Surveying the Landscape: An In-Depth Analysis of Spatial Database Workloads

Bogdan Simion
Dept. of Computer Science
University of Toronto
bogdan@cs.toronto.edu

Suprio Ray
Dept. of Computer Science
University of Toronto
suprio@cs.toronto.edu

Angela Demke Brown
Dept. of Computer Science
University of Toronto
demke@cs.toronto.edu

ABSTRACT

Spatial databases are increasingly important for a wide variety of real-world applications, such as land surveying, urban planning, cartography and location-based services. However, spatial database workload properties are not well-understood. For example, it is unknown to what degree one spatial application resembles another in terms of resource demand, or how the demand will change as more concurrent queries (i.e., more users) are added. We show that spatial workloads have a different CPU execution profile than well-studied decision support workloads, as represented by TPC-H.

We present a framework to automatically classify spatial queries and characterize spatial workload mixes. We first analyze the resource consumption (i.e., computation and I/O) of a representative set of spatial queries, which are then classified into five distinct categories. Next, we create five homogeneous spatial workloads, each composed of queries from one of these classes. We then vary database-specific parameters (e.g., the buffer pool size) and workload specific parameters (e.g., the query mix), to characterize a workload in terms of CPU utilization and I/O activity trends.

We study workloads simulating real-world spatial database applications and show how our framework can classify them and predict resource utilization trends under various settings. This can provide clues to the database administrator regarding which resources are heavily contended and can guide resource upgrades. We further validate our approach by applying it to a much larger dataset, and to a second DBMS.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS; H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Performance, Measurement

Keywords

Query classification, workload characterization, architectural stall breakdown, PostGIS, Informix

1. INTRODUCTION

Spatial database workloads are increasingly important, thanks to the advent of popular geospatial Web services such as Google Maps, in-vehicle GPS navigation systems, and a host of accompanying location-based services. Other important geospatial applications include city planning, land surveys and environment assessments. Most relational database systems now offer spatial features.

As uses of spatial databases become more widespread, there is a growing need to analyze and understand the characteristics of different spatial applications. The characterization of database workloads is pivotal in analyzing performance issues, detecting optimization opportunities and determining how well the system will scale as data volume and request rate increases. Workload characterization can also enable automatic tuning of databases, thereby reducing the cost and complexity of database management [25].

Workload characterization is the quantitative description of a workload with the objective of obtaining a model that encapsulates its essential features and dynamic behavior [5]. Often, such workload analysis is conducted informally by the Database Administrators (DBAs), who study query plans and try various ad hoc solutions to improve the performance of a query (e.g., by rewriting it or by tuning various system parameters on a trial-and-error basis). This reliance on DBA expertise is costly, and does not easily translate to new situations. Although several research projects have studied how to characterize database workloads [10] [11] [12], they have mainly focused on TPC [23] workloads such as OLTP (Online Transaction Processing) and DSS (Decision Support System).

To date, there has been no such characterization of spatial database workloads, making it difficult to answer even relatively simple questions. For instance, to what extent does one spatial application resemble another in terms of its resource demand? What optimizations or hardware upgrades will be most beneficial for a given application? To address such questions, we "survey" the spatial query execution landscape[1]. We present a framework to automatically classify spatial queries by the nature of their resource consumption (computationally intensive, data intensive, mixed, low resource utilization etc.) and characterize spatial workload mixes. Based on a series of factors, our framework can provide clues to the users (or DBAs) in a reactive manner, to offer a better understanding of which resources are heavily utilized. Furthermore, we analyze how this resource pressure changes with parameters such as the buffer pool size and query density in the workload mix.

Our goal is to identify the inherent resource demands of the queries that make up a workload, enabling the prediction of performance trends as hardware and workload intensity changes. We aim to minimize the number of measurements that are needed to

[1]We draw an analogy to land surveying, which uses measurements to establish the relative position of physical features on the earth.

characterize a new target workload. Our contributions are:

1. We demonstrate that spatial queries differ from previously analyzed decision support workloads (as represented by TPC-H), in terms of their low-level CPU characteristics such as instruction and data cache misses and branch mispredictions.
2. We analyze the resource utilization of spatial queries and determine their nature in terms of the computation and data accesses they perform (i.e., compute-intensive, data-intensive, mixed, low-resource utilization etc.). We show that queries can be classified into five distinct categories.
3. We extend the Jackpine spatial database benchmark [19] with a mechanism to create custom concurrent spatial workloads. We use this to construct five custom workloads, each composed of multiple concurrent queries drawn from one of the query classes, and characterize them by varying parameters such as the buffer pool size and the query density. We show how each class of workload reacts to these changes.
4. We show how the query classification and the characterizations of known homogeneous workloads can be used to classify new workloads and predict their behavior under varying conditions. We demonstrate this on a set of realistic applications simulated in the Jackpine macrobenchmark suite.

The rest of the paper is organized as follows. Section 2 presents background on spatial databases and the benchmarks that we use. Next, Section 3 explores the CPU and I/O behavior of spatial queries to motivate our framework for query classification and spatial workload characterization, which is presented in Section 4. We demonstrate the use of this framework, showing how it can predict trends in previously unseen workloads, in Section 5. Using a second DBMS and a larger dataset, we validate our methodology in Section 6. Related work and conclusions are in Sections 7 and 8.

2. BACKGROUND ON METHODOLOGY

In this section, we present some background on spatial databases and testing considerations that influence our methodology.

There are a wide variety of geospatial databases, ranging from open-source projects such as MySQL and PostGIS (the spatial extension to PostgreSQL) to commercial products such as Oracle Spatial, ArcGIS and Informix. These databases contain built-in support for spatial data types (e.g., point, line, polygon) and spatial operations (e.g., distance, intersects, contains, etc.) that can be performed on the data shapes. The Open Geospatial Consortium (OGC) [17] recently standardized the set of spatial operations that should be provided in any geospatial database, however many products are not yet OGC-compliant. For example, MySQL lacks support for spatial operations like ST_Distance and ST_Dwithin.

Spatial query processing, which is a two-step process comprised of filtering and refinement stages, also differs across spatial databases. In the filtering step, the records are filtered based on the minimum bounding rectangles (MBRs) of the data shapes, to narrow down the possible matches to a much smaller candidate result set. In the refinement step, the records in the candidate result set are processed to determine which shapes satisfy the spatial criteria. Most databases comply with this two-step evaluation. However, some (e.g. MySQL) only perform the filtering stage, which can lead to many false positives in the final result set.

For our purposes, we wish to experiment with an implementation that is OGC-compliant, performs both stages of the spatial query processing, and is open source to allow us to investigate the causes for the behavior we observe. These requirements lead us to select PostgreSQL with the PostGIS extension as our target database.

In our analysis we use read-only workloads. From our experience with spatial databases, we observed that many real-world spatial applications involve read-mostly workloads and updates are very infrequent. For example, datasets representing cartographic maps, land ownership information, and flood insurance rate maps are rarely updated but are heavily queried. Although certain applications such as those involving spatio-temporal workloads do involve frequent updates, our current analysis has not addressed these scenarios. Extending the workload characterization to update-heavy applications is an important avenue for future work.

