Performance of Spatial Queries in Object–Relational Database Systems

K. Kalpakis[1,2,3], J. Behnke[4], M. Pasad[1,2], and M. Riggs[1,2]

January 11, 2000

Abstract

We report on experiments with the Informix Universal Server 9.14 for the Monet catalog (USNO-A2.0) of stars. The Monet catalog consists of about 500 million objects. The focus of this report is on the performance of access methods present in a traditional relation database system (e.g. B-trees) as well as access methods for user defined datatypes (e.g. R-trees for spatial objects). In addition to the standard relational access methods available in the Informix Universal Server, we use those available in the Geodetic datablade and the Shapes2 datablade.

We experiment with the following types of queries. Spatial window queries (e.g. find all stars within a user defined window or box), spatial self–join queries (e.g. find all stars that are within a specified box centered on each star), spatial chain–joins (join a "chain" of tables using a window) and star–joins (join a "star" of tables using a window). Window queries are important for online access to the catalog. Spatial chain–join and star–join queries are important in correlating a catalog with other catalogs. Spatial self–join queries are important for mining collections of spatial objects.

For the spatial window queries, we find that the access methods provided by each one of the two datablades, as well as the standard relational access methods, are sufficient for online processing. For example, processing spatial window queries on two million stars takes about 1 sec elapsed time. However, for spatial join queries (self–join, chain–join, and star–join), relational access methods are not adequate. The R-tree access methods of the two datablades provide significant performance improvements. Moreover, the Shapes2 datablade outperformed the Geodetic datablade, primarily due to its lower space overhead.

We also find that B–tree indexes are more sensitive than R–tree indexes to the choice of clustering fields and the distribution of the data. Further, for spatial joins, the number of buffer reads is better than the number of page reads as an indicator of query response times. Further, using a method by Faloutsos and Kamel (1994), after computing the fractal dimension of the Monet catalog data, we estimate the expected number of page reads for spatial window queries. We find that clustering of data records in correspondence to their location in an R–tree index, will significantly improve the performance of spatial queries.

We conclude that a spatial datablade with approximately half the space overhead of the Shapes2 datablade that supports clustered R–tree indexes will provide better performance than that provided by any of the schemes examined in our experiments.

Finally, we present a number of user–defined–functions for extending the Informix Universal Server.

[1]Supported in part by NASA, Contract No. NAS5–32337.

[2]Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, E-mail: {kalpakis,pasad,mriggs1}@csee.umbc.edu

[3]CESDIS/USRA, Code 930.5, NASA Goddard Space Flight Center, Greenbelt, MD 20771

[4]NASA ESDIS, Code 586, NASA Goddard Space Flight Center, Greenbelt, MD 20771, Email: jeanne.behnke@gsfc.nasa.gov

Contents

1 Introduction

There are many catalogs of stellar objects and observations at various data centers that are available to the public and to astronomers through online systems and in many cases through the Web. Astronomers have greater and faster access to stars than they ever had. However, there are still problems associated with searching single and multiple data catalogs. The problems are especially compounded when the catalogs grow in size from a few thousand objects to millions of objects and involve spatial attributes of such objects. At present, there has been no evaluation of how current astronomical data centers would scale their catalogs to large catalogs. It is expected that with the new missions, such as AXAF (Advanced X–Ray Astronomy Facility) and NGST (Next Generation Space Telescope) the size of the catalogs will grow enormously.

The goals of this project are to

1. Demonstrate the use and investigate the performance of extended data types for enhanced data retrieval and storage management for large catalogs in Object–Relational database systems.
2. Investigate the utility of object relational technology for datamining.

Some of the issues addressed include: How does performance of spatial joins degrades with increasing the size and the number of the catalogs? How should catalogs be organized so that queries on non–spatial attributes be answered effectively? What is the most effective way to perform transformations on the data? (e.g. transform coordinates with user–defined functions or by precomputing the transformation?) Is there any structure information that can be derived from the information in a catalog? (e.g. is there a relationship between location and magnitude? Are yellow dwarves found in the galactic poles rather than the galactic plane?)

In choosing an application that would allow us to demonstrate these goals, we had the following criteria:

1. Size. The application should involve a large set of objects. The motivation behind this criterion is to investigate the scalability of query processing and the effectiveness of user–defined types, functions, and access methods.
2. Spatial objects. The application should involve extensive use of spatial data as well as associated traditional attribute data with the application's objects. Relational databases are ill–equipped to deal with spatial data [9]. This is because (a) they do not allow for the creation and manipulation of spatial data–types as *first–class citizens* (e.g. creating and manipulating boxes, circles, polygons etc. in the same way as integers, character strings etc.), and (b) they do not allow for the construction and/or use by the query optimizer of spatial or user–defined access methods (e.g. R-tree indexes, etc). Object–Relational database systems (e.g. Informix Universal Server/UDO) provide mechanisms to accomplish both items above to a certain extent. The issue to investigate is whether such mechanisms are efficient and effective in manipulating spatial data.

