On the Impact of Flash SSDs on Spatial Indexing

Tobias Emrich, Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Marisa Thoma
Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 Munich, Germany
{emrich, graf, kriegel, schubert, thoma}@dbs.ifi.lmu.de

ABSTRACT

Similarity queries are an important query type in multimedia databases. To implement these types of queries, database systems often use spatial index structures like the R*-Tree. However, the majority of performance evaluations for spatial index structures rely on a conventional background storage layer based on conventional hard drives. Since newer devices like solid-state-disks (SSD) have a completely different performance characteristic, it is an interesting question how far existing index structures profit from these modern storage devices. In this paper, we therefore examine the performance behaviour of the R*-Tree on an SSD compared to a conventional hard drive. Testing various influencing factors like system load, dimensionality and page size of the index our evaluation leads to interesting insights into the performance of spatial index structures on modern background storage layers.

1. INTRODUCTION

Similarity search and spatial promixity queries are an important query type in spatial, temporal and multimedia databases. In general, the task is to find all spatially close neighbors to a query object in a database of d-dimensional feature vectors. Example applications for this type of query might be to select all sensors within a 5 mile diameter around some seismic distortion or find the top-k similar songs in an audio database. For processing this type of queries the simplest solution is to scan the complete database. However, for large databases this leads to an enormous overhead in distance computations as well as in I/O. To avoid this overhead, the database community proposed spatial index structures [6, 3] organizing the database in order to avoid comparing the query object to each feature vector in the database. The most prominent spatial index structure is the R-Tree [9] and its extension the R*-Tree [2]. Though many further extension of its principles have been proposed in the last twenty years, the R-Tree is still the most used method in the area. Additionally, it is implemented in several standard database systems like MySQL[1], PostgreSQL via GiST[10] or Oracle[2].

Despite its wide use, the R-Tree or related data structures do not solve the problem of efficient similarity search by guaranteeing a logarithmic processing time for similarity queries. Instead, it only provides an average logarithmic search time. Factors having a negative impact on the search time include high dimensionality and an inappropriate data distribution.

Besides the dimensionality, the hardware underlying the R-Tree plays an important role when determining the performance advantage compared to the sequential scan. A major difference between both methods exists due to the predominant type of I/O-operations employed in both methods. The sequential scan basically reads the complete database, ideally employing a single seek on the disk. Thus, the transfer rate of the device is very important, whereas the seek time and latency are rather negligible. Searching on a hierarchical index like the R-Tree on the other hand is largely determined by random access I/O to single node pages. Thus, the cost of a similarity query on an R-tree is strongly dominated by the number of accessed nodes because accessing the page usually requires much more time than transferring its contents into main memory. Thus, in [4] the authors state that the selectivity of a query in a hierarchical index should be less than 5 % in order to clearly outperform the sequential scan. In this case, considering only the CPU time of both methods would still favor the tree because the amount of distance computations is still several times smaller than for the scan. However, since the search is I/O bound and sequentially reading the complete dataset is faster than reading 5 % of the data with random access operations, the scan still yields a performance advantage. Nevertheless, the threshold of 5 % is subject to various system parameters like latency, seek time and transfer rate of the underlying storage system. Since the performance characteristics of available background storage devices have significantly changed within the last ten years, the current threshold should be considerably different as well.

In this paper, we will therefore examine the real-time performance improvement of the R*-Tree using new hardware, i.e. flash solid-state-drives (SSDs) under various aspects. As a first result of our examination, we will show that simulating the system workload is mandatory in order to observe the influence of different background storages due to cache utilization. We argue that the use of SSDs for storing spatial indexes is quite a realistic architecture because the size

[1]http://dev.mysql.com, R-Tree indexing for 2 dimensions
[2]http://www.oracle.com, Oracle Spatial: 2 to 4 dimensions

Proceedings of the Sixth International Workshop on Data Management on New Hardware (DaMoN 2010), June 7, 2010, Indianapolis, Indiana.
Copyright 2010 ACM 978-1-4503-0189-3/10/06 ...$10.00.