For test queries and datasets, we use the open-source Jackpine spatial database benchmark [19]. Jackpine provides comprehensive coverage of common spatial operators and several representative spatial application scenarios. In its initial version, Jackpine did not support multiple concurrent scenarios of different types, so we modified it to add support for concurrent heterogeneous query execution. We use this implementation to drive our spatial workloads. Jackpine uses the TIGER [22] dataset for the state of Texas and a few additional datasets corresponding to Travis County and the city of Austin [7]. These datasets have a variety of spatial features. The cardinality of the largest table is roughly 6 million records (requiring 1.7GB of storage for data and 400MB for spatial index) containing line shapes representing roads and rivers. We also added an additional copy of each data table to artificially inflate the spatial dataset size. The total dataset size is slightly under 9GB, including data tables and indices.

3. SPATIAL DATABASE RESOURCE USAGE

Spatial and non-spatial databases use several system resources during query execution, such as the CPU (for computation), and the disk (for retrieving data records). Another common trait is the impact of buffer pool size on database performance, since obviously more cache means that more data can fit into memory and more expensive disk accesses can be avoided.

There are, however, key differences that distinguish spatial and non-spatial databases. First, spatial queries must evaluate spatial relationships between shapes, involving complex geometric computations that can saturate the CPU for a long time. Second, geometric shapes do not have a fixed length, making the storage of spatial data more complicated. For example, the polygon for Maine is much larger than the one for Wyoming, which is a simple rectangle. Thus, spatial data must be stored in variable-length fields, which creates problems with record alignment. PostgreSQL's solution is to store large shapes (such as lines and polygons) as TOAST[2] data blocks, which are kept separate from the other fields in the main table. A TOAST index allows fast access to TOAST data blocks.

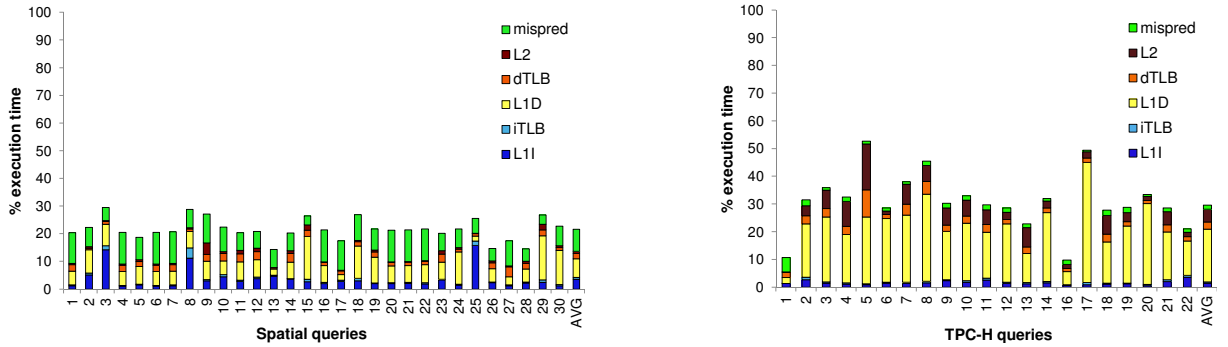
We first examine how these differences affect spatial query processing in terms of the CPU time breakdown for an in-memory dataset, before considering the effect of I/O.

3.1 Comparison of architecture-level events

Spatial applications and their associated data and queries involve a different type of processing, which imposes a different stress on system resources. An architectural breakdown of the CPU processing time illustrates these differences for two standard sets of spatial and non-spatial analytical queries.

We compare the set of microbenchmark queries from Jackpine, and those from the non-spatial TPC-H benchmark, on the same machine. Both benchmarks include a large set of read-only queries, involving a variety of complex analytical operations. In both benchmarks, there are long-running and short-running queries, and both

[2]TOAST stands for "The Oversized-Attribute Storage Technique"



(a) Spatial benchmark (b) TPC-H benchmark

Figure 1: Architectural breakdown for spatial and non-spatial workloads

involve complex query plans, with sequential scans, index scans, joins and other typical relational operators. Both test datasets are roughly equal in size, with multiple large data tables and indexes.

To eliminate I/O effects and focus on the CPU, we ensure that data is entirely memory resident by warming up main memory and caches with multiple runs for each query. Additionally, to determine the confidence intervals we repeated the experiments several times and found that the average variation is less than 4%.

To show the dominant components of the CPU processing time during query execution, we use an approach similar to Ailamaki et al. [1], who showed that for some database workloads CPU stalls are a big component of execution time. We measure several events that can cause CPU stalls: L1 data and L1 instruction cache misses, L2 cache misses, data and instruction TLB misses, and branch mispredictions. We then use a fixed cost estimate for each event to give a breakdown in terms of execution time. This is an approximation, since several stall events may overlap in time.

These events can be obtained using hardware performance counters available in most modern CPUs. The performance analysis tool "perf", included in Linux kernels, can monitor and record these counters. It provides a detailed report on the number of occurrences for each event. The cache and TLB latencies for our hardware are estimated using the Calibrator tool [4]. Branch misprediction latencies are estimated based on our processor specifications.

After running the spatial queries from Jackpine and non-spatial queries from TPC-H through the PostgreSQL engine, we show a side-by-side comparison in Figure 1. Since the queries in Jackpine had long descriptions, we ordered them alphabetically and numbered them for easy reference in the figures. We also omitted query 15 from TPC-H since it involves creating a view which is not typical of other TPC-H and spatial queries. The final bar in each graph shows the average contribution to execution time for each of the features analyzed, for both spatial and TPC-H queries. We note several interesting differences.

First, spatial queries incur fewer cache misses on average than their non-spatial counterparts. However, spatial queries spend more than twice the execution time (3.6%) in L1 instruction cache misses compared to TPC-H queries (1.5%). A few spatial queries (3, 8 and 25) involve a materialization with sequential scan on the outer relation of the join, which causes a much higher L1 instruction miss rate than others. Even if we exclude the 3 outliers, spatial queries still spend roughly 1.6 times more execution time in L1 instruction misses than TPC-H queries. As for L1 data cache misses, they account for less than 7% of execution time on average for spatial queries, compared to a whopping 19% for TPC-H queries. The L2 misses are also higher on average for TPC-H queries (roughly 5% of execution time) compared to spatial queries (less than 1% of execution time).

Second, spatial queries incur a slightly lower number of data and instruction TLB misses than non-spatial ones (2.4% of execution time on average for spatial queries, compared to 3% for TPC-H).

Third, spatial queries incur a lot more branch mispredictions, which can be very expensive since fetching the wrong instructions pollutes the caches with unnecessary instructions and causes stalls due to pipeline flushes. Branch mispredictions account for 8% of the execution time on average for spatial queries, compared to approximately 4.5% for TPC-H queries.

Overall, spatial queries are a lot more "CPU-friendly": a greater portion of execution time is spent performing useful computations, rather than stalling. Consequently, we argue that spatial workloads are a beast of a different nature, and thus warrant special attention.

3.2 Characterization of I/O behavior

At the other end of the spectrum from the CPU, database query processing requires I/O. Thus, we examined the I/O behavior, as seen at the disk level, during spatial query processing. To capture the complete I/O behavior we only use cold runs so that the data is not already cached in memory.

