An Experimental Analysis of Spatial Indexing on Hard Disk Drives and Flash-based Solid State Drives

Anderson Chaves Carniel[1], Ricardo Rodrigues Ciferri[2], Cristina Dutra de Aguiar Ciferri[1]

[1] University of São Paulo, Brazil
accarniel@gmail.com, cdac@icmc.usp.br
[2] Federal University of São Carlos, Brazil
ricardo@dc.ufscar.br

Abstract. Spatial databases and Geographic Information Systems frequently employ *disk-based spatial indices* like the R-tree and the R*-tree to speed up the processing of spatial queries, such as *spatial range queries*. These indices are originally designed for *Hard Disk Drives* (HDDs) and thus, they take into account the slow mechanical access and the cost of search and rotational delay of magnetic disks. On the other hand, *flash-based Solid State Drives* (SSDs) have widely been adopted in local data centers, and cloud data centers like the Microsoft Azure environment. Because of intrinsics characteristics of SSDs like the erase-before-update property and the asymmetric costs between reads and writes, the impact of spatial indexing on SSDs needs to be studied. In this paper, we conduct an experimental evaluation in order to analyze the performance relation of spatial indexing on HDDs and SSDs. For this purpose, we run our experiments in a local server equipped with an HDD and an SSD, as well as in virtual machines with HDDs and SSDs allocated in the Microsoft Azure. As a result, we show experimentally that spatial indices originally designed for HDDs should be redesigned for SSDs in order to take into account the intrinsic characteristics of SSDs. This means that, a spatial index that showed a good performance in an HDD do not showed the same good performance in an SSD.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Spatial databases and GIS

Keywords: benchmarking, flash memory, spatial database, spatial indexing

1. INTRODUCTION

Several advanced applications like agriculture systems, urban planning, and public transportation planning make use of geometric or spatial information in order to represent spatial phenomena. Thus, these applications commonly require specialized systems to manage, store, and retrieve spatial phenomena. *Spatial database systems* and *Geographic Information Systems* (GIS) fulfill these requirements allowing the processing of *spatial queries* on spatial objects [Güting 1994]. For instance, a spatial selection that finds all the roads intersecting the Sao Paulo state. Another example is a spatial range query that returns all the buildings overlapping a given rectangular object, called query window. Due to the expensive processing of topological predicates (e.g., intersects, overlap), spatial database systems and GIS widely employ *spatial indices* to speed up the processing of spatial queries [Gaede and Günther 1998]. Frequently, hierarchical indices like the R-tree [Guttman 1984] and the R*-tree [Beckmann et al. 1990] are employed.

In general, these indices are designed to be handled in *Hard Disk Drives* (HDD) and thus, they take into account the slow mechanical access and the cost of search and rotational delay of magnetic disks. We term them as *disk-based spatial indices*. On the other hand, *flash-based Solid State Drives* (SSDs) have widely been used in many applications [Kryder and Kim 2009; Mittal and Vetter 2016] because

Corresponding author: Anderson Chaves Carniel. Department of Computer Science, University of São Paulo, 13.560-970, São Carlos, SP, Brazil. Phone number (+55) 16-3373 9677. Email: accarniel@gmail.com.

of many positive characteristics compared to HDDs, such as (i) smaller size, (ii) lighter weight, (iii) lower power consumption, (iv) better shock resistance, and (v) faster reads and writes. As a result, SSDs are being used as the main storage device in local data centers and in cloud data centers, such as the Azure Premium Storage of the Microsoft Azure environment[1].

However, SSDs have intrinsic characteristics [Chen et al. 2009; Jung and Kandemir 2013; Mittal and Vetter 2016] that may affect the performance of many spatial database applications. The main intrinsic characteristic is the asymmetric cost between reads and writes, where a write requires more time and power consumption than a read. Another characteristic is the erase-before-update property, where an erase operation is needed in order to update the content of a flash page. Thus, a random write is much slower than a sequential write.

Because of the intrinsic characteristics of SSDs, spatial database applications increasingly require the performance evaluation of disk-based spatial indices on SSDs to analyze if there is a performance relation with the well-known obtained performance on HDDs. As a result, we can see how similar are the performance behavior of the spatial indexing on such devices. There are few approaches [Emrich et al. 2010; Fevgas and Bozanis 2015] that conduct an experimental evaluation of spatial indexing on SSDs. There are also approaches [Wu et al. 2003; Lv et al. 2011; Pawlik and Macyna 2012; Sarwat et al. 2013; Jin et al. 2015] that propose *flash-aware spatial indices* by adapting the R-tree in order to deal with the intrinsic characteristics of SSDs. The performance of these indices are measured through experimental evaluations. However, the experiments conducted on these approaches face several problems, such as the lack of an analysis between the performance relation of spatial indexing on HDDs and SSDs, the lack of focus on the spatial query processing, and the lack of an analysis with respect to the utilization of different parameter values of spatial indices on SSDs and HDDs.

To fill these gaps, this article has the following objectives. First, we aim to check the relative performance results of an index on an SSD and an HDD, that is, if a spatial index that show the best results on the HDD also shows the best results on the SSD and vice-versa. For this purpose, we conducted an extensive experimental evaluation on SSDs and HDDs using a local server and using virtual machines maintained in the Microsoft Azure environment. Second, we aim to verify the performance impact on the utilization of different parameter values in the evaluated spatial indices. For this purpose, we analyzed the performance relation between the used parameter values and the obtained results. Finally, we aim to analyze if the parameter values of a spatial index that bring benefit for its construction also guarantee a good performance in the spatial query processing.

This article is organized as follows. Section 2 surveys related work. Section 3 summarizes underlying concepts from spatial indexing. Section 4 briefly describes the intrinsic characteristics of SSDs. Section 5 details our experiments and discusses the obtained results. Section 6 concludes the paper and presents future work.

2. RELATED WORK

There are few approaches that conduct experimental evaluations of spatial indices on SSDs [Emrich et al. 2010; Fevgas and Bozanis 2015]. Further, there also approaches that propose flash-aware spatial indices based on the R-tree [Wu et al. 2003; Lv et al. 2011; Pawlik and Macyna 2012; Sarwat et al. 2013; Jin et al. 2015]. We classify these approaches according to the following characteristics: (i) the analysis of the performance of spatial indexing on HDDs and SSDs, (ii) the variety of spatial indices used in experiments, (iii) the parameter values considered in the indices, and (iv) the type of workloads focused on the experiments.

Regarding the first characteristic, almost all the approaches do not evaluate the performance of spatial indexing on HDDs and SSDs. This kind of evaluation is important to study and check if the

[1] `https://azure.microsoft.com`

performance of well-known indices (e.g., the R-tree and the R*-tree) are the same on HDDs and SSDs. Only [Emrich et al. 2010] conducts experiments considering both storage systems, HDDs and SSDs.