of the available SSDs is now sufficiently large even for very large index structures. Furthermore, datasets in large spatial or multimedia search systems are usually rather static, i.e. change operations do occur considerably less often than queries. Thus, using SSDs is feasible in spite of the limited writing capacity of flash based storage modules. Another important factor is the affordable price of flash SSDs, making the use of dedicated storage devices for indexing rather inexpensive. Finally, the most important reason for using an SSD for spatial indexing is its enormous improvement in access time compared to conventional hard drives. While the transfer rate of a modern hard disk drive (HDD) is quite comparable to an SSD, the access time of the SSD is up to two orders of magnitude faster. Thus, hierarchical index structures should significantly benefit more from this new storage device than scan-based methods. The contributions of this paper are:

- An examination of the query-time behaviour of the R*-Tree depending on the server workload for background storages using an HDD and an SSD.
- A discussion of the implications of the SSD's characteristics on tuning the page size of the R*-Tree.
- A comparison of the effects of increasing dimensionality to the same index run on both storage systems.

The rest of the paper is organized as follows. In Section 2 we briefly discuss related work on using SSDs in databases. Section 3 dicusses the changes in the access path over the last two decades. In Section 4, we formalize our testing environment for the experiments displayed in Section 5. They demonstrate the effect of the hardware advances to the performance of the R*-Tree w.r.t. system workload, page size and dimensionality. Our paper concludes with a summary and ideas for future work.

2. RELATED WORK

Publications on SSDs usually focus on their main weakness: a limited number of write operations which are relatively slow compared to reads. Lee et.al. [11] compared the transaction performance of standard SQL I/O operations on HDDs and SSDs and have found SSDs to be faster. However, the runtime advantage of SSDs decreases with an increasing number of users due to imperfect handling of write operations. As write operations are crucial for the creation and updates of index structures, there has already been research on how to design write structures for indexing on SSDs. In [17, 18] Wu et.al. proposed a method to speed up the construction and maintenance of B-Trees or R-Trees directly using the flash translation layer (FTL). In [12] Li et.al. introduced the FD-Tree, a B^+-Tree derivate of three index layers specialized to the use in flash discs.

In addition, various types of queries on SSDs have been analyzed. For join operations, the writing problem prohibits an effect of the full advantage of fast reads. Thus, [14, 15, 5] have developed methods for fast join processing on flash devices.

Write-independent queries profit stronger from exchanging HDDs by SSDs. In [7] Goetz Graefe tested query runtimes of the B-Tree on SSDs as opposed to HDDs and concluded that SSDs are not only faster but they also induce a lower optimum page size. In [13] Nath and Kansal use a cost model for the automatic adaption of flash-specialized B^+-Tree types and their parametrization (e.g. page sizes) to the varying access costs of different flash devices. It depends on the tree's height, the read and write access cost of the disk and the node's *utility* [8], the logarithm of records within a node.

3. CHANGES IN THE ACCESS PATH

In this section, we want to review the advances in computer architecture having the strongest influence on the performance of spatial index structures. For most types of similarity queries (apart from special applications like similarity joins), it still holds that the performance bottleneck derives from the I/O-operations. Thus, we will focus on changes w.r.t. the access path of the indexed data.

3.1 Caching

Commonly used cost models for estimating the I/O-costs of index structures usually only regard caching strategies implemented directly into the proposed method (like LRU-page buffers). But besides these explicit caching strategies, all methods are implicitly using caching strategies provided by the underlying operating system (OS) (unless explicitly coded differently) in order to avoid time consuming I/O operations.

A typical disc read request proceeds according to the following pattern: The process requests a certain part of a file to be read from disc. This request is directed to the file system driver which first checks the cache manager and virtual memory for the page. If the page is found (cache hit), the data is returned immediately from the cache without needing to start a time consuming I/O operation. On the other hand, if the data could not be found in the cache (cache miss), the disc driver requests the data to be read directly from the device and thus from the next cache layer, the disc cache which is implemented directly on the according device having a size of 8-128 MB (depending on type and manufacturer). If this request also results in a cache miss, the data must finally be read from disc, causing the expected I/O cost of seek- and transfer time of the requested blocks of data.