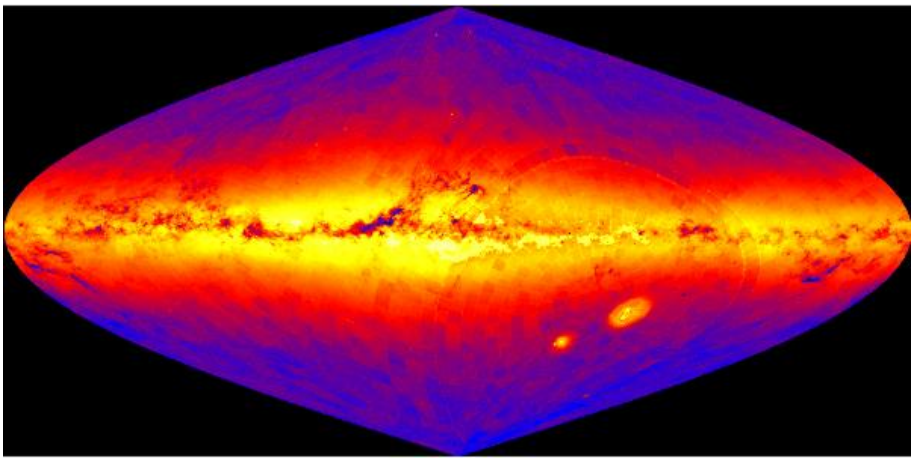


Figure 1: An image of the Monet catalog (`http://www.nofs.navy.mil/projects/pmm/universe.html`) showing the density (number of stars per square degree) on the sky. Density per square degree increases from blue (≈ 500) to yellow (≈ 150000).

3. Scientific datamining. In order to investigate an Object–Relational database system for datamining, we need a dataset for which proven hypotheses exist. Such a dataset can be used to validate the datamining algorithms and hypotheses.

1.1 The Monet Application

We chose to use the USNO-A2.0 catalog (the Monet catalog) as the test–case application. Monet is a catalog of approximately 5×10^8 stars, based on Precision Measuring Machine (PMM) scans. [5]This catalog is used for astrometry. [6] Each catalog entry representing a star is simple, essentially consisting of the celestial coordinates of the star, and the blue and red magnitudes of the star. The difference between the blue and red magnitudes of a star is an important quantity called the *color*. The color of a star gives an indication of its physical properties. Despite the simplicity of the data, there are obvious opportunities to find patterns. Some example patterns/hypotheses are: are stars of different magnitude distributed differently in the celestial sphere? do stars cluster? are *yellow* stars more common in the high northern hemisphere than anywhere else? Having Monet in an Object–Relational database, such as Informix, and using user–defined types and functions, such as the Geodetic and/or the Shapes2 datablades in Informix, should make it easier to datamine for such correlations. This is because (i) The level of abstraction provided makes applications easier to write (e.g. an application can refer directly to properties of a star object), and (ii) spatial access methods can be used to facilitate efficient query processing.

The Monet catalog was suggested to us by scientists at the High Energy Astrophysics Archive

[5]In fact, not all records represent stars; many are quasars or galaxies. We follow the convention of using star as a generic label for astronomical objects.

[6]Astrometry is the analysis and measurement of celestial bodies, their motion and positions.

(HEASARC), (http://heasarc.gsfc.nasa.gov). HEASARC is an online archive consisting of several dozen sky surveys, each stored in a Sybase Relational database system. A user can view a single catalog at a time, or can make use of a cross–correlation feature which allows a user to query two surveys simultaneously, as well as to ask for all objects seen by both surveys (i.e. to join two surveys on their coordinates.) The cross–correlation feature is of enormous importance, and is one of the main contributions of HEASARC. [7]

Table joins are expensive in terms of elapsed time, and in many cases cannot be performed online (within 3 secs or less elapsed time) on large tables (tables with tens of thousands rows). For example, we have been told that HEASARC does not usually evaluate joins with tables of more than 500 rows on behalf of web users. [8] Although Monet will soon be added to the HEASARC collection, it is clearly not easy to join another table with it using the existing architecture. We can perform spatial self–joins of tables with 1,000 rows, with 10,000 rows, with 20,000, and with 40,000 rows in approximately 3 sec, 84 secs, 182 secs, and 428 secs average elapsed time respectively.

Furthermore, the Monet catalog clearly satisfies the criteria of size (e.g. about 500 million objects), spatial objects (e.g. stars on a 2-d plane), and scientific datamining (e.g. color/location relationships) we set for a test application.