Regarding the second characteristic, the majority of the approaches [Wu et al. 2003; Lv et al. 2011; Pawlik and Macyna 2012; Sarwat et al. 2013; Jin et al. 2015] employs the R-tree as baseline in the experiments, while the remaining approaches [Emrich et al. 2010; Fevgas and Bozanis 2015] employ the R*-tree as baseline. However, it is important to consider both the R-tree and the R*-tree in a same experiment because of their good performance reported in the literature [Gaede and Günther 1998]. In addition, many approaches [Wu et al. 2003; Lv et al. 2011; Pawlik and Macyna 2012; Sarwat et al. 2013; Fevgas and Bozanis 2015; Jin et al. 2015] also propose flash-aware spatial indices and thus, they conduct experimental evaluations to check the performance behavior of these indices. But, the lack of studies about the performance of well-known spatial indices on SSDs can ignore other characteristics that could improve the performance of the proposed flash-aware spatial indices.

Regarding the third characteristic, the main parameter analyzed in the approaches is the buffer size employed in the main memory. However, parameterization plays an important role in spatial indexing and the variation of specific parameter values of a spatial index could impact on its performance (see Section 3). For instance, a well-known parameter used on spatial indexing is the page size, which is ignored in the current experiments. Thus, there is a lack of analysis of the impact of the page size used in a spatial index handled in an HDD and an SSD.

Regarding the fourth characteristic, the experiments of the approaches focus on insertion and deletion operations only since random writes are expensive operations on SSDs. However, it is also important to verify the performance of spatial queries since this is a very common operation in spatial databases and GIS. Hence, there is a lack of studies for verifying the impact of the spatial selectivity on the spatial query processing using indices stored in SSDs.

As a conclusion, we can note that there is a lack of a experimental evaluation to study the performance relation of the spatial indexing in the HDD and SSD. In this article, we conduct experiments considering different spatial indices with different parameter values on both storage systems, HDDs and SSDs. For this purpose, we consider disk-based spatial indices (i.e., the R-tree and the R*-tree) and flash-aware spatial indices, which are summarized in the next section. Further, we analyze the performance results for creating and querying spatial indices. Thus, we are able to analyze whether there is a spatial index that provides a spatial organization that benefits the spatial query processing.

We extend our previous work in [Carniel et al. 2016b] also performing our experiments in virtual machines equipped with HDD and SSDs allocated in the Microsoft Azure environment, which is a cloud data center. Further, we present details of our experimental setup, as well as discussions regarding to new obtained results. We also correlated the performance of the spatial indices stored in the different environments in order to detect guidelines for spatial database applications.

3. SPATIAL INDEXING

In general, hierarchical structures are employed to index spatial objects. Due to the complexity to store spatial objects in the indices, spatial objects are generalized to Minimum Bounding Rectangles (MBR). A MBR approximates a spatial object by using its minimum and maximum values of each dimension, that is, a MBR corresponds to the maximum extents of an n-dimensional object. Since MBRs are employed to represent spatial objects, two steps are required in order to process spatial queries [Gaede and Günther 1998]: the filtering step, and the refinement step. The filtering step makes prunes in the dataset using the spatial index and returns the possible candidates that answer a query. Then, the refinement step checks if each object really belongs to the final query. Thus, this step is a time-expensive operation since the spatial object is accessed from the storage device and complex algorithms from the geometry computational are employed to compute the topological predicate of the query [Gaede and Günther 1998].

The R-tree [Guttman 1984] is a common hierarchical spatial index used in spatial databases and GIS. It is composed of internal and leaf nodes. A leaf node comprises entries in the format (mbr_o, p_o), where mbr_o refers to the MBR of the spatial object o and p_o is a pointer that guarantees the direct access to o. An internal node comprises entries in the format (mbr_c, p_c), where mbr_c is the MBR that encompasses all the entries of the child node c and p_c is a pointer to the child node c. The minimum and maximum capacity of internal and leaf nodes are parameters determined respectively by m and M such that $m \leq M/2$. If the maximum capacity of a node is achieved in an insertion, a split operation is performed. There are different split algorithms with different complexities and results, such as the linear split and the quadratic split.

The R*-tree [Beckmann et al. 1990] is a well-known variant of the R-tree that applies other aspects to index spatial objects. For instance, the utilization of overlapping area among the entries in internal nodes, the redistribution of entries of overflowed nodes to maximize the storage utilization, and the utilization of margin of nodes. Hence, the R*-tree modifies the insertion algorithm of the R-tree in order to take into account these aspects. Further, it employs another split algorithm and applies a reinsertion policy in order to decrease the number of split operations.

With the increasing use of SSDs in applications, flash-aware spatial indices were proposed in the literature. The RFTL [Wu et al. 2003], the ARFTL [Pawlik and Macyna 2012], the LCR-tree [Lv et al. 2011], and the FOR-tree [Jin et al. 2015] are straightforward adaptations of the R-tree. A generic framework, called FAST [Sarwat et al. 2013], is proposed to efficiently convert a disk-based spatial index into a flash-aware spatial index. FAST does not change the index structure but only changes the way in which the nodes are written in the SSD. Thus, we are able to define FAST-based indices by using disk-based spatial indices as basis. For instance, the FAST R-tree and the FAST R*-tree, which are based on the R-tree and R*-tree, respectively.

Despite the particular characteristic of each flash-aware spatial index, they often make use of a buffer in the main memory that stores the most recent modifications of nodes instead of applying directly the modifications to the SSD. The goal is to avoid random writes, such as those that occur in splits. A *flushing operation*, composed of sequential writes, is performed when the buffer is full. While there are indices [Wu et al. 2003; Lv et al. 2011; Pawlik and Macyna 2012] that flush all the modifications leading to an expensive flushing operation, the FAST and the FOR-tree uses a refined *flushing policy* that chooses a set of nodes with modifications to be flushed. This set form a *flushing unit*, which has a fixed number of modified nodes to be written.

Parameterization plays an important role in spatial indexing [Gaede and Günther 1998]. Page (node) size as well as the minimum and maximum number of entries of leaf and internal nodes are examples of typical parameters. In addition, each index may include specific parameters according to its design. For instance, the reinsertion percentage of the R*-tree. Moreover, flash-aware spatial indices introduced several new parameters, such as the buffer and flushing unit sizes, which impact directly on the performance of flushing operations. Hence, there is a significant performance impact if *different* parameters are used in a spatial index in *different* datasets under *different* storage systems.

4. FLASH-BASED SOLID STATE DRIVES

Flash-based Solid State Drives (SSDs) have been very popular in many applications [Kryder and Kim 2009; Mittal and Vetter 2016]. Table I shows a comparison of the SSD and HDD used in our experiments (Section 5). SSDs have intrinsic characteristics that introduce several implications [Chen et al. 2009; Jung and Kandemir 2013] for the management of data in such devices.

SSDs store data in a block-oriented model, which means that a fixed number of flash pages composes a flash block. Commonly, the flash page size is 4KB and the flash block size is 256KB [Chen et al. 2009; Mittal and Vetter 2016]. SSDs provide the following operations: erase, read, and write (program) [Chen et al. 2009; Jung and Kandemir 2013]. Erase is a block level operation that changes

Table I. Comparison of the HDD[2] and the SSD[3] used in the experiments.