Another issue is that OS and disc cache managers analyze file access patterns to a certain amount and start reading ahead data into the caches in order to improve access speed for future reads.

3.2 New Storage Media

Thus, the most important part of the access path remains the used storage medium of the data to be indexed. With the rise of SSDs as a new possibility to store and access data, we want to examine the performance characteristics of common HDDs and SSDs in the context of spatial index structures. Traditionally, three parameters are of crucial importance in this scenario:

- Seek Time: The time to find the requested blocks on the medium.
- Latency: The time until the storage medium can access the requested blocks.
- Transfer Rate: The time to transfer the requested blocks to the processor.

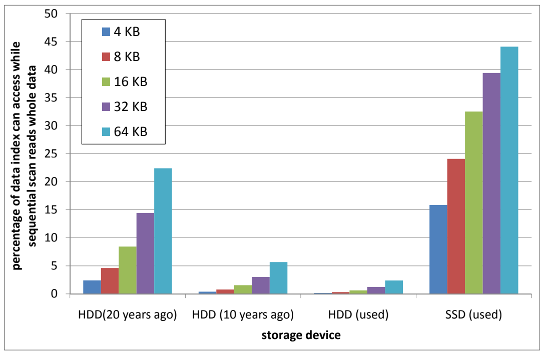

Figure 1: Fraction of data a spatial indexstructure can read before the sequential scan becomes faster

3.2.1 *Hard-Disk-Drives*

Commonly used HDDs store data on rapidly rotating platters with magnetic surfaces. The data is accessed by positioning the head of the right platter and then transfering the data. The seek time is the time required to position the head on the target platter and the correct track, whereas latency is the time passing until the platter is rotated to the position of the sector containing the requested data block. Just like the transfer rate, the average latency is therefore dependent on the rotational speed of the drive. In the last two decades, especially the transfer rate of HDDs increased ($\sim$ factor 40), whereas the seek time and latency only improved little ($\sim$ factor 3).

3.2.2 *SSD*

In contrast to HDDs, flash SSDs do not have any mechanically moving parts but use NAND flash memory chips to store the data. Each flash chip is divided into several flash blocks consisting of several flash pages. Operations on the drive are performed on page level. Due to these characteristics the only latency for a read operation derives from the mapping between logical block adresses and flash pages. This results in access times which are typically two orders of magnitude faster than the ones from HDDs. However, the structural characteristics only remove the seek process and boost the latency time of a disk, thus flash SSDs are not able to outperform HDDs w.r.t. the transfer rate until now.

3.2.3 *Theoretical Implications*

When accessing a large amount of (defragmented) data sequentially, the main parameter of interest is the transfer rate, since seek time and latency only occur once. Thus this parameter has the highest influence on scan-based methods like the sequential scan or the VA-File [16]. On the other hand a hierachical index structure has to acceess the data independently and in a random access manner. In this case, the seek time and the latency are more important for the performance of the index structure. This suggests that hierarchical spatial indexes benefit by far more from the use of a flash SSD than the sequential scan does. To confirm this assumption, in a first step we examine the performance of the sequential scan and spatial index structures from a theoretical point of view. We use the cost model from [4] to calculate the percentage of data which can be accessed by an index structure (via random access) while the sequential scan reads the whole dataset. Like the authors of [4] we assume a storage utilisation of the index of 50 %. Besides the two storage devices used in the experiemental section (summarized in Table 1), we include two HDDs into our calculation. One as it was used 20 years ago (latency + seek time = 20 ms; transfer rate = 5 MB/s) and one as it was used 10 years ago (latency + seek time = 15.3 ms; transfer rate = 32 MB/s). Figure 1 illustrates the fraction of the data a spatial index can maximally access before the sequential scan becomes faster. For each storage device, we compare several page sizes as this has an impact on the break-even point. Let us note that the optimal page size for an index is mainly dependent on the characteristics of the indexed dataset. The figure shows that the advances in the HDD technology of the last decades make it harder for an index structure to perform faster than the sequential scan. This fraction is nowadays far below the commonly assumed 5 % rate (cf. [4]). With the new technology of SSDs this trend is reversed. Following the above considerations, on an SSD a spatial index theoretically still performs better than the scan even if the accessed amount of data is far above 20% for a common page size ($\geq$ 8 KB). Thus, on SSDs a spatial index should outperform the sequential scan even if a large amount of the data has to be visited as it is the case for data which is hard to index (e.g. high dimensional). However, the real-time performance of a spatial query usually depends on several other system characteristics like cache utilization. We will therefore empirically examine the performance advantage of a SSD compared to a modern HDD in the next section.