We perform a set of experiments to demonstrate and assess the performance and utility of user–defined extensions (types, functions, and access methods) of the Informix Universal Server for the Monet catalog. The results of our experiments are discussed in section 4.

The rest of this paper is organized as follows. Section 2 describes the experimental setup, which includes the software and hardware used, the database schema, the data preprocessing and loading, the test queries generated, the indexes created, and the performance measures collected. Section 3 describes how our experiments are conducted. Section 4 presents the results of the experiments and their analyses. Section 5 discusses an analysis of spatial queries using the fractal dimension of the data for estimating the average number of page reads in processing such queries. Section 6 presents a number of user–defined functions we developed for extending the functionality of the database server. Section 7 discusses the development environment. Section 8 concludes this paper.

2 Experimental Setup

In this section we describe the hardware and software configuration we used for the experiments we conducted. We also describe the schema of the tables created for the experiments.

2.1 Software and Hardware Configuration

The hardware platform used to conduct all our experiments was a Sun Ultrasparc 60 with 512MB of physical memory running Solaris 2.6 (isc587.gsfc.nasa.gov) with two 8GB SCSI disks and two

[7] The importance of this feature is illustrated below, in section 2.2, in the discussion of spatial join queries.

[8] Currently, the largest table in HEASARC has 20,000 rows.

4GB SCSI disks.

The Object–relational database system we use is the Informix Universal Server/UDO, version 9.14, installed on the platform above. The configuration parameters for the database server are given in Appendix D.

The main advantage of Informix Universal Server is its support for the creation of user defined data types (UDTs), functions (UDFs), and access methods. Informix and its partners have developed a variety of *datablades*, each of which is a collection of UDTs and UDFs related to a particular application domain. We experimented with two such datablades: (i) the *Informix Geodetic DataBlade*, and (ii) the *Shapes2 DataBlade* which was created by Informix as an illustration of datablade design principles. Our primary interest in these datablades lay in the fact that they support *point* and *rectangle* user–defined datatypes, *contains* and *overlap* user–defined functions for these datatypes, and an R–tree access method for these datatypes, all of which are useful for processing spatial queries. The Geodetic and Shapes2 datablades are described in Appendix B.

2.2 Table Schemas

We created three different table schemas: (i) one schema that uses the relational only features of Informix, (ii) one schema that uses the features added by the Geodetic datablade, (iii) one schema that uses the features added by the Shapes2 datablade.

The ranges and dimensions for each attribute of a star in the Monet catalog are as follows (see also appendix A):

- `field` an integer in the range [1, 1431].
- `blue` an integer in the range [0, 250]. This is 10 times the actual blue magnitude of the star. We represent magnitude this way so that we can store it as `smallint` instead of `smallfloat`.
- `red` same as `blue` above.
- `RA` a real number in the range (-180, 180], representing the right ascension of a star in decimal degrees.
- `dec` a real number in the range [-90, 90], representing the declination of a star in decimal degrees.

Since there will be no inserts or updates to the table during our experiments, there is no reason to incur the overhead that comes with specifying these ranges via constraints in the `CREATE TABLE` SQL statement. All data loaded from the Monet catalog are assumed to be valid. Further, we may assume that declination and right ascension form a key for stars (but that may not be true due to measurement/instrument calibration errors in the star's location).

In the Geodetic datablade, we represent the location of a star using the `GeoPoint` datatype, an extended datatype provided by the Geodetic datablade which contains as attributes the extended datatypes `lat` and `long`, which we use to represent the declication and right ascension respectively.

In the Shapes2 datablade, we represent the location of a star using the `MyPoint` datatype, an extended datatype provided by the Shapes2 datablade that consists of the attributes x and y which we use to represent declination and right ascension respectively.

Further, due to a run–time error in using user–defined functions that create instances of the user–defined datatypes during the processing of spatial queries over large tables (see sections 6.2.1 and 6.2.2), we are precomputing the uncertainty window for each star and storing it with some of the tables below as an attribute `ebox` of datatype `GeoBox` (in the Geodetic datablade) or `MyBox` (in the Shapes2 datablade). The uncertainty window of a star captures the error in the location of a star due to calibration errors of the measuring instruments. The the uncertainty window is specified by its lower left and upper right corner coordinates. The `GeoPoint, GeoBox, MyPoint` and `MyBox` datatypes are described in Appendix B.

In addition, in order to be able to cluster the objects in a table according to attributes of their locations, and since neither the Geodetic nor the Shapes2 datablades allow us to create clustered indexes on attributes of the `GeoPoint` or `MyPoint` datatypes, for tables that use those datatypes and we want to cluster them on location attributes, we define two more attributes (`dec` and `RA`).