	4KB Random Transfers[4]		Power Consumption[5]		Endurance[6]
	Read	Write	Read	Write	
HDD	0.185MB/s	0.441MB/s	4.1W	4.1W	$> 10^{15}$
SSD	285.156MB/s	109.375MB/s	1.423W	2.052W	$10^4 - 10^5$

the bits from 0 to 1 of all pages contained in the block and thus, this is the most expensive operation. Read and write are page level operations with asymmetric costs. A read requires much less time and power consumption than a write (see Table I). A write is only able to change the bits from 1 to 0 in an erased block. Hence, to write in a previously written block (i.e., update), an *erase-before-update operation* is performed leading to a time-consuming operation. On the other hand, a write operation in a previously erased block is a lower latency operation and is denominated as a *sequential write*. Another important characteristic of SSDs is their lower endurance than HDDs (see Table I). Endurance refers to the maximum number of writes and erases in a block before its unreliableness.

To avoid erase-before-update operations, facilitates the integration of SSDs with current systems, and improve the endurance of SSDs, a Flash Translation Layer (FTL) [Chung et al. 2009] is employed. For this purpose, FTL provides an interface that allows operational systems to use SSDs as an usual storage device and only enables reads and writes for application layers. Further, FTL maps physical page addresses of the SSD into logical page addresses, which are effectively used by application layers. A logical page is marked as either free, valid, or invalid. A free logical page is a page ready to store data. A valid logical page contains data previously written. A logical page is marked as invalid if a write is performed on a valid logical page; and its new content is stored in another free logical page. This operation is termed as an *out-of-place update* and avoids an erase-before-update operation. A *garbage collection* is performed when space is required and there is a great number of invalid pages. This operation selects a set of blocks to apply erase operations causing erase-before-update operations. Hence, this is the most expensive operation performed by the FTL. The algorithms of the garbage collection and out-of-place update also consider a *wear leveling* in order to improve the endurance of flash blocks. For a survey of FTLs, see [Chung et al. 2009].

5. PERFORMANCE EVALUATION

We conduct an experimental evaluation in order to analyze the performance relation of the spatial indexing on HDDs and SSDs. Section 5.1 details the experimental setup used in the experiments. The obtained results related to the spatial index construction and spatial query processing are discussed in Sections 5.2 and 5.3, respectively.

5.1 Experimental Setup

We used a real dataset extracted from the OpenStreetMap[7], which consisted of 534,926 complex regions possibly with holes. This dataset represents the buildings of Brazil, such as hospitals, schools, universities, houses, stadiums, and so on. We used the PostgreSQL database management system with the PostGIS extension to store this dataset in a relational table.

[2] `https://support.wdc.com/product.aspx?ID=608&lang=en`
[3] `https://www.kingston.com/us/ssd/consumer/sv300s3`
[4] Measured by Iometer (`http://www.iometer.org/`).
[5] According to the manufactures[23].
[6] According to [Mittal and Vetter 2016].
[7] `http://www.openstreetmap.org/`

Table II. Configurations of the spatial indices and their corresponding specific parameters used in the experiments.

Configuration Name	Spatial Index	Specific Parameters
Linear R-tree	R-tree	Split: Linear
Quadratic R-tree	R-tree	Split: Quadratic
R-tree 20%*	R*-tree	RP: 20%
R-tree 30%*	R*-tree	RP: 30%
R-tree 40%*	R*-tree	RP: 40%
FAST Linear R-tree	FAST R-tree	Split: Linear; FP: FAST*
FAST Quadratic R-tree	FAST R-tree	Split: Quadratic; FP: FAST*
FAST R-tree 20%*	FAST R*-tree	RP: 20%; FP: FAST*
FAST R-tree 30%*	FAST R*-tree	RP: 30%; FP: FAST*
FAST R-tree 40%*	FAST R*-tree	RP: 40%; FP: FAST*

Table III. Generic parameters used in the experiments.

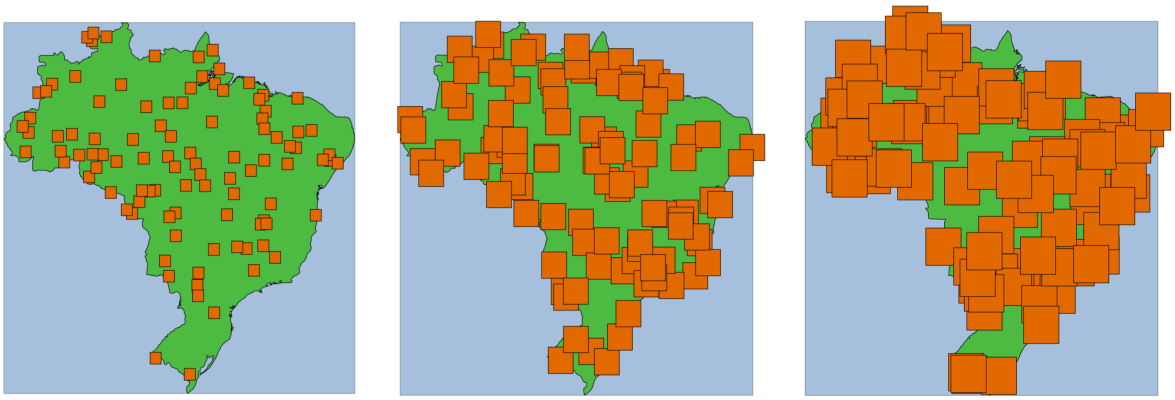
Page Size	Minimum Occupancy	Maximum Occupancy
2KB	28	56
4KB	57	113
8KB	114	227
16KB	228	455
32KB	455	910
64KB	910	1820

In order to conduct our experiments, we have extended the first version of FESTIval [Carniel et al. 2016a] by adding new functionalities like the creation of user-defined workloads. FESTIval[8] is an open-source PostgreSQL extension that enables the benchmarking of disk-based and flash-aware spatial indices with different parameter values under different storage systems by issuing workloads in a unique environment. We used FESTIval to compare the following disk-based spatial indices: the R-tree and the R*-tree. Further, we evaluate the following flash-aware spatial indices: FAST R-tree and the FAST R*-tree, which are FAST-based spatial indices (Section 3). Since we have updated the version of the FESTIval used in our previous work [Carniel et al. 2016b], in this article we have re-executed all the experiments.

FESTIval also enables the configuration of a spatial index by using specific and generic parameter values. Specific parameters determine the configuration of a specific spatial index only. For the R-tree, we varied its split algorithm by considering the linear and quadratic splits. For the R*-tree, we varied the reinsertion percentage (RP) since it impacts on the number of writes. Further, based on the recommendations for the R*-tree [Beckmann et al. 1990], we considered the close reinsert policy. For the FAST-based indices, we considered the FAST* flushing policy (FP) [Sarwat et al. 2013]. As a result, we employed the configurations depicted in Table II. We also studied the effect of the variation of the buffer and the flushing unit size. For all FAST-based indices, we executed our experiments considering the following buffer sizes: 128KB, 256KB, and 512KB. We have reported results only for the buffer equal to 512KB since it showed similar performance behavior compared to the other buffer sizes. In addition, we varied the flushing unit size from 1 to 5, which contributed to analyze the impact of this parameter in the flash-aware spatial indexing.

