An Efficient Implementation of a Drug Candidate Database[†]

Zina Ben Miled,*[,‡] Yang Liu,[§] David Powers,[#] Omran Bukhres,[§] Michael Bem,[#] Robert Jones,[#] and Robert J. Oppelt[#]

Electrical Engineering Department, Purdue School of Engineering & Technology, Indianapolis, Indiana 46202, Computer & Information Science, Purdue School of Science, Indianapolis, Indiana 46202, and Eli Lilly & Company, Indianapolis, Indiana 46202

Received April 15, 2002

The recent advances in laboratory technologies have resulted in a wealth of chemical and biological data. The rapid proliferation of a vast amount of data has led to a set of cheminformatics and bioinformatics applications that manipulate dynamic, heterogeneous, and massive data. An example of such application in the pharmaceutical industry is the computational process involved in the early discovery of lead drug candidates for a given target disease. In this paper, an efficient implementation of a drug candidate database is presented and evaluated. This study shows that high performance data access can be achieved through proper choices of data representation, database schema design, and parallel processing techniques.

1. INTRODUCTION

Modern research in chemistry and biology increasingly depends on data-intensive applications. In the pharmaceutical industry, for example, the early drug discovery process produces data accumulations that challenge the limits of current database systems. With novel screening technology, chemists can synthesize and identify hundreds to thousands of new compounds every week.[1] Advanced computational tools[2] are also available to compute and explore tens of thousands of structure-based characteristics for each compound. For a typical pharmaceutical company, millions of compounds and their characteristics are accumulated in a drug candidate database and are available to scientists for model computation. Thus, an efficient data storage and high-throughput data access system is critical for routine screening of the massive drug candidate database in search of potential lead drug candidates for a given target disease.

The primary focus of this study is on the implementation of an efficient drug candidate database application that is used in the early drug discovery process. The inherent characteristics of this drug candidate database make it difficult to obtain a high data access throughput:

• **Dynamic:** The dynamic behavior results from frequent updates that can change the entire structure of the drug candidate database. These updates include adding new descriptors, new descriptor programs, and new compounds.

• **Heterogeneous:** The heterogeneous characteristics result from the fact that the data is derived from different descriptor programs. Each descriptor program has its own descriptor sets and data distribution pattern.

• **Large:** Technological advances have made it possible for a drug candidate database to contain a massive amount of compounds and descriptors.

In this paper, we show how a drug candidate database can be efficiently designed to overcome the impact of the heterogeneous, dynamic, and large size characteristics on data retrieval performance.

The driving application used in this study is described in Section 2. Schema design and data representation are discussed in Section 3. Section 4 presents the experimental results when parallel data accessing techniques are used. Tuning the degree of the parallelism in the application is discussed in Section 5. Related work is included in Section 6. Section 7 summarizes the findings of this study.

2. APPLICATION

Given a target disease, such as cancer, the drug discovery process consists of determining a pharmaceutical therapy for this target disease. This involves finding a strategic macromolecule that is pivotal to the disease state. Knowledge of the receptor site on the macromolecule allows the scientist to find new smaller molecules that will fit it, much like a key fits a lock. These molecules constitute lead drug candidates. The more accurate the fit, the more efficient the drug candidate is in binding to the receptor site, and thus a more potent drug candidate is realized. This study focuses on the computational process involved in extracting lead drug candidates from a database containing millions of drug candidates. Once a set of lead drug candidates is extracted for a given target disease, extensive structural modifications of the candidate's molecular structure are made and biological tests are conducted to identify at least one drug candidate for advanced studies. The objective is to select a suitable drug candidate that can be proven effective in the disease and thus may be sold as a pharmaceutical therapy for the target disease.

The database consists of the descriptor values for the drug candidates, which are computed by using several descriptor programs. Descriptors refer to numerical representation of a molecular attribute of the drug candidates. Each descriptor program computes a specific set of descriptors for a

* Corresponding author e-mail: zmiled@iupui.edu.
[†] This research was supported in part by Eli Lilly & Company.
[‡] Purdue School of Engineering & Technology.
[§] Purdue School of Science.
[#] Eli Lily & Company.

Drug Candidate	Descriptor	Value
1	$desc_1$	54
2	$desc_1$	55
2	$desc_2$	56
2	$desc_3$	1.5
...	...	...
6555	$desc_1$	32
6555	$desc_3$	2.4

Figure 1. Example of a descriptor program and its corresponding relational database table.

compound. These descriptors quantify the characteristics of the molecule that may be important in fitting the structure to the receptor site. Example descriptor programs include Fingerprints,[3] Jurs,[4] Topological Indices,[5] etc. For each descriptor program, the relation between the drug candidates, the descriptors, and the descriptor values is shown as a relational database table in Figure 1. The first column (attribute) of this table is the identification number for the drug candidate (Drug ID). The second column (attribute) identifies the descriptor for which the corresponding drug candidate has a value. The third column (attribute) contains the value of the corresponding descriptor (column 2) for the given drug candidate (column 1). These descriptor values are actual numerical values representing different characteristics of the molecule such as the number of sequences of a given set of atoms that are present in the molecule (drug candidate). Fingerprint is an example of a descriptor program that counts the number of a given set of atom sequences.

Drug candidates may not have values for all the descriptors in a given descriptor program. For example, if the descriptor represents the presence of a given sequence of atoms in the drug candidate, that sequence may not be present in all the drug candidates. The "missing value" is used to refer to the drug candidate-descriptor tuple for which a descriptor value does not exist. The concept of "missing value" leads to a classification of the table corresponding to a descriptor program as dense or sparse. The descriptor programs for which descriptor values are available for nearly all the drug candidates are classified as dense. The descriptor programs for which descriptor values may be available for only a subset of the entire set of drug candidates are classified as sparse. The density level is measured by calculating the percentage of drug candidates that have a descriptor value for a given descriptor. The average of these percentages is taken over all the descriptors for a given descriptor program table.

A typical drug candidate database may include millions of drug candidates and this number grows continuously. In this study, we start with a subset of the large database containing 350 000 drug candidates. In Section 5, to evaluate the impact of the database instance size on performance, the number of drug candidates is doubled and quadrupled to 700 000 and 1 400 000 drug candidates, respectively. The database used in this study contains 12 descriptor programs. These descriptor programs are representative of the descriptor programs that could be in an actual production database used in a pharmaceutical company. They cover different types of descriptor programs. The number of descriptors, number of drug candidates, number of descriptor values, and the average density level for each descriptor program are listed in Table 1.

During the drug discovery processes, the database is screened in order to extract lead drug candidates on the basis of a computational model. In the simplest form this model is the weighted sum of several descriptor values. For example, given a target disease D, the scientist may establish that the computational model for a lead drug candidate is $0.5\times d^5{}_1(x) + 0.3\times d^7{}_3(x)$, where $d^5{}_1(x)$ and $d^7{}_3(x)$ are the values of descriptor 1 from descriptor program 5 and descriptor 3 from descriptor program 7, respectively, for the drug candidate *x*. The higher the value of this weighted sum the greater the likelihood that the drug candidate will be an adequate pharmaceutical therapy for the disease D. Of course, the actual computational models are more complex than the above example. A typical computational model may include between 100 and 500 descriptors.

Table 1. Description of the 12 Descriptor Programs in the Drug Candidate Database with 350 000 Drug Candidates

descriptor program	number of descriptors	number of drug candidate	number of descriptor values	average percentage density (%)
01	8	335 859	2 527 330	94.1
02	149	339 893	16 822 044	33.2
03	49	339 893	16 654 757	100.0
04	15 265	339 850	8 644 612	0.17
05	208	326 932	5 135 743	7.55
06	2 048	339 893	71 297 638	10.2
07	88	337 407	29 691 816	100.0
08	2	339 814	679 628	100.0
09	102	27 822	2 672 238	94.2
10	36	337 915	4 616 811	38.0
11	557	338 176	137 374 818	72.9
12	30	131 812	3 934 800	99.5

Table 2. Single Item Record (SIR) Schema[a]

drug candidate ID	**descriptor program**	**descriptor name**	descriptor value

[a] The key is represented in bold.

