Impact of Storage Technology on Cluster-Based High-Dimensional Indexing

Gylfi Þór Guðmundsson, Laurent Amsaleg, Björn Þór Jónsson

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Impact of Storage Technology on Cluster-Based High-Dimensional Indexing

Gylfi Þór Guðmundsson — Laurent Amsaleg — Björn Þór Jónsson

N° 7681

Juillet 2011

ISSN 0249-6399 ISRN INRIA/RR--7681--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Impact of Storage Technology on Cluster-Based High-Dimensional Indexing

Gylfi Þór Guðmundsson , Laurent Amsaleg , Björn Þór Jónsson*

Thème :
Équipes-Projets Texmex

Rapport de recherche n° 7681 — Juillet 2011 — 10 pages

Abstract: The scale of multimedia data collections is expanding at a very fast rate. In order to cope with this growth, the high-dimensional indexing methods used for content-based multimedia retrieval must adapt gracefully to secondary storage. Recent progress in storage technology, however, means that algorithm designers must now cope with a spectrum of secondary storage solutions, ranging from traditional magnetic hard drives to state-of-the-art solid state disks. This paper studies the impact of storage technology on a simple, prototypical high-dimensional indexing method for large scale query processing. We show that while the algorithm implementation deeply impacts the performance of the indexing method, the setup of the underlying storage technology is equally important.

Key-words: disques magnétiques, disques Flash, indexation multidimensionnelle, clustering

* School of Computer Science, Reykjavík University, Menntavegi 1, IS 101 Reykjavík, Iceland. bjorn@ru.is

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Impact des technologies de stockage sur les algorithmes d'indexation multidimensionnels basés clustering

Résumé : Ce rapport analyse l'impact de diverses technologies de stockage secondaire sur le comportement des techniques d'indexation multidimensionnelles.

Mots-clés : magnetic disks, solid state disks, SSD, high dimensional, clustering

1 Introduction

Multimedia data collections are now extremely large and algorithms performing content-based retrieval must deal with secondary storage. Magnetic disks have been around for decades, but their performance, aside from capacity, has not improved significantly. Higher performance, more reliable storage systems exist, however, by grouping many disks together and striping data as in the Redundant Array of Independent Disks approach. Recently, Solid-State Disks (SSDs) have emerged as a disruptive storage technique based on memory cells on chips. Their storage capacity grows quickly and they outperform magnetic approaches. It is therefore of high interest to study what impact secondary storage technologies can have on the design and performance of high-dimensional indexing algorithms that are a core component of all content-based retrieval approaches. This paper is an initial investigation in that direction.

Contribution: We have created two very different implementations of a rather simple, yet very effective, high-dimensional indexing strategy based on clustering. These implementations differ in the way they access secondary storage, emphasizing small vs. large I/Os or random vs. sequential I/Os. We then ran these implementations on a machine connected to various magnetic storage solutions as well as various SSDs solutions and accurately logged their respective performance.

From this analysis, we show that a good understanding of the underlying hardware is required at design time to get optimal performance. In particular, as the devil is in the details, we show that the theoretical behavior of any storage device much differs from what is observed on the battlefield and that even the members of a presumably homogeneous family of storage solutions may behave quite differently. We also show that balancing efforts on software versus hardware is not always obvious: it is key to clearly evaluate the cost of paying a very skilled programmer to nicely tune and debug a complex piece of code versus producing a naive implementation and throwing at it efficient hardware.

The Case for Reality : This investigation is done using real algorithms working on real data with standard production hardware. Simply working with various technical specifications does not accurately reflect the complex layers of interactions between application, operating system and I/O devices (disks, network) while processing real data.

We implemented a high-dimensional cluster-based indexing algorithm that is prototypical of many approaches published in the literature. The algorithm is approximate in order to cope with truly large high-dimensional collections. It has an initial off-line phase that builds an index. This is a very demanding process: the entire raw data collection is read from disk and vectors that are close in the feature space are clustered together and then written back to disk. This process is essentially I/O bound.

The search phase is quite different, but it is also prototypical of what has been published. From a query submitted by a user, few candidate clusters are identified using the index, fetched from disks, and then the CPU is heavily used for scanning the clusters in search of vectors.

