do mnożenia tą metodą należy podzielić blok danych na t części (t jest liczbą bitów reprezentacji elementu \mathbb{F}_{2^t}), na których wykonujemy jedynie operacje XOR [BKKKLZ95]. Bity w i -tej części bloku odpowiadają i -tym bitom odpowiednich elementów ciała.

Podstawową operacją kodową jest mnożenie bloku danych X traktowanego jako zbiór elementów ciała \mathbb{F}_{2^t} przez ustalony element a , wynik jest dodawany (XOR) do bloku wynikowego Y . Operacja ta może być zapisana jako

$$Y = Y + a * X$$

a składa się z działań na elementach ciała. Jej koszt można oszacować porównując ją z kosztem XOR dwu bloków, jaki odpowiada obliczaniu informacji nadmiarowych dla RAID 5. Dla macierzy reprezentującej element ciała \mathbb{F}_{2^t} koszt operacji zależy od liczby jedynek w macierzy. Macierz jednostkowa w takiej interpretacji odpowiada dodaniu dwu bloków, czyli XOR na tych blokach. Szacując średnią liczbę jedynek w macierzy reprezentującej element ciała \mathbb{F}_{2^n} możemy określić przeciętny koszt podstawowej operacji kodowej. Przykładowo dla \mathbb{F}_{2^4} średnia liczba jedynek w macierzy reprezentującej niezerowy element tego ciała wynosi 8.533, co jest ok. 2 razy większe niż liczba jedynek macierzy jednostkowej 4×4 . Stąd przeciętna

operacja kodowa dla \mathbb{F}_{2^4} ma koszt zbliżony do kosztu dwu operacji XOR na całym bloku. Odpowiednio dla \mathbb{F}_{2^8} średnia liczba jedynek wynosi 32.125, co stanowi ok. 4-krotność liczby jedynek w macierzy jednostkowej 8×8 .

3.4.2. Macierz kontroli parzystości

Macierz kontroli parzystości dla kodu Reeda-Solomona nad ciałem \mathbb{F}_q ma postać

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^{2*2} & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{(n-k)2} & \dots & \alpha^{(n-k)(n-1)} \end{pmatrix},$$

gdzie α jest elementem pierwotnym, a $n = q - 1$. Dowolne $n - k$ kolumn tej macierzy jest liniowo niezależne, a zatem kod jest optymalny typu $[n, k]$ [MS77]. Przekształcając macierz do postaci standardowej uzyskamy kod systematyczny. Aby zachować optymalność kodu, przekształcenie nie może zmniejszyć rzędu macierzy.

Oto metoda przekształcenia H do postaci standardowej:

- oznaczmy przez H_i i -tą kolumną macierzy H , wtedy $H = [H_0, \dots, H_{n-1}]$,
- niech $B = [H_k, H_{k+1}, \dots, H_{n-1}]$ oznacza macierz powstałą przez wybranie ostatnich $n - k$ kolumn macierzy H , jest to macierz kwadratowa odwracalna,
- mnożymy macierz H przez B^{-1} , co daje macierz $H' = [H'_0, \dots, H'_{n-1}] = B^{-1} * H$
- odrzucamy zbędne kolumny macierzy (skrótowanie kodu).

Innym sposobem pozwalającym uzyskać macierz kontroli parzystości w postaci standardowej jest użycie macierzy Cauchy'ego [BKKKLZ95]. Niech $\{x_1, \dots, x_m\}$, $\{y_1, \dots, y_n\}$ będą dwoma zbiorami elementów ciała \mathbb{F} spełniających warunki

- $\forall i \in \{1, \dots, m\} \forall j \in \{1, \dots, n\} : x_i + y_j \neq 0$
- $\forall \{i, j\} \in \{1, \dots, m\}, i \neq j : x_i \neq x_j$
- $\forall \{i, j\} \in \{1, \dots, n\}, i \neq j : y_i \neq y_j$

wtedy macierz

$$\begin{pmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \dots & \frac{1}{x_1+y_n} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \dots & \frac{1}{x_2+y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{x_m+y_1} & \frac{1}{x_m+y_2} & \dots & \frac{1}{x_m+y_n} \end{pmatrix}$$

jest macierzą Cauchy'ego. W macierzy tej dowolna podmacierz jest macierzą Cauchy'ego. Podmacierze kwadratowe są nieosobliwe, a ich odwrotności można znaleźć w czasie $O(n^2)$ [BKKKLZ95].

Macierz kontroli parzystości kodu można zbudować przez dołączenie macierzy jednostkowej $(n - k) \times (n - k)$ do $(n - k) \times k$ macierzy Cauchy'ego. Dowolne $n - k$ kolumn tak powstałej macierzy jest liniowo niezależne, co wynika z nieosobliwości dowolnej podmacierzy kwadratowej macierzy Cauchy'ego.

Rozdział 4

Projekt i implementacja

4.1. Założenia projektowe

4.1.1. Wstęp

Jako środowisko implementacji sterownika wybrany został system operacyjny Linux w wersji 2.6.16. Przystąpienie do implementacji wymagało rozważenia kilku kwestii. Pierwszą z nich było wykorzystanie istniejącego w jądrze podsystemu MD (por. p. 4.1.2). Kolejną był wybór optymalnego pod względem wydajności i komplikacji algorytmu obliczania informacji nadmiarowych. Należało także ustalić jakiego rodzaju ograniczenia można przyjąć na liczbę dysków i stopień nadmiarowości. Ograniczenia te mogą być związane z zastosowanym algorytmem. Użycie większej liczby dysków wiąże się także z koniecznością rozszerzenia sposobów rozdziału danych nadmiarowych.

4.1.2. Podsystem MD

Podsystem jądra MD (Multiple Devices) umożliwia budowę wirtualnych urządzeń blokowych na bazie grupy urządzeń blokowych niższego poziomu (np. dysków). W ten sposób można uzyskać macierze RAID, jak również macierze dyskowe bez redundancji. Wspólne elementy funkcjonalności różnych typów wirtualnych urządzeń blokowych zostały wydzielone w oddzielnym module. Dzięki temu realizacja konkretnego urządzenia wirtualnego wymaga jedynie dostarczenia implementacji funkcji z określonego interfejsu. Kod podsystemu MD można znaleźć w źródłach systemu Linux w katalogach `drivers/md` oraz `include/linux/raid`. Znajduje się tam także kod sterownika RAID 5, na którym w dużym stopniu oparty został opracowany sterownik.

Różne typy urządzeń wirtualnych w podsystemie MD identyfikowane są poprzez ich stopień. Dla urządzeń RAID stopień w MD dobierany jest tak, by pokrywał się ze stopniem RAID, np. dla RAID 5 jest to 5. Ponieważ nie ma dokładnie ustalonych stopni dla liczby dysków nadmiarowych większej niż 2, jako stopień MD implementowanego urządzenia wybrałem 8.

4.1.3. Obliczanie informacji nadmiarowych

Ze względu na wydajność optymalnym rozwiązaniem byłoby stosowanie różnych kodów do obliczania informacji nadmiarowej w zależności od całkowitej liczby dysków oraz liczby dysków nadmiarowych. Przykładowo dla dwu dysków nadmiarowych dobrym rozwiązaniem mógłby być kod EVEN-ODD. Jednak z uwagi na stopień komplikacji postanowiłem zrealizo-

wać sterownik bazując na jednym kodzie, pozwalającym obsłużyć maksymalnie 256 dysków, a jednocześnie wystarczająco wydajnym. Jego kolejną cechą jest możliwość przeznaczenia dowolnej liczby spośród wszystkich dysków na dyski nadmiarowe. Wybrany kod oparty jest na kodzie Reeda-Solomona nad F_{2^8} . Wstępne testy pozwoliły ustalić że podstawowe operacje kodowe dla bloku danych w tym kodzie są ok. 2,5 raza wolniejsze od operacji XOR. Jest to wynik lepszy niż można było się spodziewać analizując liczbę składowych operacji XOR czyli 4, przypadających średnio na jedną operację kodową. Tę rozbieżność może tłumaczyć przechowywanie ostatnio przetwarzanych danych w pamięci podręcznej procesora, dzięki czemu kolejne operacje XOR działające w obrębie ustalonego bloku nie muszą już sięgać do pamięci głównej.

