Record Placement Based on Data Skew Using Solid State Drives

Jun Suzuki[1(✉)], Shivaram Venkataraman[2], Sameer Agarwal[2], Michael Franklin[2], and Ion Stoica[2]

[1] Green Platform Research Laboratories, NEC, Kawasaki, Japan
j-suzuki@ax.jp.nec.com
[2] University of California, Berkeley, USA
{shivaram,sameerag,istoica}@eecs.berkeley.edu, franklin@cs.berkeley.edu

Abstract. Integrating a solid state drive (SSD) into a data store is expected to improve its I/O performance. However, there is still a large difference between the price of an SSD and a hard-disk drive (HDD). One of the methods to offset the increase in cost of consisting devices is to configure a hybrid system using both devices. In such a system, a common method to decide the placement of data records is based on *reference locality, i.e.,* placing the frequently accessed records in a faster SSD. In this paper, we propose an alternative that focuses on data skew by storing records with values that appear less often in an SSD while those that do more in an HDD. As we will show, this enhances the performance of fetching records using multi-dimensional indices. When records are fetched using one of the indices targeted for optimization, records stored in an SSD are likely be retrieved using random access, while those stored in an HDD using sequential access. Given the method does not rely on reference locality, its performance is stable between first and second accesses and it provides a performance gain even when a host memory is large enough to contain the entire working set of the application. Our implementation and experiments show that storing just 20 % records in an SSD achieves up to 76 % of the maximum reduction that would otherwise be obtained when all the records are stored in an SSD.

Keywords: SSD · Index · Hybrid data store · Data skew

1 Introduction

Integrating SSDs into data stores has been a subject of much attention given their I/O performance is higher than that of HDDs. HDDs on the other hand have been widely used as secondary storage of data stores and their capacity has been doubling every 18 months [6]. However, their increase in I/O bandwidth

J. Suzuki—Visiting scholar at University of California, Berkeley when this work was done.

© Springer International Publishing Switzerland 2014
J. Zhan et al. (Eds.): BPOE 2014, LNCS 8807, pp. 181–193, 2014.
DOI: 10.1007/978-3-319-13021-7_14

has been slow and is currently just above 100 MB/s. SSDs on the other hand, keep doubling their bandwidth every 36 months. In addition, since they do not involve a mechanical component, their random access performance is better by more than two orders of magnitude as compared to an HDD.

Although performance of SSDs are attractive, there is still a large difference between the prices of SSDs and HDDs. In addition, their performance in sequential access is not so different as that in random access (and is even comparable when RAID is applied to HDDs). Therefore, it is reasonable to configure a hybrid system of SSDs and HDDs, and adopt a data placement method that maximizes system I/O performance.

Some of the data placement methods have been proposed for hybrid database application [1–3,9]. These methods are based on reference locality of data stored in a database. By storing frequently accessed data in an SSD, system I/O performance is improved nonlinearly to the ratio of the size of data stored in an SSD and in an HDD. Some of them also distinguish random access from sequential access, and give randomly accessed data priority to be stored in an SSD. The performance gain obtained from these methods depends on both the data access patterns and the size of the working set of application. As the hottest data is cached in the buffer pool of the host, in order to obtain a performance gain, the working set needs to contain warmly accessed data that is large enough to be spilled off from the host memory and stored in an SSD.

In this paper, we propose another data placement method among SSDs and HDDs by focusing on data skew (called *Skew-Based Data Placement, SDP*, hereafter). Data skew is known to be one of general characteristics that frequently appear in stored data [5]. SDP uses it to obtain performance enhancement nonlinear to the ratio of the size of data stored in an SSD. It provides performance enhancement from the first time when data are accessed. In addition, it provides nonlinear gain even if the host memory is large enough to contain the working set of application, and other cold data is accessed less frequently and uniformly. The application of SDP is for data that is rarely updated, – such as that for log analysis.