We also varied generic parameters, which can be applied for any spatial index of FESTIval. For instance, the page size (i.e., the size in bytes of a node), and the minimum and maximum number of entries of a node. Further, we considered the DIRECT I/O to avoid operational system caching

[8] `http://gbd.dc.ufscar.br/festival/`



(a) (b) (c)

Fig. 1. Query windows employed in the processing of IRQs: query windows with 0.1% (a), query windows with 0.5% (b), and query windows with 1% (c).

in reads and writes. Table III shows the page sizes, with its minimum and maximum occupancy, employed for all the configurations in Table II.

We executed two workloads defined as follows. The first workload focused on the *index construction* and thus, we collected the elapsed time in seconds for creating a spatial index (see Section 5.2). The creation of a spatial index was performed by inserting element by element according to the original insert algorithm of the respective index.

The second workload focused on the *spatial query processing* (see Section 5.3). We executed *intersection range queries* (IRQ) [Gaede and Günther 1998] because of its wide utilization in many applications. This kind of query returns from a dataset D, a set of spatial objects R that intersects a given rectangular-shaped query window QW, i.e., $R = \{o | o \in D \wedge intersects(o, QW) = true\}$. We synthetically generated three sets of query windows, which are depicted in Figure 1. Each set was composed of 100 query windows, which had a x% of area of the total extent of Brazil (i.e., the MBR of Brazil). The first set had 0.1% (Figure 1a), the second set had 0.5% (Figure 1b), and the third set had 1% (Figure 1c). Considering selectivity as the proportion between the number of returned elements in the query and the number of total elements in the dataset, these query windows composed IRQs with low, medium, and high selectivity, respectively.

We collected the total elapsed time in seconds taken to execute the 100 IRQs of each set of query windows. The total elapsed time was calculated as follows. For a specific set of query windows, we executed 10 times each IRQ, collected the average elapsed time of its execution, and then calculated the sum of the average elapsed times of the 100 IRQs. We flushed the system cache after each execution. Further, we consider the elapsed time with respect to the index time.

The experiments were conducted on two different hardware environments. The first environment was a local server equipped with an Intel® Core™ i7-4770 with frequency of 3.40GHz and 32GB of main memory. For the experiments with HDD we used a 2TB Western Digital with 7200RPM, while for the experiments with SSD we used a 480GB Kingston V300. Hence, this environment give us experiment results under controlled conditions since we executed the workloads in a local server to avoid network latency.

The second environment was composed of two virtual machines of the Azure Microsoft in order to analyze the efficiency of the HDD and SSD in cloud servers used by many applications. The first virtual machine had the Standard A6 size (4 cores, 28GB of main memory) with an HDD of 500GB. The second virtual machine had the Standard DS12 v2 size (4 cores, 28GB of main memory) with an SSD

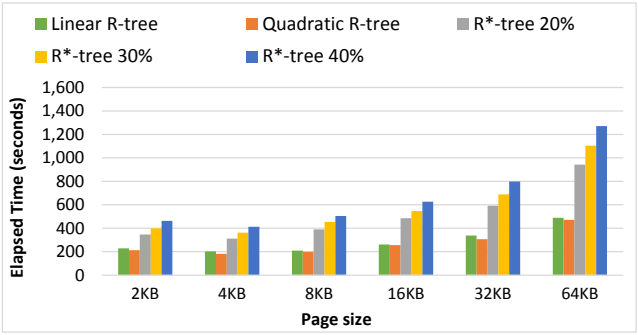
of 500GB. These two virtual machines and their storage devices were allocated in the South Central US. This environment is not as much controlled as the first environment because of the characteristics of the Microsoft Azure, such as the lack of guarantee that the storage device is in the same physical place than the unity processing. However, this environment gives us the change to analyze if the spatial indexing on HDDs and SSDs employed in such virtual machines have performance behaviors similar to the found in a controlled environment. For instance, we can find out if a spatial index that shows a good performance on the SSD in the first environment also shows a good performance on a virtual machine on a cloud data center with SSD. This would help applications hosted in the cloud to make use of the better spatial index.

The software employed in all the environments are: Ubuntu Server 14.04 64 bits, PostgreSQL 9.5, and PostGIS 2.2.0.

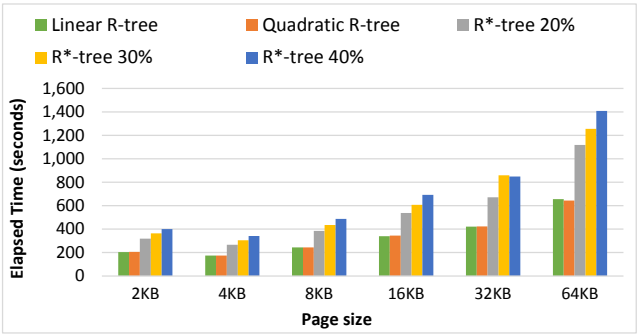
5.2 Spatial Index Construction

This section details the obtained results for the index constructions. Section 5.2.1 and 5.2.2 discuss the results for the local server and the virtual machines of the Microsoft Azure, respectively. Correlations are made in Section 5.2.3.

5.2.1 *Execution on the Local Server.* Figure 2 depicts the elapsed time for creating disk-based spatial indices on the HDD and SSD. The page size equal to 4KB gathered the best performance results for all spatial indices on both storage systems, indicating that this page size should be used for creating spatial indices. Considering this page size, the SSD showed better performance results than the HDD. The SSD showed performance gains between 4% and 16% over the HDD. A performance gain is the percentage that shows how much one configuration is more efficient than another configuration.

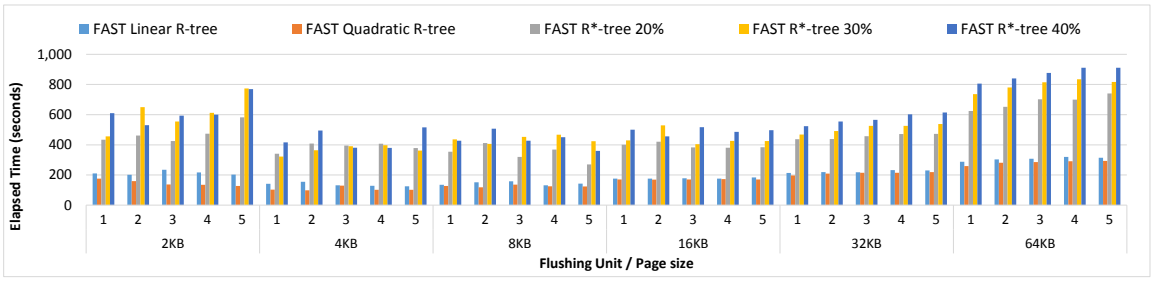


(performed on the HDD)

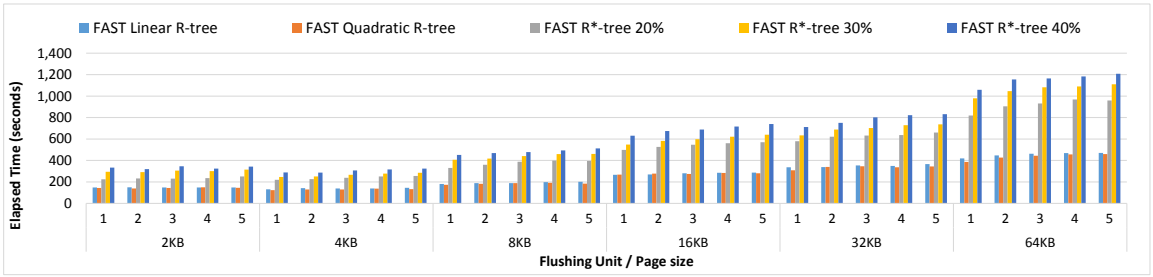


(performed on the SSD)

Fig. 2. Performance results for creating disk-based spatial indices on the local server.