Macierz kontroli parzystości dla kodu wybrałem tak, by zawierała jako podmacierz macierz Cauchy'ego. Dzięki temu jej współczynniki mogą być obliczane w stałym czasie, co pozwala uniknąć przechowywania macierzy w pamięci. Dodatkowo dowolną podmacierz macierzy Cauchy'ego można odwrócić w czasie kwadratowym. Macierz kontroli parzystości została zmodyfikowana tak, by wszystkie niezerowe współczynniki w pierwszym wierszu sprowadzić do jedynek. Wymaga to tylko jednej dodatkowej operacji przy obliczaniu współczynnika macierzy. Także koszt odwracania rośnie o jedną operację na każdy współczynnik odwracanej podmacierzy. Dzięki sprowadzeniu niezerowych elementów pierwszego wiersza do jedynek, pierwszy z bloków nadmiarowych może być obliczony przez XOR na wszystkich blokach danych.

Zaletą wybranego kodu jest koszt odtwarzania zbliżony do kosztu kodowania oraz łatwa aktualizacja bloków nadmiarowych przy zapisach w schemacie odczyt-modyfikacja-zapis. Wybrany kod ma duży zakres możliwych wartości całkowitej liczby dysków (256), a także brak ograniczeń innych niż całkowita liczba dysków na liczbę dysków nadmiarowych. Z uwagi na uproszczenie implementacji wprowadziłem jednak ograniczenie na maksymalną dopuszczalną liczbę dysków nadmiarowych. Zależy ono od stałej `PARITY_DISKS_MAX` definiowanej w pliku `raidn.h`. W obecnej wersji wynosi ono 8.

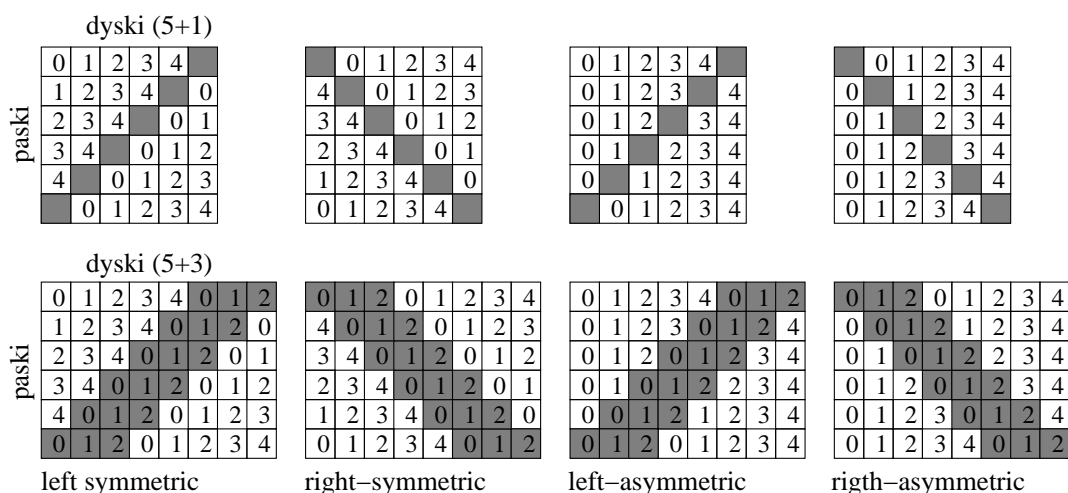
Mimo że sterownik został zrealizowany z użyciem jednego kodu, korzystna byłaby możliwość podmiiany kodowania. Aby to ułatwić dla funkcji odpowiedzialnych za kodowanie zaprojektowany został przejrzysty interfejs a one same umieszczono w oddzielnym pliku.

4.1.4. Rozdział danych nadmiarowych

Aby uniknąć wąskiego gardła związanego z aktualizacją danych nadmiarowych, dyski, na które one trafiają, wybierane są niezależnie dla każdego paska. Metody wyboru dysków dla paska uzyskałem rozszerzając to co oferował sterownik RAID 5. Są to `left-symmetric`, `right-symmetric`, `left-asymmetric` i `right-asymmetric`. Rozszerzenie polega na zastąpieniu pojedynczych dysków parzystości grupą dysków. Rysunek 4.1 przedstawia przykłady odpowiednich metod oraz ich rozszerzenia dla przypadku 5 dysków z danymi właściwymi oraz odpowiednio 1 i 3 dyskami nadmiarowymi.

4.2. Interfejs podsystemu MD

Sterownik działający w obrębie podsystemu MD musi implementować jedynie część funkcjonalności urządzenia składającego się z wielu urządzeń blokowych niższego poziomu. Funkcjonalność związana z utrzymywaniem informacji o konfiguracji, w tym jej zapis i odczyt z dysków, przechowywanie informacji o urządzeniach, ich dodawanie i usuwanie, a także komunikacja z użytkownikiem jest realizowana przez podsystem MD. Stan urządzeń wirtualnych dla potrzeb podsystemu MD przechowywany jest w obiektach typu `mddev_t`. Sterowniki,



Rysunek 4.1: Przykład rozdziału danych nadmiarowych. Bloki zaciemnione są blokami z danymi nadmiarowymi. Numeracja opowiada kolejności bloków w pasku, po wypełnieniu białych bloków paska przechodzimy do następnego.

które chcą związać z urządzeniem dodatkowe informacje mogą zdefiniować typ pomocniczy. Dowiązania do obiektów tego typu przechowywane są w polu `private` z `mddev_t`. Urządzenia składowe reprezentowane są przez obiekty `mdk_rdev_t`. Podsystem MD komunikuje się on ze sterownikami implementującymi poszczególne rodzaje urządzeń wirtualnych przez interfejs określony poniższą strukturą.

```
struct mdk_personality
{
    char    *name;
    int     level;
    struct list_head list;
    struct module    *owner;
    int  (*make_request)(request_queue_t *q, struct bio *bio);
    int  (*run)(mddev_t *mddev);
    int  (*stop)(mddev_t *mddev);
    void (*status)(struct seq_file *seq, mddev_t *mddev);
    void (*error_handler)(mddev_t *mddev, mdk_rdev_t *rdev);
    int  (*hot_add_disk)(mddev_t *mddev, mdk_rdev_t *rdev);
    int  (*hot_remove_disk)(mddev_t *mddev, int number);
    int  (*spare_active)(mddev_t *mddev);
    sector_t (*sync_request)(mddev_t *mddev, sector_t sector_nr,
                             int *skipped, int go_faster);
    int  (*resize)(mddev_t *mddev, sector_t sectors);
    int  (*reshape)(mddev_t *mddev, int raid_disks);
    int  (*reconfig)(mddev_t *mddev, int layout, int chunk_size);
    void (*quiesce)(mddev_t *mddev, int state);
};
```

Sterownik powinien zdefiniować strukturę typu `struct mdk_personality` i wypełnić jej pola wskaźnikami do funkcji, które chce obsłużyć. W trakcie inicjowania, które może się odbywać podczas ładowania modułu bądź startu systemu, w zależności od tego czy sterownik jest zbudowany jako moduł, czy wbudowany w jądro, powinien on wywołać funkcję

```
int register_md_personality(struct mdk_personality *p)
```

przekazując jej jako parametr tak wypełnioną strukturę.

Przyjrzyjmy się poszczególnym funkcjom z interfejsu, ze zwróceniem uwagi na ich zastosowanie w sterowniku.

1. `int (*make_request)(request_queue_t *q, struct bio *bio)`

Funkcja ta pozwala zlecić operację wejścia-wyjścia reprezentowaną przez strukturę `struct bio`.

2. `int (*run)(mddev_t *mddev)`

Inicjowanie kolejnej instancji urządzenia wirtualnego danego typu.

3. `int (*stop)(mddev_t *mddev)`

Zwolnienie zasobów związanych z instancją urządzenia wirtualnego.

4. `void (*status)(struct seq_file *seq, mddev_t *mddev)`

Funkcja wypisująca informacje o stanie urządzenia. Informacje te są dostępne w pliku `/proc/mdstat`.

5. `void (*error_handler)(mddev_t *mddev, mdk_rdev_t *rdev)`

Obsługa błędu jednego z urządzeń, na bazie których działa urządzenie wirtualne realizowane przez sterownik. Dzięki tej funkcji sterownik może zaktualizować swoje informacje o stanie i ewentualnie rozpocząć naprawę (na przykład odtwarzanie na urządzeniu zapasowym). Informacje o uszkodzeniach urządzeń przechowywane są także przez podsystem MD.

6. `int (*hot_add_disk)(mddev_t *mddev, mdk_rdev_t *rdev)`

Dodanie urządzenia. Urządzenie takie staje się urządzeniem zapasowym (ang. *spare*). Jeśli nastąpi bądź już nastąpiła awaria jakichś urządzeń, to zostanie ono użyte do rekonstrukcji macierzy.