For simplicity, we refer to the traditional relational schema of the database as the **B** schema, while we refer to the schema that uses the data types defined by the Geodetic (Shapes2) datablade as the **G** (**S**) schema. Table 1 lists the acronyms that we are using hereafter.

Table 1: Acronyms used in the tables that show the ranking of the various schemas and various clusterings.

BDR	**B** schema, clustering on dec–RA
BRD	**B** schema, clustering on RA–dec
GDR	**G** schema, clustering on dec–RA
GRD	**G** schema, clustering on RA–dec
SDR	**S** schema, clustering on dec–RA
SRD	**S** schema, clustering on RA–dec

Examples of SQL statements creating tables for each schema are as follows:

- **B** schema:

```
CREATE TABLE Btable(
field smallint,
blue smallint,
red smallint,
RA smallfloat,
dec smallfloat);
```

- **G** schema:

```
CREATE TABLE GeoRtableC (
dec smallfloat,
```

```
RA smallfloat,
field smallint,
blue smallint,
red smallint,
coords GeoPoint,
ebox GeoBox);
```

- **S** schema:

```
CREATE TABLE ShapesRtableC (
dec smallfloat,
RA smallfloat,
field smallint,
blue smallint,
red smallint,
coords MyPoint,
ebox MyBox);
```

2.3 Data Preprocessing and Loading

The Monet catalog consists of 24 files binary encoded files, each file representing 7.5 degrees of declination. Within each file, stars are stored in order of increasing RA. All data in our experiments are taken from the file `zone0000.cat`, the first binary file of the Monet catalog. This file contains all stars with declination between -90 degrees and -82.5 degrees, and is sorted in increasing order of right ascension.

We have written the C program `LoadFile` that reads a binary encoded file from the Monet catalog, and formats the data so that they can be loaded into database tables using the Informix `LOAD` command. Example output from the `LoadFile` program, whose usage is described in Appendix C, is as follows:

- **B** schema:
  `20003|192|178|35.384858|-84.129550|`
- **G** schema:
  `-84.129550|35.384858|20003|192|174|GeoPoint((-84.129550,35.384858),ANY,ANY)|`
  `GeoBox( (-84.162883, 35.351525), (-84.096217, 35.418191), ANY, ANY)|`
- **S** schema:
  `-84.129550|35.384858|20003|192|174| point(-84.129550,35.384858) |`
  `box(-84.162883, 35.351525, -84.096217, 35.418191)|`

We experiment with tables for each schema with different number of stars: 1,000, 10,000, 20,000, 40,000, 60,000, 100,000, and 140,000 stars each.

2.4 Queries

The Monet catalog is currently used for astrometry, the discipline of determining the precise coordinates of observed stars. This involves a single type of query, spatial window, defined below. In addition to this capability, we want to provide users with the ability to correlate records in the Monet catalog with records in other star catalogs. Traditionally, correlation is done via equi–join queries. However, since there can be errors in the measurement of the exact location of objects, correlation of stars among catalogs is better handled via a second type of query, spatial join, which we also define below.

If we only wanted to support *spatial window* and equi–join queries for the Monet catalog, then traditional relational database systems would suffice. However, since processing *spatial join* queries by relational database systems is quite inefficient (see chapter 14 in [9]), and we also want to enable datamining, we need to pay close attention to spatial join queries. [9] Note that the parameters of the queries will be different for datamining queries than for non–datamining queries. We describe these queries below. Parameters of the queries are generated randomly, in a manner described below.

The description of each query type is followed by an example of our implementation for the traditional relational schema, and examples of our implementation for the schemas for the Geodetic and Shapes2 datablades.

2.4.1 Spatial Window Queries

Spatial window queries are of the form "Find all stars within a user-defined bounding rectangle".

For astrometry, this query is only intended to return a single star. The rectangle represents the uncertainty in the position of the star. For the Monet catalog, this uncertainty is 0.25 arc seconds. Due to the limited precision of the Geodetic datablade, we selected an uncertainty value of $\epsilon = 2 \times 0.033333$ decimal degrees, and we used ϵ as the value for the extend of the spatial windows in our experiments with queries of this type. For datamining, our rectangles would be of any size. We specify a rectangle by specifying its lower left and upper right hand corner point; a point is specified by its declination (dec) and right ascension (RA).

The parameters for this query type are: the table `<tablename>` to be queried, and the two corner points of the spatial window. See section 3 for values selected for the corner points. Examples of spatial window queries for each schema, are as follows: [10]

- **B** schemas:

```
SELECT count(*), avg(field) FROM <tablename>
WHERE dec ≥ x_l AND RA ≥ y_l AND dec ≤ x_u AND RA ≤ y_u;
```

[9] A spatial join is a theta join with spatial constraints.

[10] In order to force the database to retrieve the actual records, we included the aggregate `avg(field)`.