	HDD	SSD
Avg. seek time	8.9 ms	none
Avg. latency	4.2 ms	0.09 ms
Transfer rate	93.5 MB/s	94.7 MB/s

Table 1: Performance characteristics of used devices

4. TEST BED

4.1 Used Datasets

For the majority of our experiments, we created a random test database. Since tree-based spatial index structures are most challenged by poorly clustering datasets, we chose uniformly distributed datasets. We did not simulate any clustered datasets in order to avoid overfitting of the distributions to the used index. All experiments involve 1 million 10-dimensional feature vectors unless explicitly stated otherwise, i.e. $N = 10^6, d = 10$.

4.2 Hardware

To support our assumptions, we tested several settings on two storage devices, one of each class. Our HDD is a *Western Digital Caviar Blue (WD2500AAJS)* SATA Drive with 8 MB cache, 250 GB memory and 7200 rpm. The SSD is a *Corsair P128 (CMFSSD-128GBG2D)* SATA II drive with 128 MB cache and 128 GB memory. For further specifications see Table 1. All experiments were run on a machine with two Intel Xeon 5160 3.00 GHz Dual-Core processors and 4 GB of main memory.

4.3 Software

We stored the data in a persistent R*-Tree of the ELKI framework [1]. Each accessed node in the tree results in one access to the underlying storage system. Correspondingly, the experiments showing the results of the sequential scan were programmed to access the storage device only once. An important aspect of the following results is that the implemented search system is running on top of an operating system allowing concurrent processes. In our case, we employed *openSUSE 10.3 (X86-64)*. We additionally ran most experiments on Windows XP to test whether a different operating system causes significantly different results. However, the results between both operating systems were quite comparable and thus, we present the results measured on the LINUX system. As mentioned in the previous section, there are multiple caching systems for background storages. Thus, the experiments include the caching mechanism provided by the underlying hardware and operating system. Though using a disk cache is a realistic assumption, having an otherwise idle system is not. If we think of an index as a component of a database server, it is a very unrealistic assumption that the only process currently running is the search itself. A single process experiment on an otherwise idle system leads to an unrealistically large amount of available main memory which the operating system will use for caching parts of the index or the dataset. Furthermore, if the system is only occupied with the test program, it is also quite likely that its caches will be exclusively used for parts of the index. However, assuming the search process as part of a larger server system, the system will require the resources for other processes as well. Considering the sequential scan, it is possible to scan a large data file in a consecutive way if there are no important concurrent processes accessing the disk as well. However, on a real database server, it is rather unlikely that there is no other equally important process or thread requesting to access the disk as well. Thus, in order to provide fair answering times, most systems will interrupt large scans causing multiple disk accesses for the sequential scan as well. Thus, cache utilization, available main memory and concurrent reads will be limited by the system load caused by other database server functions and other processes. A further aspect limiting the resources for a single query is the concurrent processing of several user requests at a time.

To conclude, in order to make sure that the tests are performed under realistic conditions, we have to limit the available resources for answering a query and to simulate a server workload consisting of multiple concurrent threads or processes. To achieve this result, we allocated and locked the available main memory to make sure that the test system only had access to 1 GB of main memory. Furthermore, to simulate concurrent queries being answered at the same time, we multithreaded our test program for answering multiple queries at the same time. The effect of the number of parallel queries at a time will be discussed in the following section. We measured the query performance based on the average answering times of k-nearest neighbor queries where the number of retrieved neighbors k is set to 10.

5. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments testing similarity queries based on a flash SSD and HDD background storage. We start with measuring the utilization of the background storage when scaling the system load. Afterwards, we examine the impact of the changed performance characteristics on tuning the page size of the R*-Tree. In a final set of experiments, we measure the influence under a changed size of k and different dimensionalities of the dataset.

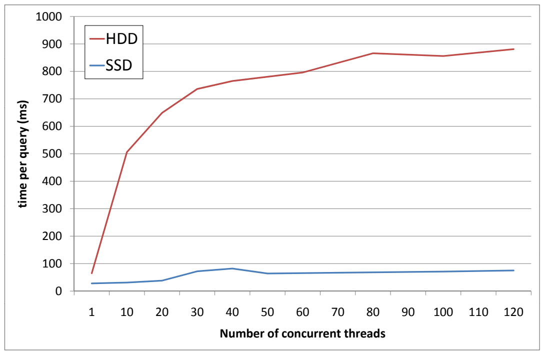

Figure 2: Querytime for an increasing number of parallel queries and databases ($k = 10, d = 10, N = 10^6$, uniformly distributed data)

5.1 System Load and Storage Device Utilization

As mentioned in the previous section, we expect the workload of the database server to have an important influence on the effect of the background storage and thus the query times. If the system load is rather low, we can expect the database server to spend more resources like caches and main memory to answering the query. Thus, the performance of the background storage should have a smaller impact. To simulate different levels of work load, we posed 5 000 queries to an R*-Tree comparing both devices while changing the number of concurrent threads processing the queries. Each R*-Tree stores one million 10-dimensional feature vectors. Let us note that we assume that each thread has its own instance of the R*-Tree simulating queries on different index structures. The results can be seen in Figure 2.

While the performance advantage of the SSD-based tree is rather small for a single query thread, the gap between both devices rapidly increases with the number of simultaneous threads. The rather small difference for a limited number of threads can be explained due to the good cache utilization for a small level of concurrency. However, with an increasing number of concurrent threads, the amount of cache misses for a dedicated thread will raise. Thus, the effect of the cache is strongly decreasing and for a number of 100 threads the impact of the storage device can be clearly observed. For more than 100 concurrent threads, the average answering time of a query is about one order of magnitude faster on the SSD than on the HDD. For the tests w.r.t. data dimensionality and the number of retreived nearest neighbours k we therefore used 100 concurrent thread.

5.2 Impact on the Page Size

In this section, we discuss the impact of the changed performance characteristics of the background storage on the parametrization of the R*-Tree. Selecting a suitable page size can have a large impact on the query performance of a spatial index structure. The impact of the page size can be explained as follows. A large page size reduces the over-

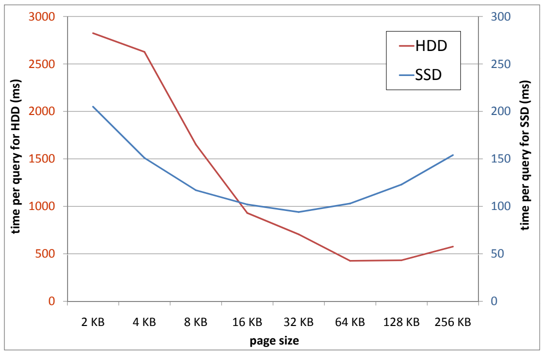

Figure 3: Querytime for different page sizes on HDD and SSD ($k = 10, d = 10, N = 10^6$, uniformly distributed)

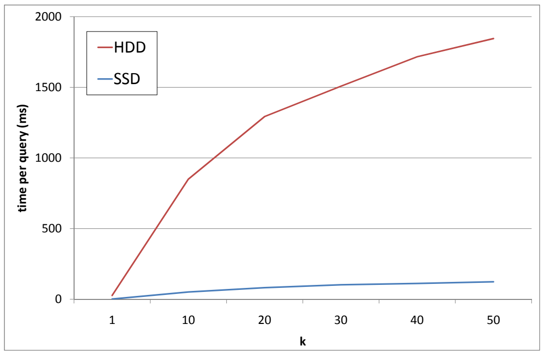

Figure 4: Querytime on for increasing k on HDD and SSD ($d = 10, N = 10^6$, uniformly distributed)

head of accessing a page on the background storage compared to the transfer time of the page content. In the extreme case the page size is large enough to keep the complete dataset and thus, the R*-Tree degenerates into the sequential scan on the root page. From a CPU-time point of view, small pages are usually more beneficial because their spatial approximations usually have a smaller spatial extension. Therefore, it is less likely that they intersect with the query region. In combination with the smaller amount of stored data objects this leads to a decreased number of distance computations.