7. `int (*hot_remove_disk)(mddev_t *mddev, int number)`

Usunięcie urządzenia, pozwala usunąć uszkodzone bądź zapasowe urządzenie.

8. `int (*spare_active)(mddev_t *mddev)`

Przełączenie urządzenia zapasowego w tryb pełnej funkcjonalności; odbywa się to zwykle po zakończeniu rekonstrukcji.

9. `sector_t (*sync_request)(mddev_t *mddev, sector_t sector_nr,
int *skipped, int go_faster)`

Żądanie synchronizacji, służy do odtworzenia uszkodzonych dysków na dysk zapasowy albo do sprawdzenia spójności danych, np. po wyłączeniu macierzy w trakcie zapisu.

10. `int (*resize)(mddev_t *mddev, sector_t sectors)`

Zmiana rozmiaru urządzenia, jeśli rozmiar jest zwiększany, to każde z urządzeń bazowych powinno posiadać odpowiednią ilość miejsca. Przy zwiększeniu rozmiaru nowy obszar jest synchronizowany.

11. `int (*reshape)(mddev_t *mddev, int raid_disks)`

Zmiana liczby dysków, wiąże się z koniecznością resynchronizacji, nie obsługiwana przez ten sterownik.

12. `int (*reconfig)(mddev_t *mddev, int layout, int chunk_size)`

Zmiana konfiguracji, np. sposobu rozmieszczenia danych na dyskach, nie obsługiwana.

13. `void (*quiesce)(mddev_t *mddev, int state)`

Przełączenie w tryb tylko do odczytu.

4.3. Struktury danych

Konieczność zachowania spójności danych na dyskach wymaga by sposób dostępu uwzględniał strukturę macierzy. Ma to znaczenie szczególnie przy zapisach, gdy trzeba aktualizować dane nadmiarowe na podstawie danych właściwych. Do tego celu potrzebny jest jednoczesny dostęp do bloków danych rozpoczynających się od tego samego sektora każdego z dysków. Realizowane jest to dzięki strukturom `stripe_dev_t` i `stripe_head_t`:

```
typedef struct {
    struct bio      req;
    struct bio_vec  vec;
    struct page     *page;
    struct bio      *toread, *towrite, *written;
    sector_t       sector;
    unsigned long   flags;

    int             disk_idx;
} stripe_dev_t;
```

```
typedef struct {
    struct hlist_node  hash;
    struct list_head  list;
    raidn_conf_t      *conf;
    sector_t          sector;
    unsigned long     state;
    atomic_t          count;
    spinlock_t        lock;
    int               bm_seq;
    stripe_dev_t      devices[0];
} stripe_head_t;
```

Służą one do opisu stanu operacji zleconej macierzy. W `stripe_head_t` znajduje się tablica `devices`, przechowująca elementy typu `stripe_dev_t`, po jednym dla każdego z urządzeń

składowych (dysków) macierzy. Każdy element tablicy zawiera wskaźnik `page` do strony pełniącej rolę bufora bloku danych. W `stripe_dev_t` mamy ponadto pola `req` i `vec`, które wraz ze stroną `page` służą do przekazywania żądań urządzeniom składowym macierzy. Wskaźniki `toread`, `towrite`, `written` przechowują listy oczekujących operacji, odpowiednio odczytu, zapisu oraz zapisu, który został skopiowany do strony buforującej. Pole `sector` dla `stripe_dev_t` określa pozycję bloku danych ze strony buforującej w macierzy RAID, natomiast `sector` w `stripe_head_t` określa pozycję tego bloku na urządzeniu składowym. Wszystkie bloki danych ze stron buforujących w tablicy pochodzą z tej samej pozycji odpowiadających im dysków. Informacje o stanie bufora dla każdego z dysków przechowywane są w polu `flags`. W tablicy `devices` najpierw znajdują się elementy odpowiadające danym właściwym, a następnie danym nadmiarowym. Umożliwia to łatwe ustalenie czy dane są nadmiarowe oraz iteracje po danych odpowiednich typów. Rozdział danych na właściwe dyski następuje przez mapowanie, realizowane z pomocą pola `disk_idx`. Operacje na obiektach typu `stripe_head_t` oraz na elementach należących do tablicy `devices` zabezpieczane są przez spinlock `lock`. Stan obiektu `stripe_head_t` przechowywany jest w `state`, a licznik użycia w `count`. Pole `hash` pozwala umieścić obiekt w tablicy haszującej, a pole `list` umożliwia trzymanie obiektu na liście. Do inicjowania obiektów `stripe_head_t` służy funkcja

```
void stripe_head_init(stripe_head_t *sh, sector_t rdev_sector) .
```

Stan urządzenia wirtualnego realizującego funkcjonalność RAID przechowywany jest w strukturze `raidn_conf_t`.

```
typedef struct {
    mdk_rdev_t      *rdev;
} disk_info_t;

typedef struct raidn_conf_t;

struct raidn_conf {
    struct hlist_head    *stripe_hashtbl;
    mddev_t             *mddev;
    int                  chunk_size, algorithm;
    int                  data_disks, parity_disks, raid_disks;
    int                  working_disks;
    int                  failed_disks;
    int                  spare_disks;

    int                  max_nr_stripes;

    struct list_head     handle_list;
    struct list_head     delayed_list;
    struct list_head     bitmap_list;
    atomic_t             preread_active_stripes;

    char                  cache_name[20];
    kmem_cache_t         *slab_cache;

    int                  seq_flush, seq_write;
```

```

        int                quiesce;
        int                fullsync;

        struct page        **page_cache;
        int                page_cache_top;

        atomic_t           active_stripes;
        struct list_head   inactive_list;

        wait_queue_head_t  wait_for_stripe;
        wait_queue_head_t  wait_for_overlap;
        int                inactive_blocked;

        spinlock_t         device_lock;
        disk_info_t        disks[0];
};

```

Pole `mddev` umożliwia komunikację z podsystemem MD. Tablica haszująca przechowująca zainicjowane elementy `stripe_head_t` wskazywana jest przez `stripe_hashtbl`. W `chunk_size` przechowywana jest wielkość bloku paskowania, a `algorithm` określa sposób rozdziału danych na dyski w zależności od paska. Pola `data_disks`, `parity_disks` i `raid_disks` oznaczają odpowiednio liczbę dysków zawierających właściwe dane, liczbę dysków nadmiarowych oraz ich sumę. Podobnie `working_disks`, `failed_disks`, `spare_disks` oznaczają liczbę dysków działających jako w pełni funkcjonalne urządzenia RAID, liczbę dysków uszkodzonych oraz liczbę aktywnych dysków zapasowych. Aktywny dysk zapasowy to dysk, który zastępuje dysk uszkodzony, zapisy są już kierowane na ten dysk, może trwać odbudowa danych, ale dane na nim nie są jeszcze spójne z resztą macierzy. Tak więc jeśli nie ma dysków uszkodzonych, czyli `failed_disks = 0`, to i `spare_disks = 0`. Rola pozostałych składowych `raidn_conf_t` zostanie omówiona przy opisie działania głównego algorytmu sterownika.

4.4. Operacje kodowe

Użyty kod wymaga realizacji działań na elementach F_{2^s} . Dodawanie i odejmowanie odpowiada operacji XOR. Do realizacji mnożenia zastosowałem tablicę logarytmów i potęg, jedno mnożenie wymaga 3 dostępu do tablic i dodawania. Dzielenie jest wykonywane z pomocą tablicy odwrotności. Wypełnienie tablic następuje w funkcji `void raidn_gf_init(void)`,wołanej przy inicjowaniu sterownika. Funkcje odpowiedzialne za operacje na F_{2^s} zostały umieszczone w plikach `gf.h` i `gf.c`.