We also use a real data collection made of more than 110 million SIFT descriptors computed on real images randomly downloaded from Flickr. This

dataset is clearly not IID; it has some very dense areas, while some others that are very sparse, which all together strongly challenge the indexing and retrieval algorithms. Observing the behavior of indexing and searching real data gives more insight on the impact of the various storage solutions.

Overview of the Paper: This paper is structured as follows. Section 2 briefly reviews the state of art in secondary storage techniques. Section 3 gives an overview of the high dimensional indexing algorithm we use in this paper while section 4 details the two implementations which stress differently the underlying hardware. Section 5 describes the experiments we performed. Section 6 concludes the paper.

2 Secondary Storage Review

We now review the state-of-the-art in storage technology.

Magnetic Disks: Magnetic disks are the standard for cheap secondary storage. During the last decade, mostly their capacity that has dramatically increased. Their read/write performance improved little as their mechanical parts are inherently slow to move. This impacts the performance of small and random operations which are significantly slower then large sequential ones. Accessing data that is contiguous on disk is key to performance. Note that sophisticated embedded software in the disk controler tries to minimize the costs of reading and writing (e.g., reordering accesses, enforcing contiguity, writing asynchronously, caching) but programmers have typically no control over these decisions.

Solid-State Disks: SSD technology is based on flash memory chips. Two types of memory cells are used: single-level chips, fast and durable, and multi-level chips, not as good, but cheaper. The memory modules are typically arranged in 128KB blocks. Since there are no slow mechanical parts, reading from SSDs is extremely fast (about 220MB/s) and sequential or random reads are equally fast. In contrast, writing is more costly as it sometimes requires a special erase operation done at the level of an entire 128KB block. The cost of writes is therefore not uniform, and this is typically unpredictable from a programmer's point of view. In addition to minimizing write costs, internal controlling algorithms do wear-levelling to extend the life span of the chips. The performance of SSDs has extensively been studied in [2].

Network Attached Storage: A NAS is typically a quite large secondary storage solution made available over a network. They usually contain an array of disks, operating in parallel thanks to advanced RAID controllers, and made available through a network connected file-servers or dedicated hardware directly connected to the network. The performance of the NAS can be very hard to evaluate as there are many layers of hardware, caching and communication, each with its own bottlenecks. Often the network links between the server and the clients limit the throughput as the links are typically shared by many clients.

Interactions with the OS: Many sophisticated routines trying to reduce the costs of accessing secondary storage exist in all operating systems. Prefetching is a common technique: instead of reading few bytes at a time from the disk,

more data than is asked for is brought into RAM, hoping the data needed later will already be in memory, thus avoiding accessing the disk again. Reads are blocking operations and the requesting process can only be resumed once the data is there. Writes might be handled asynchronously as there is no need to wait for the data to effectively reach the disk. Overall, the operating system uses buffers for I/Os and fills or flushes them when it thinks it is best, trying to overlap as much as possible the I/O and CPU loads. If reads or writes are requested too rapidly, then there is little overlapping, and performance degrades.

3 Extended Cluster Pruning

The extended version [4] of the Cluster Pruning algorithm [3] is quite representative of the core principles underpinning many of the quantification-based high-dimensional indexing algorithms that perform very well [8, 7, 5]. Overall, the extended Cluster Pruning algorithm (eCP) is very much related to the well-known k-means approach. As k-means, eCP is an unstructured quantifier, thus coping quite well with the true distribution of data in the high-dimensional space [6]. It has two phases, which work as follows.

Index Construction: eCP starts by randomly selecting C points from the data collection. They are used as representatives of the C clusters eCP will eventually build. C is typically chosen such that the average cluster size fits within one disk I/O, which is key to performance. Then, the remaining points from the data collections are read, one after the other, and assigned to the closest cluster representative (creating Voronoi cells). When the data collection is large, the representatives are organised in a multi-level hierarchy. This accelerates the assignment step as finding the representative that is closest to a point then has logarithmic complexity (instead of linear complexity). Eventually, once all the raw collection has been processed, then eCP has created C clusters stored sequentially on disks as well as a tree of representatives, also kept on disks.