In the following, we want to examine the optimal page size for the R*-tree based on the flash SSD compared to the HDD. We generated R*-trees for page sizes varying from 2 kB to 256 kB. Each tree contains the same dataset of one million 10-dimensional, uniformly distributed feature vectors. To test the query performance, 480 10-nearest-neighbor queries were simultaneously performed by 30 concurrent threads for both storage devices. The measured average processing time per query can be seen in Figure 3.

On the HDD the results indicate that the pages should be chosen considerably larger than the 4 kB disk pages of the underlying file system. We achieved the best results with a page size of 64 kB closely followed by 128 kB pages. Due to the significantly shorter access times of the flash SSD, its optimal page size should be smaller than for the hard drive. We measured the best results when using a page size of 32 kB on the flash SSD. Due to the comparatively large

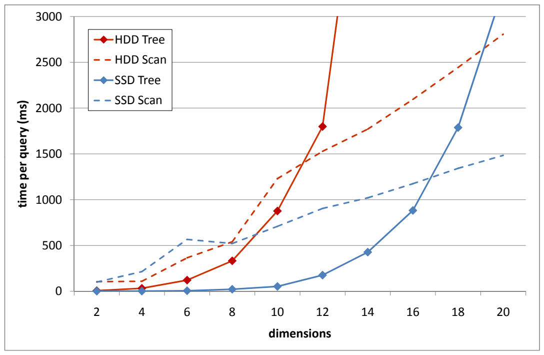

Figure 5: Performance of R*-Tree and sequential scan with increasing dimensionality on HDD and SSD ($k = 10, N = 10^6$, uniformly distributed)

dimension of our test data, this amounts to a considerably smaller discrepancy between HDD and SSD than an earlier study on B-Trees (256 vs. 2 kB for only 1 dimension) [7]. However, in general the performance of the SSD indicates no strong decrease in performance for 8 to 128 kB. This is quite remarkable because the performance of the R*-Tree seems to be a lot less dependent on a suitable page size on the SSD than on the hard drive. For example, the query time of $\sim$2 700 ms on the hard drive based system with a page size of 4 kB is more than five times higher compared to the tree having a page size of 64 kB ($\sim$500 ms). In contrast, the runtimes of the observed optimal page size of 32 kB and the worst page size of 2 kB on the SSD only differ by the factor two. Thus, due to the fast access times of the SSD, its performance is more robust towards bad selection of the page size. A further interesting observation is that the page size for the SSD is still indicating a rather different behaviour than for a purely memory-based tree. Although the access time of the SSD is significantly lower than for the hard drive, it is still considerably higher than accessing data in the main memory.

5.3 Query Size

In the next experiment, we tested the impact of different query sizes on the runtime of the spatial index on the two different devices. Therefore we measured the runtime for different parameters k. Figure 4 shows the results. Clearly, the total difference in runtime increases for a higher value of k. Interestingly, the query time of the index on the SSD is always an order of magnitude faster than the one on the HDD.