The Jackpine microbenchmark includes spatial join and spatial analysis functions. The spatial join queries (all pair join and join with a given spatial object) involve accessing multiple on-disk files, such as the corresponding data files, index files, the TOAST data files and TOAST index files. Due to the interleaved accesses to these files, the disk throughput of these queries resembles that of a random access pattern. The spatial analysis functions operate on one table and require accessing the corresponding data file, TOAST data file and TOAST index file only. These queries achieved disk throughput that resembles sequential access. This I/O behavior is expected, given the organization of the database files and the characteristics of disk drives.

3.3 Combined resource utilization

We now consider the combined demand for CPU and disk resources, as the amount of memory available to the database buffer pool varies. We observed that spatial queries can vary widely in terms of resource utilization. Some queries either do not touch many disk blocks or manage to cache blocks relatively quickly, after which most of the time is spent by the CPU processing complex geometries as part of the refinement stage. At the other end of the spectrum, we have queries that touch many data, index and/or TOAST blocks. In these cases, the main culprit for high query latency is the disk, because the slow disk accesses render the computation time almost negligible. Even for these data-intensive queries, if most of the necessary disk blocks can be cached, then CPU is

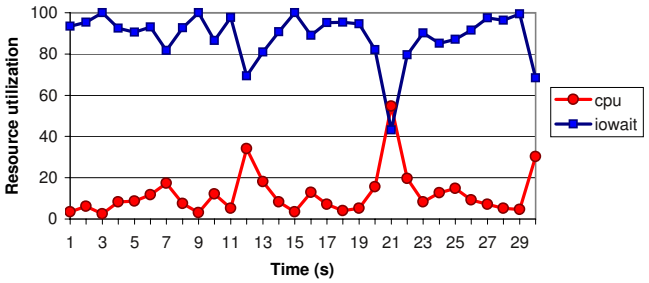


Figure 2: LineCrossesArea w/ 64MB buffer pool is I/O-bound

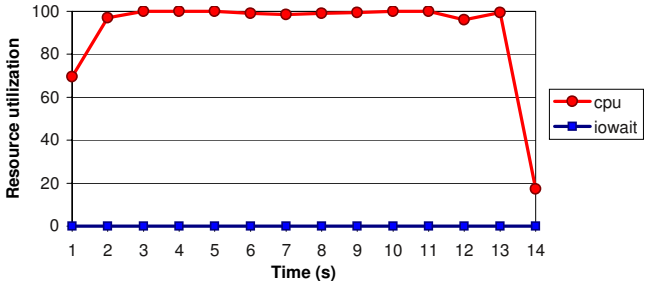


Figure 3: LineCrossesArea w/ 1GB buffer pool is CPU-bound

again the main resource used. Thus, such a query can be found at either end of the spectrum, depending on the buffer pool size.

To illustrate the impact of the buffer pool size on the execution of a spatial query, we present an example of a spatial query running at a low buffer pool setting (64MB) and a high buffer pool setting (1024MB) in Figures 2 and 3, respectively. The query is *LineCrossesArea*, a Jackpine scenario involving an all-pair join between two spatial tables, which finds all the pairs of lines (e.g., rivers, roads) and polygons (e.g. parks, lakes) that cross each other.

At a low buffer pool setting, Figure 2 shows that disk I/O takes up most of the execution time, even for warm runs, making the query I/O-bound. By increasing the buffer pool to a size where all the data fits, we observe a completely different behavior for warm runs, as shown in Figure 3. When the data is all in memory, the complex geometric computation takes up most of the execution time. Naturally, this also implies a reduction in query execution time.

Our spatial query analysis aims to incorporate all the information about I/O wait, CPU load and buffer pool size into a classification model for spatial queries, which can serve to create spatial workloads with predictable characteristics. We use system measurement tools that calculate the CPU utilization and the I/O wait time during the execution of a given spatial query. Since we aim to expose the inherent resource demands of the queries executed by the database, we must guard against interference from the operating system (OS) or other applications. OS interference can make it hard to observe the effect of changing the buffer pool size, since blocks can be provided from the OS cache rather than from disk. Many OSs provide an option for direct I/O, which bypasses the OS caches. However, PostgreSQL does not make use of this facility. To ensure that disk access performance is not enhanced by OS caches, we periodically clear the OS caches in all our experiments. Application interference can be avoided by running the database on a dedicated machine.

4. QUERY CLASSIFICATION AND WORKLOAD CHARACTERIZATION

Our framework for spatial query classification and workload characterization is based on resource utilization and various database-specific and workload-specific parameters. Stage I of our analysis classifies spatial queries based on the maximum stress they impose on CPU and disk resources, respectively. Next, we build spatial workloads fitting certain characteristics of their component queries, and show how these spatial workloads can be characterized in terms of their resource utilization at various settings.

4.1 Stage I: Spatial query classification

Spatial query resource utilization is very complex and depends on many factors, such as the spatial operations performed (complex geometric computations translate into heavy CPU usage), the database operators involved (e.g., complex joins, sequential scans, etc.) and query planner optimizations (e.g., whether spatial indexes can be used), and the table size (smaller data involves less disk activity). These factors are not easy to quantify and can introduce a multitude of combinations that are hard to cover exhaustively.

To classify spatial queries automatically, we need a measure that captures the query nature (compute intensive, data intensive, mixed resource, or low resource utilization), *without* analyzing all these factors. Essentially, we want a black-box method that can accurately determine the nature of the query in a particular setting (a known spatial DBMS and a known dataset), without any internal knowledge such as which spatial functions or database operators the query is using, what tables it is operating on, or what the query plan looks like. Our insight is that by using extreme buffer pool size configurations for our test database, we can extract information about the inherent query nature without analyzing the multitude of factors that contribute to the observed behavior. Although Kutner et al. [15] warn against using extreme endpoints for 2^K experimental designs because the system's performance at the midpoint may be nothing like its performance at either extreme, the miss-ratio curve (MRC) corresponding to a database query is monotonically decreasing in the database bufferpool size [21]. Therefore selecting extreme points is sufficient and accurate.

Our classification model is not purely analytical. We use experiments and measurements to determine where a spatial query fits. We use the full set of Jackpine microbenchmark queries, which are representative of most spatial operations and data types. However, we have extended the dataset with additional copies of the tables, to increase the pressure on the buffer pool.

We run the test spatial queries using two configurations:

a) BP_{min}: A configuration with a small buffer pool, which will not allow full caching of the disk blocks for large tables (resulting in lots of disk activity), but will allow small tables to be cached fairly well. In our tests, we use 64MB for BP_{min}, but this can be configured based on the dataset table sizes (significantly smaller than the largest table but able to fit some of the smaller tables).

b) BP_{max}: A configuration with a large buffer pool, which will allow a large number of disk blocks (if not all of them) to be cached in the buffer pool, even for large tables. We use 1GB for BP_{max}.

For each of these configurations, we record the average CPU load and average I/O-wait over the duration of the query. We use standard Linux measurement tools (such as *vmstat* or *sar*) that report for each 1 second sample, the proportion of time spent performing CPU computations or time spent waiting for I/O. These samples are then combined to give the overall CPU and I/O-wait measurements, each ranging between 0 and 100.

