Allocating Inverted Index into Flash Memory for Search Engines

Bojun Huang
Microsoft Research Asia
Beijing, China
bojhuang@microsoft.com

Zenglin Xia
Microsoft Research Asia
Beijing, China
zlxia@microsoft.com

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Algorithms, Experimentation

Keywords

Flash memory, caching, inverted index

1. INTRODUCTION

Although most large-scale web search engines adopt the standard DRAM-HDD storage hierarchy, the usage of hard disk is greatly limited by its long read latency. On the other hand, NAND Flash memory is 100x faster than hard disk and 10x cheaper than DRAM [2]. Therefore, it's possible to allocate a significant portion of DRAM data into Flash memory, so as to save money on storage.

This paper considers the optimal policy that allocates the DRAM portion of inverted index into Flash memory as much as possible. Note that the original hard disk portion of index data is still left in hard disk in our scheme, which actually results in a three-layer storage hierarchy. To our best knowledge, we are the first to show that it's possible to get substantially better system performance for web index serving by trying some *Flash-aware* storage management approaches, rather than just plugging in a SSD and treating it as super hard disk.

We limit our discussion in the static scenario, where posting lists are allocated *atomically* in either Flash memory or DRAM only when the index updates and no other data movement is performed at run time. The problem is very similar to static index caching/pruning [1] [4], except that the caching here is *exclusive* and the target storage is Flash memory. Note that previous work suggested that static policies work well for inverted index caching, compared with their dynamic counterparts [1].

2. ANALYSIS

In static index allocation problem, a subset of index terms is selected to stay in Flash memory until the index is reallocated. Following the traditional analysis framework for hard disk caching, [1] and [4] modeled it as knapsack problem, and proposed the standard greedy algorithm to approximate the optimal solution. However, the distinct characteristics of Flash memory requires modifications on existing optimization model. Specifically, our analysis considers the problem from following aspects:

Throughput constrained optimization - Throughput and latency are two major constraints for storage management. Traditional disk caching systems concern mostly about latency, this may be partially because the long access latency of hard disk tend to hurt the overall performance first, and partially because the bandwidth of hard disk is instable across different workloads. However, the latency requirement in single index server is less restrictive since we could reduce the latency by splitting index into more partitions and serving them simultaneously. Furthermore, the throughput of Flash read is relatively insensitive to workloads and can be considerably large in its high-performance setup [3]. Thus, we take throughput/bandwidth as the main constraint in our optimization model.

Cache size oriented optimization - Traditional disk caching systems try to minimize cache hit rate given a *fixed* cache size. This paradigm makes sense for latency constrained optimization, because a "consistently good" caching policy within traditional framework *implicitly* minimizes the cache size for a given cache hit rate, and a required cache hit rate can be uniquely translated to a required end-to-end latency. Note that the underlying assumption of this constraint translation is that the access latency of any storage remains the same while its size changes. Unfortunately, similar assumption doesn't hold in throughput/bandwidth constrained optimization, because the bandwidth of Flash memory grows as Flash chips added. Consequently, we have to explicitly minimizes (maximizes) the size of DRAM (Flash memory) while ensuring the throughput in Flash memory to be sustainable.

Finally, the problem can be modeled as, given a set of index terms T where each index term $t \in T$ is associated with a bandwidth requirement B_t and a capacity requirement C_t, also given a capacity-bandwidth function of Flash memory $F_{flash} : R^+ \rightarrow R^+$ which indicates the sustainable bandwidth provided by a specified amount of Flash memory, then the goal is to select a subset $S \subseteq T$ for Flash memory and solve the optimization problem

$$\arg\max_{S \subseteq T} \sum_{t \in S} C_t \text{ , subject to } \sum_{t \in S} B_t < F_{flash}(\sum_{t \in S} C_t) \tag{1}$$

Although F_{flash} depends on specific package setup of the

Copyright is held by the author/owner(s).
WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

Flash memory used, it is convenient to arrange a bunch of Flash chips in single board [3] and to uniformly distribute the access requests (e.g. by stripping), in order to achieve fully parallelized integrated bandwidth. In this case, the integrated bandwidth will increase by a fixed amount for each Flash chip added, and thus is proportional to its capacity. This means F_{flash} can be seen as a linear function $F_{flash}(x) = \alpha x$. Interestingly, it can be proved that in this case the greedy algorithm based on individual ratio $\frac{C_t}{B_t}$ performs at least as good as what it does as the standard heuristic in knapsack problem, by noticing that it is still an ILP problem with the constraint of

$$\sum_{t \in S} (B_t - \alpha C_t) < 0 \qquad (2)$$

The proof is omitted due to space limitation.

