Optymalizacja Minix File System

Filip Plata
Uniwersytet Warszawski
Pasteura 5
Warszawa, Polska
fp371335@students.mimuw.edu.pl

ABSTRACT

Sprawozdanie z próby usprawnienia Minix File System poprzez niezapisywanie na dysk bloków samych zer.

1. OPIS MODYFIKACJI

Minix File System (dalej MFS) w wersji 3.3.0 nie przeprowadza żadnej optymalizacji przechowywanych danych. Wprowadzona optymalizacja jest nieduża i opiera się na niezapisywaniu na dysk każdego bloku danych, który zawiera same zera. Ponieważ MFS domyślnie czyta same zera z nieprzydzielonych bloków, wystarczy, że w przypadku wykrycia zapisu bloku zer, nie zrobimy nic.

1.1 Oczekiwany wpływ na wydajność

Wstępne oczekiwania co do takiej optymalizacji były następujące:

- Wzrośnie czas zapisu losowych i naturalnie powstających w wyniku używania plików ze względu na dodatkowe kopiowanie pamięci.
- Znacznie zmniejszy się czas zapisu plików zawierających dużę ilości bloków samych zer.
- Kopiowanie pliku samych zer powinno być niewiele wolniejsze, gdyż nie powinno wykonywać operacji na dysku.
- Łatka będzie kompatybilna z poprzednią wersją systemu plików.

1.2 Uzyskane rezultaty

Czasy uzyskane dla testu wydajnościowego "test2.sh", zakładającego rozmiar bloku na dysku 4096B. Zapisywał on pliki coraz większych rozmiarów składające się z samych zer oraz końcowo kontrolnie losowy plik (korzystając z /dev/urandom) - przy tym ostatnim czasy powinny być porównywalne. Tak uzyskano na niezmodyfikowanym systemie dla zapisu(podane czasy real z komendy time):

DOI: 10.475/123_4

Rozmiar	Odczyt i Zapis	Kopiowanie
196MB	1.6s	10.2s
392MB	14.0s	33.6s
587MB	29.1s	53.0s
783MB	35.2s	75.3s
979MB	50.8s	107.8s
196MB	15.6s	16.4s

Ostatni wynik dotyczy pliku losowego. Czasy uzyskane po optymalizaji:

Rozmiar	Odczyt i Zapis	Kopiowanie
196MB	0.9s	0.5s
392MB	1.8s	0.8s
587MB	2.7s	1.3s
783MB	3.6s	1.7s
979MB	4.4s	2.1s
196MB	13.4s	12.1s

Dodatkowo "test3.sh"mierzył też prędkość zapisu i kopiowania plików składających się z bloków na przemian zer i losowych. Czasu są tu istotnie większe ze względu na konstrukcję testu i konieczność wielokrotnego uruchamiania programu dd. Uzyskane czasy poniżej dla zmodyfikowanego systemu:

Rozmiar	Odczyt i Zapis	Kopiowanie
196MB	58s	55s
392MB	152s	45s
587MB	276s	50s
783MB	452s	62s
979MB	716s	104s

Jak widać, nie widać znaczącej poprawy w stosunku do oryginalnego MFS i zapisu samych zer. Najprawdopodobniej takie wyniki to efekt nadmiernego obciążenia systemu operacyjnego hosta (Debian) podczas testu, zwłaszcza w pierwszym i drugim co do wielkości pliku - oczekiwny rezultat w takim przypadku dla kopiowania to oczywiście średnia czasów z zapisu samych zer z i bez modyfikacji.

Oprócz tego, kontrolnie poniżej czasy kopiowania danych z dysku (urządzenie c0d0, "test4.sh") przed modyfikacją:

Rozmiar	Odczyt i Zapis	Kopiowanie
196MB	16.9s	9.8s
392MB	45.7s	30.9s
587MB	80s	54s

oraz po:

Rozmiar	Odczyt i Zapis	Kopiowanie
196MB	10.6s	2.0s
392MB	26.0s	7.5s
587MB	55.5s	28.0s

Znowu widać przyspieszenie, co jednak po przejrzeniu zawartości dysku nie zaskakuje: zawiera on wiele bloków zer.

1.3 Uzyskane rezultaty w świetle założeń

Nie udało się zauważyć zauważalnego wpływu na zapis plików o losowych danych. Pewien narzut musi oczywiście wystąpić, lecz jest on istotnie mniejszy niż wariancja rozkładu czasów zapisu na dysk. Sprawdziły się za to pozostałe przewidywania: czasu zapisu plików zawierających same zera jest mały i rośnie liniowo z rozmiarem pliku, a kopiowanie jest jeszcze szybsze - gdyż pomijamy odczyt z urządzenia /dev/zero.

Trudniej jest skomentować wyniki tworzenia plików z danymi naprzemiennymi. Czasy ich stworzenia to niemal wyłącznie narzut shella, natomiast przy kopiowaniu widać duże czasy dla relatywnie niedużych plików (około 200MB i 400MB).

2. ZASTOSOWANIE OPTYMALIZACJI

Oczywiste jest, iż używanie takiego systemu plików ma sens tylko, gdy spodziewamy się plików z dużą ilością kolejnych zer. Pytanie jakie należy sobie zadać to więc: od jakiej procentowej ilości bloków samych zer w plikach narzut zostanie zniwelowany. W tym celu należy go najpierw oszacować. Traktując narzut oraz czas zapisu jako zmienne losowe oraz zakładając, że czas zapisu jest sumą wielu zmiennych losowych o rozkładzie Gaussa, a narzut ma rozkład punktowy, wystarczy osczacować wartości oczekiwane poprzez średnią i odjąć od siebie. W ten sposób dla oryginalnego systemu plików otrzymujemy średni czas 10.9s, natomiast dla po modyfikacji 11.35s dla plików wielkości 196MB(przprowadzamy po kilka pomiarów dla oryginalnego i zmodyfikowanego MFS). Zatem narzut szacujemy na 2.30 ms/MB danych - dla wszystkich bloków. Z danych z tabeli dla zapisu otrzymujemy dla ostatnich wierszy zysk około 47.4 ms/MB zer. Stąd wnioskujemy, że punkt opłacalności można oszacować przez około 5% (iloraz liczb powyżej). Tego typu szacunki obarczone są oczywiście dużym błędem ze względu na trudność zmierzenia narzutu, lecz co ważne: nie zależy od bezwzględnych liczb, zatem powinien kształtować się na tym poziomie dla dowolnej maszyny, która również korzysta z HDD - gdyż wydaje się, że więcej RAM na cache nie będzie miało aż tak dużego wpływu, gdyż dane i tak muszą zostać zapisane na dysk.

Innym aspektem jest fakt, iż nie zapisujemy zer na dysk, co w przypadku bardzo duzych plików oszczędza dysk.

3. PROWADZENIE POMIARÓW

Pomiary były wykonane na maszynie wirtualnej z systemem operacyjnym MINIX za pomocą quemu z włączonym kvm i przydzielonymi 512MB pamięci RAM. Główny system operacyjny: debian 4.9.0.2-amd64 z około 3 GB RAM, dysk HDD. Testy wydajnościowe zakładają rozmiar bloku na dysk 4096B(zdefiniowany w common.sh); 'test1.sh' zawiera asercje na poprawność modyfikacji.