5.4 Dimensionality

One of the most interesting questions for spatial index structures is: How many dimensions can be indexed by the structure before the sequential scan is faster? Generally, with increasing dimensionality of a dataset, the pages of an R*-Tree overlap more and more which leads to a higher percentage of pages which have to be visited at query time. Section 3.2.3 already gave an indication, that R*-Trees can still perform better than the sequential scan on SSDs, even for a large ratio of read pages. To test this hypothesis we performed queries on uniformly distributed datasets with increasing dimensionality and measured the query time for the sequential scan and the R*-Tree on HDD and SSD. Figure 5 shows the results. As expected, the query time using a sequential scan increases roughly linearly (due to increasing data volume), with faster runtimes on the SSD. Since the transfer rate of both media is comparable, this effect is probably caused by interruptions of the sequential scan, resulting in new seeks, which can be performed much faster on the SSD. Comparing the performance of the R*-Tree on the two devices shows that on the SSD the index is around one order of magnitude faster than on the HDD, regardless of the dimensionality. This is caused by the lower seek and latency times of SSDs which play a central role in the performance of spatial index structures. We conclude our observations with a comparison of the query times of the R*-Tree and the sequential scan on each device. On the HDD, the scan outperforms the index at a dimensionality of about 11. On the SSD, this break-even point occurs later at approximately 17 dimensions. This result confirms the theoretical assumptions from the previous sections and shows that spatial index structures can greatly benefit from the use of SSDs.

6. CONCLUSIONS

In this paper, we examined the impact of flash SSDs on spatial index structures. Due to the fast page accesses, SSDs should largely improve the query times for similarity queries. Therefore, we examine the impact of SSDs of the performance of an R*-Trees. An important aspect we observed is that the impact of the storage device is rather small if the experiments are run on an idle system. However, simulating a server workload, we do observe a strong influence of the storage device. Our screenings for optimal page sizes indicated a lower sensitivity of the SSD to improperly-chosen page sizes. Furthermore, in our experiments we observe that the trade-off w.r.t. the dimensionality between index and scan is about 6 dimensions higher for the SSD. For future work, we plan to examine the use of various different page sizes in one spatial index structure in order to better utilize the characteristics of SSDs.

Acknowledgements

This research has been supported in part by the THESEUS program in the MEDICO and CTC projects. They are funded by the German Federal Ministry of Economics and Technology under the grant number 01MQ07020. The responsibility for this publication lies with the authors.

7. REFERENCES

[1] E. Achtert, T. Bernecker, H.-P. Kriegel, E. Schubert, and A. Zimek. ELKI in time: ELKI 0.2 for the performance evaluation of distance measures for time series. In *Proc. SSTD*, 2009.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

[3] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM CSUR*, 33(3), 2001.

[4] C. Böhm and H.-P. Kriegel. Dynamically optimizing high-dimensional index structures. In *Proc. EDBT*, 2000.

[5] J. Do and J. M. Patel. Join processing for flash SSDs: remembering past lessons. In *Proc. DaMoN*, pages 1–8, 2009.

[6] V. Gaede and O. Günther. Multidimensional access methods. *ACM CSUR*, 30(2):170–231, 1998.

[7] G. Graefe. The five-minute rule twenty years later, and how flash memory changes the rules. In *Proc. DaMoN*, pages 1–9, 2007.

[8] J. Gray and G. Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Rec.*, 26(4):63–68, 1997.

[9] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57, 1984.

[10] M. Kornacker. High-performance extensible indexing. In *Proc. VLDB*, 1999.

[11] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A case for flash memory SSD in enterprise database applications. In *Proc. SIGMOD*, pages 1075–1086, New York, NY, USA, 2008. ACM.

[12] Y. Li, B. He, Q. Luo, and K. Yi. Tree indexing on flash disks. In *Proc. ICDE*, pages 1303–1306, 2009.

[13] S. Nath and A. Kansal. FlashDB: dynamic self-tuning database for NAND flash. In *Proc. IPSN*, pages 410–419, 2007.

[14] M. A. Shah, S. Harizopoulos, J. L. Wiener, and G. Graefe. Fast scans and joins using flash drives. In *Proc. DaMoN*, pages 17–24, 2008.

[15] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query processing techniques for solid state drives. In *Proc. SIGMOD*, pages 59–72, 2009.

[16] R. Weber, H. J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. VLDB*, pages 194–205, 1998.

[17] C.-H. Wu, L.-P. Chang, and T.-W. Kuo. An efficient B-Tree layer for flash-memory storage systems. In *Proc. RTCSA*, pages 17–24, 2003.

[18] C.-H. Wu, L.-P. Chang, and T.-W. Kuo. An efficient R-Tree implementation over flash-memory storage systems. In *Proc. ACM GIS*, pages 17–24, 2003.