Podstawową operacją używaną do kodowania jest przemnożenie bloku danych przez element F_{2^s} . Korzystając z niej można wykonywać obliczenia na blokach danych tak jakby to były elementy F_{2^s} . Funkcją wykonującą tą operację jest

```
void raidn_block_mul(gfv_t x, void *b_out, const void *b, int b_size) .
```

Działanie funkcji można opisać równaniem $B_{out} = x * B + B_{out}$, gdzie B oraz B_{out} to bloki rozmiaru `b_size` wskazywane odpowiednio przez `b_out` oraz `b`. Wynik operacji umieszczony jest w bloku `b_out`, blok `b` nie ulega zmianie. Ponieważ rolę buforów pełnią strony, funkcja ma także swoją wersję działającą na stronach

```
void raidn_page_mul(gfv_t x, struct page *page_out, struct page *page).
```

Pozostałe funkcje wykorzystywane w realizacji operacji kodowych to

```
int raidn_page_is_clr(struct page *page)
void raidn_page_clr(struct page *page)
void raidn_page_cpy(struct page *page_out, struct page *page)
void raidn_page_xor(struct page *page_out, struct page *page).
```

Używane są one kolejno do testowania czy strona jest wyzerowana `raidn_page_is_clr`, zerowania strony `raidn_page_clr`, kopiowania strony `raidn_page_cpy`, operacji XOR na stronach `raidn_page_xor`. Ponadto do kopiowania danych między stroną a buforami listy operacji służą

```
void raidn_page_biolist_cpy(struct bio *bio, struct page *page, sector_t sector)
void raidn_biolist_page_cpy(struct page *page, sector_t sector, struct bio *bio).
```

Pierwsza kopiuje dane ze strony do buforów listy operacji, a druga z buforów listy operacji na stronę. Powyższe funkcje odpowiedzialne za operacje kodowe na blokach danych znajdują się w pliku `dataop.c`.

Do realizacji macierzy RAID wykorzystano następujące operacje kodowe:

- obliczanie danych nadmiarowych w schemacie odczyt-modyfikacja-zapis,
- obliczanie danych nadmiarowych w schemacie rekonstrukcja-zapis,
- sprawdzanie spójności danych nadmiarowych,
- odtwarzanie danych.

Funkcje realizujące je znajdują się w pliku `parity.c`. Niektóre z wymienionych operacji do wykonania potrzebują pomocniczej strony. Pomocnicza strona jest albo niezbędna do obliczeń, albo pozwala na ich przyspieszenie. W celu szybkiego pobierania stron stworzyłem strukturę przechowującą pulę stron prealokowanych. Wykorzystuje ona pola `page_cache`, oraz `page_cache_top` w `raidn_conf_t`. Pobieranie i zwalnianie prealokowanych stron odbywa się za pomocą funkcji

```
struct page *raidn_cache_get_page(raidn_conf_t *conf)
void raidn_cache_put_page(raidn_conf_t *conf, struct page *page) .
```

W przypadku gdy nie uda się przydzielić strony pomocniczej, do obliczeń używana jest jedna ze stron buforujących. Dane na takiej stronie oznaczane są jako nieaktualne, co może wiązać się z koniecznością wczytania ich z dysku, gdy będą potrzebne.

Funkcje wykonujące operacje kodowe korzystają z macierzy kontroli parzystości, której współczynniki są wyliczane ze wzoru $a_{ij} = \frac{X0+j+P}{i+j+P}$, gdzie $X0 = 0$, a P to liczba dysków parzystości. Funkcją obliczającą współczynniki macierzy jest

```
gfv_t mx_cauchy_xy(int x, int y) .
```

Do obliczania macierzy odwrotnej podmacierzy macierzy kontroli parzystości służy funkcja

```
void mx_cauchy_rev_init(mx_t *mx, int *x, int *y, int n)
```

wykorzystywana przy obliczaniu brakujących bloków danych.

4.4.1. Obliczanie danych nadmiarowych w schemacie odczyt-modyfikacja-zapis.

Przy zapisie możemy obliczyć nowe dane nadmiarowe na podstawie poprzedniej wartości zapisywanych danych oraz poprzedniej wartości danych nadmiarowych. Funkcją wykonującą tę operację jest

```
void raidn_parity_compute_rmw(stripe_head_t *sh) .
```

Powinna być ona wołana z buforami zawierającymi aktualne bloki danych dla dysków, na które chcemy dokonać zapisu, oraz dla dysków nadmiarowych. Dyski, na które chcemy dokonać zapisu, mają niepustą listę `towrite` w odpowiednim elemencie tablicy `devices`. W trakcie działania funkcji następuje przekopiowanie danych z buforów operacji znajdujących się na liście `towrite` do strony buforowej `page`, lista po przekopiowaniu przenoszona jest do pola `written`. Do kopiowania danych z listy buforów na stronę używana jest funkcja `raidn_biolist_page_cpy`. Funkcja `raidn_parity_compute_rmw` korzysta ze strony pomocniczej dla przyspieszenia obliczeń.

4.4.2. Obliczanie danych nadmiarowych w schemacie rekonstrukcja-zapis

Bloki danych nadmiarowych mogą być obliczone na podstawie wszystkich bloków danych właściwych. Do tego obliczenia wykorzystywana jest funkcja

```
void raidn_parity_compute_rcw(stripe_head_t *sh, int syncing).
```

Oczekuje ona, by aktualne były bufor dla dysków z danymi właściwymi, do których nie zapisujemy lub zlecone operacje nie nadpisują całego bufora. Funkcja ta ma dodatkowy parametr określający czy została ona wywołana do zapisu, czy do synchronizacji/rekonstrukcji. W przypadku gdy wołana jest do zapisu, dla każdego z elementów tablicy `devices` z niepustą listą `towrite` kopiuje dane z buforów listy operacji `towrite` do strony buforowej `page`, a lista przenoszona jest do pola `written`. Przy wołaniu do synchronizacji nie kopiuje danych z buforów listy operacji `towrite`. Funkcja nie oblicza danych nadmiarowych dla urządzeń, które uległy uszkodzeniu. Nie potrzebuje strony pomocniczej.

4.4.3. Sprawdzanie spójności danych nadmiarowych.

Do sprawdzenia spójności służy funkcja

```
int raidn_parity_check(stripe_head_t *sh).
```

Jej działanie polega na ponownym obliczeniu wartości danych nadmiarowych na podstawie danych właściwych. W przypadku stwierdzenia różnicy funkcja przekazuje zero, a bufor z blokami nadmiarowymi, dla których stwierdzono różnicę, oznaczane są jako wymagające ponownego obliczenia. Funkcja ta oczekuje aktualnych danych w buforach dla wszystkich dysków. Korzysta ze strony pomocniczej.

4.4.4. Odtwarzanie danych.

Odtwarzanie danych wykonuje funkcja

```
void raidn_data_compute(stripe_head_t *sh).
```

Oczekuje ona by liczba aktualnych buforów bloków danych nadmiarowych była nie mniejsza niż liczba nieaktualnych buforów bloków danych właściwych. Oblicza ona wszystkie nieaktualne bloki danych właściwych i zmienia stan ich buforów na aktualny. W trakcie obliczeń funkcja korzysta z funkcji obliczającej macierz odwrotną do podmacierzy Cauchy'ego, używa także strony pomocniczej.

4.5. Synchronizacja i rekonstrukcja

Zapis do macierzy RAID z dyskami nadmiarowymi wiąże się z chwilową utratą spójności danych. Wynika to z faktu, że każdy zapis wymaga co najmniej dwu zapisów na urządzenia składowe macierzy. Jeśli w trakcie trwania zapisu nastąpi wyłączenie macierzy, np. awaria zasilania, to macierz może nie być spójna. Aby rozpoznać taką sytuację, przed każdym zapisem do macierzy informacja o potencjalnym braku spójności logowana jest na dysk. Aby uniknąć zbytniego obciążenia logowaniem informacji o stanie, logowanie związane z odzyskaniem spójności jest opóźniane. Gdy przy uruchomieniu macierzy stwierdzony zostanie brak spójności następuje synchronizacja macierzy. W jej trakcie sprawdzane jest, czy dane nadmiarowe są zgodne z danymi właściwymi. Aby uniknąć konieczności sprawdzania całej macierzy, dodatkowo przed zapisami zapamiętywane jest miejsce, do którego zapis nastąpi. Tutaj także stosowane jest opóźnione logowanie informacji o odzyskaniu spójności. Dzięki pamiętaniu informacji o miejscach zapisu synchronizacja wykonywana jest tylko w ograniczonym obszarze, co pozwala znacznie przyspieszyć jej przebieg. Funkcjonalność związaną z logowaniem informacji o braku spójności udostępniana jest przez podsystem MD. Do logowania informacji o rozpoczęciu i zakończeniu zapisu służą funkcje

```
void md_write_start(mddev_t *mddev, struct bio *bi)
void md_write_end(mddev_t *mddev).
```

Dwie kolejne

```
int bitmap_startwrite(struct bitmap *bitmap,
                     sector_t offset, unsigned long sectors, int behind)
void bitmap_endwrite(struct bitmap *bitmap,
                    sector_t offset, unsigned long sectors, int success, int behind)
```

logują ponadto informacje o miejscach zapisu co pozwala na ograniczenie, synchronizacji tylko do miejsc, które jej wymagają. Funkcje logujące informacje o miejscach zapisu wymagają dodatkowej pamięci dyskowej do przechowywania bitmapy. Bitmapa może być przechowywana w oddzielnym pliku bądź w wydzielonym obszarze urządzeń składowych macierzy. Możliwe jest działanie macierzy bez korzystania z dodatkowej pamięci na bitmapę, jednak wiąże się to z długim czasem synchronizacji. Dla macierzy RAID1 opisane funkcje pozwalają na zwiększenie przepustowości przy korzystaniu z dysków zdalnych. Dzięki zalogowaniu informacji o rozpoczęciu zapisu oraz pamiętaniu liczby niezakończonych zapisów zdalnych, operacja zapisu do macierzy uznawana jest za zakończoną po wykonaniu zapisów na dyski lokalne. Zagadnienia projektowe związane z logowaniem informacji o zapisach do macierzy w systemie Linux opisano w [CB03].

