ITAT 2013 Proceedings, CEUR Workshop Proceedings Vol. 1003, pp. 65–68
http://ceur-ws.org/Vol-1003, Series ISSN 1613-0073, © 2013 M. Kruliš, M. Čermák, Z. Falt, J. Yaghob

Revize metod externího třídění pro moderní hardware

Martin Kruliš, Miroslav Čermák, Zbyněk Falt a Jakub Yaghob *

Katedra softwarového inženýrství, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze, Malostranské nám 25., Praha 1
{krulis,cermak,falt,yaghob}@ksi.mff.cuni.cz

Abstrakt: Metody externího třídění, tedy třídění využívajícího vnější paměť, jsou velmi dobře známy již mnoho desetiletí. Tyto metody byly původně navrženy pro systémy s malým množstvím interní paměti a magnetickými páskami coby vnější pamětí. Magnetické pásky jsou specifické čistě sekvenčním přístupem k datům, který také ovlivnil návrh metod externího třídění. Pásky byly nahrazeny pevnými disky s magnetickými plotnami, které přinesly možnost náhodného přístupu k datům, avšak sekvenční přístup zůstal nadále výrazně výkonnější. Většina hardwarových předpokladů, na kterých je externí třídění postaveno se za poslední desetiletí výrazně změnila, zejména s příchodem SSD disků a vývojem nonvolatilních pamětí. V tomto článku představujeme nový přístup k externímu třídění, který reflektuje parametry současného hardware. Dále předkládáme empirické srovnání s již existujícími metodami, které se hojně používají v soudobých systémech.

Klíčová slova: třídění, vnější paměť, algoritmus, optimalizace, moderní hardware

1 Úvod

Třídění patří k základním stavebním kamenům řady algoritmů a aplikací. V databázových systémech patří společně s operací JOIN k nejpoužívanějším operacím vůbec. I přes rostoucí kapacity a klesající ceny operačních pamětí není vždy možné řešit tuto úlohu čistě v rámci vnitřní paměti počítače. Pro tyto situace máme k dispozici algoritmy externího třídění, které využívají diskové úložiště k odkládání mezivýsledků.

Algoritmy vnějšího třídění byly navrženy a zmapovány již v dobách, kdy se jako externí paměť používaly magnetické pásky. I přes jejich původní cíle se tyto algoritmy s drobnými úpravami používají dodnes. Jejich hlavní nevýhodou je, že předpoklady, ze kterých tyto algoritmy vychází dnes již neplatí. Nejvýznamnější předpoklady můžeme shrnout takto:

- Externí paměť je organizována sekvenčně, nebo je sekvenční přístup výrazně výkonnější.
- Máme k dispozici pouze malé množství externích pásek, resp. můžeme rozumně pracovat pouze s malým množstvím otevřených souborů.
- Externí paměť je o mnoho řádů pomalejší než interní paměť, takže dominantní složkou vyjadřující časovou složitost je počet operací s externí pamětí.

V tomto článku bychom rádi vyvrátili tyto předpoklady pomocí empirických experimentů, jejichž cílem je identifikovat vlastnosti soudobých pevných disků s magnetickými plotnami a SSD pevných disků. Na základě opravených předpokladů pak navrhujeme změny pro algoritmy vnějšího třídění, které by měly značně vylepšit jejich výkon.

Tento článek je organizován následovně. Sekce 2 shrnuje přehled souvisejících prací zaměřených na externího třídění. Popis současných algoritmů se nachází v sekci 3. Sekce 4 definuje nové předpoklady o současném hardware a na jejich základě navrhuje změny v metodách externího třídění. Výsledky experimentů, které podporují naše závěry, se nachází v sekci 5 a sekce 6 uzavírá článek.

2 Související výzkum

Problém externího třídění, tedy třídění za použité vnější paměti, je jedním z hlavních problémů v oblasti algoritmů pro externí paměť. Částečně je tomu tak proto, že třídicí operace tvoří signifikantní část počítačových operací [6], a částečně proto, že třídění je důležitým paradigma v návrhu efektivních algoritmů nejen pro externí paměť. Studium těchto problémů a analýza algoritmů používajících vnější paměť mají své počátky před více než 50ti lety v Demuthově doktorské tezi [3], která se zaměřovala zejména na třídění. V 70tých letech provedl Knuth [6] rozsáhlou studii třídění v rámci svých knih pojednávajících o umění moderního programování. V knize o třídění a vyhledávání se zabývá mimo jiné strategiemi výběru a nahrazení a metodami polyfázového slévání za použití magnetických pásek a magnetických disků. Od té doby vzniklo mnoho nových a upravených algoritmů pro třídění za pomoci externí paměti [11], avšak všechny tyto metody sdílí společné předpoklady definované Knuthem.