In SDP, the target columns for which the placement of records are optimized are given by an administrator or an overlying application. It then decides record placement so that the performance of fetching records using indices on any of those columns is enhanced. SDP configures a composite key consisting of the target columns. It then decides whether the records are to be stored in an SSD or in an HDD depending on the value of the composite key. To optimize the overall placement, data skews in all of the target columns are considered simultaneously. This process is formulated as integer linear programming (ILP) problem. With SDP, records stored in an SSD are likely to be retrieved using random access, while those stored in an HDD are likely to be fetched using sequential access. Concentrating random accesses to an SSD enables performance enhancement nonlinear to the ratio of records stored in an SSD and in an HDD.

We implemented a prototype using MySQL and evaluated it using a customer statistics table provided by an internet company, Conviva Inc. We show that by

storing 20 % of records in an SSD we can achieve up to 76 % of the maximum reduction that would otherwise be obtained when all the records are stored in an SSD.

The rest of the paper is organized as follows– comparison of an SSD and an HDD is given in Sect. 2, the data skew of the table used in the prototype evaluation is discussed in Sect. 3, the proposed SDP is described in Sect. 4, the implemented prototype and its evaluation results are presented in Sect. 5, the related work is discussed in Sect. 6, and finally we conclude in Sect. 7.

2 Comparison of SSD and HDD

An SSD is generally composed of an SSD controller and multiple flash memory packages [4]. An SSD controller is connected to a host using some host connection interface such as SATA or PCI Express. It transforms block I/O requests from a host to read and write I/O operations to the flash memory packages. These I/O operations are parallelized among multiple packages to enhance performance of an SSD.

Table 1 compares the performance [7,8] and price [6] of a SATA SSD and an HDD. Here, we discuss read performance since that is the focus of the proposed method. Because an SSD does not contain a mechanical component and I/O requests are served in parallel among flash packages, its random access performance is better by more than two orders of magnitude than that of an HDD. On the other hand, the difference of sequential access is not as much as that of random access. When the prices of the two devices are considered, combination of RAID method and HDDs is a reasonable choice if majority of I/O traffic of application are performed in sequential access.

A data placement method of a hybrid system of SSDs and HDDs need to consider these performance characteristics. Performance gain obtained by serving random accesses using an SSD is larger than that obtained by serving sequential accesses. Therefore, to make the best of SSDs, data should be placed so that random accesses be served by SSDs while sequential accesses be done by HDDs.

Table 1. Comparison of SSD and HDD.

Device	Random 4K Read IOPS	Sequential Read BW	Price
SSD	41,000–89,000	500–550 MB/s	1 $/GB
HDD	>91–118[a]	125–156 MB/s	0.05–0.07 $/GB

[a]Calculated using read seek average time.

3 Skew in Data Distribution

It is widely known that in a column of a table, the numbers of entries of values appearing in that column are frequently not equal and have skew. For example,

if a column is on places of residence of customers, there are many entries of big cities such as New York and Los Angeles. On the other hand, there are many other smaller cities which appear less often. Therefore, in general, there is a small number of values that often appear in a column while many others do less often.

Figure 1 shows the skew of the table we used in our prototype evaluation. It is the table on customer statistics of an Internet company, Conviva. It has 104 attributes and we investigated the skew on the values of the combination of the four columns, namely the endedFlag, customerId, country, and city. They are often used to select records in the company for data analysis. The figure shows the cumulative distribution function of the number of the occurrence of each value appearing in the table and that of its storage consumption. It shows that 90 % of values just appear in 6 % of records in the table, while 95 % of values does in 10 %. Therefore, 90 % of records are occupied by just 5 % of major values.

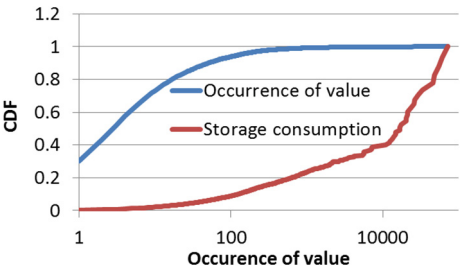


Fig. 1. Cumulative distribution function of the number of the occurrence of each value appearing in a table.

4 Optimizing Data Placement

4.1 Skew-Based Data Placement (SDP)

SDP uses the data skew appears in records stored in a table. Unlike conventional methods, it does not depend on the skew or locality of reference of application.