(Performed on the HDD considering buffer equal to 512KB)



(Performed on the SSD considering buffer equal to 512KB)

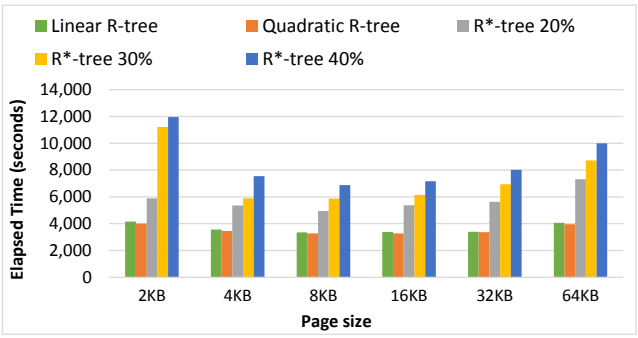
Fig. 3. Performance results for creating FAST-based spatial indices on the local server.

On the other hand, for the page sizes equal to 16KB, 32KB, and 64KB, we obtained best performance results by using the HDD. In these page sizes, the SSD introduced a performance loss ranging from 6% to 38% over the HDD. The main reason for this performance degradation is due to the interleaved writes and reads that the index construction performs on the storage device. Thus, workloads that mix many writes and reads tend to degenerate the performance of SSDs, such as discussed in [Lee and Moon 2007; Chen et al. 2009]. In addition, the lack of a special treatment for random writes was also determinant since writes are the most expensive operations of SSDs.

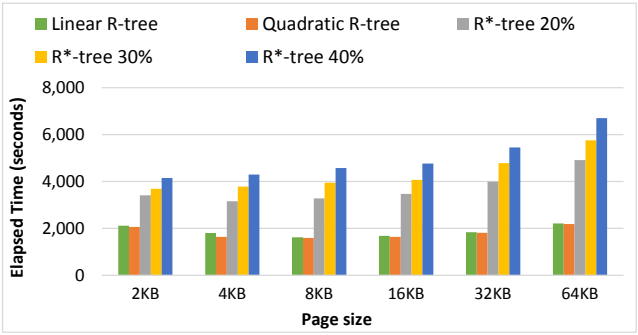
The results demonstrated that it is important to take into account the intrinsic characteristics of SSDs to achieve good performance in these memories. Thus, the redesign of spatial indices originally designed for HDDs is needed, such as the framework provided by FAST that transforms a disk-based spatial index into a flash-aware spatial index.

Figure 3 shows the performance results for constructing spatial indices by using FAST. In general, FAST-based indices improved the performance of their counterparts on both storage systems. For instance, the *FAST Linear R-tree* (Figure 3) showed lower elapsed times than the *Linear R-tree* (Figure 2) on both storage systems. The *FAST Linear R-tree* imposed performance gains ranging from 27% to 36% on the SSD. Although FAST is designed for SSDs, it also showed performance gains varying from 12% to 41% on the HDD.

An interesting fact was that the performance behavior for construction FAST-based indices on the SSD was different from the performance behavior of the HDD. We emphasize two main differences. The first difference was that the majority of the spatial indices showed best performance results on the HDD by utilizing the page size equal to 4KB and the flushing unit size equal to 5. On the other hand, all the spatial indices showed the best performance results on the SSD by employing the page size equal to 4KB and the flushing unit equal to 1. The second difference was that with the increase of page and flushing unit sizes in the index construction on the SSD, the time processing also increased. This was even much slower than the construction performed on the HDD. For instance, for the page size equal to 64KB, the results demonstrated that the HDD was faster than the SSD since big writes turned out problematic on the SSD.



(performed on the virtual machine with HDD)



(performed on the virtual machine with SSD)

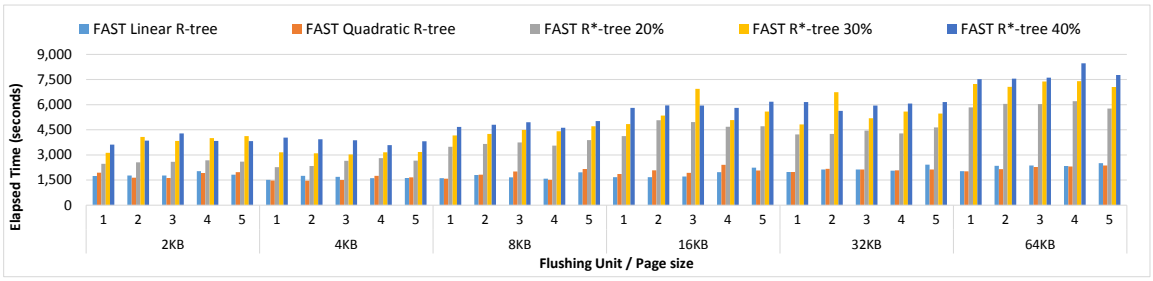
Fig. 4. Performance results for creating disk-based spatial indices on the virtual machines of the Microsoft Azure.

5.2.2 *Execution on the Microsoft Azure.* Figure 4 depicts the elapsed time for creating disk-based spatial indices on the virtual machines of the Microsoft Azure. Note that we use different scales in the graphs to better compare the performance results. Clearly, the construction of spatial indices on the virtual machine with SSD required much less processing time than the construction of the same spatial indices on the virtual machine with HDD. The main reason is that the Microsoft Azure gives a special attention for the use of SSDs, denominated as *Storage Premium Storage*[9]. It has a maximum throughput of 200MB/s and maximum of 5000 input/output operations per second (IOPS), while a common storage device (i.e., HDD) has a maximum throughput of 60MB/s and a maximum of 500 IOPS. It contributed to performance gains varying from 31% to 67% for constructing spatial indices on the virtual machine with SSD. In general, the page size equal to 8KB showed the best elapsed times for both the virtual machines.