Datová komunikace mezi rychlou interní pamětí a pomalejší externí pamětí je považována za úzké hrdlo při zpracování velikých dát [2, 7, 11]. Většina algoritmů se snaží dosáhnout odbourání tohoto úzkého hrdla minimalizací počtu vstupně výstupních operací, optimální prací s vyrovnávací pamětí [7], nebo asynchronním načítáním dat [2]. Další metodou je nasazení více pevných disků, které se využívají buď nezávisle, nebo pomocí prokládání (stripingu), který bývá často efektivnější než komplikované algoritmy pro nezávislé využití disků [10]. Nejnovějším přístupem je využití distribuovaného prostředí při

*Článek byl podporován Grantovou agenturou Univerzity Karlovy, projekty č. 472313 a 277911, Grantovou agenturou ČR (GAČR) P103/13/08195S a grantem SVV-2013-267312.

vhodném rozdělení dat a výpočtů [8]. Pokroku ve výkonu disků (především SSD) si všimli také autoři energeticky efektivních algoritmů [1, 9], jejichž hlavním cílem je opět minimalizace počtu diskových operací a tedy i minimalizace cyklů slévání.

3 Existující metody vnějšího třídění

Většina algoritmů externího třídění má dvě části. V první části se generují takzvané *běhy*, tedy sekvence setříděných dat. Druhá část pak provádí postupné slévání těchto běhů, dokud nevznikne běh jediný, který reprezentuje setříděná data. Algoritmy se však liší tím, jakým způsobem běhy generují a jak je slévají.

3.1 Generování běhů

Nejjednodušší metodou je přímočaré generování běhů pomocí jednoho bufferu[1], jehož velikost je zvolena tak, aby využíval veškerou dostupnou interní paměť. Buffer je poté naplněn vstupními daty a setříděn vhodnou metodou interního třídění, například Quicksortem [5]. Data z bufferu jsou následně přesunuta do vnější paměti, čímž je vytvořen jeden běh. Tento postup je opakován, dokud se nachází nezpracovaná data ve vstupním souboru.

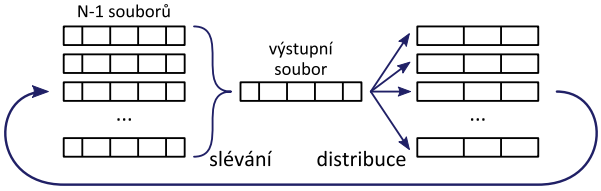
Na první pohled by se mohlo zdát, že není možné generovat běhy delší, než je velikost vnitřní paměti. Existuje drobné vylepšení, které zajišťuje generování běhů větších délek, pokud jsou vstupní data vhodně koncipována. Tato metoda používá dvě prioritní fronty (typicky reprezentované 2-regulárními haldami), které se dělí o veškerou dostupnou paměť.

Na počátku zabírá první halda veškerou paměť a je zcela naplněna daty ze vstupního souboru. V každém kroku je z haldy odebráno minimum a zapsáno do právě generovaného běhu na disku. Jako náhrada za toto minimum je ze vstupního souboru načten další prvek. Pokud je tento prvek větší nebo roven prvku právě zapsanému do výstupního běhu, může být nový prvek začleněn do první haldy. Pokud je větší, první halda zmenší svou velikost o jedna a prvek je vložen do haldy druhé, která se tím zároveň zvětší. Generování běhu končí v okamžiku, kdy je první halda vyčerpána a druhá zabírá celou paměť. V tomto okamžiku se prohodí význam obou hald a může začít generování dalšího běhu.

Experimenty na náhodně uspořádaných datech s uniformním rozložením ukazují, že běhy generované pomocí dvou hald mají v průměru dvojnásobnou délku, než je velikost dostupné paměti.

3.2 N-cestné dvoufázové třídění

Předpokládejme, že daný systém je schopen otevřít nejvýše N souborů, resp. může současně používat nejvýše N pásek. Nejjednodušší implementace slévání běhů potom využívá $N-1$ souborů jako vstup a jeden soubor pro uchování výstupu. Na počátku jsou při generování běhy rozdistribuovány rovnoměrně mezi vstupní soubory. Proces slévání pak pracuje iterativně a každá iterace má dvě fáze. V první fázi slévá současně běhy z $N-1$ souborů a výsledné běhy zapisuje do výstupního souboru. V druhé fázi je přečten celý výstupní soubor a běhy z něj jsou rovnoměrně rozdistribuovány mezi vstupní soubory. Algoritmus končí, když ve výstupním souboru zůstane pouze jeden běh.