Next, we compute the maximum average resource utilization for both resources in the two buffer pool configurations. As a general rule, the maximum strain on the CPU will always be at a high buffer pool configuration, while the maximum strain on the disk is when the database operates with very little buffer cache. Thus,

$$Max_{CPU} = AvgCPUload(BP_{max}) \quad (1)$$

and

$$Max_{IO} = AvgIOwait(BP_{min}) \quad (2)$$

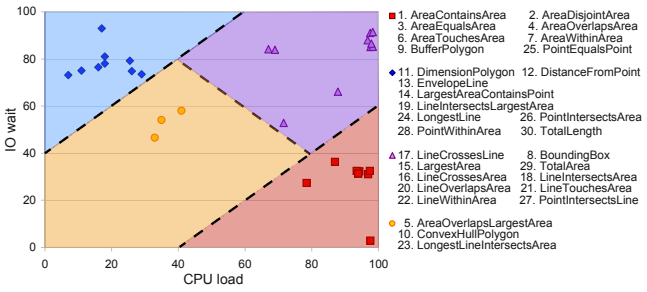


Figure 4: Spatial query classification - color coding represents the 4 query classes

These numbers represent the spatial query nature in terms of data-intensiveness (if Max_{IO} is high at BP_{min}, when disk activity should be dominant), and computational intensity (if Max_{CPU} is high at BP_{max}, when everything should theoretically fit in memory). The difference between these two figures represents the variation between the highest CPU-utilization and the highest I/O activity. This is a good indicator if a query has a more computational nature (e.g, performing complex spatial operations on complex spatial types), or if data fetching activity is predominant.

In Figure 4, we plot the values of Max_{CPU} and Max_{IO} for each query on a two-dimensional scatterplot with the CPU utilization on the x-axis and the I/O-wait on the y-axis. Intuitively, the queries located in the upper left corner of the resource space shown in Figure 4 are data-intensive, since they have high I/O-wait times but do not saturate the CPU even at BP_{max}. Similarly, we can see that the queries from the lower right corner are compute-intensive, since the I/O-wait is never a large factor, even at BP_{min}. The queries in the upper right corner are of a mixed nature, depending on the buffer pool size, while the queries close to the lower left corner utilize few resources (low computation, few data accesses).

From this plot we can see that spatial queries differ widely in terms of their maximum resource demand, with some being compute intensive while others are inherently data intensive. Based on these observations, we can determine the query classes by splitting the resource usage domain using a set of delimiting line segments:

$$y = x - a \tag{3}$$

$$y = x + b \tag{4}$$

$$y = -x + c \tag{5}$$

In our work, we determined empirically that a = b = 40 and c = 120 work well as the thresholds. With these equations, the resource usage space is divided as shown by the colored regions in Figure 4. Using equations (3) and (4), we can determine the nature of a given query point Q(x,y):

i. if $x - y >= a\% \Rightarrow$ Query is compute-intensive (red region)

ii. if $x - y <= -b\% \Rightarrow$ Query is data-intensive (blue region)

iii. if $-b\% < x - y < a\% \Rightarrow$ Query is mixed (orange+purple regions)

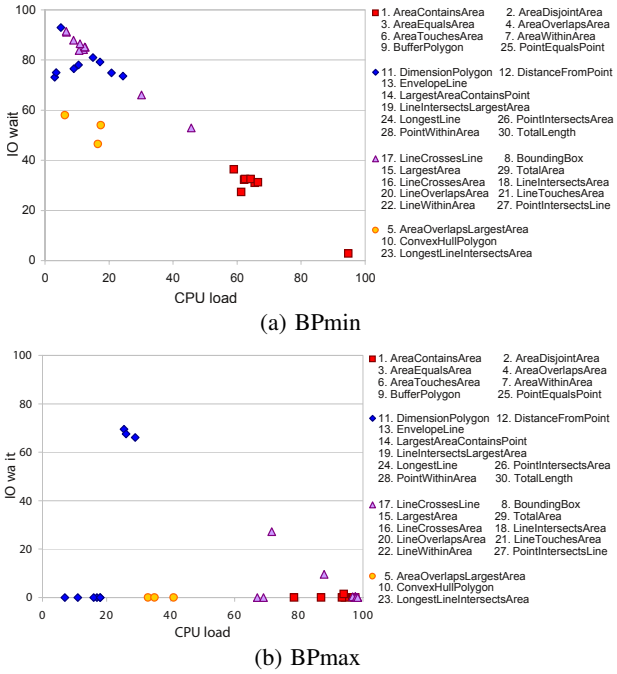
Category iii) is further split using equation (5):

iii.a) if $y + x < c\% \Rightarrow$ low resource utilization (orange region)

iii.b) if $y + x >= c\% \Rightarrow$ mixed resource usage (purple region)

Mixed usage queries arise either because varying the buffer pool size migrates the query from the data-intensive end of the spectrum to the compute-intensive end, or simply because the query is using both resources in roughly equal proportions.

The visual representation in Figure 4 shows that this approach groups queries into 4 classes: compute-intensive (class C), data-intensive (class D), mixed resource (class M1) and low resource utilization (class M2). The legend lists the queries in each category.



(a) BPmin

(b) BPmax

Figure 5: Mixed queries migrate from a data-intensive state at BP_{min} to a high computation region at BP_{max}

Although the use of delimiting line segments works well for this set of experiments, the empirical choice of thresholds is unsatisfactory. As a result, we use the intuition behind the different segments to construct a k-means clustering model.

4.1.1 *Clustering-based classification model*

We implemented a clustering method which places the queries into 4 clusters (corresponding to the 4 categories mentioned above). We use a k-means algorithm with 4 centroids and Euclidean distance measure. The centroids are artificial points in the resource space, representing the "average location" of a particular cluster. The 4 centroids are placed initially at (90,10), (10,90), (90,90) and (10,10), corresponding to compute-intensive tasks, data-intensive tasks, mixed-resource tasks and low-resource tasks respectively.

The algorithm computes the distances from each point to each of the centroids, then marks each point as belonging to the cluster whose centroid is closest. Once the points are grouped into clusters, the centroids are recomputed by averaging the coordinates for each of the points in the cluster. The process repeats until the centroids stabilize (they remain the same after two successive iterations).

The query classes represented by the clusters are identical to the ones obtained using the delimiting lines, further supporting that we accurately capture the query nature and produce a correct classification. Figure 4 lists the queries contained in each cluster.

4.1.2 *Discussion*

We observe that mixed resource queries (class M1) come in two forms. The first type consists of queries that use roughly equal execution time to perform computations and to fetch data blocks. Based on our experiments, this type is rare; only one example can be found in Jackpine, namely the LineCrossesLine query. This scenario was limited to return 5 records, since it was the longest running query (an all pair join between two 6-million record tables, representing all the intersections between line shapes such as roads

and rivers). When a data-intensive query such as LineCrossesLine only has to fetch a limited amount of data, the computation becomes significant and thus the query is evenly balanced in terms of computation and time spent accessing data.

The second type consists of queries that are at opposite ends of the spectrum depending on the buffer pool size. In Figures 5(a) and 5(b) we illustrate how mixed-nature queries (class M1, marked with purple triangles in the figures) queries migrate from I/O-bound at a low buffer pool size to CPU-bound at a high buffer pool size.

It is important to note that the query classification does not tell us whether a query is I/O-bound or CPU-bound. Instead, the class gives us insight into how the query will behave in different configurations. Figure 5 shows the behavior at the two extreme buffer pool configurations, illustrating how the behavior changes for the different query classes. This figure motivates one additional refinement to our classification. We observe that data-intensive queries (class D) come in two types. The first type (class D1) is I/O-bound with a low buffer pool and low-CPU with a high buffer pool, while the second category (class D2) stays I/O-bound, even with a significantly large (yet reasonable size) buffer pool. Class D1 contains queries that cannot saturate the CPU even when all the data fits in the buffer cache; it accounts for most of the data-intensive queries from Jackpine. The queries that are included in class D2 are EnvelopeLine, LongestLine and TotalLength.

