CACHOVACÍ ALGORITMY V DISTRIBUOVANÝCH SYSTÉMECH SOUBORŮ

Pavel BŽOCH[1]

1 Úvod

Potřeba uchovávat a sdílet data v posledních letech velice roste. Uchovávaná data mohou být různých formátů (např. multimédia, dokumenty, vědecké výpočty a další). Takto vyprodukovaná data můžeme uchovávat na lokálním systému souborů, na vzdáleném serveru nebo na distribuovaném systému souborů.

Lokální systém souborů poskytuje data velice rychle v porovnání s ostatními možnostmi uložení. Je však náchylný na chyby hardware, těžce škálovatelný a uložená data jsou přístupná pouze lokálně, tzn., že data není možné přistupovat odkudkoli.

Ukládání dat na vzdálený server přináší proti uložení dat na lokálním stroji možnost přistupovat data vzdáleně. Oproti lokálnímu přístupu je ovšem nutné používat autentizační algoritmy pro ověřování uživatele, což zpomaluje přístup k datům. Uložená data jsou navíc stejně náchylná na chyby v hardware, vzdálený server je opět těžce škálovatelný.

Distribuovaný systém souborů (DSS) se snaží minimalizovat nevýhody vzdáleného serveru. DSS používá pro uložení dat více uzlů. Uložená data mohou být replikována, čímž se zvyšuje spolehlivost a dostupnost. Často přistupovaná data mohou být uložena v uzlech, které mají vysoký výkon, čímž se zvyšuje výkon celého systému. Pokud je kapacita DSS nedostačující, je možné do DSS připojit další uzly. Systém je tedy dobře škálovatelný. V dalším textu se budeme zabývat komponentou cache, která slouží k dalšímu urychlení přístupu k datům.

2 Cache a cachovací algoritmy

Cache je komponenta, která uchovává často přistupovaný obsah. Tento obsah uchovává v rychlé paměti (obecně v rychlejší, než na jaké jsou data běžně uložena). U dat v cache se předpokládá, že budou v budoucnosti opět používána. V DSS můžeme cache používat jak na straně uživatelské, tak na straně serveru. V dalším textu se budeme zabývat nasazením cache v uživatelské aplikaci.

Nevýhoda cache spočívá v tom, že má omezenou kapacitu. Není tedy možné do cache uložit všechna data, která uživatel požaduje, ale jen některá. Pokud je cache již zaplněna a je do ní potřeba uložit nová data, je potřeba „vyhodit“ některé bloky dat, aby uvolnily místo. Algoritmy, které slouží k tomuto označování bloků dat, se označují jako cachovací algoritmy nebo caching policy v anglickém jazyce. Cachovací algoritmy lze rozdělit do několika kategorií: jednoduché, sofistikované a hybridní. Nyní si představíme blíže jednotlivé kategorie včetně zástupců cachovacích algoritmů.

A. Jednoduché cachovací algoritmy

Tyto algoritmy pro rozhodnutí o vyhození bloků z cache nepoužívají žádné informace o těchto blocích. Příkladem jednoduchého algoritmu může být např. algoritmus **RND**. RND nebo také Random mechanismus označuje bloky pro nahrazení náhodně. Dalším jednoduchým algoritmem je algoritmus **FIFO**, který reference na bloky uchovává ve frontě a který v cache nahazuje soubory, které jsou v ní nejdéle.

[1] Ing. Pavel Bžoch, student doktorského studijního programu Inženýrská informatika, obor Informatika a výpočetní technika, e-mail: pbzoch@kiv.zcu.cz

B. Sofistikované cachovací algoritmy

Sofistikované algoritmy používají pro rozhodnutí o vyhození z cache statistiky, které vytvářejí z požadavků uživatele. Algoritmus **LFU** (Least Frequently Used) si pro každý cachovaný blok dat uchovává počet přístupů. Při plné cache označuje na vyhození blok, který má nejmenší počet přístupů. Algoritmus **LRU** (Least Recently Used) si pro každý blok uchovává čas posledního přístupu. Pokud má algoritmus LRU označit blok na nahrazení, jedná se vždy o blok, který nebyl nejdelší dobu přistupován.

C. Hybridní cachovací algoritmy

Oba zmiňované sofistikované algoritmy mají určité nevýhody. LRU např. může z cache vyhodit blok, který je periodicky požadován, pokud je mezi periodou mezi přístupy požadavek na velké množství jiných nově příchozích bloků. Nevýhoda LFU spočívá v tzv. stárnutí bloků v cache. Pokud byl blok v určitou dobu v minulosti často přistupován, ale v současné chvíli už na něj nejsou žádné požadavky, je jeho počet přístupů vysoký a není jej možné v cache nahradit.

Hybridní cachovací algoritmy jsou algoritmy, které vznikly kombinací LFU a LRU. Tyto algoritmy se snaží využít výhod obou těchto algoritmů při předvídání budoucího chování uživatelů. Jako příklad lze uvést algoritmus **2Q.** Tento algoritmus používá pro nově příchozí bloky FIFO algoritmus. Pokud je blok v FIFO požadován podruhé, použije se pro něj LRU algoritmus. Algoritmus LRU tedy spravuje soubory se dvěma a více referencemi. Při požadavku na nahrazení bloku jsou nejprve nahrazovány bloky, které spravuje FIFO, až dále jsou nahrazovány bloky, které jsou spravovány pomocí LRU.

3 Vývoj nového cachovacího algoritmu

Během mého dalšího studia se chci věnovat návrhu nového cachovacího algoritmu, který bude použitelný pro DSS. Výše zmíněné cachovací algoritmy berou v potaz pouze lokální chování uživatele. Pro vývoj nového cachovacího algoritmu bychom chtěli použít i statistiky, které budou sledovány na straně serveru. Tyto statistiky jsou počet přístupu k bloku na čtení, počet přístupu k bloku na zápis a celkový počet přístupů na čtení u všech bloků. V rámci tohoto přístupu chceme v lokální uživatelské cache zvýhodnit nově příchozí bloky dat, které budou často přistupovány na serveru a mají tedy velkou pravděpodobnost, že budou opakovaně přistupovány uživatelem a měly by tudíž zůstat v cache.

4 Závěr

Cache je komponenta, která urychluje zpracování požadavků uživatele na přístup k datům. Protože je kapacita cache malá, je potřeba navrhnout algoritmy, které uvolní bloky při plné cache pro další bloky. V distribuovaných systémech souborů se v současné době používají cachovací algoritmy, které používají pouze lokální statistiky přístupů. V naší budoucí práci bychom chtěli vytvořit cachovací algoritmus, který bude pro rozhodování o vyhazování bloků z cache používat i statistické údaje, které získá ze serveru.

Literatura

Benjamin Reed and Darrell D. E. Long, "Analysis of caching algorithms for distributed file systems," in *ACM SIGOPS Operating Systems Review, Volume 30 Issue 3*, New York, NY, USA, 1996, pp. 12-17

Theodore Johnson and Dennis Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," in *In VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 439-450.