The method is intuitively illustrated using Fig. 2(a). A table which is used as an example to explain the method has two columns: customer ID and city. Although the cardinality of the customer ID is three, because of its data skew, the number of occurrence of "1" is larger than that of "2" and "3". In the same way, on the city column, the number of occurrence of "New York" is larger than that of "Berkeley", because New York is a much larger city than Berkeley.

When records, which are rows of the table, are sorted according to one column to optimize fetching records using its index, the performance of fetching records using an index on another column is not generally optimized. In a case considered here, records are fetched using either the index on the customer ID or the city. Figure 2(a) shows the order of records in the table that are sorted using

a composite key of the customer ID and the city. Because the customer ID is the first column in the composite key, sorted records are clustered on it. In other words, records that share an identical value of the customer ID are continuously placed in the table. When clustered records are stored in an HDD, fetching those records are performed using sequential access. An HDD provides good performance for sequential access. On the other hand, when records are fetched using the index on the city – as the records that share an identical value of the city are separated in the table – fetching is performed using random access. When the table is stored in an HDD, fetching records that correspond to "New York" requires three seeks. Because seek time of an HDD is considerably large, the performance of fetching records using the index on the city is much worse than that of the customer ID.

SDP solves this kind of cases and enhance the performance of fetching records using indices in a multi-dimensional way. It focuses on records with less frequent values and stores them in an SSD. In the example of Fig. 2(a), when it is allowed to move up to three records from an HDD to an SSD, moving records with the customer ID of "2" and "3" reduces the number of seeks to fetch the records with "New York" from three to one. This means that by moving 33 % of records of the table, the number of seeks is reduced by 66 %. In this way, by moving records with less frequent values, SDP reduces the number of seeks made to an HDD nonlinearly to the ratio of the size of records stored in an SSD and an HDD.

Figure 2(b) schematically shows the reduction of the cardinality of the composite key of records stored in an HDD, The horizontal and vertical axises correspond to the columns consisting the composite key. Although the number of axises or columns in the composite key is two for the explanation purpose, SDP is not limited to it. Each square in Fig. 2(b) corresponds to a possible value of the composite key which are combination of values in each column. The colored squares represent that there are records with the corresponding value. By storing less frequent values of the composite key in an SSD, large number of colored

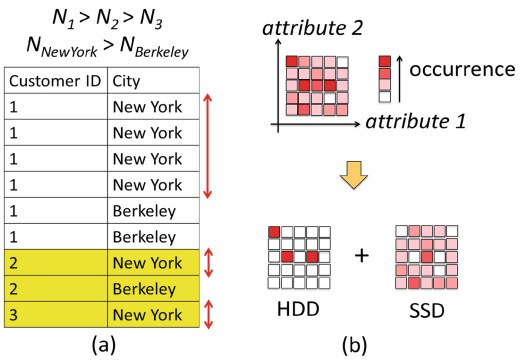


Fig. 2. Enhancement of fetching records using data skew.

squares moves to an SSD. As a result, an HDD stores a small number of colored values that has many entries. This results in the nonlinear reduction of the number of seeks in fetching records using indices of the columns in the composite key except for the first column.

4.2 Optimization Formulation

In this subsection, the optimization of data placement between an SSD and an HDD is formulated. It is supposed that data are stored in a single table, and records are retrieved using either of multiple indices. Therefore, the motivation of the data placement is to enhance the performance of fetching records using indices in multi-dimensional way.

It is also shown that the optimization of data placement is formulated as integer linear programming. I/O cost to fetch records is the optimized function under the constraint of SSD resources. Because integer linear programming is known as an NP-hard problem, we used greedy method in our prototype described in Sect. 5 to perform optimization calculation.