When screening the database, scientists may give a customized drug candidate input list or require the entire database to be searched. In the first case, the drug candidate IDs are randomly selected from the database and the data access to the database is random. In the second case, the drug candidate IDs in the drug candidate input list are sequential numbers and the database access is consecutive. Using the drug candidate list given by the scientists and the descriptor list from the model, the corresponding descriptor values are retrieved from the database and used to apply the model to find the lead drug candidates.

3. SCHEMA DESIGN

The schema design for large databases is a complex task that can be further complicated by the inherent dynamic and heterogeneous characteristics of the drug candidate database. Schema design can have an impact on data access performance through reduced storage size and more efficient indexing. The major goal of the schema design in the case of the drug candidate database is achieving high data access rates.

3.1. Alternative Schemas. Tables 2 and 3 show two alternative schema for the drug candidate database. These schema are defined as follows:

• SIR (Single Item Record): each record consists of the drug candidate's ID, the descriptor program number, the descriptor name and the descriptor value. The entire drug

Table 3. All Items Record (AIR) Schema[a]

drug candidate ID	*desc*$_1$	*desc*$_2$	...	*desc*$_n$

[a] The key is represented in bold.

candidate database can be represented under this schema by using a single table. Three attributes-the drug candidate ID, the descriptor program number, and the descriptor name-form the primary key of this table. A primary key uniquely identifies a given record in the table. This primary key is identified in bold in Table 2.

• AIR (All Items Record): each record consists of the drug candidate's ID followed by the list of descriptor values for all the descriptors in a given descriptor program. The first attribute, the drug candidate ID, is the primary key of the table. The primary key for the AIR schema is identified in bold in Table 3. A null value is used to indicate that a given drug candidate has a missing value for a given descriptor. With this schema, 12 tables are needed to represent the drug candidate database, where each table corresponds to a given descriptor program.

A similar example to the one shown in Tables 2 and 3 was used in ref 6. As stated in ref 6, the SIR and AIR schemas contain semantically the same information and correctly portray the application domain. Additionally, it is easy to translate from the AIR schema to SIR schema and vice versa. However, the two schemas differ in their data access performance in the following four aspects:

• Indexing: Database indexing allows fast access to data records stored in the database. Most modern relational DBMSs use tree-based indexes (i.e., B+Tree). The size of the index, which is the number of index entries, is directly related to the number of internal tree nodes. In turn, access performance is based on the number of tree levels traversed in the path from the root to the leaf node which points to a spe cific record. The AIR schema has a smaller and more efficient index than the SIR schema. Since the indexes for the SIR and AIR tables are all based on the primary key, the number of index entries equals the number of data records in the table. For each AIR table, the number of index entries equals the number of drug candidates. For the SIR table, the number of index entries equals the total number of descriptor values (i.e., number of drug candidates $\times$ number of descriptors for a given descriptor program). The AIR tables have a simple primary key that consists of a single attribute (the drug candidate ID) represented in bold in Table 3. Whereas, the SIR tables have a composite primary key that consists of three attributes (drug candidate ID, descriptor program number, and descriptor name) as shown in bold in Table 2. The composite index key of the SIR schema makes its index structure much larger and with more redundant entries compared to the index for the AIR tables. The simple and small index structure of the AIR tables is more efficient and leads to a reduced search and data access time.

• Storage size: Depending on the density of the descriptor program, either the SIR schema or the AIR schema may lead to more efficient storage. Intuitively, using the SIR schema for a dense descriptor program is inefficient since the SIR schema allows redundancy in the drug candidate ID and descriptor program number. A reduction in storage size directly impacts data retrieval performance since it leads to a reduced traffic between disk and main memory.

• Query complexity: In a modern relational database, Structured Query Language (SQL) is used to access the data organized in a relational model. Schema design can affect the data access performance through different query algorithms. Queries based on simple database operations such as selection and projection are more efficient and faster than the queries based on complex database table join operations. In the SIR schema, a single SQL query is used to retrieve the descriptor values. However, this single query requires an expensive three table join operation. This join is between the input descriptor list, the input drug candidate list, and the descriptor program table. In the AIR schema, multiple SQL queries are needed to retrieve the descriptor values, where each query is a simple operation composed of selection and projection operations.

• Partitioning: Data partitioning splits very large tables and indexes into smaller and more manageable parts. This partitioning allows data access to be performed against the partitions rather than the entire table or index. Thus, data partitioning can improve data access performance by reducing the search space for a given query, and it can lead to increased disk I/O performance when parallel processing techniques are used. The SIR table can be partitioned according to the descriptor program. In this case, the descriptor values from the same descriptor program are stored in the same partition. The partitioned SIR table is more comparable to the AIR tables. The descriptor values in different partitions can be distributed across different disks. Similarly, different AIR tables can be stored on different disks. The SIR table used in this study is partitioned according to the descriptor program.

The SIR and AIR schema also differ from a maintenance perspective. Under the SIR schema, one table is needed for the entire database, whereas the AIR schema requires separate tables for each descriptor program. The entire database for the SIR schema uses a uniform data representation. Thus, the same query can be used to access or manipulate data in different descriptor programs. For the AIR schema, since different descriptor programs use different tables, each table has its own structure. Thus, different queries are used to retrieve descriptor values from different descriptor programs.

Adding descriptors or descriptor programs does not alter the SIR schema since the addition affects only the content of the table. With partitioned SIR table, new descriptor programs are added into the database by creating additional table partitions without changing the underlying schema. However, adding descriptors or descriptor programs to the AIR schema requires a schema transformation. If new descriptors are added to an existing descriptor program, new attributes need to be added to the AIR table. In the case of adding a new descriptor program, a new AIR table and a new set of queries to manipulate the data in the new table are needed.

The AIR schema can suffer from practical constraints imposed by the DBMS, whereas the SIR schema may not. For example, certain DBMSs such as Oracle[7] impose a limit on the number of attributes in a table. The AIR schema may exceed those limits. In this pharmaceutical drug candidate database, two descriptor programs, 4 and 6, exceed the 1000 attributes limit imposed by Oracle8i.

Table 4. Data File Size for Each Descriptor Program in the Drug Candidate Database Using Oracle DBMS

descriptor program	SIR schema (MBytes)	AIR schema (MBytes)	average percent density
01	83.9	41.9	94.1
02	304.1	104.8	33.2
03	346.0	104.8	100.0
04	199.2	N/A	0.17
05	104.9	104.8	7.55
06	1467.9	N/A	10.2
07	629.1	209.71	100.0
08	31.4	21.0	100.0
09	73.4	41.9	94.2
10	125.8	41.9	38.0
11	2778.7	524.3	72.9
12	157.3	104.8	99.5

Table 4 shows the sizes in MBytes of the tables for each descriptor program under the SIR and AIR schema with 350 000 drug candidates. This table indicates that with the exception of 4, 5, and 6, all the other descriptor programs have smaller size tables under AIR than under SIR. The above result is counter-intuitive, since it was expected that dense tables have a smaller size tables in the AIR schema than in the SIR schema, whereas the opposite should be true for sparse tables. The average percentage density measure of Table 1 is repeated in Table 4 for convenience. In a different study,[8] the storage size of the descriptor program was shown to be sensitive to the density of the descriptor values when the same implementation is used for all the descriptor programs under the two schema. The counter-intuitive results of Table 4 show that the schema of choice is implementation dependent. Thus, an analytical approach to schema selection such as the one proposed in ref 9 may not lead to an optimal solution.