In summary, the Stage I analysis groups spatial queries into 5 major classes: compute-intensive (class C), data-intensive (classes D1 and D2), and mixed (classes M1 and M2). The next stage selects queries of a given class to create spatial workloads with a known homogeneous nature.

4.2 Stage II: Spatial workload characterization

In the second stage of our framework, our goal is to characterize spatial workloads as we vary database-specific parameters (most significantly, the buffer pool size) and workload-specific parameters (e.g., the query density in the workload mix). Given the classification of all the test queries in Stage I, we aim to determine if a workload comprised of queries from a given class is CPU-bound, I/O-bound or a mix of both, in different configurations.

4.2.1 Jackpine workload support

To test and analyze spatial workloads, we extend the Jackpine benchmark to support fully concurrent heterogeneous query execution. Jackpine only had limited support for concurrent query execution. Specifically, it could launch multiple threads within the same scenario to execute multiple instances of the same query. However, to provide realistic workloads containing any combination of distinct spatial queries running concurrently we require more flexible support for concurrent scenarios.

We use the modified Jackpine benchmark to drive custom workloads for the Stage II analysis. Jackpine now launches a workload, based on a configuration file that contains the descriptions of the scenarios to be executed concurrently. Each scenario in turn has its own configuration file, containing the description of the spatial scenario class, and a set of parameters. One of these parameters specifies how many instances of that query should be included in the workload mix. Consequently, a workload can contain multiple distinct scenarios (corresponding to distinct spatial queries), and each scenario can have multiple instances of the same query. All of these queries are run concurrently.

To isolate the performance of the database from that of the benchmark driver, we run Jackpine on a separate machine; queries are issued to the database using JDBC connections.

4.2.2 Workload characterization via resource usage

Our goal is to characterize spatial workloads by varying several factors that can have a significant impact on the resource usage of a given workload. As we have seen, one critical factor is the buffer pool size, which determines whether the data and index blocks needed by a workload can fit entirely in memory, thus eliminating expensive disk accesses. Based on the buffer pool size, a workload can be characterized as either I/O-bound, mixed, or CPU-bound. If all needed data fits in the buffer pool, we can determine whether the CPU usage is high or low without the interference of fetching data from disk. In contrast, some queries perform sequential scans on large tables which will not fit into even a large buffer pool.

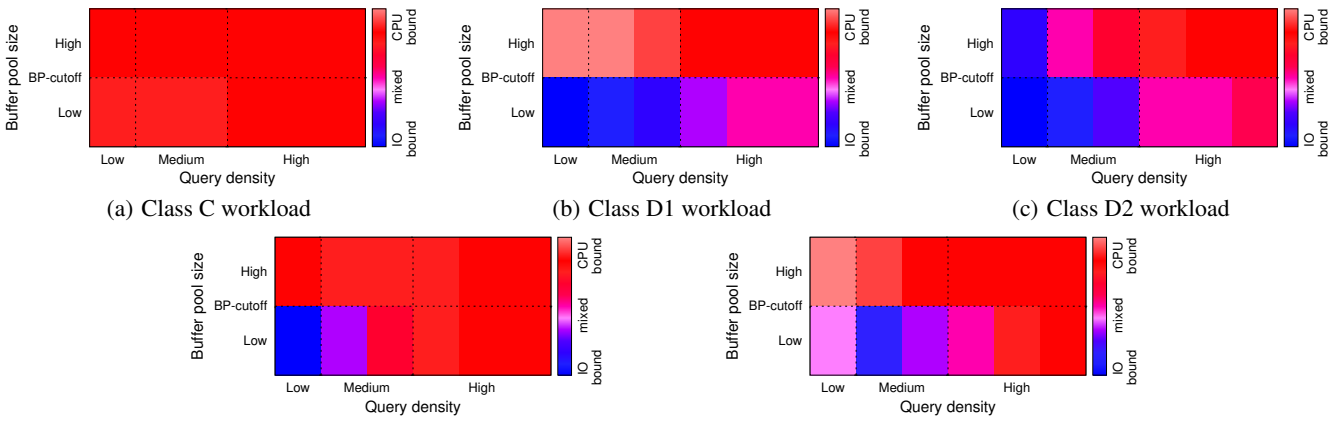
Another important parameter is query density. In generic database workloads, there are a finite number of tables in a dataset, but a large number of query types and combinations. This leads us to the safe assumption that the number of queries in the average workload mix is much larger than the number of tables used. In turn, this means that many queries will be accessing the same tables concurrently, which would enable sharing of buffer pool pages. Based on the density of the queries in the workload mix, this sharing effect can have a significant impact on the workload nature as it attracts workloads towards the CPU-intensive end of the spectrum. Basically, workloads that are already CPU-bound will be even more dominantly CPU-bound, while for I/O-bound workloads the disk activity dimension will become less predominant.

Because disk accesses are very expensive, some queries are I/O-bound even though they perform heavy computations. However, when the query density is high, a large number of I/O-bound queries will be accessing the same tables and will inevitably share pages in the buffer pool. The result is a higher hit ratio, which in turn reduces the amount of disk accesses. This buffer pool sharing could result in a CPU-bound workload comprised of individually I/O-bound queries, given that the CPU-work cycles for each individual query in the workload cannot be shared.

We design 5 types of workloads corresponding to the query classes described in Section 4.1. The workloads are homogeneous in the query class types, but contain a mix of distinct queries. Thus, workloads are composed of either C, D1, D2, M1 or M2-class queries. The buffer pool size and query density are the testing parameters; by varying them, the workload nature can change dramatically for D-class and M-class workloads, but not for the C-class, as we will show. We use 2 buffer pool settings (low/high) corresponding to BP_{min} and BP_{max} configurations. We use 3 query density settings: low, medium and high. With the low setting, the workload contains a single query of the respective query class. A medium setting includes a mix of 6 queries, while a high setting contains 30 concurrent queries. To better show the trends, we chose one more mix (of 12 queries) for the medium setting and two extra mixes (24 and 48 concurrent queries) for the high setting. The basic characterization can be obtained with just the initial 3 mixes, however.

As in Stage I, CPU load and I/O activity measurements are taken at 1 second intervals and averaged over the execution time of the workload. The average CPU load and the average I/O-wait represent the percentage of time executing CPU instructions and the percentage of time fetching either data or index pages from disk.

We use a visual representation of the resource usage for these workloads in a two-dimensional space, where the coordinates are query density (x-axis) and buffer pool size (y-axis). We draw a line between the low and high buffer pool settings, corresponding to the size where the miss-ratio curve (MRC) drops to zero buffer pool misses. This *BP-cutoff* will occur at different buffer pool sizes for different queries. We do not determine the actual cutoff, we merely observe that behavior will be different above and below this line.



(a) Class C workload (b) Class D1 workload (c) Class D2 workload

(d) Class M1 workload (e) Class M2 workload

Figure 6: Resource usage maps. Color coding is used for resource usage: CPU computations are shown in shades of red (darker means more computation), I/O activity is in blue nuances, while mixed is in shades of purple (depending on the mix of CPU and I/O).