- **G** schemas:

```
SELECT count(*), avg(field) FROM <tablename>
WHERE inside(<GeoPoint>, <GeoBox>);
```

  where `<GeoPoint>` is the location attribute (`coords`) of the object and `<GeoBox>` is the spatial window, set to `'((`x_l`, `y_l`), (`x_u`, `y_u`),ANY,ANY)'::GeoBox`.

- **S** schemas:

```
SELECT count(*), avg(field) FROM <tablename>
WHERE within(<MyPoint>, <MyBox>);
```

  where `<MyPoint>` is the location attribute (`coords`) of the object and `<MyBox>` is the spatial window, set to `'box(`x_l`, `y_l`, `x_u`, `y_u`)'`.

2.4.2 Spatial Or–Window Queries

Spatial Or–Window queries are of the form "Find all the stars within at least one of two different bounding rectangles." This query is like the Spatial Window Query except two bounding rectangles are specified instead of one. The purpose of this query was to check the efficiency of the schemas that use the datablades. It was expected that only one pass of the R–tree index will be needed to perform the query while the **B** schemas might have to make two passes over the entire index. The description of the query is the same as that of the Spatial Window query. The two random windows were generated independently in a similar manner as for the spatial window queries. Examples of the Or–Window are as follows:

- **B** schemas:

```
SELECT count(*), avg(field)
FROM <tablename>
WHERE (dec ≥ x_l1 AND RA ≥ y_l1 AND dec ≤ x_u1 AND RA ≤ y_u1 )
UNION
SELECT count(*), avg(field)
FROM <tablename>
WHERE (dec ≥ x_l2 AND RA ≥ y_l2 AND dec ≤ x_u2 AND RA ≤ y_u2 );
```

- **G** schemas:

```
SELECT count(*), avg(field) FROM <tablename>
WHERE inside(<GeoPoint>, <GeoBox_1>)
UNION
SELECT count(*), avg(field) FROM <tablename>
WHERE inside(<GeoPoint>, <GeoBox_2>);
```

where <GeoPoint> is the location attribute (coords) of the object and <GeoBox$_1$> and <GeoBox$_2$> are the spatial windows. Each GeoBox is of type '((x_l, y_l), (x_u, y_u),ANY,ANY)'::GeoBox.

- **S** schemas:

```
SELECT count(*), avg(field) FROM <tablename>
WHERE within(<MyPoint>, <MyBox1>)
UNION
SELECT count(*), avg(field) FROM <tablename>
WHERE within(<MyPoint>, <MyBox2>);
```

where <MyPoint> is the location attribute (coords) of the object and <MyBox$_1$> and <MyBox$_2$> are the spatial windows. Each <MyBox> is of type 'box(x_l, y_l, x_u, y_u)'.

2.4.3 Spatial Self–Join

Spatial self–join queries are of the form "Find all stars in the table whose "uncertainty boxes" overlap the uncertainty boxes of other stars".

We consider this type of query for several reasons. First, spatial joins are important for correlating different catalogs, especially when there is uncertainty in the location of the stars in those catalogs (almost always the case), in which case the traditional equi–join catalog correlation is not applicable, and the general theta–join is inefficient (as we will see from the experimental results for spatial self–joins over the **B** schemas). Second, spatial self–join is indicative of the performance of spatial joins with tables of similar size. Third, self–joins form an important query type for datamining, appearing in many datamining algorithms.

The utility of spatially joining two tables is illustrated by an example. Suppose a scientist has a new theory about the relationship between the x–rays and gamma–rays emitted by stars. To test her theory, she will want to know all stars that were captured by both x–ray and gamma–ray surveys. That is, she will want to join tables of x–ray data with tables of gamma–ray data. Ideally, she would want to perform an equi–join on the coordinates of the stars in the two tables. But, due to calibration problems the same star might have slightly different coordinate values in the two tables. To allow for such differences she will want to perform a spatial join query.

Spatial self–join are expensive queries. Without an appropriate access method, it can take $O(n^2)$ block accesses to evaluate a spatial self–join on a table with n blocks.

The parameters for the spatial self–join queries are: the name <tablename> of the table to be joined, and the uncertainties ϵ_1 and ϵ_2 on the declination and right ascension respectively. Examples of spatial self–join queries for each schema, are as follows:

- **B** schemas:

```
SELECT count(*), avg(A.field), avg(B.field)
FROM <tablename> A, <tablename> B,
WHERE B.dec ≤ ( A.dec + 2 * ε1 ) AND B.dec ≥ ( A.dec - 2 * ε1 ) AND
      B.RA ≤ ( A.RA + 2 * ε2 ) AND B.RA ≥ ( A.RA - 2 * ε2 );
```