The sizes of the data files for descriptor program 4 and 6 under AIR schema are missing from Table 4 since the number of attributes, 15 265 and 2048 of descriptor program 4 and 6, respectively, exceeds the 1000 attribute limit imposed by Oracle.[10,7]

3.2. Data Access Performance of Individual Descriptor Program under SIR and AIR Schema. To gain insight into the impact of AIR and SIR schema on performance, experiments were designed to compare the data access time for retrieving descriptor values from a single descriptor program under different schema.

The platform used to conduct the experiments consists of a Sun Enterprise E3000 running Oracle 8i.[7,10] All of the applications were developed by using the Oracle OCI interface. The drug candidate database created in Oracle8i includes 350 000 drug candidates with descriptor values from 12 descriptor programs. A single SIR table partitioned by the descriptor program is created in the database by using the schema defined in Table 2 for all 12 descriptor programs. In addition, 10 separate AIR tables were created in the same Oracle database instance by using the schema defined in Table 3 for all the descriptor programs except descriptor programs 4 and 6.

The descriptor list includes a set of 100 randomly selected descriptors from 10 out of the 12 descriptor programs. Descriptor programs 4 and 6 were excluded because they cannot be represented by using the AIR schema. The choice of 100 for the number of descriptors is driven by the fact that it corresponds to the average number of descriptors requested by the scientist in a given experiment. Two drug candidate lists including 5000 consecutive and random drug candidate IDs are used in this experiment. The time used to retrieve descriptor values from each single descriptor program are recorded and shown in Figures 2 and 3.

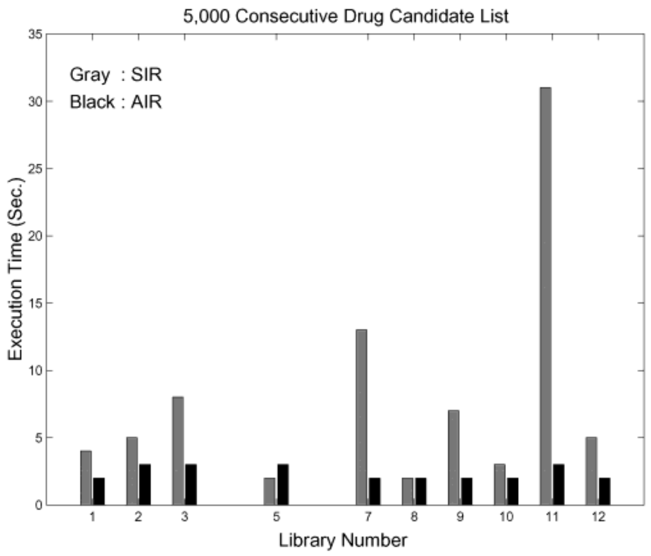

Figure 2. Performance of single descriptor program when the 5000 consecutive drug candidate list is used as input.

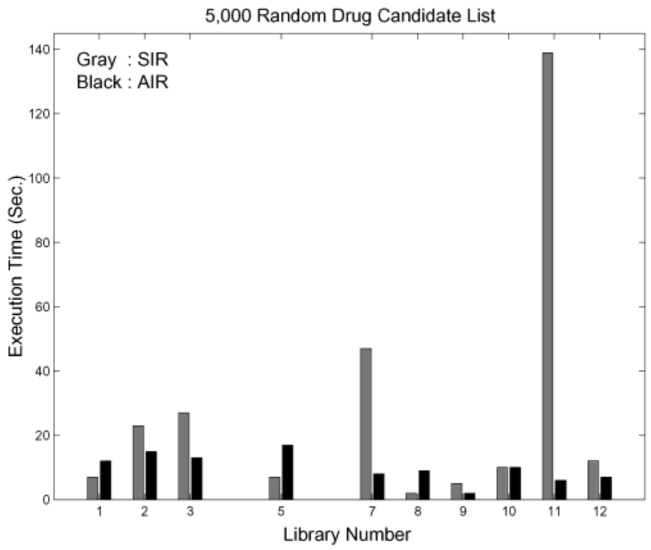

Figure 3. Performance of single descriptor program when the 5000 random drug candidate list is used as input.

Figures 2 and 3 indicate that, with the exception of descriptor programs 5 and 8, the AIR schema is the schema of choice for all the descriptor programs and types of access patterns under study. The times for descriptor programs 4 and 6 are not shown in the Figures 2 and 3 because these two descriptor programs cannot be represented by the AIR schema. These two figures show the substantial gain that can be achieved by transforming a descriptor program table from SIR to AIR. For example, descriptor program 11 shows more than 6-fold improvement with this transformation for both random and consecutive access. This substantial improvement is due to the fact that descriptor program 11 has a large number of descriptors and a high-density level. By transforming it from the SIR schema to the AIR schema, the size of the data files and the complexity of the indexes for desciptor program 11 are reduced dramatically. When

Table 5. Performance of the SIR and Hybrid Schema When the Consecutive Input Drug Candidate List Is Used as Input

drug candidate ID list (size)	SIR schema		hybrid schema	
	time (sec)	throughput	time (sec)	throughput
5000	116	43.10	48	104.17
10 000	272	36.76	98	102.04
20 000	721	27.74	190	105.26

Table 6. Performance of SIR and Hybrid Schema When the Random Input Drug Candidate List Is Used as Input

drug candidate ID list (size)	SIR schema		hybrid schema	
	time (sec)	throughput	time (sec)	throughput
5000	265	18.87	202	24.75
10 000	376	26.60	267	37.45
20 000	577	34.66	346	57.80

the access to the data from the entire drug candidate database instead of each individual descriptor program is measured, the AIR schema can improve the throughput by a factor of 3 compared to the SIR schema.[11]

The SIR schema leads to a slightly better performance than that of the AIR schema for descriptor programs 5 and 8. Descriptor program 5 is very sparse. Its average percentage density is less than 10% (Table 1). Descriptor program 8 has only two descriptors (Table 1). These two results hint that the schema of choice may be dependent on the density level as well as the cardinality of the descriptor program.

3.3. Data Access Performance under SIR and Hybrid Schema. In the previous experiment, the AIR schema shows better data access performance in almost all the descriptor programs compared to the SIR schema. Because of practical limitations, the entire drug candidate database cannot be represented by using the AIR schema. To circumvent this practical limitation imposed by Oracle[7,10] on the number of attributes in a given table, a hybrid schema in which descriptor program 4 and 6 are represented by using the SIR schema and the remaining descriptor programs are represented using the AIR schema is constructed.

The performance of the hybrid schema is compared to the case where all the descriptor programs are represented by using the single SIR schema. A descriptor list containing 100 descriptors randomly selected from all 12 descriptor programs is used in this experiment. Besides the 5000 drug candidate lists used before, 10 000 and 20 000 drug candidate lists with both random and consecutive drug candidate IDs are added to study the data access throughput under varying workloads. The results of this comparison are shown in Tables 5 and 6 for the consecutive and random drug candidate lists, respectively. In addition to the execution times, Tables 5 and 6 also show the access throughput as defined by Equation 1. This measure allows a performance comparison for increasing numbers of drug candidates in the input drug candidate list.

$$\text{throughput} = \frac{\text{number of drug candidates}}{\text{execution time in seconds}} \qquad (1)$$

The results of this experiment indicate that the hybrid schema has better performance than the SIR schema. Furthermore, the performance of the hybrid schema remains constant as the number of drug candidates increases for the consecutive drug candidate input list (Table 5). The performance of the hybrid schema improves with increasing number of drug candidates in the random drug candidate input list (Table 6).

The hybrid schema is an efficient alternative to the SIR and AIR schema because (1) it does not suffer from the practical limitation of the AIR schema, and (2) it achieves better performance than the SIR schema.