W trakcie synchronizacji spójność testowana jest przy pomocy funkcji `raidn_parity_check`. Jeśli stwierdzony zostanie jej brak, to użyta zostanie funkcja `raidn_parity_rcw` w trybie synchronizacji – obliczy ona właściwe wartości bloków nadmiarowych. Bloki te zostają następnie zapisane na dyski. Synchronizacja macierzy pozbawionej części dysków może grozić utratą danych.

Operacją podobną do synchronizacji jest rekonstrukcja. Polega ona na obliczaniu danych dla tych urządzeń, które nie zawierają aktualnych danych. Mogą to być urządzenia zapasowe, które zastąpiły urządzenia uszkodzone lub też urządzenie na jakiś czas odłączone od systemu. W drugim przypadku dzięki logowaniu informacji o miejscach zapisu rekonstrukcja wykonuje się tylko w miejscach, do których były zapisy w czasie, gdy urządzenie było odłączone. Rekonstrukcja korzysta zarówno z funkcji `raidn_parity_rcw` w trybie synchronizacji, jak i z `raidn_data_compute`, w zależności od tego czy obliczane bloki są blokami danych czy parzystości, a to zmienia się dla ustalonych urządzeń ze względu na przeplot danych nadmiarowych.

4.6. Główny algorytm

Stan zainicjowanych urządzeń `raidn` przechowywany jest w obiektach typu `raidn_conf_t`. Dla każdego urządzenia przy jego inicjacji tworzony jest wątek jądra `mdN_raidn`, gdzie `N` – numer urządzenia. Wątek monitoruje stan urządzenia korzystając z

```
void md_check_recovery(mddev_t *mddev)
```

oraz przetwarza listę `handle_list` w `raidn_conf_t` przy pomocy funkcji

```
void handle_stripe(stripe_head_t *sh).
```

Obiekty typu `stripe_head_t` umieszczane są na liście `handle_list` przy zleceniu operacji na macierzy, oraz przez funkcje obsługi zakończenia operacji na urządzeniu składowym.

Przy zapisach, dla których wymagane są odczyty, nie dochodzi do natychmiastowego zlecenia odczytów. Obiekty typu `stripe_head_t` wymagające odczytów trafiają do kolejki oczekujących `delayed_list` w `raidn_conf_t`. Obiekty z tej listy są aktywowane gdy liczba aktywnych operacji zapisu, wymagających odczytów spadnie poniżej wartości `IO_THRESHOLD`. Zwiększa to prawdopodobieństwo, że kilka zapisów wykonanych zostanie za jednym razem. Aktywowanie kolejki `delayed_list` może także zostać spowodowane przez żądanie zrzucenia buforowanych danych dla urządzenia.

Obiekty reprezentujące operacje zapisu, dla których należy zalogować informacje o braku spójności oczekują na zakończenie logowania na liście `bitmap_list`.

4.6.1. Funkcja `raidn_make_request`

Za pomocą funkcji

```
int raidn_make_request(request_queue_t *queue, struct bio *bio)
```

zlecana jest operacja odczytu lub zapisu określonych sektorów macierzy RAID. Funkcja wołana jest pośrednio przez procesy korzystające z macierzy. Dla obiektu `queue` określany jest odpowiedni obiekt `conf` typu `raidn_conf_t`. Przy użyciu funkcji

```
sector_t raid2ddev_sector(raidn_conf_t *conf, sector_t raid_sector, int *ddev_idx)
```

obliczane są numer urządzenia składowego macierzy oraz sektor urządzenia, do których odnosi się operacja. Na podstawie obliczonego sektora znajdujący jest obiekt typu `stripe_head_t`. Służy do tego funkcja

```
stripe_head_t *get_active_stripe(raidn_conf_t *conf, sector_t sector, int noblock).
```

Najpierw obiekt jest poszukiwany w tablicy haszującej, jeśli go tam nie ma to następuje próba pobrania pierwszego elementu z listy wolnych `inactive_list` obiektu `conf`. Jeśli lista jest pusta to proces wołający funkcję `raidn_make_request` zasypia w oczekiwaniu na jej zapełnienie. Z `get_active_stripe` można korzystać także bez blokowania, zależy to od znacznika `noblock`, jego ustawienie powoduje że zamiast usypiania procesu zwracany jest `NULL`. W przypadku pobrania obiektu z listy wolnych jest on inicjowany przy pomocy funkcji `stripe_head_init`. Następnie zlecona operacja dodawana jest do listy `toread` lub `towrite` odpowiedniego elementu tablicy `devices`, służy do tego funkcja

```
int add_stripe_bio(stripe_head_t *sh, struct bio *bio, int ddev_idx, int forwrite).
```

Ostatnim krokiem działania `raidn_make_request` jest wywołanie funkcji `handle_stripe`.

4.6.2. Funkcja `raidn_sync_request`

Do synchronizacji służy funkcja

```
sector_t raidn_sync_request(mddev_t *mddev, sector_t sector_nr,
                           int *skipped, int go_faster).
```

Jest ona wołana przez funkcję `md_check_recovery`, którą uruchamia wątek `mdN_raidn`. Rozpoczyna ona działanie od sprawdzenia czy synchronizacja dla żadanego zakresu jest konieczna. Jeśli tak, to podobnie jak `raidn_make_request` znajduje odpowiedni obiekt `stripe_head_t`, ustawia go w stan synchronizacji i woła funkcję `handle_stripe`.

4.6.3. Funkcja `handle_stripe`

Funkcja

```
void handle_stripe(stripe_head_t *sh)
```

obsługuje poszczególne etapy operacji na macierzy. Może zostać wywołana przy zleceniu operacji lub przez wątek `mdN_raidn`.

W pierwszym kroku sprawdza stan obiektu typu `stripe_head_t` i na tej podstawie podejmuje odpowiednie działania. Od razu przekazywane są odczyty, dla których istnieją aktualne dane oraz wykonane zapisy, dla których wszystkie bloki parzystości zostały zapisane.

Następnie urządzeniom składowym macierzy zlecane są operacje odczytu potrzebne do obsługi żądania. Wczytanie bloku jest konieczne w przypadku odczytów, synchronizacji lub zapisów, które nie nadpisują całego bufora lub które potrzebują dodatkowych danych do obliczenia bloków nadmiarowych. Przed zleceniem odczytów sprawdza się, czy można obliczyć wymagane dane, jeśli tak odczyt zastępowany jest obliczeniem. Jeśli konieczne jest wczytanie bloku, dla którego urządzenie zostało uszkodzone, to zlecane są odczyty potrzebne do obliczenia brakujących danych. Gdy jest coś do zapisania, wybierana jest ta metoda obliczenia danych nadmiarowych, która wymaga mniej operacji odczytów z urządzenia.

Jeśli wszystkie dane potrzebne do zapisu zostały zgromadzone, obliczane są dane nadmiarowe i zlecane zapisy. Przy synchronizacji sprawdzane są dane nadmiarowe, i jeśli trzeba nadpisywane. W przypadku gdy część urządzeń uszkodzonych zastąpiono zapasowymi procedura synchronizacji odpowiada za rekonstrukcję. Dla rekonstruowanych urządzeń obliczane są uszkodzone dane i zlecane zapisy.

Po zakończeniu `handle_stripe` przetwarzany obiekt typu `stripe_head_t` może trafić na jedną z list `handle_list`, `delayed_list`, `inactive_list` bądź oczekiwać na zakończenie operacji na urządzeniach składowych macierzy. Jeśli zlecono operacje urządzeniom składowym

obiekt ponownie powraca na listę `handle_list` po ich zakończeniu. Odpowiadają za to funkcje obsługi zakończenia operacji na urządzeniach składowych. Po powrocie obiektu na listę `handle_list` wątek `mdN RAID` ponownie wywołuje funkcję `handle_stripe` na obiekcie realizując kolejny etap przetwarzania operacji macierzowej.