Table 2. Variables to formulate I/O cost

Variable	Explanation
N_i	Cardinality of column c_i
K_i	Cardinality of combinatorial column (c_1, c_2, ..., c_i)
F_i	The number of fragmentation of column c_i in HDD
B_{SSD}, B_{HDD}	Bandwidth of SSD or HDD
S_{SSD}, S_{HDD}	Size of data stored in SSD or HDD
T_{seek}	Average seek time of HDD
$y_i(x_1, x_2, ..., x_i)$	Whether records with $(x_1, x_2, ..., x_i)$ are stored in SSD
$s(x_1, x_2, ..., x_n)$	Total size of records with combinatorial values $(x_1, x_2, ..., x_n)$
C_{SSD}	Constraint of SSD consumption

To formulate the I/O cost, the variables shown in Table 2 are introduced. F_i represents the number of fragments that include records with the same value on column c_i in an HDD. i represents the order within the target columns that are optimized for fetching records. In the example shown in Fig. 2, if the whole table is stored in an HDD, F_2 is five because "New York" appears in three fragments while "Berkeley" does in two.

The average I/O cost to fetch records using an index of c_i by specifying a value in the column is given by dividing total I/O cost to fetch all records by the cardinality of the column. The total I/O cost is the sum of the I/O cost to fetch records reside in both an SSD and in an HDD. We ignored the latency of I/O requests other than the seek time of an HDD, because our interest is the

difference of the cost between the two devices. Then, the average I/O cost is described as

$$\frac{1}{N_i}\left(T_{seek}F_i + \frac{S_{HDD}}{B_{HDD}} + \frac{S_{SSD}}{B_{SSD}}\right). \tag{1}$$

When the size of an SSD that can be used to store records of a table is given as the constraint of the data placement optimization, the second and third terms in formula (1) are constant. Then, only the first term varies depending on the placement of records. Therefore, we formulate the I/O cost to be optimized by linearly combining the costs of each target column as

$$\sum_i R_i \frac{Fi}{N_i} \tag{2}$$

Here, R_i adjusts the relative importance among the target columns, and T_{seek} is included into it. We introduce two parameters, y_i and s; $y_i(x_1, x_2, ..., x_i) \in \{0, 1\}$ denotes whether records with the combination of the values in target columns of $(x_1, x_2, ..., x_i)$ are stored in an SSD; $s(x_1, x_2, ..., x_n)$ denotes the total size of the records with the combination of the value of $(x_1, x_2, ..., x_n)$. n denotes the number of columns to be optimized and $1 \leq i \leq n$. In SDP, records which share the same combination of the value $(x_1, x_2, ..., x_n)$ are stored either in an SSD or an HDD. Then, the next relation holds for y_i and y_{i+1}.

$$y_{i+1}(x_1, x_2, ..., x_i, x_{i+1}) - y_i(x_1, x_2, ..., x_i) \geqq 0 \tag{3}$$

That is, for $y_i(k_1, k_2, ..., k_i)$ of the specific combination of constants of $(k_1, k_2, ..., k_i)$ to be one, $y_{i+1}(k_1, k_2, ..., k_i, x_{i+1})$ needs to be one for all possible value of x_{i+1}. Then, the number of fragments F_i in an HDD is described as

$$F_i = K_i - \sum_{x_1}\sum_{x_2}...\sum_{x_i} y_i(x_1, x_2, ..., x_i). \tag{4}$$

The constraint of the consumption of SSD resources is described as

$$\sum_{x_1}\sum_{x_2}...\sum_{x_n} s(x_1, x_2, ..., x_n) y_n(x_1, x_2, ..., x_n) \leq C_{SSD}. \tag{5}$$

Substituting formula (4) into formula (2) gives the I/O cost that are described with variables $y_i(x_1, x_2, ..., x_i)$. Then, calculating the combination of $y_i(x_1, x_2, ..., x_i)$ which minimizes the I/O cost under the constraints of formulas (3) and (5) is an integer linear programming since $y_i(x_1, x_2, ..., x_i) \in \{0, 1\}$. An integer linear programming is known as an NP-hard problem.

5 Prototype Evaluation

5.1 Implementation

To evaluate SDP, we implemented a prototype shown in Fig. 3 using MySQL. It is consisted of two layers: data placement optimizer and MySQL. The data placement optimizer optimizes the placement of records in an original table between

an SSD an HDD. An application program is supposed to create individual index on each of the target columns and use either of them to fetch records by specifying value on that column.