The resource utilization is color-coded (similar to heat maps) into *resource usage maps* for each of the test workloads.

Figure 6 (a)-(e) shows the resource usage maps for each of the workload types. Our experiments show that the compute-intensive class C queries (Figure 6(a)) are CPU-bound regardless of buffer pool size or query density. This is an expected result, since increasing the buffer pool size and/or query density can only increase the demand on the CPU. However, for data-intensive and mixed query classes, the workload categorization gets considerably more complex, as the workloads can turn out to be either I/O-bound, mixed or even CPU-bound, in different configurations. In the following text, we use the abbreviation "QD" to refer to query density.

The D1 category is I/O-bound at a (low QD; low BP) configuration, and low CPU usage at a (low QD; high BP) setting. This shows that each of the queries in this category individually exhibit this behavior, as expected. As query density increases, at a low BP setting the concurrent queries start sharing pages in the buffer pool and therefore the workload migrates towards a mixed resource utilization (for QD=high). With a high BP setting and increasingly higher QD, the workload performs more computation exerting more pressure on the CPU and (predictably) becoming CPU-bound. The D2 category exhibits a trend of slowly transitioning from I/O-bound towards mixed and CPU-bound as query density increases. The M1 category is similar to D1, except that having more computation speeds up the transition from an I/O bound to a CPU-bound behavior at a low buffer pool setting and leads to consistently CPU-bound behavior at a high buffer pool setting. The M2 category exhibits a low resource utilization behavior for a low query density, but as the number of queries in the workload mix increases, the workload behavior changes. By adding more queries (which use more data tables), the workload first becomes I/O-bound at a medium QD setting, only to migrate towards CPU-bound as more queries are added at a high QD setting (with the number of data tables staying the same). With a high buffer pool setting, an M2 workload transitions from low-CPU usage to CPU-bound behavior as QD increases.

As a general observation, once a workload becomes CPU-bound increasing the query density can only leave the workload CPU-bound, if the number of data tables in the dataset stays the same. If each query would operate on its own copy of the data table, the situation would be different. However, as noted earlier, we make the reasonable assumption that the number of queries in a spatial workload far exceeds the number of tables in the dataset.

These resource usage maps provide valuable information on the behavior of any spatial workload. Database administrators and database users can use these maps to understand which resources are heavily utilized and what to expect with changes in the workload query density. Given a buffer pool size and the workload mix, these categories can pinpoint the nature of the workload (CPU-bound, I/O-bound, mixed etc.). Furthermore, this helps determine where to focus performance upgrades to deal with increased pressure on a given resource.

5. APPLICATION TO NEW WORKLOADS

To demonstrate that our characterization provides useful guidance on the behavior of a target workload, we test realistic spatial workloads drawn from the Jackpine macrobenchmark scenarios. We show that the classifications obtained by our framework accurately predict the behavior of these workloads at different buffer pool size and query density settings. Further, we show how this characterization can serve to guide resource upgrades.

5.1 Experimental setup

For the query classification and workload characterization stages, we use an Intel Core 2 Duo E7500 at 2.93GHz, 3MB cache, and 4GB RAM, running Ubuntu Linux 10.04, 64-bit with 2.6.26 kernel version, and PostgreSQL 8.4.4 with PostGIS spatial extension. All the results in the previous sections were obtained on this machine.

When running on a machine with multiple cores, Linux measurement tools average out the CPU load and I/O wait time among the cores, whether active or not. To obtain correct measurements, we disabled one core when running the workloads. On Linux, this can be done dynamically by writing "0" to a system file (e.g. "`echo 0 > /sys/devices/system/cpu/cpu1/online`").

5.2 Classifying realistic workloads

To test the accuracy of our categorization framework on real-world spatial applications, we use all but one of the Jackpine workloads from the macrobenchmark suite (Map Browsing, Land Information Management, Flood Risk Analysis, Toxic Spill and Reverse Geocoding). We omitted the Geocoding scenario, since it does not use spatial shapes, just latitude-longitude coordinates.

In Stage I, each macrobenchmark scenario is classified into data-intensive, compute-intensive, or mixed (as well as the appropriate subcategories). In Stage II, we select two interesting scenarios in terms of resource usage (MapBrowsing and Land Information Management, which are also among the most widely-used spatial applications) to demonstrate that the resource usage maps for their

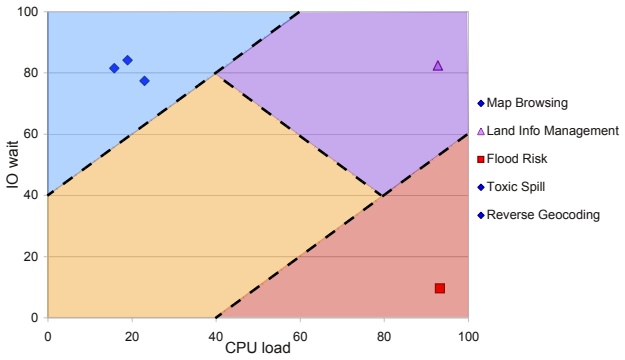


Figure 7: Classifying realistic application workloads.

respective classes can be used to accurately predict the resource-boundness of the workloads in a wide range of configurations.

The Map Browsing and Land Information Management scenarios are described in detail in [19]. For completeness, we provide a short description for each of them here.

Map Browsing. Searching for locations of interest by keywords and viewing the results on a map is a popular use of online mapping applications. For example, a tourist visiting a new city may look for nearby airport, hotels, restaurants, and popular sites. The Map Browsing scenario models such use cases by first performing a search query for a place of interest. Then a series of 5 queries are executed that fetch the spatial objects inside a bounding box centered around the found location of interest. The sequence of searching for a nearby place of interest and 5 map display queries are repeated a number of times to reflect typical browsing behavior.

Land Information Management. Land information management is vital to the maintenance of property ownership and the enforcement of land use regulations. Digital land information management systems, where parcels of property are stored as shapes in a geospatial database, are a valuable tool to many local governments. The land information data set used in Jackpine was obtained from the City of Austin GIS data sets. Several land use related queries are executed in sequence during the scenario.

We first show how the Stage I process can be used to classify the spatial workloads into one of the 5 categories. Each of the macro scenarios are individually run and classified using the delimiting line segments (clustering with the other queries yields the same classification). The results are shown in Figure 7. Map Browsing has a highly data-intensive nature, belonging in the D2 class, whereas Land Information Management is of a mixed nature, fitting into class M1. The Flood Risk analysis is CPU-bound (C-class) and thus is not as interesting to study since the resource usage map is CPU-bound at all settings. Toxic Spill is similar to Map Browsing and fits into the D2 category, while Reverse Geocoding matches the D1 class. Note that only two measurements are needed to obtain the classification for a given workload.

Based on this classification, the resource maps from Figure 6 in Section 4.2 can be applied to predict how these workloads change at different settings. Due to space constraints, we selected for analysis the Map Browsing and Land Information Management scenarios, as they are representative of the most interesting query classes (D2 and M1), which provide a wide range of resource usage variation.