4.7. Narzędzia administracyjne

Wykorzystanie funkcjonalności realizowanej przez sterownik `raidn` wymagało przeróbki narzędzia administracyjnego. Tym narzędziem był `mdadm` w wersji 2.4.

Pakiet `mdadm` pozwala na administrowanie z poziomu użytkownika sterownikami podsystemu MD. W skład pakietu `mdadm` wchodzi komenda `mdadm` wraz z dokumentacją. Dodanie obsługi nowego sterownika wiązało się z rozszerzeniem obsługiwanych przez `mdadm` stopni RAID, jak również z koniecznością przekazania nowego parametru, jakim jest liczba dysków nadmiarowych. Pełne rozszerzenie funkcjonalności `mdadm` wymagało też przeróbek w dokumentacji pakietu. Rozszerzona komenda `mdadm` rozpoznaje nowy stopień RAID, jako parametr `raidn` lub `N` dla opcji `--level`. Liczbę dysków parzystości można przekazać jako parametr dla opcji `--parity-devices`. Pozostałe parametry dla `raidn` są takie same jak dla macierzy stopni 4, 5 i 6.

Rozdział 5

Testy

5.1. Wprowadzenie

Celem testów było sprawdzenie poprawności działania oraz określenie wydajności sterownika.

5.2. Metody testów

Testy poprawności działania polegały na sprawdzeniu macierzy w różnych sytuacjach użytkowych. Szczególną uwagę poświęcono działaniu macierzy w przypadku uszkodzenia części urządzeń. Każdy z testów obejmował sekwencję operacji, pozwalających określić działanie macierzy w określonym stanie.

Testy wydajnościowe oparte były na pomiarze czasu wykonania operacji zapisu i odczytu, dla wybranych konfiguracji macierzy.

5.3. Urządzenia pomocnicze

W części testów rolę dysków pełniły urządzenia blokowe udostępniane przez sterownik **ramdisk**. Umożliwia on dostęp do bloków pamięci przez interfejs urządzenia blokowego. Rozmiary takich urządzeń ograniczone są przez wielkość pamięci RAM, charakteryzują się jednak dużą szybkością, co ułatwia ocenę wydajności urządzeń zbudowanych na ich bazie. Zostały one użyte w testach wydajnościowych.

Do testów funkcjonalnych wymagających większej liczby dysków wykorzystane zostały urządzenia realizowane przez sterownik **loop**. Sterownik ten pozwala budować urządzenia blokowe na bazie plików.

Część testów funkcjonalnych wymagała metod pozwalających w kontrolowany sposób symulować uszkodzenia urządzeń. Do tego celu zostało użyte urządzenie blokowe **faulty** z podsystemu MD. Z jego pomocą można symulować różne typy uszkodzeń, a działa ono jako urządzenie pośredniczące w dostęпах do właściwego urządzenia blokowego.

5.4. Funkcjonalność

Podczas testowania poprawności działania wykonano sekwencje operacji, z których każda rozpoczyna się od utworzenia macierzy, co wiąże się z automatycznym rozpoczęciem synchronizacji. Aby macierz mogła być w pełni użyteczna, należy odczekać do zakończenia synchro-

nizacji, dlatego operacja **utworzenie macierzy** zawiera w sobie takie oczekiwanie. Wszędzie gdzie występują odczyty bądź zapisy do niepełnej macierzy, liczba urządzeń, których brakuje w macierzy, nie przekracza maksymalnej dopuszczalnej liczby uszkodzonych urządzeń. Testy funkcjonalne składały się z następujących schematów operacji:

- Zapis danych
 - utworzenie macierzy,
 - zapis danych,
 - restart macierzy,
 - odczyt danych i porównanie z danymi zapisanymi.

W tym schemacie sprawdzana jest poprawność zapisu danych do macierzy. Restart macierzy ma służyć wymuszeniu odczytu danych z dysków, bez restartu istnieje możliwość, że odczytane zostaną dane z buforów. Schemat ten był wykonany także w wersji bez restartu, odnosi się to do wszystkich kolejnych przypadków gdzie występuje ta operacja.

- Odczyt z niepełnej macierzy
 - utworzenie macierzy,
 - zapis danych,
 - usunięcie części dysków,
 - restart macierzy,
 - odczyt danych i porównanie z danymi zapisanymi.

Schemat ten sprawdza odtwarzanie danych w trakcie odczytu, które jest wymuszone przez brak części dysków.

- Zapis i odczyt z niepełnej macierzy
 - utworzenie macierzy,
 - usunięcie części dysków,
 - zapis danych,
 - restart macierzy,
 - odczyt danych i porównanie z danymi zapisanymi.

- Rekonstrukcja danych
 - utworzenie macierzy,
 - usunięcie części dysków,
 - zapis danych,
 - restart macierzy,
 - dodanie brakujących dysków,
 - oczekiwanie na zakończenie synchronizacji,
 - usunięcie części dysków obecnych w trakcie zapisu,
 - odczyt danych i porównanie z danymi zapisanymi.

W tym schemacie sprawdzane jest czy na dyskach odtworzonych w trakcie rekonstrukcji znajdują się właściwe dane. Usunięcie części dysków obecnych w czasie zapisu gwarantuje, że będą wykorzystane dane z dysków odtworzonych.

- Naprawa błędu odczytu
 - utworzenie macierzy,
 - zapis danych,
 - uruchomienie odpowiedniej symulacji uszkodzenia,
 - odczyt danych i porównanie z danymi zapisanymi.

Testowana jest tu korekcja błędu odczytu. Korekcja błędu odczytu polega na obliczeniu uszkodzonych danych, zapisie w uszkodzone miejsce i ponownym kontrolnym odczycie. Postępuje się w ten sposób, ponieważ część błędów odczytu nie musi się wiązać z trwałym uszkodzeniem dysku i taka naprawa czasem jest skuteczna. W przypadku gdy przekroczony zostaje limit prób naprawy, urządzenie uznawane jest za uszkodzone i odłączane. Obie te sytuacje zostały zasymulowane za pomocą urządzenia **faulty**, które pozwala zarówno na symulację trwałego uszkodzenia, jak i jednorazowego błędu odczytu.

- Uszkodzenia
 - utworzenie macierzy,
 - zapisy i odczyty,
 - uruchomienie jednej z symulacji uszkodzeń,
 - zapisy i odczyty,
 - sprawdzenie skutków uszkodzenia.

Wszystkie operacje usuwania części dysków wykonywane były przez administracyjną zmianę stanu dysków na uszkodzone. Do wnioskowania o aktualnym stanie sterownika stosowany był odczyt stanu z użyciem narzędzia **mdadm**, za pomocą którego wykonywane były wszystkie inne operacje administracyjne. Dodatkowe informacje można było uzyskać w pliku **/proc/mdstat** oraz analizując logi systemu.

Testy funkcjonalne przeprowadzano kolejno rozpoczynając od **zapisu danych**, a kończąc na **naprawie błędu odczytu**. W przypadku stwierdzenia błędu, po usunięciu jego przyczyny, testy wykonywane były od początku. Dzięki testom udało się usunąć kilka błędów popełnionych przy implementacji.

Skrypty pozwalające wykonać opisane testy znajdują się na dołączonej płycie CD.

5.5. Wydajność

W testach wydajnościowych badano wpływ liczby urządzeń nadmiarowych na szybkość zapisu, wpływ liczby urządzeń uszkodzonych na odczyt, a także porównano niektóre parametry sterownika **raidn** z dostępnymi w źródłach jądra sterownikami **raid0**, **raid5** i **raid6**. Sposób przeprowadzenia testów umożliwił ustalenie wpływu obliczeń nadmiarowych na wydajność sterownika. Ponieważ testy przeprowadzone były na urządzeniach odbiegających parametrami od rzeczywistych dysków, na ich podstawie można tylko częściowo wnioskować o zachowaniu sterownika w normalnych zastosowaniach. Podstawową różnicą między dyskiem a użytym do testów urządzeniem udostępnianym przez sterownik **ramdisk** jest czas dostępu, który dla dysku jest rzędu milisekund, natomiast tutaj jest zanedbywalnie mały. Z tego względu testy te lepiej odpowiadają sytuacjom, gdy korzystamy z macierzy operując na dużych blokach danych.

