

Optymalizacja programów Open-Source

Dostęp do dysku

Krzysztof Lichota
lichota@mimuw.edu.pl

Wprowadzenie

Dostęp do dysku a wydajność

- Dysk jest kilka rzędów wielkości wolniejszy od procesora czy pamięci (patrz prezentacja Federico)
- Samo przesłanie danych z dysku jest stosunkowo szybkie, najwolniejsze jest ustawienie głowicy (seek) oraz opóźnienie podejścia sektora pod głowicę
- Przepustowość standardowego dysku ATA przy dostępie sekwencyjnym wynosi około 60 MB/s, w przypadku dostępu „random access” spada do 1 MB/s albo mniej
- Dlatego należy unikać (niepotrzebnych) dostępuów losowych, zwłaszcza czytania wielu małych plików.

Cechy dostępu do dysku

- Dyski dzielą się na talerze, które mają ścieżki podzielone na sektory
- Ścieżki są oddzielone od siebie (odmiennie niż płyty CD, które mają jedną spiralną ścieżkę), więc żeby odczytać coś z następnej, trzeba przesunąć głowicę
- Czas odczytu sektora = czas przesunięcia (*seek time*) + czas ustawienia nad sektorem + czas przesłania sektora (sektorów)
- Nie wszystkie sektory są odczytywane tak samo szybko – zależy to od położenia ścieżki (patrz ZCAV)

Cechy dostępu do dysku (2)

- Czas przesunięcia nad ścieżkę jest mniej więcej proporcjonalny do odległości między ścieżkami
- Logiczne bloki są wewnętrznie remapowane na fizyczne bloki przez dysk, ale spodziewa się on, że system operacyjny będzie używał dysku w ten sposób, że dane używane razem są na bliskich ścieżkach
- System operacyjny wprowadza dodatkowe przemapowania związane z układem bloków plików na dysku, itp. ale zwykle pliki w jednym katalogu są umieszczane na dysku blisko siebie

Wnioski z cech dostępu do dysku

- Dane, które mają być używane razem należy umieszczać jak najbliżej na dysku
- Jak coś wczytujemy, to należy wczytywać strumieniowo, w szczególności czasem bardziej opłaca się wczytać więcej i wyrzucić część niż wczytywać po kawałku
- Małe dane lepiej upakować w jednym dużym pliku zamiast rozsiewać w kilku małych

Odczyty a zapisy

- Kluczowe w wydajności programów desktopowych są zazwyczaj odczyty
- Zapisy są buforowane w pamięci i rzucane później, kiedy są ku temu możliwości
- Nie można buforować odczytów, bo program czeka na odczytane dane, można tylko przewidywać, czego program będzie potrzebował i wczytać to wcześniej (readahead) – również dzięki podpowiedziom programu (hints)

Analiza korzystania z dysku

Narzędzia do analizy dostępuów do dysku

- Nie ma zbyt wielu narzędzi pozwalających na analizę dostępuów do dysku
- Narzędzia dzielą się na rejestrujące dostępy do plików (ścieżka, położenie w pliku) i rejestrujące dostępy do dysków albo urządzeń blokowych (numer bloku na urządzeniu)
- W przypadku urządzeń rejestrujących dostępy do urządzeń blokowych należy pamiętać, że w przypadku urządzeń LVM/RAID dostępy do prawdziwego dysku mogą być inne

strace

- Przydatne wywołania:
 - `strace -e trace=open,read,write,lseek,_llseek` – śledzi odczyty, zapisy i przesunięcia w plikach
 - `strace -e trace=file` – śledzi inne operacje na plikach (`open`, `stat`, itp.)
- Zalety:
 - Dokładnie pokazuje odczyty i zapisy
- Wady:
 - Nie uwzględnia dostępu przez `mmap()`
 - Nie pokazuje jak odczyty i zapisy przekładają się na dostęp do dysku (mogą być w cache) ani jaka jest pozycja bloków

vm/block_dump

- Loguje operacje na przestrzeni adresowej (zapisy i odczyty stron) oraz zapisy i-węzłów
- Pokazuje nazwę procesu, pid, numer bloku i nazwę urządzenia
- Logowanie odbywa się do bufora na komunikaty systemowe (przez printk)
- Włączenie/wyłączenie przez zapisanie 1/0 do pliku `/proc/sys/vm/block_dump`