- **G** schemas:

```
SELECT count(*), avg(A.field), avg(B.field)
FROM <tablename> A, <tablename> B
WHERE inside ( <GeoPoint(B)>, <GeoBox(A)> );
```

  where `<GeoPoint(B)>` is the GeoPoint attribute (`coords(B.coords)`) of table B, and `<GeoBox(A)>` is a `GeoBox` centered on the GeoPoint attribute (`coords`) of A and extending $4\epsilon_1$ and $4\epsilon_2$ on declination and right ascension respectively. We have written a function `MakeGeoBox` that dynamically constructs, using the expression `'MakeGeoBox(coords(A.coords),` $2 * \epsilon_1$, $2 * \epsilon_2$`)'`, such a `GeoBox` for each object in A. However, as for the spatial window queries, the database server was running out of shared memory when we attempted to use this function on tables with 100,000 objects or more (see section 6.2.1). So, instead, we precomputed these `GeoBox`es and stored them with the data tables.

- **S** schemas:

```
SELECT count(*), avg(A.field), avg(B.field)
FROM <tablename> A, <tablename> B
WHERE within(<MyPoint(B), <MyBox(A)>);
```

  where `<MyPoint(B)>` is the location attribute (`coords`) of B, and `<MyBox(A)>` is a rectangle similar to `<GeoBox(A)>` above. We have written a function `MakeShapesBox` that constructs that rectangle dynamically via the expression `'MakeShapesBox(A.coords,` $2 * \epsilon_1$, $2 * \epsilon_2$`)'`. As before, the server ran out of memory (see section 6.2.2), and therefore, we precomputed these boxes and stored them with the data tables.

It is interesting to contrast the complexity of the queries for the **B** schemas with the simplicity of the queries for the **G** and **S** schemas. From the point of view of amount and complexity of code that an application developer needs to write, clearly the use of the datablades is beneficial, even when use of UDFs for queries over large tables is problematic.

2.4.4 Multiple Join Queries

This type of queries is used to evaluate the performance of the various indexes as the number of spatial joins increases. There are two ways the joins could be performed. In the *chain–join*, the result of one join is used for the next join in a chain manner. For example, in a 3–chain–join involving tables A, B, and C, we join table A with B and table B with table C. In a *star–join*, we join multiple tables with one table. For example, in a 3–star–join involving tables A, B, and C, we join table B with table A and table C with table A as well.

2.4.5 Spatial Window Chain–Join Queries

Spatial window chain–join queries are defined as k–chain–join queries using the same window for all the spatial joins. For $k = 2$ it is of the form "Find all stars in the table B that are within error (ϵ_1, ϵ_2) of stars of table A within a given window". For $k = 3$ it is of the form "Find all stars in the table C that are within error (ϵ_1,ϵ_2) of stars in table B, only for those stars in table B that are within error (ϵ_1, ϵ_2) of stars in table A that are within a given window". In general, a spatial window k–chain–join is of the form $\sigma_F(A_1) \wedge (((A_1 \bowtie_W A_2) \bowtie_W A_3) \bowtie_W \ldots \bowtie_W A_k)$ where $\bowtie_W$ denotes a spatial join with window W, and σ_F is a select with spatial window constraint F. In our experiments, F is a window constraint with the window centered on a randomly selected point and having extend in both directions 10ϵ, and W is a window of extend 2ϵ in both directions. Examples of 3–chain-join queries for the various schemas are given below:

- **B** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE (A.dec ≥ x_l AND A.RA ≥ y_l AND A.dec ≤ x_u AND A.RA ≤ y_u) AND
      (B.dec ≤ (A.dec + ε1) AND B.dec ≥ (A.dec - ε1) ) AND
      (B.RA ≤ (A.RA + ε2) AND B.RA ≥ (A.RA - ε2) ) AND
      (C.dec ≤ (B.dec + ε1) AND C.dec ≥ (B.dec - ε1) ) AND
      (C.RA ≤ (B.RA + ε2) AND C.RA ≥ (B.RA - ε2) );
```

- **G** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE inside(A.coords, <GeoBox>) AND
      inside(B.coords, A.ebox) AND inside(C.coords, B.ebox);
```

  where `GeoBox` is of type `'((`x_l`, `y_l`), (`x_u`, `y_u`), ANY, ANY)'::GeoBox`.