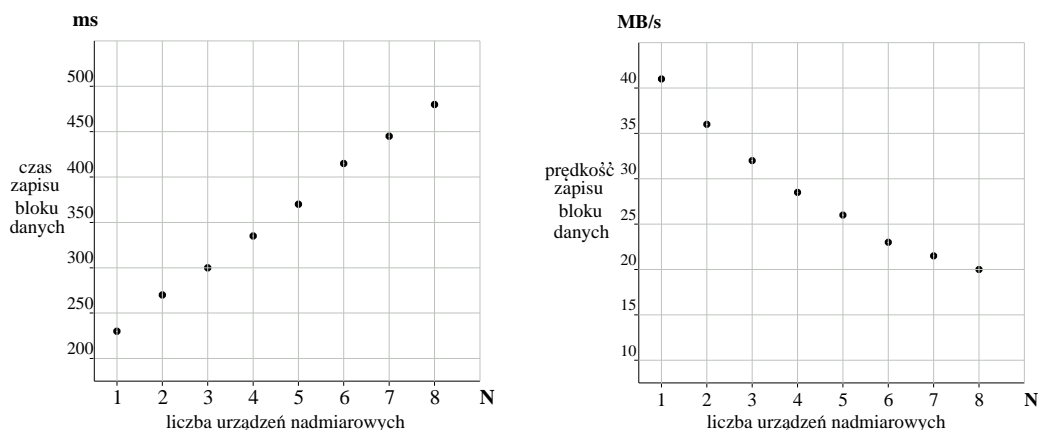
Testy zostały przeprowadzone na maszynie z procesorem PIII 928 MHz.

5.5.1. Liczba dysków nadmiarowych a czas odczytu

W testach nie udało się stwierdzić zauważalnego wpływu liczby dysków nadmiarowych na czas odczytu. Wynika to z faktu użycia do testów urządzeń, dla których odczyt danych z urządzenia może być szybszy niż ich obliczenie oraz sposobu dostępu polegającego na czytaniu dużych bloków danych.

5.5.2. Liczba dysków nadmiarowych a czas zapisu

Związek między liczbą dysków nadmiarowych przy ustalonej liczbie dysków przeznaczonych na dane a czasem zapisu przedstawiony jest na rysunku 5.1.



Rysunek 5.1: Czas zapisu bloku danych rozmiaru 9.6 MB oraz odpowiadająca mu prędkość zapisu w zależności od liczby urządzeń nadmiarowych N . Liczba urządzeń podstawowych jest stała i wynosi 8. Całkowita liczba urządzeń to $8 + N$.

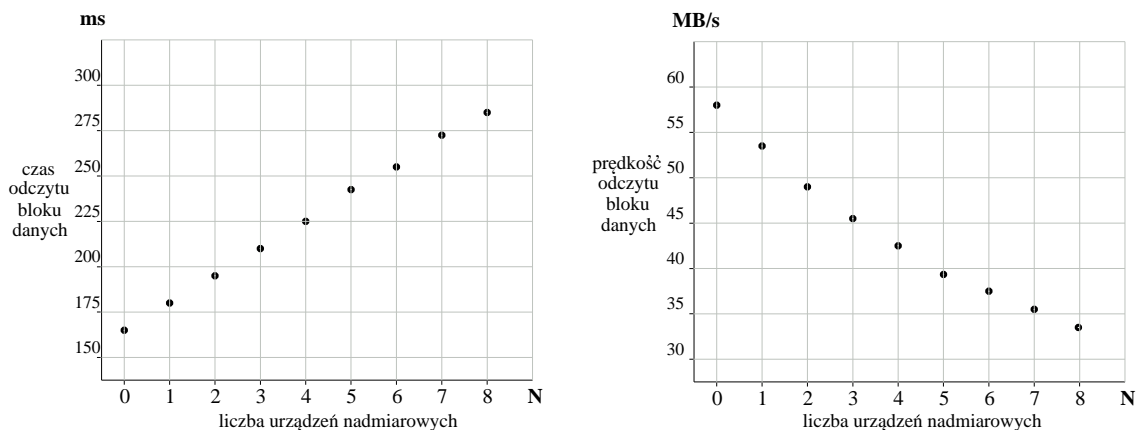
Na podstawie wyników można stwierdzić zbliżony do liniowego charakter zależności między czasem zapisu a liczbą dysków nadmiarowych. Przyczyną wzrostu czasu przy wzroście liczby dysków nadmiarowych jest konieczność dodatkowych obliczeń nadmiarowych. Można zauważyć, że koszty związane z obliczaniem informacji nadmiarowych zaczynają przeważać nad pozostałymi kosztami dopiero przy ok. 8 dyskach nadmiarowych. Prawdopodobnie dla szybszych procesorów ta zależność będzie jeszcze lepsza. Zakładając liniową zależność między liczbą dysków nadmiarowych a czasem zapisu bloku, możemy czas zapisu bloku wyrazić wzorem

$$T_{\text{zapisu}} = T_0 + a * N,$$

gdzie N – liczba dysków nadmiarowych. Przy takim założeniu wyznaczone metodą najmniejszych kwadratów współczynniki równania wynoszą odpowiednio $T_0 = 196,1$ ms i $a = 35,5$ ms. Wartości zaznaczone na wykresie stanowią średnie z 4 pomiarów, średnie odchylenie standardowe to ok. 8 ms.

5.5.3. Liczba dysków uszkodzonych a czas odczytu

Odczyt danych z macierzy w przypadku braku części dysków powoduje konieczność obliczania danych z brakujących dysków. Zależność między liczbą uszkodzonych dysków a czasem odczytu przedstawiona jest na rysunku 5.2.



Rysunek 5.2: Czas odczytu bloku danych rozmiaru 9.6 MB oraz odpowiadająca mu prędkość odczytu w zależności od liczby uszkodzonych urządzeń N . Liczba wszystkich urządzeń wynosi 16, z czego 8 to urządzenia nadmiarowe.

Tutaj także zauważyć można, że czas odczytu rośnie proporcjonalnie do liczby dysków uszkodzonych. Narzut związany z obliczaniem brakujących danych nie przekracza pozostałych kosztów odczytu nawet przy 8 uszkodzonych dyskach. Wpływ na to ma rozdział danych nadmiarowych między różne dyski w zależności od paska. Dzięki temu nawet przy 8 uszkodzonych dyskach mając 8 dysków przeznaczonych na dane dla niektórych pasków odczyt może być wykonany bez wykonywania obliczeń. Przy założeniu liniowej zależności obliczone współczynniki zależności to $T_0 = 164,3$ ms oraz $a = 15,3$ ms. Każda z wartości jest średnią z 4 pomiarów, średnie odchylenie standardowe to ok. 5 ms.

5.5.4. Liczba dysków uszkodzonych a czas zapisu

Przy wzroście liczby dysków uszkodzonych czas zapisu maleje. Wynika to stąd, że sterownik nie oblicza danych nadmiarowych dla takich dysków. Odpowiada to sytuacji obniżenia liczby dysków nadmiarowych, choć przez stosowanie rozdziału danych nadmiarowych zależność ta jest trochę złagodzona.

5.5.5. Porównanie z innymi sterownikami RAID

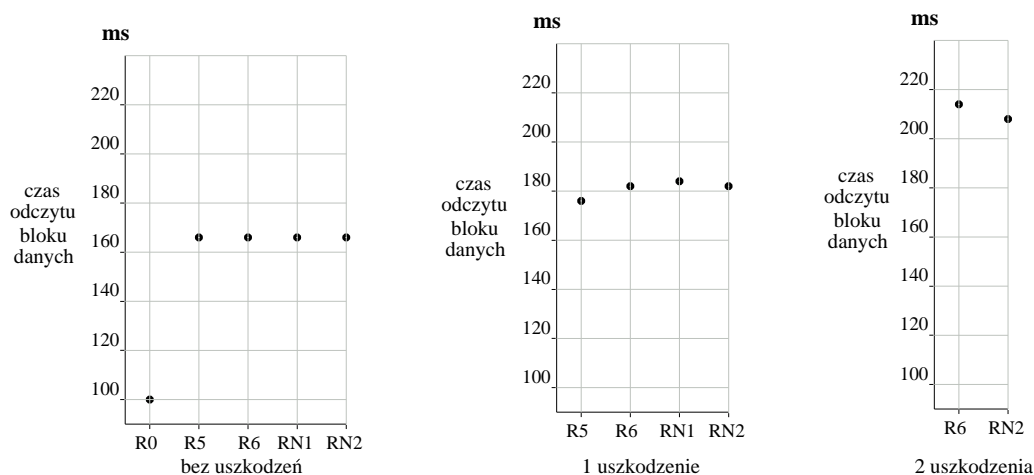
W celu określenia wydajności sterownika względem innych implementacji zostały przeprowadzone testy porównawcze. Aby konfiguracja macierzy realizowanej przez sterownik `raidn` była porównywalna z macierzami udostępnianymi przez sterowniki `raid5` i `raid6`, do testów użyte zostały dwie macierze `raidn` z jednym i dwoma dyskami nadmiarowymi.