Zalety vm/block_dump

- Pokazuje zarówno dostępy przez mmap() jak i zwykłe odczyty/zapisy (wszystkie idą przez operacje na przestrzeni adresowej)
- Pokazuje dostępy opóźnione (np. demona kjournald)
- Pokazuje stan całego systemu
- Pozwala zobaczyć, czy coś działa w tle i zakłóca pomiary

Wady vm/block_dump

- Opóźnione/przedwczesne odczyty/zapisy przez mmap() mogą być przypisywane do innych procesów niż żądające (np. kswapd, kjournald)
- Nie wszystkie odczyty/zapisy są logowane - jeśli dane są już w pamięci podręcznej, to nie są ściągane (można wyczyścić cache za pomocą vm/drop_caches)
- Nie pokazuje, którego pliku dotyczy odczyt/zapis
- Logowanie ma spory narzut, więc nie nadaje się do ciągłego analizowania

Wady vm/block_dump (2)

- Granularność na poziomie stron
- Logowanie na dysk powoduje zakłócenia poprzez dostępy do dysku na logowanie, można wyłączyć demona klogd, ale wtedy bufor się szybko przepełni – alternatywnie można zrzucić wynik „dmesg” na własną rękę, np. przez sieć
- Loguje mnóstwo informacji do bufora na informacje jądra, który ma ograniczoną wielkość i może się przepełnić

blktrace

- Modyfikacja jądra pozwalająca śledzić operacje na blokach w warstwie blokowej jądra Linuksa
- Dostępny standardowo od wersji jądra 2.6.17, wcześniej jako patche
- Wymaga włączenia przy kompilacji jądra, nie jest standardowo dostępny
- Może śledzić operacje odczytu/zapisu do dysku (urządzenia blokowego), ale również bardziej skomplikowane operacje w schedulerze urządzenia (np. sklejanie lub podział żądań)

bltrace (2)

- Zdarzenia są buforowane w jądrze i mogą zostać odczytane za pomocą specjalnego programu pomocniczego poprzez relayfs
- Pokazuje nazwę i pid procesu, typ, pozycję i rozmiar żądania oraz operację

Zalety blktrace

- Pozwala śledzić co się dzieje z żądaniem odczytu programu w warstwie blokowej, np. czy nie czeka na sklejenie z innym
- Bardzo mały narzut (około 2% w skrajnych przypadkach)
- Pozwala filtrować zdarzenia według typu i uruchamiać śledzenie tylko dla niektórych urządzeń
- Ślad ma stosunkowo mały rozmiar, więc można go przechowywać nawet na RAM-dysku
- Posiada tryb do przesyłania śladu siecią

Wady blktrace

- Nie pokazuje powiązania między numerem bloku a plikiem na dysku
- Trzeba samemu skompilować jądro
- Brak strony domowej, dokumentacja jedynie w paczce z programem

Użycie blktrace

- Uruchomić system pod skompilowanym jądrem
- Zamontować system plików debugfs: `mount -t debugfs none /debug`
- Uruchomić: `blktrace -d urządzenie -r /debug -o nazwa`
- Wykonać operacje, które chcemy śledzić
- Przerwać blktrace za pomocą „blktrace -k” lub Ctrl-C, ślad jest w pliku `nazwa-blktrace.numer`
- Przetworzyć ślad na formę tekstową za pomocą `blkparse -i nazwa-blktrace.numer`

Użycie blktrace (2)