3. EXPERIMENTATION

We compare our algorithm with the state-of-art hard disk caching policy, as well as two simplified criteria in the greedy framework. We use a query log containing 0.7 million user queries which are already filtered by query result caching, and an index file covering 7 million web pages, both of which are from real search engine. Dynamic index pruning and adaptive matching algorithms are employed, which means posting lists may be partially scanned for serving a query. To achieve the full potential of Flash memory, we implemented a customized board with 36 Flash chips which has integrated bandwidth of at most 1.8 GB/s (shown in Figure 1). Specifically, the four caching policies we examined are

- **C/B** the Flash-aware policy discussed in last section, which is based on the individual ratio $\frac{C_t}{B_t}$. B_t is estimated from the run-time logs. The run-time overhead for logging is found to be ignorable.
- **QTF/DF** is based on the ratio of term access frequency and document frequency. It's the "optimal" policy in hard disk caching [1] [4].
- **Capacity** is solely based on capacity requirement, which doesn't need any run-time logging.
- **QTF** is solely based on term access frequency. Note that this policy is equivalent to C/B when posting lists are always completely processed.

Figure 2 illustrates our model, as well as the results of different policies, in a more intuitive way. For each policy, the index terms are sorted according to its respective criterion, thus the accumulative capacity requirement and accumulative bandwidth requirement of index terms can be drawn as a curve. All index terms on the left side of the intersection point between the curve and F_{flash} will be allocated into Flash memory, and those on the right side will be in DRAM. To maximize Flash size, the intersection point should go right as much as possible.

C/B does the best job and put more than 95% of the index data into Flash memory for system with 1GB Flash chip array (85% and 75% for 2GB chip and 4GB chip respectively). QTF, as a previously sub-optimal policy [1], has a similar curve with C/B and gives comparable allocation result for system with 4GB Flash chips. However, for 1GB Flash chips, the DRAM size of QTF is roughly 100% larger than

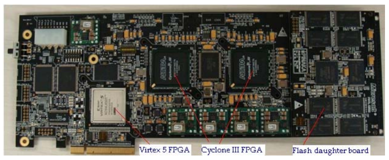

Figure 1: The customized board with 36 Flash chips

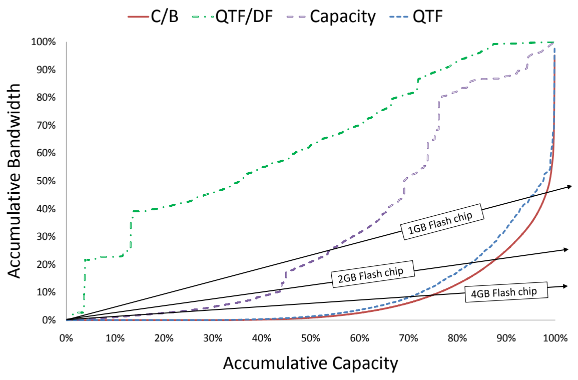

Figure 2: Accumulative bandwidth-capacity distributions of different allocation policies

that of C/B, because the intersection point in this case falls into the most different parts of these two curves. QTF/DF results in a nearly straight line, implying that it almost acts like randomly picking index terms into Flash memory, and performs even worse at the starting end of the curve (due to *stopwords* actually). Because the curve of QTF/DF is consistently higher than all the F_{flash} lines, almost all index terms have to be allocated into DRAM under this policy.

4. CONCLUSION

Experimental evidence shows that the previously optimal policy for hard disk doesn't work for Flash memory. The Flash-aware optimization should explicitly maximize the Flash size, under the constraint of memory throughput. When the integrated bandwidth of Flash memory grows linearly, the classic greedy algorithm for knapsack is still "good", and in our experiments it is able to allocate at most 95% of index data into Flash memory. Meanwhile, a previously sub-optimal policy becomes close-to-optimal in certain situations.

5. REFERENCES

[1] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *SIGIR*, 2007.

[2] A. M. Caulfield, L. M. Grupp, and G. Ganger. Gordon: using flash memory to build fast power-efficient clusters for data-intensive applications. In *ASPLOS*, 2009.

[3] Fusion-IO. http://www.fusionio.com/.

[4] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *SIGIR*, 2007.