Pierwszym parametrem, który był porównywany dla wszystkich macierzy był czas odczytu przy braku urządzeń uszkodzonych. W teście tym uczestniczyła także macierz `raid0`. Wyniki testu przedstawia rysunek 5.3. Między macierzami `raid5`, `raid6` a `raidn` nie ma praktycznie różnicy. Od reszty wyraźnie lepsza jest macierz `raid0`, jest to spowodowane najprawdopodobniej dodatkowym kopiowaniem danych w przypadku macierzy realizujących nadmiarowość. Dane przy odczycie przechodzą przez bufor używane do obliczeń nadmiarowych. W niektórych przypadkach może to być opłacalne, jednak przy bezpośrednim odczycie powoduje pogorszenie szybkości.

Kolejnym testem był odczyt danych dla jednego dysku uszkodzonego. Uczestniczyły w nim

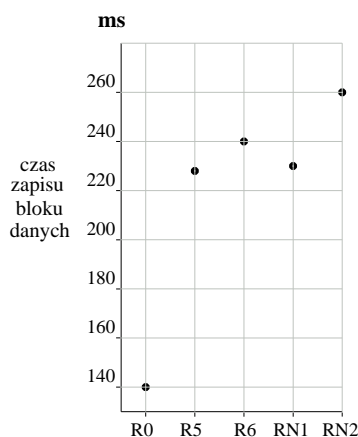
tylko te macierze, które mogą działać mimo takiego uszkodzenia. Wyniki wszystkich macierzy są porównywalne i nie przekraczają wartości odchylenia standardowego.

W następnym teście przedstawione są czasy odczytu danych dla dwu dysków uszkodzonych. Tutaj trochę lepsza jest macierz `raidn`. Wyniki są średnimi z 8 pomiarów, odchylenie standardowe dla poszczególnych macierzy to około 8 ms.



Rysunek 5.3: Czas odczytu bloku 9.6 MB dla różnych macierzy RAID, w zależności o liczby dysków uszkodzonych. RN1, RN2 – macierze RAIDn z jednym i dwoma dyskami nadmiarowymi

Ostatni test porównuje czasy zapisu bloku danych dla różnych macierzy RAID. Na rysunku 5.4 przedstawione są jego wyniki.



Rysunek 5.4: Czas zapisu bloku 9.6 MB dla różnych macierzy RAID. RN1, RN2 – macierze RAIDn z jednym i dwoma dyskami nadmiarowymi

Wyniki są średnimi z 8 pomiarów, odchylenie standardowe to około 10 ms. Najlepszy wynik macierzy `raid0` wynika zarówno z braku dodatkowego kopiowania danych jak również braku obliczeń nadmiarowych dla tej macierzy. Najgorzej wypada macierz `raidn` z dwoma dyskami nadmiarowymi. Jest gorsza od analogicznej macierzy z dwoma dyskami nadmiarowymi `raid6`. Wynika to prawdopodobnie z dość ogólnego charakteru sterownika `raidn`,

jak też optymalizacji obliczania informacji nadmiarowych w macierzy `raid6` dla najczęściej spotykanych procesorów.

Rozdział 6

Podsumowanie

Praca miała na celu implementację sterownika obsługującego macierz RAID o konfigurowalnym stopniu nadmiarowości. Został on wykonany i przetestowany w różnych konfiguracjach sprzętowych spełniając większość założonych wymagań. Implementację ułatwił przejrzysty interfejs podsystemu MD oraz dostępność kodu sterowników macierzy RAID innych stopni w systemie Linux. Trudność stanowił wybór metody kodowania danych nadmiarowych. Większość publikacji opisuje metody zoptymalizowane pod względem obliczania danych nadmiarowych, jednak gorzej zachowujące się przy odtwarzaniu. Z uwagi na chęć wykorzystania operacji odtwarzania w normalnej pracy macierzy ostatecznie wybrałem metodę o przeciętnym czasie kodowania, jednak dość wydajną przy odtwarzaniu.

W trakcie testów okazało się, że macierz realizowana przez sterownik jest porównywalna jeśli chodzi o wydajność, z macierzami dotychczas dostępnymi w systemie. Jej stosowanie ma więc sens głównie wtedy gdy chcemy wykorzystać jej podstawową funkcjonalność, czyli macierz z więcej niż dwoma dyskami nadmiarowymi. Zastosowania takie będą ograniczone do sytuacji gdzie potrzeba szczególnie wysokiego poziomu bezpieczeństwa danych. Osiągnięta tym sposobem niezawodność nie przekroczy jednak poziomu wyznaczonego przez pozostałe elementy systemu. Dlatego stosowanie większej liczby dysków nadmiarowych wydaje się racjonalne w przypadku korzystania z dysków o dużym prawdopodobieństwie awarii. Zastosowaniem tego rodzaju może być budowa macierzy z użyciem dysków, do których dostęp odbywa się po zawodnych kanałach komunikacyjnych np. dysków dostępnych zdalnie.

6.1. Możliwe rozszerzenia

Część z proponowanych rozszerzeń związana jest z wybranym do budowy sterownika środowiskiem podsystemu MD w systemie Linux.

- Wybór sposobu kodowania danych nadmiarowych. Rozszerzenie to pozwalałoby na użycie innych algorytmów kodowania.
- Inne algorytmy kodowania danych nadmiarowych.
- Możliwość wyłączenia kopiowania przy odczycie. W trakcie odczytów dane przechodzą przez system buforujący sterownika co wymaga dodatkowego kopiowania. W pewnych zastosowaniach, głównie tam gdzie odczyty znacznie przeważają nad zapisami, opłacalne może być pominięcie tego etapu. Rozszerzenie to powinno być parametrem sterownika.
- Możliwość zmiany parametrów działającej macierzy np. zmiana liczby dysków danych, liczby dysków nadmiarowych, sposobu kodowania itp. .

Dodatek A

Zawartość płyty dołączonej do pracy

Na płycie znajdują się:

- wersja elektroniczna tekstu pracy – katalog `tekst`,
- kod źródłowy sterownika macierzy – katalog `raidn`,
- kod źródłowy narzędzia administracyjnego `mdadm` w wersji 2.4 – katalog `mdadm-2.4`,
- kod źródłowy systemu Linux w wersji 2.6.16 – katalog `linux-2.6.16`,
- skrypty do testów poprawności działania – katalog `testy/poprawnosc`,
- skrypty do testów wydajnościowych – katalog `testy/wydajnosc`,

Bibliografia

- [Anvin] H. Peter Anvin, *The mathematics of RAID-6*,
<http://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf>.
- [BBBM95] Mario Blaum, Jim Brady, Jehoshua Bruck, Jai Menon, *EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures*, IEEE Transactions on Computers, 44(2), s. 192-202, February 1995.
- [BBDKMV99] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, D. Verworner, *Linux kernel, jadro systemu*, Wydawnictwo MIKOM, Warszawa 1999.
- [BKKKLZ95] J. Bloemer, M. Kalfane, M. Karpinski, R. Karp, M. Luby and D. Zuckerman, *An XOR-Based Erasure-Resilient Coding Scheme*, ICSI Technical Report, TR-95-048, August 1995.
- [BR99] M. Blaum and R.M. Roth, *On Lowest-Density MDS Codes*, IEEE Transactions, Information Theory, 45(1), s. 46-59, January 1999.
- [CB03] Paul Clemens, James E.J. Bottomley, *High Availability Data Replication*, Linux Symposium 2003, Ottawa, s. 119-126, <http://www.linuxsymposium.org>.
- [Chen93] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, *RAID: high-performance, reliable secondary storage*, ACM Computing Surveys, 26(2) s. 145-185, June 1994.
- [MS77] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-holland, Amsterdam, 1977.
- [Patterson88] David A. Patterson, Garth Gibson, Randy H. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, International Conference on Management of Data (SIGMOD) s. 109-116, 1988.
- [Rub99] Alessandro Rubini, *Linux. Sterowniki urzadzeń*, Wydawnictwo RM, Warszawa 1999.
- [Sin64] Richard C. Singleton, *Maximum distance q-nary codes*, IEEE Transactions on Information Theory, 10 s. 116-118, April 1964.
- [Vah01] Uresh Vahalia, *Jadro systemu UNIX nowe horyzonty*, Wydawnictwa Naukowo-Techniczne, Warszawa 2001.