한국정보과학회
KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

http://www.kiise.or.kr

2016 한국컴퓨터종합학술대회

Korea Computer Congress 2016

대한민국의 힘, 지능정보기술!

2016. 6. 29(수)~7. 1(금), 제주ICC&부영호텔

- 빅데이터 플랫폼 기술(6.29)
- 인공지능 최근 동향 워크샵(6.29)
- 가상현실 핫 이슈와 전망(6.29)
- 정보통신 소사이어티 기술워크샵(6.29)
- 인공지능 소사이어티 워크샵(6.30)

후 원

플래티넘 NAVER KOFST

골 드 Jeju CVB 제주컨벤션뷰로 Jeju 제주특별자치도

O3.3 컴퓨터시스템 II	7.1(목) 12:40-14:40 제주ICC 3층 304호
▷ 좌장: 이혁준(서강대)	▷ 평가위원: 박기진(아주대)

O3.3-01 Hadoop의 데이터 블록 개수에 따른 시작 프로세스 오버헤드 분석
노승준·강동현·엄영익(성균관대)

O3.3-02 F2FS 파일시스템의 공간 효율성 분석
박종규·강동현·엄영익(성균관대)

O3.3-03 에너지 수집 무선 센서 네트워크에서 잔여 에너지양 예측을 통한 데이터 병합 기법
정세미·김혁·노동건·윤익준(숭실대)

O3.3-04 OpenCL을 이용한 이미지 처리 프로그램 자동 최적화 방법
신재호·조강원·이일구·이재진(서울대)

O3.3-05 [우수논문] NVM 기반 시스템에서의 다중 타입 객체 지원 가비지 수집 기술
이도근·손성배·이성진·원유집(한양대)

O3.3-06 [우수논문] 웹 애플리케이션 성능 개선을 위한 V8 JavaScript 엔진의 아키텍처 레벨 프로파일링
강윤지·홍경환·신동군(성균관대)

O4.1 데이터베이스 I	7.1(목) 09:20-11:40 제주ICC 3층 300호
▷ 좌장: 김정동(선문대)	▷ 평가위원:

O4.1-01 [우수논문] K데이터를 포함하는 P개-최소MBR 탐색
김건우·김영훈(한양대)

O4.1-02 [우수논문] 재현율 기반 효과적 평가척도에 관한 연구
임지영·송종수·이철기·이우기(인하대)

O4.1-03 Opportunity of using Multi-Streamed SSD in MongoDB
Trong-Dat Nguyen·Sang-Won Lee(성균관대)

O4.1-04 [우수논문] 요약 그래프 상에서 효과적인 그래프 압축 기법
서호진·박기성·이영구(경희대)

O4.1-05 [우수논문] 대용량 데이터를 공개하기 위한 맵리듀스 기반 개인정보 보호 알고리즘
송광호·심규석(서울대)

O4.1-06 빅데이터 플랫폼의 병렬성 측면에서의 HDFS Archival Storage 성능 분석
김재형·박상현(연세대)

O4.1-07 로그 데이터 이상 탐지를 위한 지수 가중 이동 평균 및 3-시그마 활용 기법
손시운·길명선·문양세(강원대)

O4.2 정보보호	7.1(목) 09:20-11:40 제주ICC 3층 301A호
▷ 좌장: 최윤호(부산대)	▷ 평가위원:

O4.2-01 [우수논문] 결정트리 기반의 기계학습을 이용한 익명화기법 연구
김영기·홍충선(경희대)

O4.2-02 매쉬업 서비스를 위한 Smart Mediator에서의 효율적인 익명화 기법 연구
이다은·홍충선(경희대)

Opportunity of using Multi-Streamed SSD in MongoDB

Trong-Dat Nguyen Sang-Won Lee
College of Information and Communication Engineering, Sungkyunkwan University
datnguyen@skku.edu, swlee@skku.edu

Abstract

MongoDB is popular NoSQL DBMS that shares some common mechanisms with traditional RDBMS. Thus some techniques from RDBMS can also be used in MongoDB. This paper evaluates and analyzes the IO pattern of MongoDB using YCSB benchmark and blktrace. There is a chance to apply multi-streamed SSD in MongoDB due to block allocation mechanism inside WiredTiger storage engine.