4. PARALLEL DATA ACCESS

The experimental results presented in Section 3 show that the hybrid schema, which is the combination of AIR and SIR data representations, leads to an increase in throughput compared to the SIR schema. Since the data access under the hybrid schema requires multiple database queries (i.e., each query retrieves descriptor values from a single descriptor program table), to improve throughput, these queries can be implemented in parallel. In this section, the throughput obtained using parallel query execution is compared to that of the sequential execution.

4.1. Implementation of Parallel Data Access Techniques. There are two different levels at which parallelism can be exploited:[12,13] inter-query parallelism and intra-query parallelism. Inter-query parallelism allows multiple queries to be executed concurrently. This type of parallelism is used for queries from multiple users. Intra-query parallelism allows different sections of a single query to be executed concurrently.

During the data access to the drug candidate database, the SIR schema can use intra-query parallelism because a single SQL query is used to retrieve data. Under the hybrid schema, neither one of the two types of parallelism meets the drug candidate database process requirement directly. Multiple queries are issued to retrieve the data under the hybrid schema, and each query retrieves descriptor values from a single descriptor program table. The multiple queries share the same data during their execution (i.e., the input drug ID list and the input descriptor list), and the entire data access process should be considered as an atomic transaction rather than separate user requests. Furthermore, intra-query parallelism cannot be exploited efficiently since there are multiple separate queries.

To exploit parallelism under the hybrid schema, a thread-based parallel data access application was designed. Multithreading is a parallel processing paradigm that is based on lightweight threads running concurrently on a shared multiprocessor system (SMP). Threads are lightweight compared to processes because they share the same memory space. Multithreading is attractive for applications, such as databases, where parallelism can only be exploited through extensive data sharing between concurrent threads or processes.

Under the multithreaded data access application, multiple threads are spawned, and each thread retrieves descriptor values from a single descriptor program. The degree of the parallelism can be adjusted explicitly to achieve optimal performance. The multithreaded application is implemented by using the Oracle OCI thread package.[10]

4.2. Performance of the Multithreaded Data Access. For the multithreaded application, the drug candidate input list and the descriptor input list used are the same as the ones used to compare the SIR and hybrid schema in Section

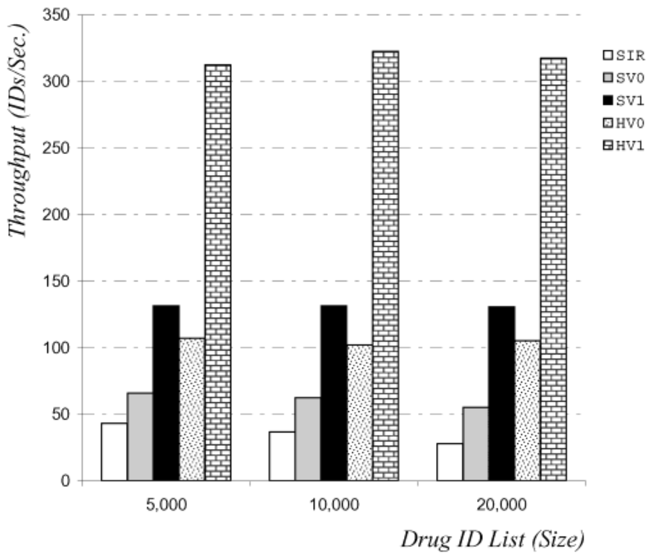

Figure 4. Parallel versus nonparallel performance using the consecutive drug ID input list.

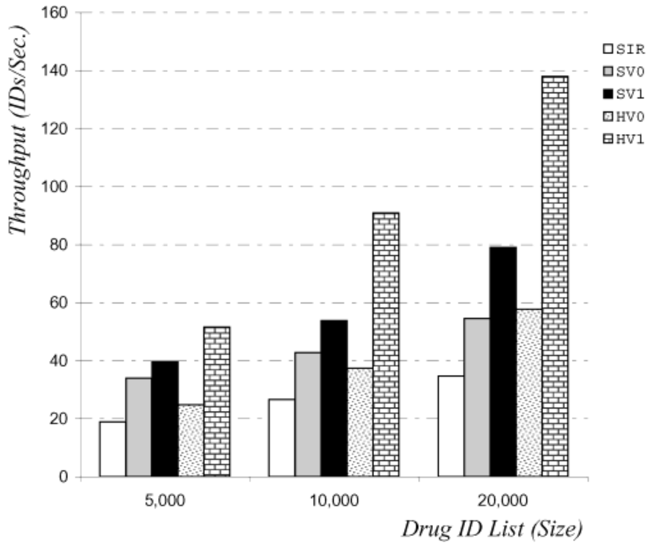

Figure 5. Parallel versus nonparallel performance using the random drug ID input list.

3.3. The tables under SIR and hybrid schema created in the oracle database are also the same.

Under the SIR schema, two different parallel processing approaches were used. In the first approach, parallelism is achieved through intra-query parallelism by supplying a parallelism hint in the database query. The degree of parallelism is set to 12, which is equal to the number of partitions in the SIR table. In this approach, the parallel query processing is fully dictated by the Oracle query optimizer, and thus it is transparent to the user. In the second approach, multiple threads are spawned where each thread retrieves data from a single partition of the SIR table, and each partition corresponds to a single descriptor program. The degree of parallelism is also set to 12. Under the hybrid schema, only the second approach is used.

The test results using the consecutive and random drug candidate input lists are shown in Figures 4 and 5, respectively. The original nonparallel approach under the SIR schema is denoted by SIR. The intra-query parallelism approach under the SIR schema is denoted by SV0. The multithreaded approach under the SIR schema is denoted by SV1. The original nonparallel approach under the hybrid schema is denoted by HV0. The multithreaded approach under the hybrid schema is denoted by HV1.

Table 7. Multithreaded Data Access Applications with Different Degree of Parallelism

application	total number of threads	number of threads per descriptor program
HV1	12	1
HV2	24	2
HV3	36	3
HV4	48	4

The following observations can be derived from the above experiments.

1. The multithreaded approach under the hybrid schema has the best performance in all the tests.

2. A comparison of the SV0 and SV1 with the original SIR shows the throughput of the multithreaded approach is higher than the approach that uses the Oracle intra-query parallelism and the original SIR for both consecutive and random data access.

3. The performance improvement gained through the multithreaded approach in the case of the consecutive drug input list is higher than that in the case of the random drug input list. In the consecutive data access case shown in Figure 4, the improvement is by a factor of more than 3, whereas in the random data access case shown in Figure 5, the improvement factor is about 2.

4. For the multithreaded approach, the performance with varying size of consecutive drug input list is nearly constant. However, the performance in the case of the random drug input list scales up with increasing number of drug candidates in the drug candidate input list.

Given that the multithreaded data access approach under the hybrid schema had the best performance, it was selected to study the degree of parallelism which is discussed in the next section.

5. TUNING THE DEGREE OF PARALLELISM

Using the multithreaded data access application, the degree of the parallelism can be specified explicitly by varying the number of threads that are spawned per descriptor program. When the number of threads is one per descriptor program, each thread will be used to query a single descriptor program table. This case is the HV1 case described in Section 4.2. If there is more than one thread per descriptor program, the input drug candidate list is partitioned into a number of ranges equal to the number of threads per descriptor program, and each thread is assigned one of these ranges. For example, when three threads are issued per descriptor program, the input drug candidate list is divided into three partitions and each of these partitions is assigned to one thread. Besides the HV1, three more multithreaded data access applications with different degree of parallelism, HV2, HV3, and HV4, are used in this experiment. These applications are described in Table 7.

Several tests were conducted to explore the machine scalability and the impact of the dynamic data on the data access to the drug candidate database.