Obrázek 1: Princip dvoufázového třídění

Proces slévání je znázorněn na obrázku 1. Samotné slévání probíhá ve vnitřní paměti a je možné jej implementovat například pomocí prioritní fronty (tzn. haldy), která si udržuje první dosud nezpracovaný prvek z každého běhu.

Hlavní nevýhodou tohoto postupu je nutnost provádět opětovnou redistribuci běhů ve druhé fázi každé iterace. Jedním z možných řešení, je použít pouze $N/2$ souborů jako vstupních a generované běhy distribuovat rovnou mezi $N/2$ výstupních souborů. Tento postup má ale nevýhodu v tom, že slévá vždy pouze $N/2$ běhů místo $N-1$ běhů, díky čemuž může potřebovat větší množství iterací.

3.3 Polyfázové třídění

Hlavní nevýhodu dvoufázového třídění do jisté míry řeší polyfázové třídění. Jeho idea spočívá v nerovnoměrné distribuci běhů, která je chytře navržena tak, aby bylo možné běhy neustále slévat bez jejich opětovného rozhazování. Přitom je vždy $N-1$ souborů používáno jako vstupních a výsledné běhy se ukládají do jednoho souboru.

Cílem je, aby při posledním slévání byl v každém z $N-1$ vstupních souborů právě jeden běh. V takovém případě proběhne poslední krok slévání optimálním způsobem. Počáteční rozložení běhů lze dopočítat reverzním naplánováním všech iterací slévání od optimální koncové konfigurace. Situace pro $N=3$ a 21 počátečních běhů je znázorněna v následující tabulce.

soubor	zač.	#1	#2	#3	#4	#5	#6
#1	13	5	0	3	1	0	1
#2	8	0	5	2	0	1	0
#3	0	8	3	0	2	1	0

Tabulka 1: Optimální slévání 21 běhů pro $N=3$

Na počátku je v prvním souboru 13 běhů a ve druhém 8. V každém kroku se vezme nejvyšší možný počet běhů, který je možný slít přímo (např. v prvním kroku 8), čímž se

[1] Z důvodu nedostatku vhodné terminologie budeme v tomto článku používat anglický termín *buffer* v jeho počeštěné podobě.

vždy právě jeden soubor uvolní. V případě dvou vstupních a jednoho výstupního souboru je rozložení běhů založeno na Fibonacciho číslech.

4 Moderní přístup k vnějšímu třídění

V této sekci nastíníme nové předpoklady o hardware, zejména o pevných discích, a na jejich základě opravíme existující algoritmy vnějšího třídění.

4.1 Vlastnosti hardware

Na základě empirických pozorování, jejichž nejdůležitější výsledky jsou shrnuty v sekci 5, můžeme postulovat následující předpoklady:

- Soudobé magnetické disky sice stále preferují sekvenční přístup, avšak pokud je k datům přistupováno po dostatečně velkých blocích (řádově jednotky až desítky MB), je možné aplikovat nad těmito bloky náhodný přístup bez výrazného poklesu výkonu. U SSD disků je možné používat i bloky menší.
- Průměrná rychlost čtení a zápisu výrazně převyšuje rychlost vnitřního sériového třídění a je srovnatelná s rychlostí paralelního vnitřního třídění (na běžně dostupném hardware).
- Použití prioritní fronty (resp. haldy) k vnitřnímu třídění je výrazně pomalejší než použití Quicksortu.
- Soudobé operační systémy zvládají bez problémů pracovat i s tisíci otevřenými soubory současně.
- Poměr velikosti vnitřní a vnější paměti je na běžných hardwarových konfiguracích menší než 1 : 1000.

Z výše uvedených předpokladů vyplývají dvě věci. Externí třídění již není potřeba optimalizovat na počet diskových operací, neboť za běžných podmínek je možné provést slévání všech vygenerovaných běhů najednou. Díky tomu je každý prvek právě dvakrát čten z disku a právě dvakrát na disk zapsán. Za druhé, možnost paralelního zpracování tříděných dat ve vnitřní paměti je výrazně důležitější pro budoucí škálovatelnost, protože rychlost pevných disků již přesáhla propustnost třídicích algoritmů pro vnitřní paměť.