Searching the Index: When searching, the query point is compared to the nodes in the tree structure to eventually find the closest cluster representative. Then, the corresponding cluster is accessed from the disk, fetched into memory, and the distances between the query point and all the points in that cluster are computed to get the k-nearest neighbors. The search is approximate as some of the true nearest neighbors may be outside the Voronoi cell under scrutiny. Experience from different application domains has shown that the quality results of eCP can be improved when searching in more than one cell because this returns better neighbors. In this case, however, more cells must be fetched from the disks and more distance computations performed.

4 Index Creation Policies

We have designed two implementations of the eCP index creation that have quite different access patterns to secondary storage. They differ during their *assignment phase*, when performing the assignment of vectors to their clusters, and also during their *merging phase*, when forming the final file that is used

Disk	Ave. seek time	Ave. rot. latency	Cache size	Meas. Seq. R/W MB/s
Seagate	11.0ms	4.16ms	8 MB	46/40
Fujitsu	11.5ms	4.17ms	16 MB	68/53
STalent	<1 ms	-	Unkn.	124/34
Intel	<1 ms	-	16 MB	220/66

Table 1: Key figures for the different storage devices used, both specified and observed values

during the subsequent searches. We have not changed at all the search process of eCP.

Both index creation policies start by building their in-memory index tree by picking leaders from the raw collection. Then, they allocate a buffer, called *in-buff*, for reading the raw data collection in large pieces. They then iterate through the raw collection via this buffer, filling it with many not-yet-indexed vectors. The index is used to quickly identify the leader that is the closest to each vector in *in-buff*, respresenting the cluster each vector must be assigned to. Once all vectors in *in-buff* have been processed, one of the two policies described below is used to process the buffer.

Policy 1: TempFiles (`TF`): This first policy uses temporary files, one for each cluster, that are stored on a disk. One cluster file contains all the vectors assigned so far to that cluster. When called, the `TF` policy loops on the leaders identified for the vectors in *in-buff*. When processing one such leader, its associated cluster file is opened and the appropriate vectors from *in-buff* appended to it, before it gets closed. When all vectors from *in-buff* have been processed, then a new large piece from the raw collection is read into *in-buff*, and eCP loops. After having assigned all vectors from the collection, all these temporary files are then concatenated into a unique file.

In terms of access patterns, `TF` performs, at cluster assignment time, fairly large sequential reads to fill *in-buff* with new vectors as well as many rather small random writes, one per cluster, every time all the vectors in *in-buff* have been processed. When creating the final file, it also performs cluster-sized sequential reads (one per cluster, typically 128KB, see section 5) as well as large sequential writes for the final file.

Policy 2: ChunkFiles (`CF`): This second policy generally follows a sort-merge principle. When called, `CF` sorts *in-buff* on increasing values of the leader identifiers. It then creates a new chunk file on disk, and flushes *in-buff* into that chunk file before closing it. It then reads another large piece from the raw collection into *in-buff* and loops. After having processed all vectors from the raw collection, `CF` merges all chunk files into a unique file.

In terms of access patterns, `CF` performs, at cluster assignment time, large sequential reads (typically 128MB) to fill *in-buff* and large sequential writes when creating each chunk file. When creating the final file, it performs many small random reads to get data from all chunk files and large sequential writes for the final file.

	Total Time I/O+CPU (s)		Assignment Time (s)		Merging Time (s)	
Disk	TF	CF	TF	CF	TF	CF
Seagate	43,144	12,949	12,556	548	18,829	1,299
Fujitsu	32,895	12,689	9,975	388	11,145	1,207
STalent	32,540	11,528	17,120	149	3,529	236
Intel	14,164	11,398	2,028	46	402	244
NAS	22,335	14,564	5,314	610	2,349	202

Table 2: Performance of eCP index creation policies, single drive setups with different storage devices

5 Experiments

To evaluate the performance of TF and CF, we ran extensive measurements using a collection of more than 110 million SIFT descriptors of 128 dimensions extracted from 100,000 images randomly downloaded from Flickr. This collection is about 14.5GB. We reused the parameters that the authors of [4] found to work best, i.e., the depth of the index was 3, and the size of leafs was 128KB, resulting in 111,424 clusters stored on disk. Note that clusters are not equally filled as the true distribution of vectors in space is not balanced (30% of the clusters are smaller than 64KB, while 21% are larger than 192KB). In all experiments the size of *in-buff*, and thus each chunk file, is 128MB. Note that this is much larger than the cluster size. We use an ext3 file system, a Dell Precision T3400, 3GHz Intel E6400 CPU with 6MB cache and 4GB RAM.