5.1. High-End versus Midrange SMPs. To explore the machine scalability of the multithreaded data access application under hybrid schema, the data access throughput of a

Table 8. Configuration of Midrange Sun E3000 SMP

CPU	4 × 400 MHz 1MB Ecache
memory	1GBytes
disk	A5000 disk array: 14 × 18GBytes 10k rpm disks

Table 9. Configuration of High-End Sun E10000 SMP

CPU	24 × 400 MHz 4MB Ecache
memory	24GBytes
disk	12 × A5200 disk arrays 22 × 9GBytes 10k rpm disks

Midrange SUN E3000 and a High-End Sun E10000 server were compared. Each machine may have "an optimal" number of threads. Issuing more threads that a system can handle will mostly lead to performance degradation. Therefore, investigating the degree of parallelism is very important in migrating the application between computer systems with different capabilities.

The system configurations of the E3000 and the E10000 servers are presented in Tables 8 and 9, respectively. An Oracle database instance under the hybrid schema was created on both systems containing 350 000 drug candidates. The only difference between these two databases is the physical data layout. On the E3000, two descriptor program tables share the same physical disk. On the E10000, each database table corresponding to a single descriptor program is stored on a separate virtual disk.[14] Each virtual disk is striped across six disk arrays. The distribution of the data from different descriptor program tables on separate virtual disks is designed to reduce the I/O contention when data from different descriptor program tables is accessed in parallel.

The multithreaded data access applications, HV1 to HV3, are compared in this test. Increasing the number of threads increases the degree of parallelism. However, increasing the number of threads beyond a certain threshold can also lead to resource contention which can in turn lead to reduced performance. The optimal number of threads is application-specific because it depends on the number of computations and the number of I/O accesses performed by each thread.

The descriptor list used in this test is the same as the one used in Section 4.2. Two new drug candidate input lists including 50 000 consecutive and random drug IDs are used in addition to the previous 10 000 drug ID input lists.

Figure 6 shows the throughput of HV1 through HV3 running on the E10000 and the E3000 with 10 000 consecutive drug candidates and 100 descriptors. The random drug candidate case is shown in Figure 7. In Figures 8 and 9, the number of drug candidates is increased to 50 000.

A comparison of the throughput delivered by the E3000 and the E10000 (Figures 6, 7, 8 and 9) shows that the throughput scales up as more computing resources are added. Furthermore, the number of threads affects the throughput delivered by each machine differently. For the example of the 10 000 consecutive drug candidates, the performance delivered by the E10000 increases as the number of threads increases, whereas that delivered by the E3000 remains nearly constant (Figure 6). In the case of the 10 000 random drug candidates (Figure 7), the two machines experience opposite trends: the throughput delivered by the E10000 increases, whereas the throughput delivered by the E3000 decreases for increasing number of threads.

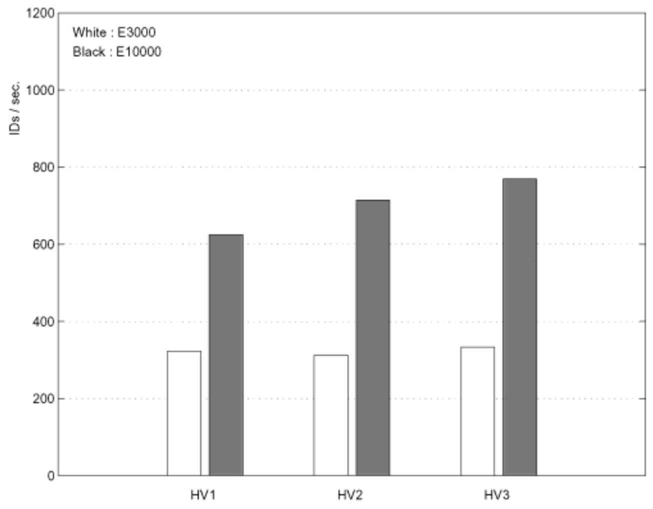

Figure 6. Multithreaded data access performance for the E3000 and the E10000 when the 10 000 consecutive drug candidate ID input list is used.

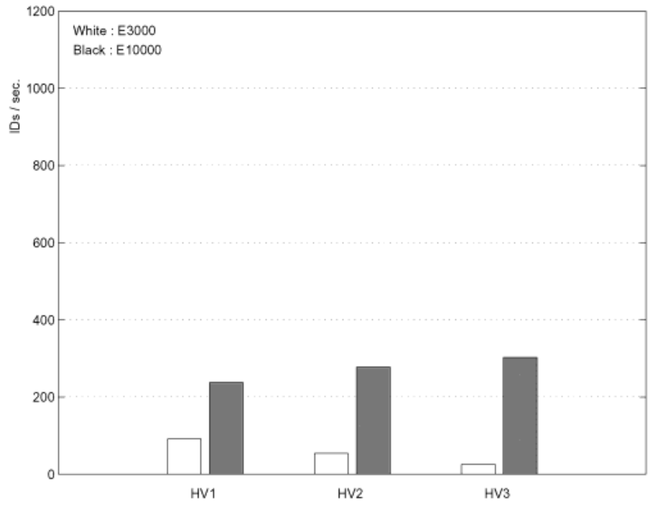

Figure 7. Multithreaded data access performance for the E3000 and the E10000 when the 10 000 random drug candidate ID input list is used.

To explain this behavior, several system parameters were monitored[15] during the application execution on the E3000, including the number of context switches and the portion of time running in user mode with some process waiting for I/O. The value of this latter parameter explains the difference between the consecutive access and the random access behavior for the E3000. In the consecutive access case, requests for I/O from multiple threads can be consolidated because of the spatial locality properties of the data on-disk. Consolidating requests for I/O from multiple threads in the random access case is often not possible because the target data is randomly distributed across the database. Thus, increasing the number of threads increases the mismatch between the degree of parallelism in the application and the degree of I/O parallelism available in the machine. Efficient disk access mechanisms can hide this mismatch in the case of consecutive accesses but not in the case of random accesses to the database.

Contention for I/O resources in the random data access case can be alleviated by increasing the number of disk arrays attached to the E3000. The same behavior was not observed on the E10000 because the physical data layout on this

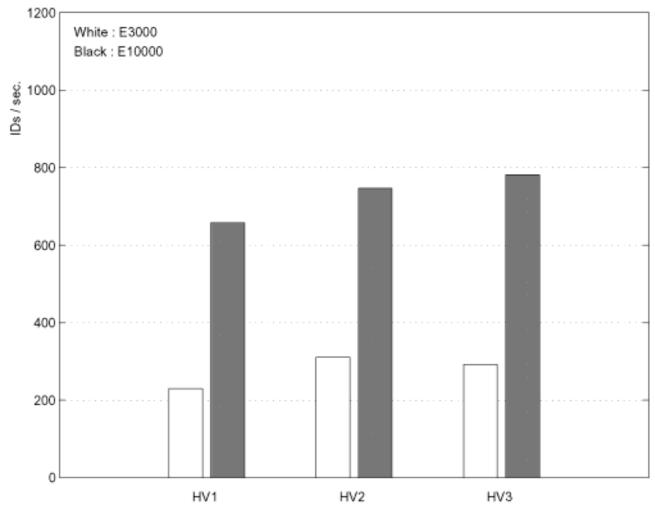

Figure 8. Multithreaded data access performance for the E3000 and the E10000 when the 50 000 consecutive drug candidate ID input list is used.