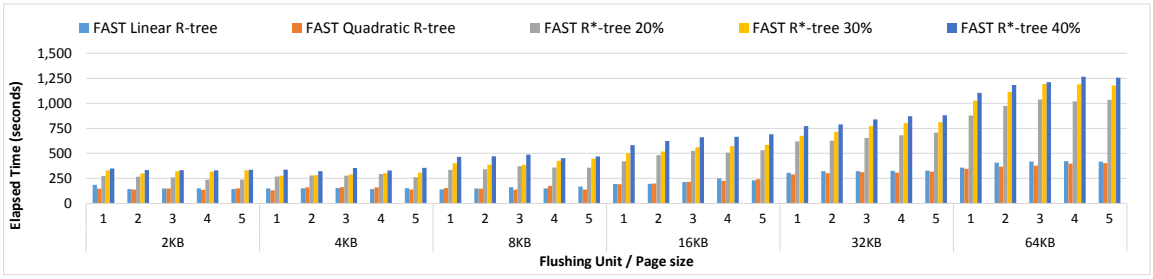
Although the expressive performance gains of the SSD over the HDD, these gains were yet improved when constructing spatial indices by using FAST. Figure 5 depicts the performance results obtained to construct FAST-based indices on the virtual machines of the Microsoft Azure. The performance gains imposed by FAST, ranged from 19% to 72% for constructing indices on the HDD. These gains were increased for constructing indices on the SSD, varying from 82% to 93%. It shows that FAST improved the performance of this kind of workload on the virtual machine using SSD because it takes into account intrinsic characteristics of these storage devices. Thus, we note that these intrinsic characteristics are still present on the cloud data center like the Microsoft Azure.

5.2.3 *Correlations Between the Results of the Local Server and the Microsoft Azure.* The main difference among the results was that the disk-based spatial indices executed on the local server with HDD showed, in general, better performance results than the execution on the local server with SSD.

[9] `https://docs.microsoft.com/en-us/azure/storage/storage-about-disks-and-vhds-linux`



(Performed on the virtual machine with HDD considering buffer equal to 512KB)



(Performed on the virtual machine with SSD considering buffer equal to 512KB)

Fig. 5. Performance results for creating FAST-based spatial indices on the virtual machines of the Microsoft Azure.

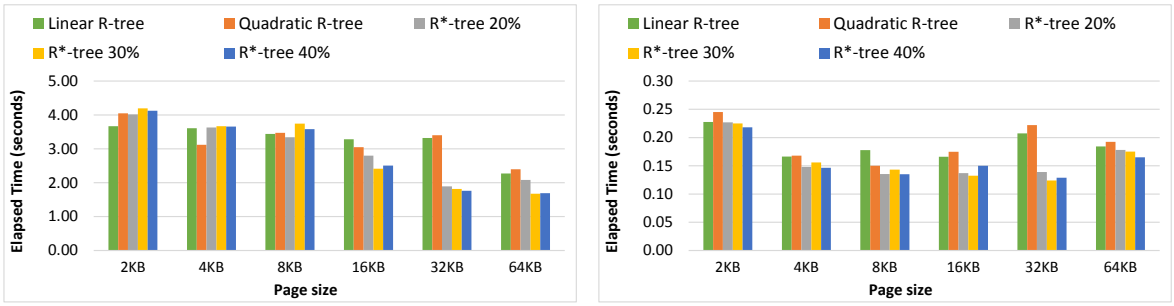
This performance behavior does not happened in the virtual machines of the Microsoft Azure. The main reason is that this cloud environment performs a special treatment for the SSDs, where only specific virtual machines are able to use this kind of storage system.

On the other hand, we can enumerate the following similarities among the results. First, since FAST takes into account the intrinsic characteristics of SSDs, it showed the best performance results for the environments using the SSD as the main storage device. It also improved the performance results for the environments using the HDD thanks to the use of a buffer in the main memory. Second, in general, the page size equal to 4KB gathered the best performance results for the FAST-based indices on the SSD and HDD. Third, the use of page sizes greater than 4KB required an increasing elapsed time to construct a FAST-based spatial index in the SSD, as the page size and flushing unit size also increases. Finally, in both the environments, the *FAST Quadratic R-tree* showed the fastest elapsed times, while the *Quadratic R-tree* was the best configuration among the disk-based spatial indices.

5.3 Spatial Query Processing

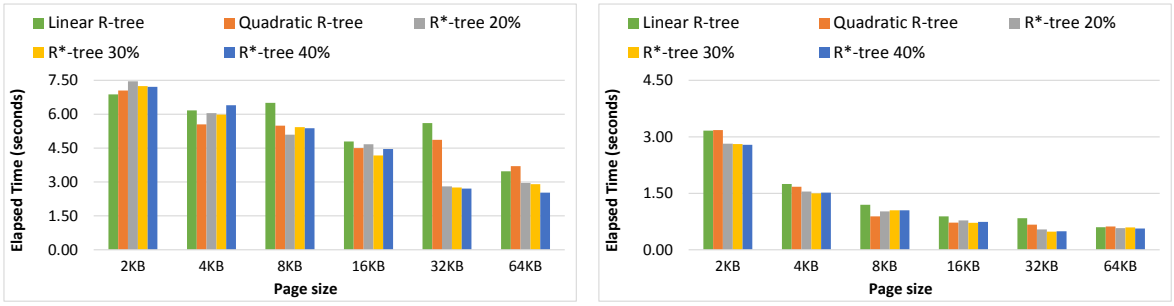
This section details the obtained results for spatial query processing on the local server (Section 5.3.1) and the virtual machines of the Microsoft Azure (Section 5.3.2). Finally, these results are correlated in Section 5.3.3.

5.3.1 *Execution on the Local Server.* Figures 6, 7, and 8 depict the obtained results for processing spatial queries by employing IRQs with 0.1%, 0.5%, and 1%, respectively. In these figures, we use a different scale to report the results for the SSD, which is always a sub set of the scale used to report the results for the HDD. Further, we considered only configurations based on the R-tree and the R*-tree (Table II) since FAST does not change the structure of a spatial index (e.g., the *Linear R-tree* using 4KB as page size has the same structure than the *FAST Linear R-tree* using 4KB as page size). Thus, we are able to compare the performance behavior of the underlying index that can organize spatially the data in different forms.



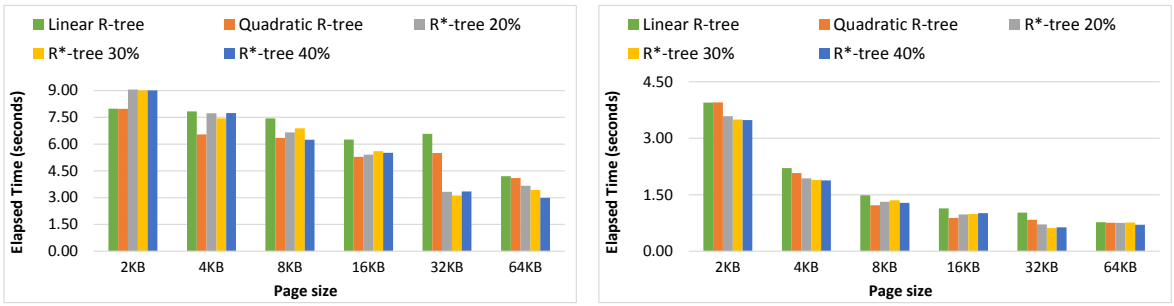
(performed on the HDD) (performed on the SSD)

Fig. 6. Performance results of spatial queries considering IRQs with 0.1%.