We tested two types of magnetic devices: 3.4" Seagate Barracuda 7200.10 and 2.5" Fujitsu MHZ2160BJ. Both are 7200 rpm disks with similar seek time and rotational latency. We also used two types of SSDs: SuperTalent FTM28GL25H and Intel X-25M, type SSDSA2MH080G1GC. We also used a NAS 3070 from NetApp. Table 1 provides more details. The three first columns are filled using vendor figures. The last column shows the sustained sequential read and write performance we observed on our system. Accurately measuring the performance of the NAS is much more complicated.

We than ran two families of experiments. The first family uses measurements performed using a single drive: the file containing the raw collection, the temporary files/chunk files, as well as the final cluster file are all stored on a single disk. In this case some reads and writes overlap in time and compete for the disk. This causes slower performance as enforcing truly sequential accesses is much more difficult.

The second family uses measurements performed using two drives. In this case, the raw collection is kept on one drive and the temporary files/chunk files are stored on another physical device, eliminating any competition between reads and writes at assignment time. Similarly, the final file and thetemporary files/chunk files are stored on physically different devices; it is sufficient to put that final file back on the first drive where the raw collection is. Again we have eliminated any competition, this time for merging. We now detail the performance measurements for these two families.

Single Drive Experiment: Table 2 shows the performance measurements when using the single drive setup. The total time (this is wall clock time) includes the time for I/Os as well as for executing the many distance calculations

Two Drive Setup	Total Time I/O+CPU (s)		Assignment Time (s)		Merging Time (s)	
	TF	CF	TF	CF	TF	CF
F-I-F	13,467	11,640	1,977	370	208	220
I-I'-I	13,484	11,301	1,666	67	180	188

Table 3: Performance of eCP index creation policies, two drive setups with different storage devices

on the CPU. The CPU usage is almost identical for both TF and CF and equal to 11,000 seconds on average, divided into 10,930 seconds for assignment and 70 seconds for merging. The second and third columns show the overall time it takes to perform the assignment of vectors to leaders and the final merging. These times include the time spent on I/Os but exclude the almost constant CPU costs.

Overall, focusing on the total time, regardless of the storage device used, the first key observation is that CF always outperforms TF. The TF policy repeatedly opens, writes to, and then closes clusters, forcing the OS to flush data on disks, doing synchronized blocking writes. TF accesses many relatively small files (111,424 files of 128KB), in contrast to CF which accesses fewer but larger files (109 files of 128MB). The performance of TF differs much from CF with magnetic storage devices as many arm movements are done. Interestingly, for TF, the SuperTalent SSD performs poorly—unfortunately, not all SSDs are equal, as reported in [2]. In contrast, the Intel SSD completely outperforms all other type of storage solutions, showing it handles random reads and writes very well, as expected.

Turning to the assignment phase, Table 2 shows that CF spends very little time waiting for I/O. Furthermore, during that phase, we observed much overlapping between the CPU computations and the disk requests thanks to OS and device optimizations which keep the processor (usefully) busy while waiting for I/O completion. This explains the very small times for CF, in particular when using the Intel SSD which proves to handle competing reads and writes very well. With CF, the performance of the assignment phase is CPU bounded, suggesting parallelism is worth looking at.

The merging phase for TF is costly due to the multitude of (relatively) small file accesses compared to CF. Merging for CF also greatly benefit from the prefetching done by the OS: the few large files are brought in memory before the data gets processed, lowering I/O times. Prefetching is less profitable for merging with TF as many small files are involved.