1. Introduction

MongoDB is document store NoSQL DBMS that shares some core features with traditional RDBMS e.g. transaction processing, multi-version concurrency control. With WiredTiger as storage engine, MongoDB become a high performance, scalable NoSQL that support both row-oriented storage and column-oriented storage.

Typically, NoSQL DBMS prefers BASE [1] properties (Basic availability, Soft-state, and Eventually consistent) than ACID (Atomic, Consistency, Isolation, and Durability). However, MongoDB ensure ACID at document granularity level with adapting of WiredTiger storage engine [2] . Read committed isolation level is used as default, for ensuring atomicity and durability, WiredTiger use write-ahead log approach (WAL) that write the log records on persistent device before flush the data pages.

Low-level SSD's Flash Translation Layer (FTL) does not aware of information in application layer. In other words, FTL works in same manner regardless of hot pages or cold pages are used. Thus, hot pages and cold pages may locate in the same physical block that is not good for the sake of low level SSD performance due to garbage collection (GC) mechanism. If an application can distinguish hot pages and cold pages in the user space, by adapting multi-streamed SSD technique [3] , one can separately write hot pages and cold pages in different physical blocks.

In this paper, we experiment MongoDB with YCSB to understand the IO pattern, especially write pattern. Due to space management mechanism internally inside WiredTiger storage engine, the result shows that there is asymmetric in number of page writes based on block address that can aid to distinguish hot pages and cold pages. Thus there is an opportunity to adapt multi-streamed SSD in MongoDB.

2. Background

2.1. Multi-streamed SSD technique

Originally, FTL does not aware of hot pages and cold pages in user space and treat them similar, that hot pages and cold pages may locate in the same block. Because hot pages have more access frequency than cold pages, when that block become a victim for garbage collection, FTL need to copy cold pages in new block before erase the old block. That not only leads to increasing overhead for GC process but also effect wear-leveling of flash memory. The main idea of multi-streamed SSD[3] is provide different stream-id for each page write system call from user space that help distingue page written based on their own characteristic e.g. hot and cold. The result is hot pages and cold pages locate in different blocks, that lead to two advances: 1) erase hot page blocks are less expensive due to copying cold pages are eliminated, and 2) number of erasing operations are reduced that decrease wear-leveling of flash memory.

2.2. Block allocation mechanism in WiredTiger

For a shake of efficiently space management, WiredTiger use three kinds of extend list named **`avail`**, **`alloc`**, and **`discard`** for keeping metadata of available blocks, allocated blocks and discard blocks accordingly. Extend list is implemented as skip-list data structure that include of extends, each extend store metadata for a data block i.e. block offset and size on the physical storage device.

Whenever a write request comes from the client, the update changes are generated as dirty pages. Those pages are written to persistent device during eviction time or checkpoint time. In such cases, the system need to find the available block on disk that can fit the dirty pages. WiredTiger using two approaches to searching for a fit block: 1) **`first-fit`** i.e. fitted by offset, that start searching from the beginning of

underlying file until find the first block that can fit, and 2) **best-fit** i.e. fitted by size, that start searching in size skip-list until find the block that can fit.

Technically, each checkpoint in the current transaction kept three extend lists to keep the block management updated. As showed in Figure 1, those metadata is flush to persistent device at checkpoint time and fetching back to D-RAM for the next checkpoint. WiredTiger only need two checkpoints to ensure the durability of the system, the live checkpoint in D-RAM is kept changes in current transaction and the previous checkpoint that usually on disk. At the checkpoint time, the previous checkpoint is fetch from disk to D-RAM then merge with live checkpoint as showed in Figure 1a. After the merging is finished, the previous checkpoint in D-RAM will be release and the live checkpoint is written to disk as demonstrated in Figure 1b, merging previous checkpoint can cause some pages become invalid and can be re-used i.e. overwritten for the next checkpoint.