- **S** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE within(A.coords, <MyBox>) AND
      within(B.coords, A.ebox) AND within(C.coords, B.ebox);
```

  where `<MyBox>` is the spatial window, set to `'box(`x_l`, `y_l`, `x_u`, `y_u`)'`.

2.4.6 Spatial Window Star–Join Queries

Spatial window star–join queries are defined as k–star–join queries over a window. For $k = 3$ it is of the form "Find all stars in the tables B and C that are within error (ϵ_1,ϵ_2) of stars in table A that are within a given window". Note that spatial 2–star joins are the same as spatial 2–chain–joins. However, for $k \geq 3$, k–star–joins are different than k–chain–joins. In general, a spatial window k–star–join is of the form $\sigma_F(A_1) \wedge (A_1 \bowtie_W A_2) \wedge (A_1 \bowtie_W A_3) \wedge \ldots \wedge (A_1 \bowtie_W A_k)$ where $\bowtie_W$ denotes a spatial join with window W, and σ_F is a select with spatial window constraint F. We generated F and W in the same way as for the k–chain–join queries. Examples of 3–star-join queries for the various schemas are given below:

- **B** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE (A.dec ≥ yl AND A.RA ≥ xl AND A.dec ≤ yu AND A.RA ≤ xu) AND
      (B.dec ≤ (A.dec + ε1) AND B.dec ≥ (A.dec - ε1)) AND
      (B.RA ≤ (A.RA + ε2) AND B.RA ≥ (A.RA - ε2)) AND
      (C.dec ≤ (A.dec + ε1) AND C.dec ≥ (A.dec - ε1)) AND
      (C.RA ≤ (A.RA + ε2) AND C.RA ≥ (A.RA - ε2));
```

- **G** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE inside(A.coords, <GeoBox>) AND
      inside(B.coords, A.ebox) AND inside(C.coords, A.ebox);
```

  where `<GeoBox>` is of type `'((`x_l`, `y_l`), (`x_u`, `y_u`), ANY, ANY)'::GeoBox`.

- **S** schemas:

```
SELECT count(*), avg(A.field), avg(B.field), avg(C.field)
FROM <tablename1> A, <tablename2> B, <tablename3> C
WHERE within(A.coords, <MyBox>) AND
      within(B.coords, A.ebox) AND within(C.coords, A.ebox);
```

  where `<MyBox>` is the spatial window, set to `'box(`x_l`, `y_l`, `x_u`, `y_u`)'`.

2.5 Indexes

For each table schema, we create a number of indexes on location attributes of stars. If there are attributes of spatial datatypes for which R–tree indexes can be created, e.g. points or boxes, then

we create such R–tree indexes. Otherwise, we only create B–tree indexes. In certain cases, we also convert a non-clustered index to a clustered index in order to cluster the objects on some location attributes. The motivation behind different clusterings of objects (e.g. distribution of stars into database pages) is to investigate the effect of such clusterings on the performance of spatial queries. Note that, if there has not been any clustered index on a table then the objects are distributed into pages according to the order they have been inserted or loaded. Further, since in the Monet application there will not be frequent insert or update operations creating many indexes is not a critical issue for the performance of such operations.

In particular, for each table schema we create the following indexes:

- **B** schemas:

```
CREATE INDEX <index name> ON <tablename>(RA,dec) USING BTREE;
CREATE INDEX <index name> ON <tablename>(dec,RA) USING BTREE;
```

- **G** schemas:

```
CREATE INDEX <index name> ON <tablename>(coords GEOOBJECT_OPS) USING
RTREE;
CREATE INDEX <index name> ON <tablename>(ebox GEOOBJECT_OPS) USING
RTREE;
CREATE INDEX <index name> ON <tablename>(RA,dec) USING BTREE;
CREATE INDEX <index name> ON <tablename>(dec,RA) USING BTREE;
```

- **S** schemas:

```
CREATE INDEX <index name> ON <tablename>(coords SHAPE2_OPS) USING
RTREE;
CREATE INDEX <index name> ON <tablename>(ebox SHAPE2_OPS) USING
RTREE; CREATE INDEX <index name> ON <tablename>(RA,dec) USING BTREE;
CREATE INDEX <index name> ON <tablename>(dec,RA) USING BTREE;
```

For each table size, we have a set of test queries S to be executed. We describe how this set of queries is constructed in section 3. In conducting the experiments, for each table size, we cluster the table on RA–dec, we run the queries in S, then we cluster on dec–RA, and rerun the queries in S. For example, assuming a table with no clustered indexes [11], we convert an ordinary index to a clustered index by the command

```
ALTER INDEX <index name> TO CLUSTER;
```

and we can convert a clustered index to an unclustered index by the command