Two Drive Experiment: By using separate physical disks for the reading and writing, any potential competition for the disk is eliminated. We observed that the larger costs are when writing the assigned vectors to disk and then reading them back as in both cases many random accesses are performed. Using an SSD is ideal to speed up this part of the indexing. We therefore defined two setups: First, we use an Intel SSD disk for the intermediate files but input and final output is on the Fujitsu (the magnetic disk with the best observed performance); this setup is called F-I-F. The second setup uses two identical Intel SSDs; this setup is called I-I'-I. The performance measurements for these setups are reported in Table 3.

Observing the total time values, using the SSD for costly random operations provides dramatic performance improvements, regardless of the type of the other device . With the F-I-F setup, the Intel 66MB/s write speed closely matches the Fujitsu 68MB/s read capacity. Replacing this magnetic device with an SSD does not help much as their total times are very similar. One key lesson is that it is not necessary to put SSDs everywhere, which could be terribly expensive, but to use them solely where random accesses are massively needed. This greatly reduces costs, both in terms of performance and money. Note that it is the CPU cost that dominates the time for `CF`, with I/Os being relatively cheap. The `TF` policy, however, suffers again from the multitude of small files.

Turning to assignment, `CF` again outperforms `TF` since it is dealing with a multitude of small files with blocking write accesses. `CF` with F-I-F is limited by the time it takes to read the data from the magnetic source: it has a lot of CPU to do once the *in-buff* buffer gets filled and the disk is not accessed again for some time, long enough to have the disk entering a power-saving mode, typically reducing its rotational speed. This, in turn, increases the cost of the next data request. The I-I'-I setup has no such problems and its performance is extremely good. It turns out to be slightly above what was observed in Table 2; the reasons are unclear, but some fluctuations have been observed. Note, however, that 67 seconds are insignificant with respect to the overall time of more than 13 thousand seconds.

The quite small values, for both policies and both setups, during merging show the improvements from the lack of competition between reads and writes as they are directed to different drives. Even SSDs suffer from I/O competition.

Other Results: We have also checked the impact of a larger *in-buff* on the performance, both for the single drive and two drive setups. As expected, enlarging *in-buff* speeds up `TF` as large *in-buff* reduces the number of random writes that are necessary. In contrast, larger writes slow down `CF` because the OS is better at overlapping CPU and I/Os when performing writes in bursts since large writes overwhelm buffers.

6 Conclusion

The type of secondary storage hardware used when doing large scale high-dimensional indexing is key to performance. On the one hand, a carefully crafted implementation can get good performance when using traditional magnetic-based storage solutions. On the other, simpler implementations potentially saving RAM can perform very well when high performance SSDs devices are used, as they cope very well with random accesses. In practice, it is therefore key to evaluate the cost of an extremely sophisticated implementation versus buying efficient storage devices and placing them along the performance-critical paths. SSDs, however, are not the magical answer to all performance problems: their capacity is still limited, their price is so far very high, although this will probably quickly change. More importantly, their observed performance varies tremendously from one model to the other, making it very difficult to determine which is best for some particular application settings. Studies like [2] or [1] are more than necessary.

Acknowledgement

We thank the project Quaero for its financial support.

References

[1] M. Bjørling, L. L. Folgoc, A. Mseddi, P. Bonnet, L. Bouganim, and B. T. Jónsson. Performing sound flash device measurements: Some lessons from uFLIP. In *Proc. SIGMOD*, 2010.

[2] L. Bouganim, B. T. Jónsson, and P. Bonnet. uFLIP: Understanding flash IO patterns. In *Proc. CIDR*, 2009.

[3] F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal. Finding near neighbors through cluster pruning. In *Proc. PODS*, 2007.

[4] G. Gudmundsson, B. T. Jónsson, and L. Amsaleg. A large-scale performance study of cluster-based high-dimensional indexing. In *Proc. ACMMM–Workshop on Very-Large-Scale Multimedia Corpus, Mining and Retrieval*, 2010.

[5] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011. to appear.

[6] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348 – 1358, 2010.

[7] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proc. CVPR*, 2008.

[8] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. ICCV*, 2003.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399