- Przydatne opcje blktrace:
 - -w sekundy – zakończenie po podanej liczbie sekund
 - -a typ – śledzi tylko określony typ zdarzeń (np. READ, WRITE, ISSUE, SYNC, COMPLETE)
- Można śledzić „na żywo”:
 - blktrace -d /dev/sda -o - | blkparse -i -

Inne narzędzia

- kprobes/jprobes – narzędzie do instrumentacji i śledzenia wywołań w jądrze Linuksa w trakcie pracy, może być użyte do przechwytywania odczytów/zapisów

Własne narzędzia

- Ponieważ każde z istniejących narzędzi ma swoje wady, czasem trzeba napisać własne, które robi dokładnie to, czego od niego oczekujemy
- Przy optymalizacji układu plików na live CD napisałem własną modyfikację modułu SquashFS jądra Linuksa, która wpina się w funkcję odczytu strony (z pamięci adresowej) i zapisuje nazwę pliku, offset w pliku i numer bloku

Poprawianie korzystania z dysku

Zmiana architektury programu

- Najlepszym rozwiązaniem jest zmiana zachowania programu tak, by nie używał losowych dostępuów, trzymał wszystkie pliki w jednym katalogu, używał możliwie dużych plików
- Szczegółowe dostosowanie wymaga zanalizowania wzorca zachowań programu i dostosowania struktur na dysku tak, by wczytywać i zapisywać dane strumieniowo
- Skrajny przykład – system plików GoogleFS, który działa strumieniowo

Scalanie plików

- Jeśli program używa wielu małych plików, można je scalić w jeden plik i wczytywać ten jeden plik
- Jeśli pliki mogą być modyfikowane, tworzymy plik, który służy jako pamięć podręczna – sprawdzamy, czy któryś plik nie był zmodyfikowany (za pomocą stat), jeśli nie, to wczytujemy duży plik (tak działa np. ksycoca w KDE)

readahead

- Jeśli wiemy, których danych będziemy potrzebować, należy z wyprzedzeniem wskazać systemowi operacyjnemu, żeby je wczytał do pamięci podręcznej
- System operacyjny robi to też na własną rękę, jeżeli program zaczyna robić odczyty sekwencyjne przez jakiś czas, ale działa to z opóźnieniem i nie zawsze odpowiada dokładnie zachowaniu programu, więc może nawet przeszkadzać

posix_madvise/posix_fadvise

- Funkcje systemowe podpowiadające systemowi operacyjnemu w jaki sposób będziemy korzystać z pliku
- posix_fadvise – dostęp przez deskryptor, przez funkcje read()/write()
- posix_madvise() - dostęp przez mmap (zakres przestrzeni adresowej procesu)
- readahead() - specyficzna dla Linuksa, odpowiada posix_fadvise() z parametrem WILLNEED

posix_madvise/posix_fadvise (2)

- Typy odpowiedzi (to nie są dokładne nazwy flag):
 - SEQUENTIAL – dostęp do pliku będzie sekwencyjny, więc należy wczytać więcej kolejnych bloków z wyprzedzeniem
 - RANDOM – dostęp do pliku będzie losowy, więc nie należy wczytywać z wyprzedzeniem
 - WILLNEED – te bloki będą potrzebne, więc należy je wczytać
 - DONTNEED – te bloki nie będą potrzebne w najbliższej przyszłości
 - NOREUSE (tylko fadvise) – te dane będą użyte tylko raz, potem można je wyrzucić

Bibliografia

- <http://www.coker.com.au/bonnie++/zcav/results.html>
- http://www.cs.ucsb.edu/research/tech_reports/reports/20
- <http://lxr.linux.no/source/Documentation/filesystems/proc>
- http://www.gelato.org/pdf/apr2006/gelato_ICE06apr_blktr
- <http://brick.kernel.dk/snaps/>
- <http://www.redhat.com/magazine/005mar05/features/kpro>
- <http://lwn.net/Articles/132196/>
- <http://lichota.net/~krzysiek/projects/kubuntu/dapper-livecd>