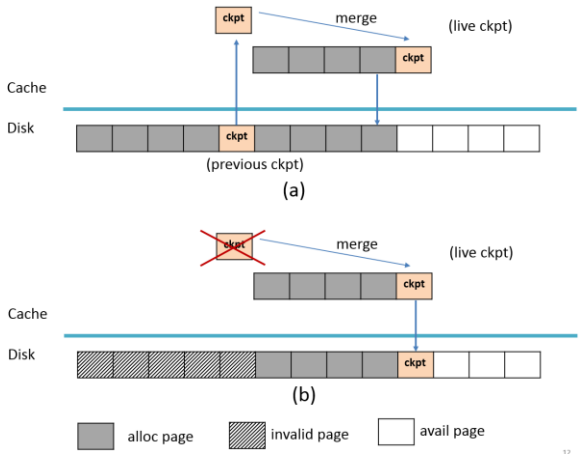

Figure 1 Re-using pages in WiredTiger

3. Evaluation

3.1. Experiment setup

The experiment is setup by running YCSB 0.5 [3] as the client request on top of MongoDB 3.2 with WiredTiger as storage engine. We use YCSB workloada with some modifies that all update requests are used. For evaluation concurrency processing, we set number of threads in YCSB is 40 and using 30 millions of documents and update operations. All configuration on WiredTiger is kept as default. We use Linux kernel 3.19.0 with Ubuntu 14.04.01 distribution. We use XFS file-system and set optimized configuration follow the official recommendation from MongoDB. Experiment is run on a commodity machine with a 48 cores Intel Xeon 2.20GHz processor, 32GB D-RAM with Samsung SSD 850 Pro as the storage device.

Blktrace is used for analysis IO pattern at block level. Due to blktrace does not have knowledge in higher level layer of the DBMS, we modify original MongoDB source code to trace some necessary statistic information in file-system level e.g. page type, or allocation approach using in WiredTiger storage engine.

3.2. Results

For understanding the internal IO pattern, we modified the MongoDB source code to get more information e.g. which type of pages are written (internal page, leaf page, root page) and which allocation approach are used (i.e. first-fit or best-fit) and how frequency a page is written in a specific block offset on physical storage device.

Table 1 Amount of data written

	Amount of data written (MB)	Fraction (%)
Collection	286,749	88.64
Index	0	0
Journal	36,719	11.35
Others	12	0.003

We only interested in write requests, as described in Table 1, almost data written to underlying storage device is come from collection. Because we use all update workload, there is no write in index pages, journal writes has small fraction 11.35% and contribute less the overall system's performance.

Further analysis inside the collection pages, there is more than 99 percentages of collection pages written to storage device is leaf pages that almost cause by eviction dirty pages during the benchmark. The internal pages, root pages and other pages are written to disk mainly when checkpoint is called. Those phenomenon can be explained as the number of leaf pages are majority larger and the LRU replacement policy is used in buffer manager, internal pages and root pages are more frequency called compare to leaf pages so there is less chance those page is evicted.

Figure 2a plots the IO pattern of write operation of leaf pages from file-system level. Almost writes are randomly. At first, the server startup call checkpoint for recovery unfinished works in previous running, some writes occur during this checkpoint. As the requests from YCSB come, pages are fetched from storage device and keep in buffer pool until it reach a

threshold. When the buffer pool reach the threshold, LRU replacement policy is used to evict dirty pages to storage device.