The data placement optimizer takes the statistics on the occurrence of values in the target columns. It then decides whether records with each combination of values are stored in an SSD or in an HDD. It uses the data placement optimization explained in Sect. 4 in the constraint of the SSD resources given by an administrative interface. It also creates an additional column, $ssd\_flag \in \{0, 1\}$, to an original table. The flag denotes whether a record containing a flag is stored in an SSD or not.

Because the calculation of the data placement optimization is NP-hard, greedy method is used. It continued choosing the next combination of values that reduced the I/O cost per SSD consumption described by formula (2) most until the total consumption of an SSD reached the given constraint.

The range partition function of MySQL 5.6 was used to divide storing records between an SSD and an HDD. Records with ssd_flag of "1" were stored in an partition made in the directory where an SSD was mounted while ones with ssd_flag of "0" were stored in the HDD directory.

In the evaluation, SSD resources consumed by the indices and the ssd_flag column was ignored because it was small compared to the original data table that had 104 columns. We used a single host with two 8-core xeon CPUs and 128-GB memory. It contained both a SATA SSD (Intel 520 series) and a SATA HDD (Seagate ST91000640NS Constellation.2) that were used for SDP evaluation.

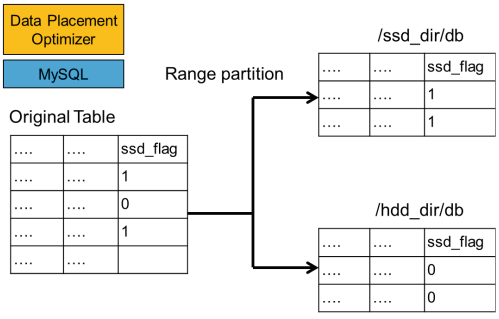


Fig. 3. Prototype implementation.

5.2 Evaluation

We performed two kinds of experiments. In the first experiment, the columns that were frequently used to select records in the data analysis in an Internet company, Conviva were selected as the target of optimization. However, because these columns were correlated, records that shared the same value in the target column were not fragmented so much and the gain of the performance by SDP was limited. Therefore, in the second experiment, different columns were selected

and the size of records was halved to artificially increase the affect of seek of an HDD to the system performance. The table used in the experiment is the same as the one analyzed in Sect. 3 that had 1,062,701 records, each record consisted of 104 columns, and the size of each record was about 2 KB. For the simplisity of evaluations, columns that were not in the interests were defined as a single large blob column.

In the first experiment, data placement was optimized on the columns that were frequently used for analysis. Table 3 shows the cardinality of each of the four columns chosen to be optimized, namely endedFlag, customerId, country, and city. It also shows the cardinality of their combination.

In SDP, records in an SSD and an HDD are sorted using a composite key that is consisted of the target columns for optimization. Except for the first column in the composite key, how much the records sharing the same value in a column are fragmented depends on the cardinality of its preceding columns. Therefore, to minimize the fragmentation of records in an HDD, the order of columns in a composite key is decided in ascending order of their cardinality; in the first experiment, the composite key was (endedFlag, customerId, country, city).

Table 3. Cardinality of each and composite column used in the experiments.

Key	Cardinality
endedFlag	2
customerId	7
country	179
city	6086
connType	12
isp	130
(endedFlag, customerId)	12
(endedFlag, customerId, country)	462
(endedFlag, customerId, country, city)	10813
(city, customerId, connType, isp)	19587

The performance of SDP was evaluated by the average cost of fetching records selecting single value on the focused column. Figure 4 shows the evaluated performance on the select queries on city, which was the last column in the composite key. The results show that by storing 20 % of records in an SSD, the average response time of the queries was reduced by 52 % of the maximum reduction which was obtained when all records were stored in an SSD. On the other hand, the reduction of the response time of 90th percentile was 75 % of the maximum reduction.