To demonstrate that the resource usage maps for the corresponding classes (obtained with the homogeneous workloads) can accurately predict workload behavior, we pass the workloads through Stage II and measure resource usage at various buffer pool and query density settings. The resulting resource usage maps are plotted in Figures 8(a) and 8(b). As can be seen from these maps, the Map Browsing workload clearly matches the behavior of the D2 class at different buffer pool sizes and query densities, while Land Information Management matches the behavior of the M1 class.

We conclude that employing the resource usage maps for workloads of a particular nature (i.e., class) can accurately predict their behavior in various configurations.

Note that our goal is not to predict exact performance under a given configuration, but rather to capture the resource usage trend of an application as database-specific and workload-specific parameters change. Consequently, the buffer pool size and query density settings must not be taken as absolute values, but rather as a means to capture the workload behavior trend. We argue that the methodology behind our framework creates resource utilization maps which are widely-applicable in this respect. For example, in production databases, the dataset will probably be much larger than our dataset, but so will the buffer pool size settings. Even if the dataset and buffer pool size increases are not proportional, the performance trend will still be captured in the resource utilization maps. Our ultimate goal is to provide clues to DBAs in capacity planning decisions, by using these maps as a predictor of how workload changes can impact the performance trends.

6. VARYING DBMS AND DATASET

To validate the classification methodology, we apply it to a second database, as well as to a much larger dataset.

6.1 Informix

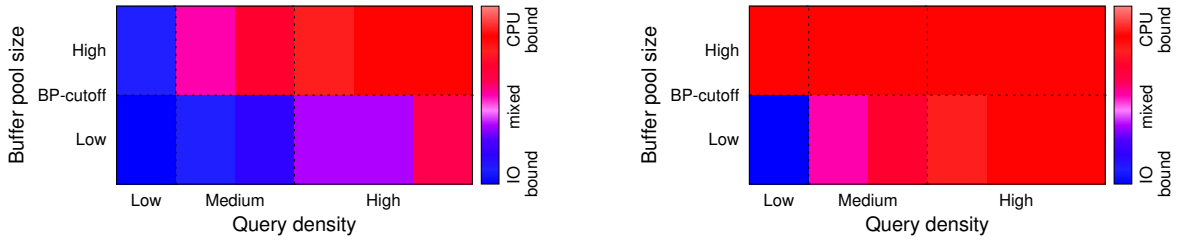
Different DBMSs implement spatial processing in different ways, and not all are OGC-compliant. For example, MySQL does not perform the refinement step. Other DBMSs use different implementations of the spatial functions and/or perform various optimizations for the relational operators (e.g., sequential and index scans, joins etc). Consequently, the results of our framework on one DBMS cannot be blindly applied across all DBMSs. The framework may place the same query executed on different DBMSs in different categories, simply because different implementations of spatial operators across DBMSs, as well varying database internals (such as buffer pool management and query optimizations) may entail different resource utilization.

However, an important aspect of our analysis is to determine if the query classification method can be used for any DBMS. Thus, we perform a validation with a different DBMS, by applying the same strategy to Informix, a closed-source commercial DBMS. In figure 9 we show the resource usage of the same set of spatial queries, executed through the Informix engine and its spatial extension. Since Informix does not support the ST_Buffer spatial function, we exclude one query involving this functionality.

Overall, the classification is quite similar to PostgreSQL in terms of the resource usage of different queries, with the exception that Informix seems to exhibit better I/O performance for some of the queries, such as TotalLength, TotalArea, LongestLine, LargestArea and LineCrossesLine. As a result, these queries get classified in a different category than for PostgreSQL.

Since Informix is closed-source, we could not confirm with the source code the exact reason for better I/O performance, but we did notice that all the aforementioned queries include a sequential scan of a large table (edges_merge and areawater_merge) as part of the query plan. As a result, we speculate that Informix is better optimized for sequential scans of large data tables. Additionally, since Informix stores data on disk as contiguous large blocks called "dbspaces" and "sbspaces" (for spatial data), we speculate that the prefetcher may also contribute to the better I/O performance.

Even without having information about the query plans, inter-



(a) Map Browsing (b) Land Info Management

Figure 8: Resource usage maps for realistic workloads

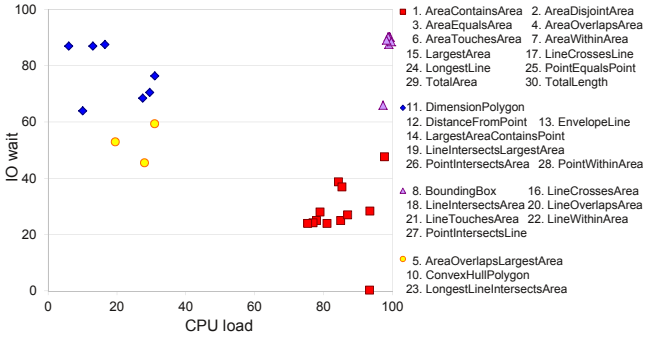


Figure 9: Classification for Informix.

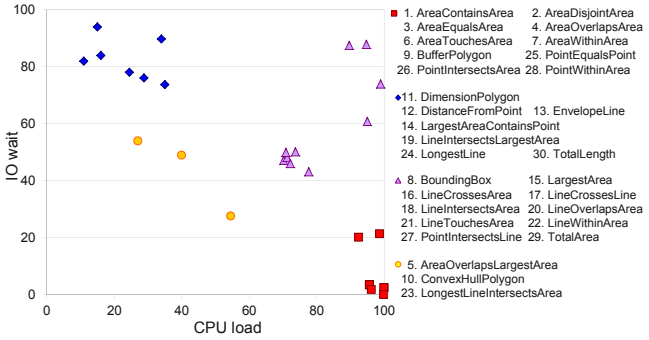


Figure 10: Classification with larger dataset (contiguous US).

nal query processing optimizations, data layout on disk, and other database internals of Informix, the classification framework was able to accurately place these spatial queries according to their resource utilization. Moreover, although Informix seems to perform more computation and do slightly more I/O for some of the queries classified as low utilization for PostgreSQL (marked with orange circles), the classification model is still able to distinguish these relatively short-running queries from the data-intensive cluster of queries and mark them as low utilization for Informix as well.

6.2 Dataset scalability

An important aspect of our classification is that it can be performed for any size spatial dataset. To illustrate this, we validated our approach against a much larger dataset. We used TIGER data for the entire contiguous US (48 states, instead of just the state of Texas), which amounted to roughly 32GB. We adjusted the buffer pool sizes in line with our concept of testing at extreme low and high settings to accurately determine the nature of a spatial query (computationally-intensive, data intensive, or mixed).

Figure 10 shows that the overall trend for spatial queries on the US dataset is towards more computation than for the Texas dataset, for all queries analyzed, with I/O making slightly less of an impact. In most cases, the spatial queries remain in the same classes as before, with two surprising exceptions. Two of the queries that were previously classified as data-intensive (PointIntersectsArea and PointWithinArea) on the Texas dataset have now migrated towards computationally intensive. The trend towards more computation is explicable since in the case of complex queries more objects lead to more complex geometry calculations, and computations on larger data end up outweighing the increased disk activity.

The queries labelled low-utilization with the Texas dataset remain low-utilization on the US dataset as well, and are classified as such. However, there is a slight trend towards either the data becoming more dominant (e.g., LongestLineIntersectsArea), or the computation (e.g., ConvexHullPolygon).