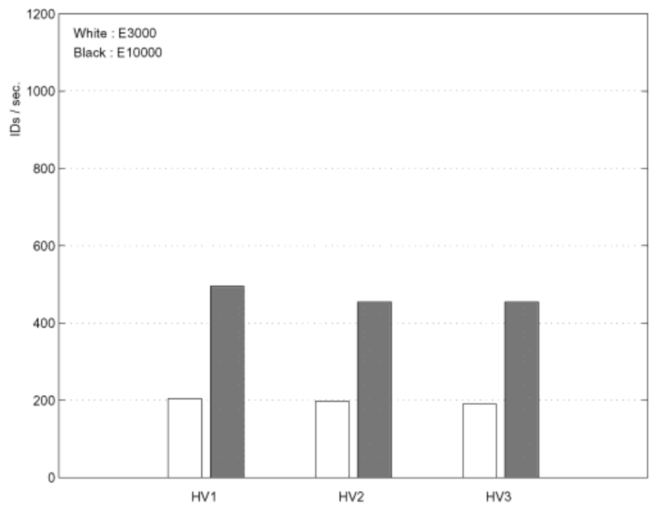

Figure 9. Multithreaded data access performance for the E3000 and the E10000 when the 50 000 random drug candidate ID input list is used.

machine was distributed across 12 virtual disks. Thus, the impact of I/O contention on E10000 is minimized.

When the workload increases to 50 000 drug candidates, both the consecutive and random access have higher throughput on the E10000. This is also true on the E3000 in the random access case. The exception to this trend is for the E3000 during consecutive access to the database. The 10 000 consecutive drug candidates case (Figure 6, E3000) has higher throughput than the 50 000 drug candidates case (Figure 8, E3000) for HV1, HV2 and HV3. For the consecutive case, the percentage of the process waiting for I/O time increases as the size of the workload increases. For the random case, this parameter decreases as the size of the workload increases. This can be explained by the fact that in the consecutive access case, the percentage of the data retrieved from disk that is needed by the application is high. Increasing the load will increase the amount of data retrieved, but it will not change the percentage of the data retrieved from disk that is needed by the application. In the random access case, this percentage increases as the size of the load increases.

When evaluating the impact of increasing the number of threads for different workload sizes, we can make the following observations. For the consecutive access and 50 000 drug candidates case (Figure 8), the throughput increases as the number of threads increases. This is similar to the trend shown in Figure 6. However, in the case of the random access to the database, the 10 000 (Figure 7) and the 50 000 (Figure 9) drug candidates input lists yield different throughput trends. In particular, the throughput for the E3000 decreases as the number of threads increases for the input list with 10 000 drug candidates. For the 50 000 drug candidates input list, the throughput remains nearly the same as the number of threads increases. This is the result of different I/O demands. For the first case, process waiting for I/O time increases sharply as the number of threads increases. In the second case, the differences in process waiting for I/O time among HV1, HV2 and HV3 are small, thus explaining the nearly equal throughput for these three applications.

One of the major findings of this experiment is that most of the performance gain was obtained with the implementation of the thread-based solution. Increasing the number of threads may lead to about 10% improvement for a high-end machine that can handle the additional threads. However, if this machine is not capable of efficiently handling the I/O from the added threads, a substantial loss of performance can be observed. This case applies to the E3000 as shown in Figure 7.

5.2. Impact of Dynamic Data. The data in the drug candidate database under consideration is highly dynamic. The number of drug candidates can increase rapidly. To evaluate the impact of this dynamic characteristic of the data, the following experiment was designed and conducted on the E10000 with the configuration shown in Table 9.

In addition to the 350 000 drug candidate database used before, two new database instances, 700 000 and 1 400 000 drug candidates, were created under the hybrid schema. The new database instances also have 12 descriptor programs, and the descriptors in each descriptor program remain the same.

For each database instance (350 000; 700 000 and 1 400 000), three tests were conducted. In the first test, the number of descriptors in the input descriptor list was 100 and the number of drug candidates in the input drug candidate list was 10 000. In the second experiment, the number of drug candidates was raised to 50 000. In the third experiment, the number of descriptors was set to 500 and the input drug candidate list contained 10 000 drug candidates. The same descriptor lists and consecutive drug candidate lists were used in the experiment for all three database instances. However, the random drug candidate lists were constructed differently based on the drug candidate ID included in each drug candidate database instance.

The normalized throughput was measured as follows:

$$\text{normalized throughput} = \frac{\text{size of drug ID list} \times \text{size of descriptor list}}{100 \times \text{execution time in seconds}} \quad (2)$$

A normalization factor of 100 was selected because the smallest computational model contains 100 descriptors.

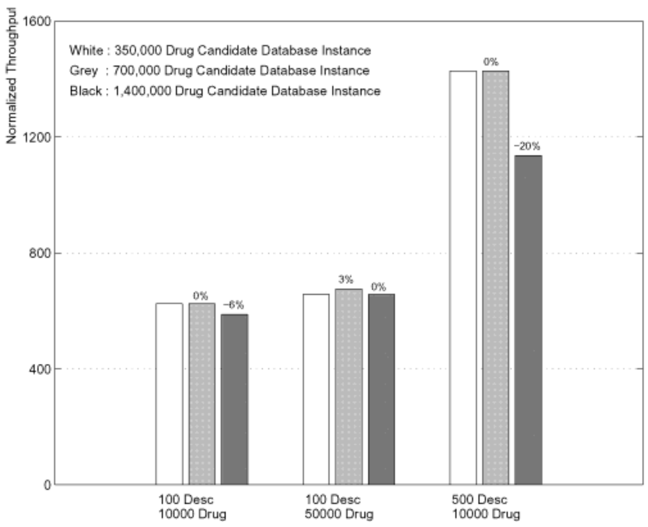

Figure 10. Normalized throughput of HV1 with consecutive drug candidate input list.

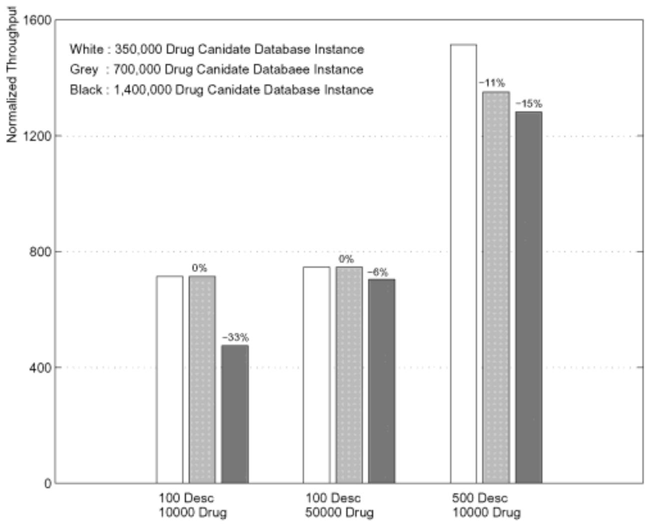

Figure 11. Normalized throughput of HV2 with consecutive drug candidate input list.

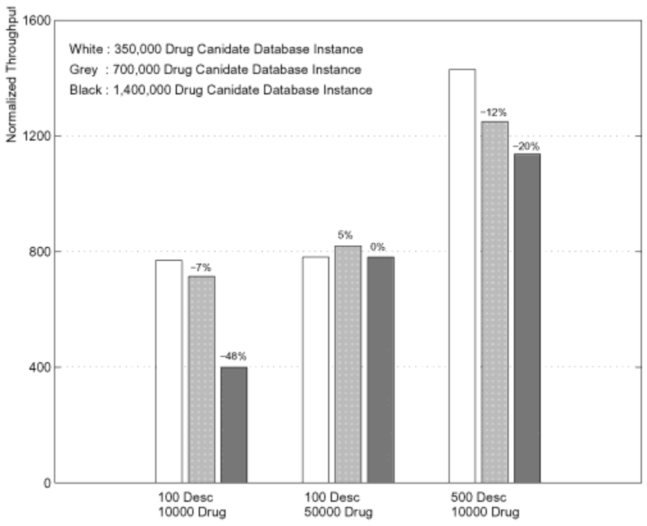

Figure 12. Normalized throughput of HV3 with consecutive drug candidate input list.