4.2 Změny v algoritmech

Základní koncept externího třídění zůstává nadále nezměněn. Každý algoritmus má tedy dvě části – generování běhů a jejich následné slévání. Tyto části jsou do jisté míry nezávislé, a proto se jim můžeme věnovat samostatně.

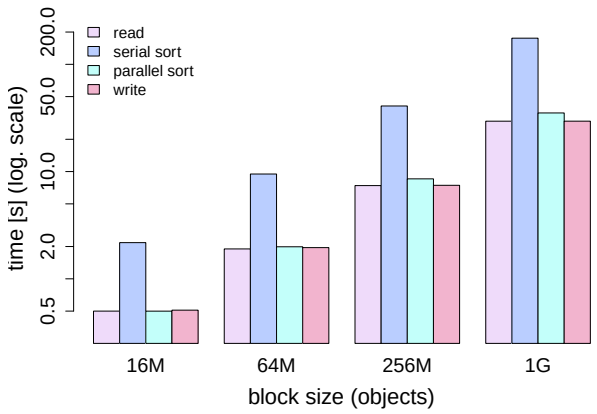
Při generování běhů jsme empiricky ověřili, že použití dvou hald sice vede k vytvoření menšího počtu delších běhů, avšak tento proces je výrazně pomalejší než použití optimalizovaných algoritmů. Hledání minima pomocí haldy navíc není možné (na rozdíl od ostatních třídicích algoritmů) jednoduše paralelizovat. V tomto okamžiku se jako nejvíce vhodný algoritmus jeví rozdělení paměti na dva (případně tři) stejně velké úseky, přičemž data v jednom úseku jsou tříděna zatím co data ve druhém (a případně třetím) úseku jsou přenášena z disku nebo na disk. Jako algoritmus vnitřního třídění poslouží nejlépe paralelní verze Quicksortu.

Jak jsme již naznačili, samotný proces slévání je za běžných okolností možné provést v jediném kroku. Pokud je vnitřní paměť řádově přibližně tisíckrát menší než největší možná data ve vnější paměti, pak první část třídění vygeneruje nejvýše tisíc běhů, které můžeme mít uloženy v tisíci nezávislých souborech. K samotnému slévání potom můžeme použít buď prioritní frontu, jak navrhuje Knuth [6], nebo upravenou techniku paralelního proudového slévání, kterou představil ve své práci Falt [4].

5 Experimenty

Experimenty byly prováděny na běžném PC s procesorem Core i7 (4 fyzická, 8 logických jader) vybaveném 16 GB RAM. Data byla uložena na samostatném disku (1 TB, 7200 otáček) a druhý (identický) disk byl použit pro dočasné soubory s vygenerovanými běhy. Testovací data reprezentoval soubor 32 bitových celočíselných hodnot, které byly náhodně vygenerovány s uniformním rozdělením.

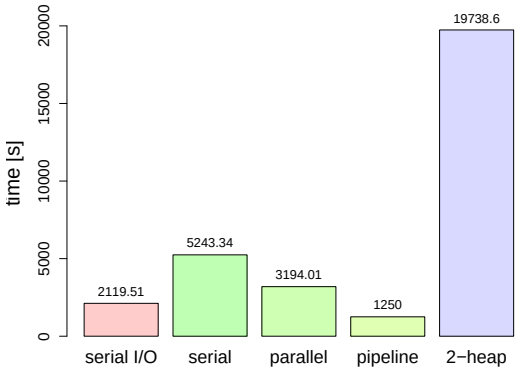
První sada experimentů zkoumá poměr časů potřebných k setřídění bloku dat pomocí interního třídění a časy potřebné k přečtení resp. zapsaní těchto dat na disk. Obrázek 2 prezentuje naměřené časy pro různě velké bloky dat a srovnává jednovláknové třídění Quicksortem s jeho paralelní verzí. Z výsledků je patrné, že časy diskových operací jsou výrazně nižší, než doba potřebná k setřídění dat pomocí jednoho jádra a přibližně srovnatelné s tříděním, které využívá všech 8 logických jader procesoru.