The results of city in Fig. 4 also shows that the response time of the 90th percentile is faster than that of the average. This is caused by the data skew in the city column; a small number of major cities in the column largely increases the average response time. In addition, the reduction of the response time of the 90th percentile is larger than that of the average. We consider this is due to the difference of the ratio of the seek and read time between major and minor values. On major values, because there are many records, the ratio of read time is larger than that of minor values. Therefore the performance gain by reducing the number of seeks could be suppressed for major values.

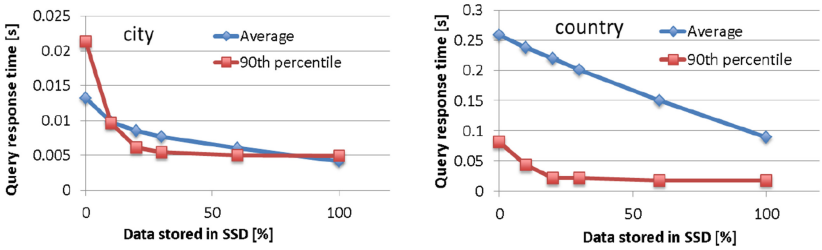
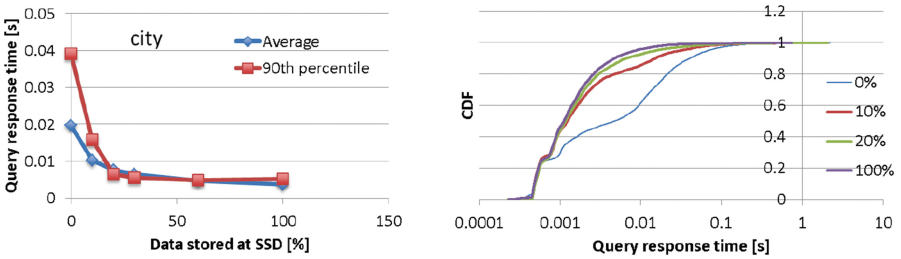


Fig. 4. Query response time on city and country in the first experiment.

In the first experiment, however, the nonlinear performance enhancement was not obtained in the preceding columns to the city in the composite key. Figure 4 also shows the performance of the queries on country, which was the third column in the composite key. Although the response time of the 90th percentile is slightly improved nonlinearly, the reduction of the average response time is linear. This shows that on the preceding columns including the country, the bottleneck of the performance to fetch records is the bandwidth of the sequential read of the HDD. Therefore, by storing some of records in an SSD, the performance is improved linearly and its slope is decided by the difference of the bandwidth of the SSD and the HDD.

In the second experiment, the different columns were chosen from the first experiment and the length of the records was halved to increase the affect of the seek of an HDD. The target columns for optimization were customerId, connType, isp, and city, and the composite key was configured using these columns in this order. However, also in this case, the bottleneck for all the column except the city was the read bandwidth of the HDD. Therefore, the order of the column was changed and the composite key of (city, customerId, connType, isp) was also tried. In this case, the bottleneck for all of the column was the seek time of the HDD. The cardinality of regarding columns are shown in Table 3.

Figure 5(a) shows the query response time on city when (customerId, connType, isp, city) was used as the composite key. Because the affect of the seek time is increased from one in the first experiment, by storing 20 % of records, 76 % of maximum reduction that is obtained when all records are stored in the SSD is obtained. Figure 5(b) shows the cumulative distribution function of the



(a) Query response time (b) CDF

Fig. 5. (a) Query response time and (b) its cumulative distribution function (CDF) on city in the second experiment. Legend in CDF is the ratio of records stored in SSD.

query response time on the city. When the ratio of the records stored in an SSD is increased, the response time rapidly approached to the performance with the maximum reduction of response time.

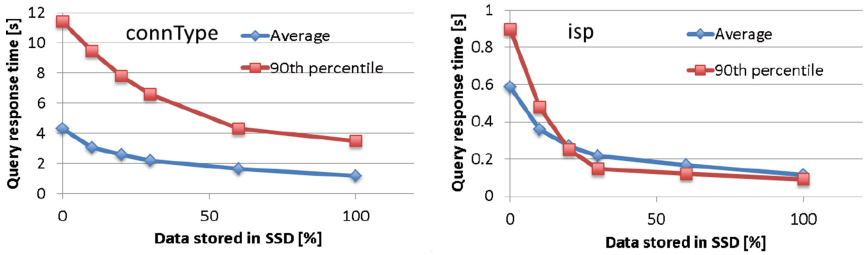


Fig. 6. Query response time on connType and isp in the second experiment.