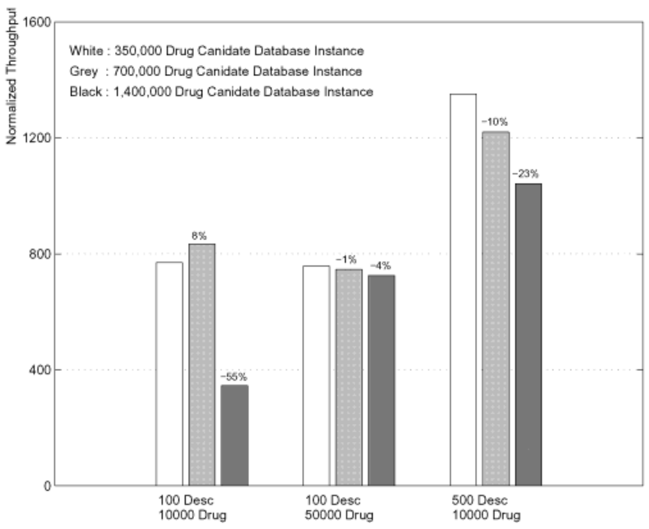

Figure 13. Normalized throughput of HV4 with consecutive drug candidate input list.

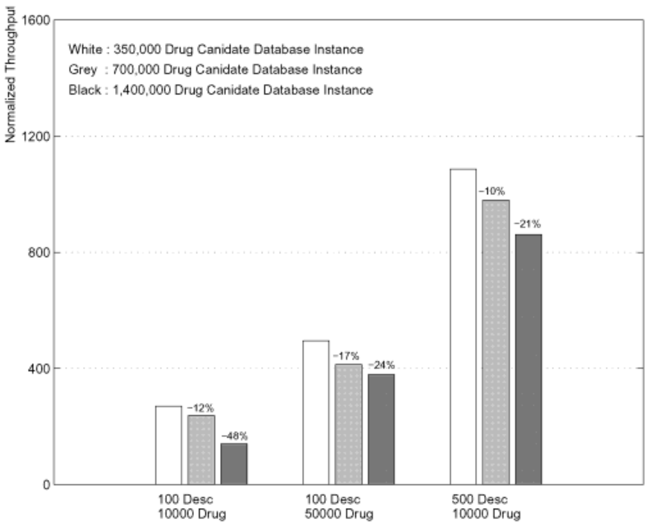

Figure 14. Normalized throughput of HV1 with random drug candidate input list.

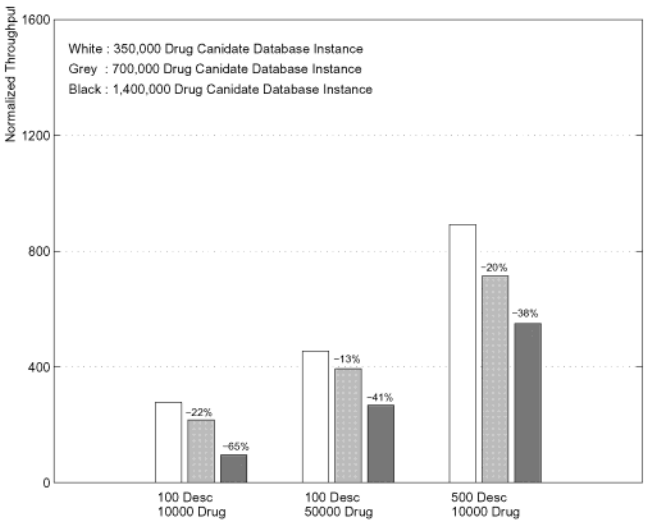

Figure 15. Normalized throughput of HV2 with random drug candidate input list.

Figures 10, 11, 12 and 13 show the performance of HV1 through HV4 in the case where consecutive drug candidate input lists are used. Similarly, Figures 14, 15, 16 and 17 show the performance of these applications in the case where random drug candidate input lists are used. The percentage on each bar in all the figures represents the percentage change

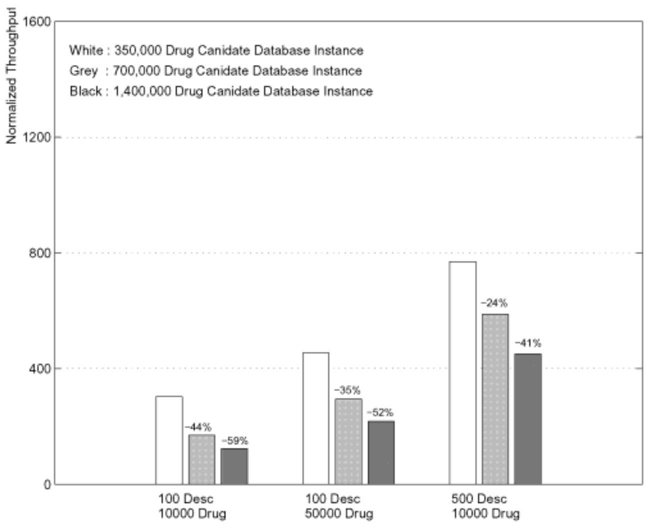

Figure 16. Normalized throughput of HV3 with random drug candidate input list.

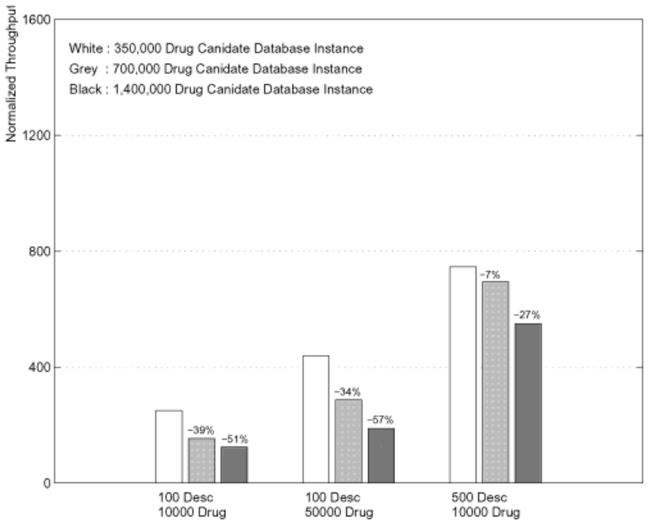

Figure 17. Normalized throughput of HV4 with random drug candidate input list.

in throughput compared to the database instance with 350 000 drug candidates.

For the input drug candidate list with consecutive drug candidate IDs (Figures 10, 11, 12 and 13), the performance of the applications HV1 through HV4 increases when the number of descriptors increases. Furthermore, in each of these cases, the throughput for the database with 1 400 000 drug candidates is consistently lower than the other two database instances. Additionally, for the large size database, increasing the number of threads results in a lower throughput for the consecutive drug candidate input list. For example, the throughput for the 1 400 000 drug candidate database is above 600 for HV1 (Figure 10) and it is below 400 for HV4 (Figure 13) when the consecutive drug candidate input list contains 10 000 drug candidates and the descriptor input list contains 100 descriptors. This behavior is more prominent for small descriptor lists.

Figures 14, 15, 16 and 17 show the throughput associated with random access to the database. As in the case of the consecutive access to the database, this throughput also increases when the number of requested descriptors increases. This trend is due to the fact that most of the descriptor program tables are in AIR format. In this format, descriptors for a given compound and from a given descriptor program table are stored in the same record on disk. If multiple descriptor values are needed from a given descriptor program for a drug candidate, the AIR format reduces the amount of data that has to be fetched from disk. Increasing the number of requested descriptors increases the likelihood that more than one descriptor will be retrieved from the same descriptor program table. Additionally, Figures 14, 15, 16 and 17 show that the throughput decreases as the number of threads per descriptor program increases.