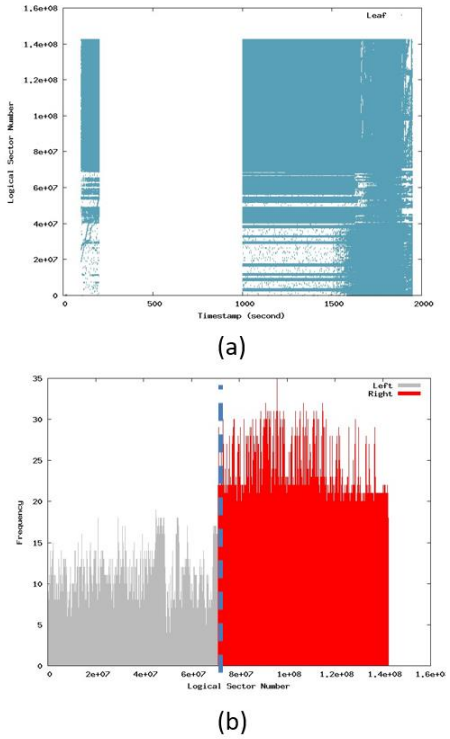

Figure 2 (a) Plotting of leaf pages write in file-system level; (b) frequency of write by logical sector number

There is a gap between the frequency of write of pages as show in both Figure 2a and Figure 2b. The hotter area is in the right hand size of the figure that has 1.75 folds larger than the left hand size.

Figure 3a shows the IO plotted by blktrace in block layer, the pattern is mixed between sequential and randomly. It due to WiredTiger use both its buffer manager and OS cache, write from file system are cached in kernel space and scheduled write to disk by OS. The same asymetric in frequency write offset can be seen in Figure 3b that the hot area has 12.4 folds number of writes larger then the cold area. That is promissing opportunity to apply multi-streamed SSD in WiredTiger, one can use different stream-id for hot area and cold area that lead to reduce the overhead of GC as well as reduce wear-leveling of flash memory.

4. Conclusion and Future Works

Our experiment shows that the majority pages written in WiredTiger is from collection and inside the written collection, almost pages are leaves. There is an unbalance of frequency of page writes based on logical sector number that can lead to an opportunity to adopt multi-streamed SSD technique, different stream-id can be used for hot area and cold area that reduce overhead of GC as well as wear-leveling in flash memory.

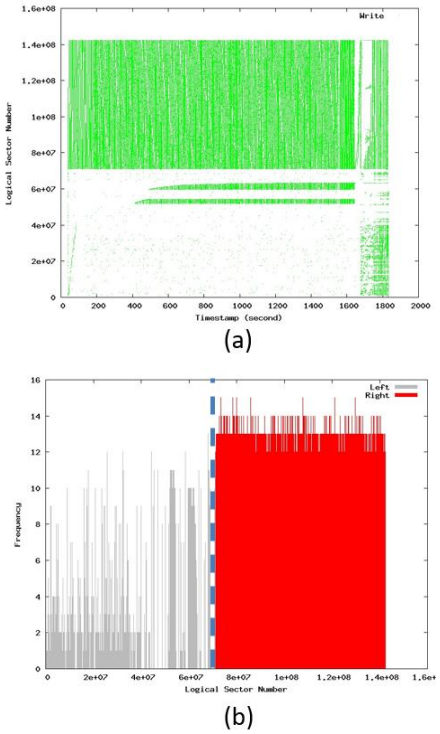

Figure 3 (a) Plotting of collection pages write by blktrace; (b) frequency of write by logical sector number

5. Acknowledgment

This work was supported by Institute for Information & Communications Technology Promotion(IITP) grant funded by the Korea government(MSIP). (R0126-16-1108, NVRam Based High Performance Open Source DBMS development), and by the Technology Innovation Program (10049445, Development of UFS 2.0 controller SoC and embedded SW for mobile storage) funded by the Ministry of Trade, Industry & Energy(MI, Korea).

REFERENCES

[1] Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

[2] WiredTiger, http://source.wiredtiger.com/2.7.0/index.html

[3] Kang, Jeong-Uk, et al. "The multi-streamed solid-state drive." *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*. 2014.

[4] Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB."*Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010.