(performed on the HDD) (performed on the SSD)

Fig. 7. Performance results of spatial queries considering IRQs with 0.5%.

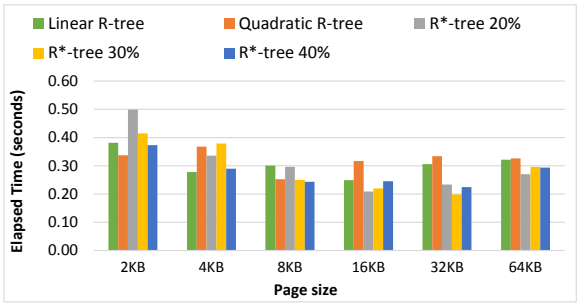


(performed on the HDD) (performed on the SSD)

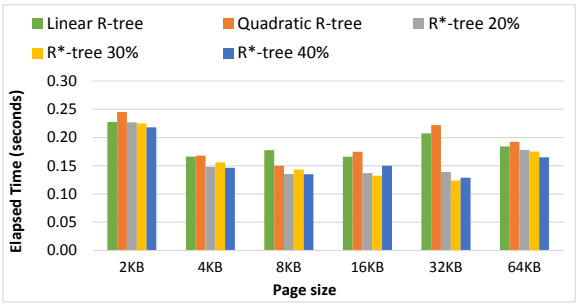
Fig. 8. Performance results of spatial queries considering IRQs with 1%.

Clearly, the performance of spatial query processing on the SSD overcame the HDD in all experiments because of its faster random reads. Considering all the page sizes, the performance gains of the SSD varied from 89% to 96% for the query windows with 0.1%. Gains ranging from 54% to 86% and from 51% to 85% were obtained for the query windows with 0.5% and 1%, respectively. The performance gains were decreasing as the selectivity increases because of the number of elements returned by the spatial queries. That is, the performance of the SSD beneficed more spatial queries with low selectivity than the other spatial queries because of the number reads made on the storage device.

With respect to the execution on the HDD, almost all the configurations improved their performance by increasing page sizes. The reason is that if a spatial index employs large page sizes, more elements

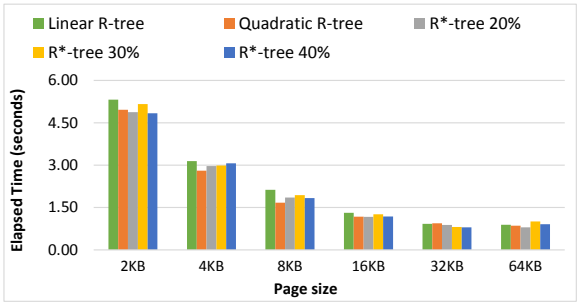


(performed on the virtual machine with HDD)

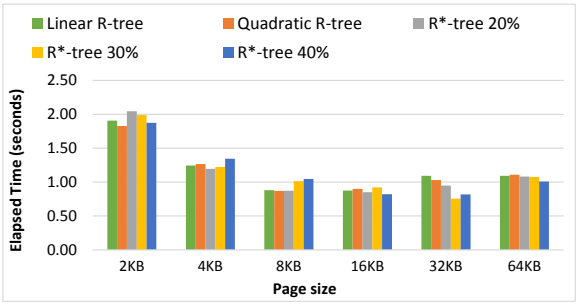


(performed on the virtual machine with SSD)

Fig. 9. Performance results of spatial queries, executed on the virtual machines of the Microsoft Azure, considering IRQs with 0.1%.



(performed on the virtual machine with HDD)



(performed on the virtual machine with SSD)

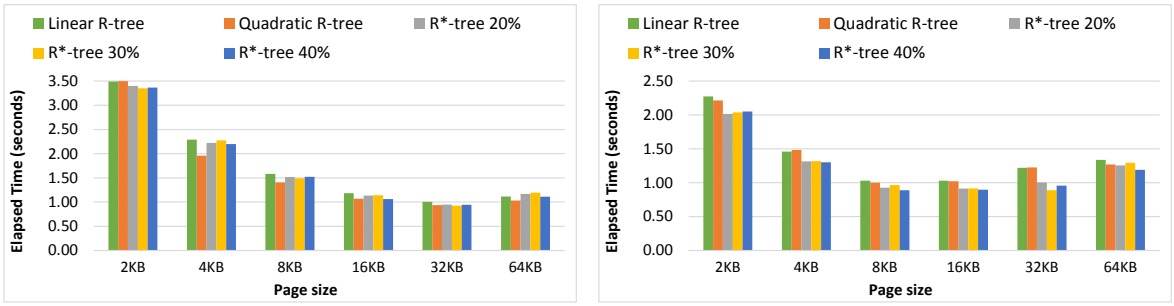
Fig. 10. Performance results of spatial queries, executed on the virtual machines of the Microsoft Azure, considering IRQs with 0.5%.

are processed in the main memory with only a few reads from the storage device. For the query windows with 0.1% (Figure 6), the *R*-tree 30%* using the page size equal to 64KB provided the best results. For the IRQs with medium and high selectivity (Figures 7 and 8), the best configuration was the *R*-tree 40%* using the page size equal to 64KB. The reason is that these queries returned more elements than the spatial queries using the other query windows. Since the *R*-tree 40%* improved the storage utilization and reduced the number of nodes, less accesses to the disk was required, improving the elapsed time.

With respect to the execution on the SSD, the performance behavior was slightly different than the HDD. For all IRQs, we obtained the best results by using the *R*-tree 30%* with the page size equal to 32KB. For the majority of the configurations, the use of the page size equal to 64KB deteriorated the performance, if compared to the use of the 32KB. We observed this behavior here because the SSD has poor performance to retrieve large node sizes, thus degenerating the performance of the reads. The results of this experiment demonstrated that the spatial organization for an efficient spatial query processing on SSDs tends to be different than on HDDs. A main finding is that in order to exploit the good performance of random reads, we cannot read nodes with a large number of elements.

5.3.2 *Execution on the Microsoft Azure.* Figures 9, 10, and 11 show the obtained results for processing spatial queries on the virtual machines of the Microsoft Azure. As in other figures, we used a different scale for better comparison of the results and considered only configurations based on the R-tree and the R*-tree (Table II).

For the majority of the cases, the virtual machine equipped with the SSD showed better perfor-



(performed on the virtual machine with HDD) (performed on the virtual machine with SSD)

Fig. 11. Performance results of spatial queries, executed on the virtual machines of the Microsoft Azure, considering IRQs with 1%.

mance results to process spatial queries. Considering the query windows with 0.1% (Figure 9), the performance gains ranged from 27% to 54%. For both virtual machines, the R*-tree 30% using the page size equal to 32KB showed the fastest elapsed time to process the spatial queries.