Overall, the experiments with a larger dataset confirm that our classification method exposes the inherent nature of spatial queries. However, these results must be considered in the context of the spatial dataset, since the inherent features of a large dataset may sway some of the spatial queries towards a different class.

7. RELATED WORK

A survey of the early research on workload characterization can be found in [6]. Various research projects attempted to characterize database workloads using different approaches. Yu et al. [26] presented a method based on the analysis of the structure of SQL statements, the behavior of transactions, and the composition of tables and views. Several projects used clustering techniques to group workloads based on similarity metrics, such as response time. Some of the early works that used clustering to classify workloads are summarized in [18]. Wasserman et al. [24] described a database workload classification approach based on system resource attributes. A few studies focused on characterizing the database access patterns. For instance, in [8] the authors identified three types of access patterns. Additionally, they addressed the issue of characterizing the random access pattern and predicting its buffer hit probabilities with various buffer pool configuration. Sapia [20] studied the high level access patterns of OLAP (Online Analytical Processing) queries with an eye towards improved caching techniques. Using trace-driven simulations Hsu et al. [12] systematically examined the properties of production workloads in relation to the TPC-C and TPC-D workloads and concluded that the production workloads are dynamic and time-varying in their characteristics. Oh and Lee [16] conducted a study to analyze how changes in resource size impact resource usage and proposed a method to identify the resources with significant influence over system performance.

Other research projects investigated database workloads to study hardware system behavior and performance issues in commercial applications. Barroso et al. [3] examined the memory system behavior of the OLTP and DSS workloads. In [14] the database performance on SMT processors was analysed, whereas, in [13] the authors studied the effectiveness of several architectural features of a multiprocessor SMP running commercial database workloads.

Previous database workload characterization research dealt with traditional database workloads, such as those from the TPC suites. In contrast, we focus on spatial database workloads. An important aspect of our approach is the small number of profile runs that are required per workload to produce the resource usage map, in contrast to other approaches such as [16] that require many more profile runs to characterize resource usage.

Databases are not unique in being the subject of workload characterization. Arlitt [2] characterized user sessions from web page accesses. Eeckhout et al. focused on Java workload characterization [9]. Many other examples of workload characterization research can be found.

8. CONCLUSIONS

Spatial workloads are different from previously studied database workloads. Using microprocessor performance events, we show that spatial queries stress the CPU in different ways from decision support queries in TPC-H. This may point to opportunities for improving the implementation of spatial operators.

We then introduce a two-stage framework for classifying spatial queries based on their nature (compute-intensive, data-intensive or mixed) and characterizing spatial workloads based on resource utilization at various settings. Our approach is unique because we incorporate concurrent query interaction by varying query density. Thus, we capture the dynamic nature of the resource consumption of spatial workloads with the change in query density and system configuration parameters. Finally, we show how this framework can characterize any spatial workload simulating real-world applications. We also present a novel resource usage map that provides a concise, yet information-rich description of the workload behavior with various system configurations and load conditions. This serves as a tool for database administrators (DBAs) to use in determining which resources are heavily utilized and thus to perform resource upgrades in a smart way.

We validate our methodology using a second DBMS and a second, much larger dataset. As future work, our goal is to apply this process dynamically to detect changes in workloads online, thereby supporting dynamic provisioning.

9. ACKNOWLEDGMENTS

We would like to thank Ryan Johnson and the anonymous reviewers for their valuable feedback. We also thank Eric Logan for assistance with machine configurations. This research was supported by an NSERC Discovery Grant. Suprio Ray is supported by an NSERC PGS-D scholarship.

10. REFERENCES

[1] A. G. Ailamaki, D. J. Dewitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go. In *Intl Conference on Very Large Data Bases*, pages 266–277, 1999.

[2] M. Arlitt. Characterizing web user sessions. *Sigmetrics Performance Evaluation Review*, 28:50–63, 2000.

[3] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Intl Symposium on Computer Architecture*, pages 3–14, 1998.

[4] The Calibrator, a cache-memory and TLB calibration tool. http://homepages.cwi.nl/ manegold/Calibrator/.

[5] M. Calzarossa and L. Massari. Workload characterization issues and methodologies. In *Performance Evaluation: Origins and Directions, Lect. Notes Comput. Sci.*, 2000.

[6] M. Calzarossa and G. Serazzi. Workload characterization: a survey. *Proceedings of The IEEE*, 81:1136–1150, 1993.

[7] City of Austin GIS Data Sets. ftp://ftp.ci.austin.tx.us/GIS-Data/Regional/coa_gis.html.

[8] A. Dan, P. S. Yu, and J. Chung. Characterization of database access pattern for analytic prediction of buffer hit probability. *The VLDB Journal*, 4:127–154, 1995.

[9] L. Eeckhout, A. Georges, and K. D. Bosschere. How Java programs interact with virtual machines at the microarchitectural level. In *Proc. of OOPSLA*, pages 169–186, 2003.

[10] S. Elnaffar. *Towards workload-aware DBMSs: Identifying Workload Type and Predicting its change*. PhD thesis, Queen's University, Canada, 2004.

[11] S. Elnaffar, P. Martin, and R. Horman. Automatically classifying database workloads. In *Intl Conference on Info. and Knowledge Management*, pages 622–624, 2002.

[12] W. W. Hsu and A. J. Smith. Characteristics of production database workloads and the TPC benchmarks. *IBM Systems Journal*, 40:781–802, 2001.

[13] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a Quad Pentium Pro SMP using OLTP workloads. In *Intl Symposium on Computer Architecture*, volume 26, pages 15–26, 1998.

[14] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Intl Symposium on Computer Arch.*, pages 39–50, 1998.

[15] J. N. Michael H. Kutner, Christopher J. Nachtsheim and W. Li. *Applied Linear Statistical Models (5th edition)*. McGraw-Hill International, New York, 2005.

[16] J. S. Oh and S. H. Lee. Resource selection for autonomic database tuning. In *ICDE Workshops*, 2005.

[17] Open Geospatial Consortium. http://www.opengeospatial.org/ogc.

[18] K. E. E. Raatikainen. Cluster analysis and workload classification. *Sigmetrics Performance Evaluation Review*, 20:24–30, 1993.

[19] S. Ray, B. Simion, and A. D. Brown. Jackpine: A benchmark to evaluate spatial database performance. In *Intl. Conf. on Data Engineering*, April 2011.

[20] C. Sapia. PROMISE: Predicting query behavior to enable predictive caching strategies for OLAP systems. In *Data Warehousing and Knowledge Disc.*, pages 224–233, 2000.

[21] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza. Dynamic resource allocation for database servers running on virtual storage. In *Conference on File and Storage Technologies*, pages 71–84, 2009.

[22] TIGER®, TIGER/Line® and TIGER®-Related Products. http://www.census.gov/geo/www/tiger.

[23] The Transaction Processing Performance Council. http://www.tpc.org.

[24] T. J. Wasserman, P. Martin, D. B. Skillicorn, and H. Rizvi. Developing a characterization of business intelligence workloads for sizing new database systems. In *Intl Workshop on Data Warehousing and OLAP*, pages 7–13, 2004.

[25] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Intl Conference on Very Large Data Bases*, pages 20–31, 2002.

[26] P. S. Yu, M. Chen, H. Heiss, and S. Lee. On workload characterization of relational database environments. *IEEE Transactions on Software Engineering*, 18:347–355, 1992.