Comparing Figures 10, 11, 12 and 13 with Figures 14, 15, 16 and 17 shows that higher throughput is obtained for consecutive accesses. In general, the data access process, which requires consecutive access to the database, is more efficient than the data access process which requires random access to the database. Also, the application favors screening workloads with more descriptors.

6. RELATED WORK

Several ongoing database research projects[16,17] are related to the computational-based drug discovery process. In ref 17, a data warehouse using a dimensional data model[18] is designed for the pharmaceutical drug discovery research. This data warehouse stores the drug candidate structure data. The attributes for the drug candidate associated with the pharmaceutical data warehouse used in ref 17 are static, which is different from the dynamic pharmaceutical drug candidate database discussed in this paper.

The process of conceptual schema design can be facilitated in traditional database applications through the use of the enhanced entity relation model.[19] In this case, the conceptual schema always results from a clear understanding of the relationships between the different entities in the application. Traditional business databases are relatively easy to conceptualize since the relationships between different entities are known a priori and are well understood. For nontraditional databases[20] and for scientific databases, this is often not the case. As stated in ref 20, for this type of application, a rigid schema can lead to a noneffective design.

There are several studies that provide foundations for conceptual schema design;[9,21] however, there are fundamental differences between these studies and the approach adopted in this paper. In refs 9 and 21, an analytical approach to cost-efficiency of schema design is adopted, whereas in this paper the approach used is experimental and is guided by the results obtained through the analysis of the execution time of different access approaches.

7. CONCLUSION AND FUTURE WORK

In this paper, a schema was designed to overcome the impact of the heterogeneity of the data on performance in a drug candidate database. By comparing various queries issued against the AIR and SIR schema, it was concluded that the efficient data representation, single key index structure, and simplified data access path under the AIR schema is more efficient for both consecutive and random data access, and it can lead to as much as a factor of 3 improvement in performance.

Because of the practical limitation imposed by DBMS as well as the density level factor observed in another study,[8] a hybrid schema, which is the combination of AIR and SIR schema, was developed to represent the entire drug candidate database. The hybrid schema does not hide the heterogeneity of the data from the user. The performance gain afforded through the hybrid schema comes at the expense of an increase in the complexity of the application program and database maintenance. This fact suggests one direction for future work: The selection of the schema for performance purposes needs to be incorporated into the internal schema design level in a DBMS, which will abstract the complexity of some of the schema from the user view.

A multithreaded approach combined with the hybrid schema was designed to support parallel data access in the drug candidate database. The multithreaded application improves the performance by another factor of 3 over the nonparallel application. This multithreaded approach also achieves better performance than the Oracle inter-query parallelism.

Since the degree of parallelism can be set explicitly when using the multithreaded approach, several tests were conducted to investigate the impact of dynamic and massive characteristics of the data on performance for a Midrange Sun E3000 and a High-End E10000. These tests show that the degree of exploited parallelism should be tuned to avoid I/O contention especially for random accesses to the drug candidate database. Additionally, the optimal degree of parallelism is machine specific. Fewer threads should be spawned on a Midrange SMP than on a High-End SMP, particularly for random accesses to the drug candidate database.

In conclusion, optimal access performance to a drug candidate database can be obtained by 1) using the appropriate data representation in order to overcome the impact of the heterogeneity of the data on performance and 2) selecting a new degree of parallelism as the database instance increases or decreases in order to reduce the impact of the dynamic and massive characteristics of the data on performance. Additionally, as the application migrates from one machine to another, the degree of parallelism should be tuned to the new machine in order to avoid loss of performance.

In this study, all the experiments conducted are based on querying by compound IDs. Data search and retrieval from a pharmaceutical drug candidate database may also include structure-based similarity queries or descriptor values based range queries. Future work includes the analysis of the performance of these latter types of queries.

REFERENCES AND NOTES

(1) Borman, S. Combinational Chemistry. *Chem. Eng. News* February 24th 1997.

(2) Finn, P.; Kavraki, L. Computational Approaches to Drug Design. *Algorithmica* **1999**, *25*, 347−371.

(3) Wild, D.; Blankley, C. J. Comparison of 2D Fingerprint types and hierarchy level selection methods for structural grouping using Ward's clustering. *J. Chem Inf. Comput. Sci.* **2000**, *40*, 155−162.

(4) Stuper, A. J.; Brugger, W. E.; Jurs, P. C. *Computer-Assisted Studies of Chemical Structure and Biological Function;* Wiley-Interscience: New York, 1979.

(5) Balaban, A. T. Highly Discriminating Distance-Based Topological Index. *Chem. Phys. Lett.* **1982**, *89*(5), 399−404.

(6) Miller, R. J.; Ioannidis, Y. E.; Ramakrishnan, R. Understanding Schemas. *Res. Issues Data Eng.: Interoperability Multidatabase Systems* **1993**, 170−173.

(7) Oracle Corporation. *8.1.5 Oracle8i Generic Documentation Set Release 8.1.5*. Oracle Press: December 1998.

(8) Ben Miled, Z.; Zaitsev, A.; Bukhres, O.; Bem, M.; Jones, R.; Oppelt, R. Efficient Data Representation in Very Large Databases: A Case Study of a Pharmaceutical Data Repository. *15th Intl. Conf. Compute. Their Applications* **2000**, 140−145.

(9) van Bommel, P. Database Design by Computer Aided Schema Transformations. *Software Eng. J.* **1995**, *10*(4), 125−132.

(10) Oracle Corporation. *Oracle Call Interface Programmer's Guide Release 8.1.5*. Oracle Press: March 1999.

(11) Liu, Y.; Ben Miled, Z.; Bukhres, O.; Bem, M.; Jones, R.; Oppelt, R. Efficient Schema Design for a Pharmaceutical Data Repository. In *Proceedings of the 13th IEEE symposium on Computer-Based Medical Systems*, 47−254, Houston, Texas, June 2000.

(12) Oracle Corporation. *8.1.5 Oracle8i Tuning Release 8.1.5*. Oracle Press: May 1999.

(13) Oracle Corporation. *8.1.5 Oracle8i Parallel Server Concepts and Administration Release 8.1.5*. Oracle Press: June 1999.

(14) Technical White Paper. Sun Microsystems, Inc., http://www.sun.com/storage/white-papers/arch-array.wp.html, 1998.

(15) Ben Miled, Z.; Liu, Y.; Powers, D.; Bukhres, O.; Bem, M.; Jones, R.; Oppelt, R.; Milosevich, S. Data Access Performance in a Large and Dynamic Pharmaceutical Drug Candidate Database. In *Proceedings of Supercomputing 2000*; Dallas, Texas, November 2000.

(16) Lavalle, S.; Finn, P.; Kavraki, L.; Latombe, J.-C. Efficient database screening for rational drug design using pharmacophore-constrained conformational search. In *Proceedings of the 3rd ACM International Conference on Computational Biology (RECOMB)* 250−260, Grenoble, France, April 1999.

(17) Axel, M. G. and Song, I.-Y. Data Warehouse Design for Pharmaceutical Drug Discovery Research. In *Proceedings of 8th International Workshop on Database and Expert Systems* 644−650, Toulouse, France, September 1997.

(18) Inmon, W. H. *Building the Data Warehouse,* 2nd ed.; John Wiley & Sons: 1996.

(19) Elmasri, R.; Navathe, S. B. *Fundamentals of Database systems*; Addison-Wesley Publishing Company, 3rd ed.; 1999.

(20) Fordham, B.; Abiteboul, S.; Yesha, Y. Evolving Databases: An Application to Electronic Commerce. *Intl. Database Eng. Applications Symp.* **1997**, 191−200.

(21) van Bommel, P., Kovacs, Gy., and Micsik, A. Transformation of Database Populations and Operations from the Conceptual to the Internal Level. *Information Systems* **1994**, *19*(2), 175−191.

CI0255275