Next, when (city, customerId, connType, isp) was used as the composite key, the nonlinear reduction of query response time was obtained for all of the target columns. Figure 6 shows the query response time on the connType which was third in the composite key and the isp which was the last. When 20 % of records are stored in an SSD, the average response time of the select queries is reduced by 55 % and 68 % of the maximum reduction for the connType and the isp, respectively.

These evaluation results showed that SDP provides nonlinear reduction of query response time against the ratio of the size of records stored in an SSD and an HDD. It also reduces the response time in multi-dimensional way. How large the nonlinear performance gain is depends on several factors such as the cardinality, the correlation, and the record size of the data stored in a system.

6 Related Work

There are several methods that have been proposed for hybrid database application. These methods can be broadly divided into two categories – ones that handle both an SSD and an HDD as different devices which configure the same storage tier [1] and others that use an SSD as an intermediate cache device between a host memory and an HDD [2,3,9]. The difference between the two is that the former provides increased I/O performance when data are accessed for the first time, while the latter does so from the second time. However, the former needs profiling of application to decide as to what data should be stored in an SSD which the latter does not.

All of these methods are based on reference locality of data stored in a database. By storing frequently accessed data in an SSD, system I/O performance is improved nonlinearly to the ratio of the size of data stored in an SSD and in an HDD. On the other hand, in this paper, we propose an alternative that focuses on data skew by storing records with values that appear less often in an SSD while those that do more in an HDD.

7 Conclusions

In this paper, we proposed SDP to enhance the performance of fetching records stored in a hybrid data store of SSDs and HDDs using indices on different columns. It is based on the data skew of stored data, and provides nonlinear performance gain to the ratio of records stored in SSDs and HDDs. Because SDP uses the data skew, unlike caching, it provides stable performance enhancement between first and second data accesses. It can also enhance performance even when a system memory is large enough to contain the working set of application. The evaluation of the implemented prototype using the data from the internet company showed that the performance of fetching records using different indices are simultaneously enhanced. By storing 20 % of records, up to 76 % of the maximum reduction of query response when all records are stored in an SSD is obtained. Our future work includes comparing the performance of SDP and others that are based on reference locality, and evaluation using real queries that are used in the data analysis.

References

1. Canim, M., Mihaila, G.A., Bhattacharjee, B., Ross, K.A., Lang, C.A.: An object placement advisor for DB2 using solid state storage. In: VLDB, pp. 1318–1329 (2009)
2. Canim, M., Mihaila, G.A., Bhattacharjee, B., Ross, K.A., Lang, C.A.: SSD bufferpool extensions for database systems. In: VLDB, pp. 1435–1446 (2010)
3. Do, J., Zhang, D., Patel, J.M., DeWitt, D.J., Naughton, J.F., Halverson, A.: Turbocharging DBMS buffer pool using SSDs. In: SIGMOD (2011)

4. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahy, R.: Design tradeoffs for SSD performance. In: 2008 USENIX Annual Technical Conference (ATC'08), pp. 57–70 (2008)
5. Walton, C.B., Dale, A.G., Jenevein, R.M.: A taxonomy and performance model of data skew effects in parallel joins. In: VLDB, pp. 537–548 (1991)
6. Stoica, I.: Warehouse-Scale Computing and the BDAS Stack. http://ampcamp.berkeley.edu/amp-camp-one-berkeley-2012/
7. Intel SSD Product Comparison. http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-ssd.html
8. Seagate Desktop HDD. http://www.seagate.com.edgekey.net/staticfiles/docs/pdf/datasheet/disc/desktop-hdd-data-sheet-ds1770-1-1212us.pdf
9. Liu, X., Salem, K.: Hybrid storage management for database systems. In: VLDB, pp. 541–552 (2013)