On the other hand, the virtual machine with the HDD overcame the virtual machine with the SSD to process the query windows with 0.5% and 1% (Figures 10 and 11). This behavior occurred by using the page sizes equal to 32KB and 64KB. The performance less of the SSD is due to the great size of the nodes to be processed by the virtual machine, which may have its storage device in a different location from the unity processing. Smaller page sizes improved the performance. For the majority of the indices, the page size equal to 16KB showed the best performance results in the virtual machine with SSD. Considering this page size, gains ranging from 23% to 33% were obtained for the query windows with 0.5% and gains varying from 4% to 20% were obtained for the query windows with 1%. While for the query windows with 0.5% the *R*-tree 30%* was the best configuration in both virtual machines, the *R*-tree 40%* was the best configuration for the query windows with 1%.

5.3.3 *Correlations Between the Results of the Local Server and the Microsoft Azure.* As main different among the results, we can cite that the SSD always showed better performance results than the HDD in the local server. On the other hand, the Microsoft Azure showed the opposite for greater page sizes (i.e., 32KB and 64KB) to process query windows with medium and high selectivity because of the peculiarity of the cloud environment. But, this behavior disappeared for smaller page sizes. This means that such page sizes are preferable to process these kind of queries in the Microsoft Azure, while greater page sizes would be preferable for the local server.

On the other hand, a common behavior was the benefit of use great pages in the HDD to process the spatial queries. In addition, the *R*-tree 30%* showed the best performance results for the HDD and SSDs for the majority of the spatial queries.

6. CONCLUSIONS AND FUTURE WORK

In this article, we conducted an extensive experimental evaluation to check the performance relation of spatial indexing on HDDs and SSDs under a local server and a cloud server. We considered the disk-based spatial indices R-tree and R*-tree because of their positive characteristics known in the literature. We also considered FAST-based spatial indices, which are flash-aware spatial indices that take into account the intrinsic characteristics of SSDs. By varying several parameters, we were able to analyze at least 210 distinct configurations of spatial indices in our experiments.

As main conclusions we can cite the following performance behaviors, which can lead us to a set of guidelines to further improve the performance of spatial indexing on SSDs. Firstly, in order to build

spatial indices in the SSD, our experiments showed that the direct employment of disk-based spatial indices did not guarantee good performance results. In fact, the HDD can be even faster than the SSD, in the local server. This means that to efficiently build spatial indices on the SSD, these indices should consider the intrinsic characteristics of SSDs, such as the poor performance of random writes.

Secondly, the experiments showed that FAST-based indices improved the elapsed time for the creation of spatial indices on the SSD and even on the HDD since it uses a buffer in the main memory. In general, the performance evaluation showed that employing the FAST for the R*-tree with page sizes varying from 4KB to 16KB did not require much time for creating the indices and provided a good performance in the spatial query processing. Finally, considering these page sizes, the use of smaller pages are recommended for queries with low selectivity, while the use of greater page sizes are recommended for queries with higher selectivities. This behavior can be applied for the both environments considered in this article. Thus, spatial database applications may employ these guidelines to improve the spatial query processing in local and cloud servers.

Future work will consider new workloads by including insertions, deletions, and updates of spatial objects in order to analyze the performance behavior for maintaining spatial indices. We will also extend the experiments by considering other spatial indices, like the Hilbert R-tree [Kamel and Faloutsos 1994]. Finally, we aim to propose a new flash-aware spatial index that takes into account the findings of this article.

Acknowledgments. This work has been supported by the Brazilian federal research agencies CAPES and CNPq as well as by the São Paulo Research Foundation (FAPESP). A. C. Carniel has been supported by the grant #2015/26687-8, FAPESP. R. R. Ciferri has been supported by the grant #311868/2015-0, CNPq. We also thanks to the support of the Microsoft Azure Sponsorship from the Microsoft Research.

REFERENCES

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. The R*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD International Conference on Management of Data*. pp. 322–331, 1990.

Carniel, A. C., Ciferri, R. R., and Ciferri, C. D. A. Experimental evaluation of spatial indices with FESTIval. In *Proceedings of the Brazilian Symposium on Databases - Demonstration Track*. pp. 123–128, 2016a.

Carniel, A. C., Ciferri, R. R., and Ciferri, C. D. A. The performance relation of spatial indexing on hard disk drives and solid state drives. In *Brazilian Symposium on GeoInformatics*. pp. 263–274, 2016b.

Chen, F., Koufaty, D. A., and Zhang, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. pp. 181–192, 2009.

Chung, T.-S., Park, D.-J., Park, S., Lee, D.-H., Lee, S.-W., and Song, H.-J. A survey of flash translation layer. *Journal of Systems Architecture: the EUROMICRO Journal* 55 (5-6): 332–343, 2009.

Emrich, T., Graf, F., Kriegel, H.-P., Schubert, M., and Thoma, M. On the impact of flash SSDs on spatial indexing. In *International Workshop on Data Management on New Hardware*. pp. 3–8, 2010.

Fevgas, A. and Bozanis, P. Grid-file: Towards to a flash efficient multi-dimensional index. In *Int. Conf. on Database and Expert Systems Applications*. pp. 285–294, 2015.

Gaede, V. and Günther, O. Multidimensional access methods. *ACM Computing Surveys* 30 (2): 170–231, 1998.

Güting, R. H. An introduction to spatial database systems. *The VLDB Journal* 3 (4): 357–399, 1994.

Guttman, A. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*. pp. 47–57, 1984.

Jin, P., Xie, X., Wang, N., and Yue, L. Optimizing R-tree for flash memory. *Expert Systems with Applications* 42 (10): 4676–4686, 2015.

Jung, M. and Kandemir, M. Revisiting widely held SSD expectations and rethinking system-level implications. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. pp. 203–216, 2013.

Kamel, I. and Faloutsos, C. Hilbert R-tree: An improved R-tree using fractals. In *International Conference on Very Large Data Bases*. pp. 500–509, 1994.

Kryder, M. H. and Kim, C. S. After hard drives–what comes next? *IEEE Transactions on Magnetics* 45 (10): 3406–3413, 2009.

Lee, S.-W. and Moon, B. Design of flash-based DBMS: An in-page logging approach. In *ACM SIGMOD International Conference on Management of Data*. pp. 55–66, 2007.

Lv, Y., Li, J., Cui, B., and Chen, X. Log-Compact R-tree: An efficient spatial index for SSD. In *International Conference on Database Systems for Advanced Applications*. pp. 202–213, 2011.

Mittal, S. and Vetter, J. S. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems* 27 (5): 1537–1550, 2016.

Pawlik, M. and Macyna, W. Implementation of the aggregated R-tree over flash memory. In *International Conference on Database Systems for Advanced Applications*. pp. 65–72, 2012.

Sarwat, M., Mokbel, M. F., Zhou, X., and Nath, S. Generic and efficient framework for search trees on flash memory storage systems. *GeoInformatica* 17 (3): 417–448, 2013.

Wu, C.-H., Chang, L.-P., and Kuo, T.-W. An efficient R-tree implementation over flash-memory storage systems. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. pp. 17–24, 2003.