Obrázek 2: Porovnání časů třídění a diskových operací

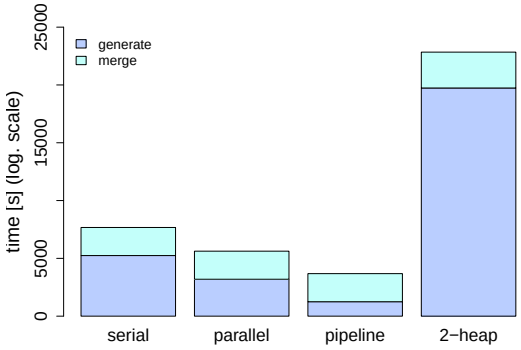
Další experimenty se týkají generování běhů. V těchto experimentech byl použit vstupní soubor o velikosti 64

GB (16 miliard čísel) a buffer pro vnitřní třídění o velikosti 1 GB (tedy pro 256 milionů čísel). Tyto experimenty srovnávají sériový přístup, který realizuje všechny diskové operace i třídění v jediném vlákně, paralelní přístup, který provádí diskové operace sériově, ale ke třídění dat využívá všechna dostupná vlákna, přístup založený na pipeline, který provádí diskové operace asynchronně a zároveň třídí paralelně, a konečně generování běhů pomocí dvou hald.



Obrázek 3: Časy potřebné pro generování běhů

Metoda generování běhů pomocí dvou hald má sice pozitivní dopad na délku (a tedy i počet) běhů [6], avšak z grafu na obrázku 3 je patrné, že tento postup je výrazně pomalejší z důvodu značného nárůstu výpočetních operací a náhodnému přístupu do vnitřní paměti, který špatně využívá vyrovnávací paměti procesoru.



Obrázek 4: Celkové časy externího třídění

Celkové časy třídění potom prezentuje obrázek 4. Ke slévání běhů byl použit mechanismus vícecestného slévání pomocí 2-regulární haldy. Slévání 128 běhů, které vygenerovaly první tři metody trvalo přibližně 2400 sekund, zatímco slévání 64 běhů vygenerovaných pomocí dvou hald trvalo 3100 sekund. Tento překvapivý výsledek se nám zatím nepodařilo uspokojivě vysvětlit.

6 Závěr

V tomto článku jsme aktualizovali některé zažité předpoklady týkající se vnějších pamětí a na jejich základě jsme navrhli změny třídicích algoritmů, které měly pozitivní dopad na výkon a budoucí škálovatelnost. Prokázali jsme, že počet diskových operací již není hlavním kritériem optimalizace těchto algoritmů, ale vývoj je třeba směřovat do paralelních implementací. V pokračování této práce se chceme zaměřit na vylepšení vícecestného slévání pro paralelní systémy a provést rozsáhlejší testy zejména za použití diskových polí a SSD disků.

Reference

[1] Andreas Beckmann, Ulrich Meyer, Peter Sanders, and Johannes Singler. Energy-efficient sorting using solid state disks. *Sustainable Computing: Informatics and Systems*, 1(2):151–163, 2011.

[2] Paolo Bertasi, Marco Bressan, and Enoch Peserico. psort, yet another fast stable sorting software. *Journal of Experimental Algorithmics (JEA)*, 16:2–4, 2011.

[3] Howard B Demuth. *Electronic data sorting*. Dept. of Electrical Engineering, 1956.

[4] Zbyněk Falt, Martin Kruliš, and Jakub Yaghob. Optimalizace třídicích algoritmů pro systémy proudového zpracování dat. *Informačné Technológie - Aplikácie a Teória*, pages 69–74, 2011.

[5] C.A.R. Hoare. Quicksort. *The Computer Journal*, 5(1):10, 1962.

[6] Donald Ervin Knuth, Donald Ervin Knuth, and Donald Ervin Knuth. *Sorting and Searching*. Addison-Wesley, 2003.

[7] Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, and Dave Lomet. Alphasort: A cache-sensitive parallel external sort. *The VLDB Journal – The International Journal on Very Large Data Bases*, 4(4):603–628, 1995.

[8] Alexander Rasmussen, George Porter, Michael Conley, Harsha V Madhyastha, Radhika Niranjan Mysore, Alexander Pucher, and Amin Vahdat. Tritonsort: A balanced large-scale sorting system. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 3–3. USENIX Association, 2011.

[9] Vijay Vasudevan, Lawrence Tan, Michael Kaminsky, Michael A Kozuch, David Andersen, and Padmanabhan Pillai. Fawnsort: Energy-efficient sorting of 10gb. *Sort Benchmark final*, 2010.

[10] Darren Erik Vengroff and J Scott Vitter. Supporting i/o-efficient scientific computation in tpie. In *Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on*, pages 74–77. IEEE, 1995.

[11] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001.