```
ALTER INDEX <index name> TO NOT CLUSTER;
```

[11] A table can have at most one clustered index.

2.6 Performance Measurements

We run queries using the ESQL–C program `sqlplus`, a command line interface that we developed to process queries, and collect performance measures in executing those queries. The `sqlplus` program writes, for each query it executes, log information into three files: it writes the executed query into the `queries.stats` file, the query plan for the executed query into the `sqexplain.out` file, and the values for the measures above into the `measures.stats` file. Usage instructions for `sqlplus` are given in Appendix C.

The following performance measures are collected by the `sqlplus` program using appropriate C library functions.

- *clientElapsedTime:* total elapsed time observed by the client.
- *clientTotalCpuTime:* total CPU time (in both system and user mode) consumed by the client.
- *clientUserModeCpuTime:* total CPU time in user mode consumed by the client.
- *clientSystemModeCpuTime:* total CPU time in system (kernel) mode consumed by the client.

The following measures are collected by `sqlplus` by querying the Informix system table `sysmaster.sysvpprof` before and after each query.

- *serverUserModeCPUTime:* The number of microseconds of user CPU time, totaled over all server virtual processors.
- *serverSystemModeCPUTime:* The number of microseconds of system CPU time, totaled over all server virtual processors.
- *serverTotalCPUTime:* the sum of the above two measures.

The following performance measures are collected by `sqlplus` by querying the Informix system table `sysmaster.syssesprof` before and after each query.

- *IsReads:* The number of ISAM read calls (usually SELECT).
- *IsWrites:* The number of ISAM write calls (usually INSERT).
- *IsDeletes:* The number of ISAM delete calls.
- *IsCommits:* The number of times a server performs a commit.
- *IsRollbacks:* The number of times a transaction is rolled back.
- *LongTransactions:* The number of transactions that fill the log buffers and need to be aborted.
- *BufferReads:* The number of times a read occurred against the buffer pool in shared memory.

- *BufferWrites:* The number of writes made against a page in the buffer pool.
- *SequentialScans:* The number of sequential scans that have occurred. Due to our heavy use of indexes, this number should be very low.
- *PageReads:* The number of pages read from disk.
- *PageWrites:* The number of pages written to disk.
- *TotalSorts:* The total number of sorts.
- *DiskSorts:* The number of sorts that did not fit into memory.

REMARK.
The timing measurements (e.g. client elapsed time, etc) by the `sqlplus` program should be treated cautiously since the resolution of the clock is limited to the 1 microsec resolution of the `gettimeofday` C function, as well as interference with the Operating System due to multiprogramming.

3 Experiments

We loaded the first 1,000, 10,000, 20,000, 40,000, 60,000, 100,000, and 140,000 stars in `zone000.cat` file from the Monet catalog into appropriate tables from each schema in the database. We also build, for each table, all the unclustered indexes specified in section 2.5.

We generate sample queries using the C program `gen_queries` we have written. The `gen_queries` program randomly generates the arguments for all query types (uniformly distributed over the declination and right ascension range of the stars loaded into the corresponding tables.

For each table size, the `gen_queries` program generates a set of test queries S consisting of 20 spatial window queries, 20 spatial Or–window queries, 3 spatial self–join queries, 20 k–chain–join queries and 20 k–star–join queries, for $k = 2, 3$, and 10 k–chain–join queries and 10 k–star–join queries for $k = 4, 5$. This set of queries S is written to a file. The parameters $x, y, \epsilon, \epsilon_1, \epsilon_2, x_l, y_l, x_u, y_u$ for the queries (see section 2.4) are set as follows. Randomly (uniform distribution) select a point (x, y) among the points loaded in the table. Set $\epsilon = \epsilon_1 = \epsilon_2 = 0.033333$ decimal degrees. For the spatial window and or–window queries, set $x_l = x - \epsilon_1$, $y_l = y - \epsilon_2$, $x_u = x + \epsilon_1$, and $y_u = y + \epsilon_2$. For the spatial chain–join and star–join queries, set $x_l = x - 3\epsilon_1$, $y_l = y - 3\epsilon_2$, $x_u = x + 3\epsilon_1$, and $y_u = y + 3\epsilon_2$. Note that the same window is used for all schemas and the same query type.

We execute those queries using the `sqlplus` program to collect performance measures for the unclustered tables. Then, we alter the appropriate B–tree indexes (on declination; on right ascension; or on their combinations) from non–clustered to clustered, one at a time; re-execute the same queries and collect the performance measurements for that clustered table.

We have also collected performance measurements for loading data into database tables, creating B–tree and R–tree indexes, and altering a non–clustered index to a clustered index.

We compute statistics for the measurements obtained for those queries using the C program `analyze` we developed. The `analyze` program computes the the minimum, maximum, average, and standard deviation for each measurement for each query type and each schema and clustering. It also outputs appropriate commands for generating a variety of plots using Matlab. Further, it creates LaTeXfiles tabulating the statistical information above.

Usage instructions for `gen_queries` and `analyze` are given in Appendix C.

4 Analysis of Experimental Results

4.1 Disk Space Requirements

In this section, we summarize the disk space requirements for the different schemas and clusterings. Tables 2 and 3 show the space requirements for the various schemas per star. The average and standard deviation for the number of bytes per star are computed by considering the following quantity: number of disk blocks times size of a disk block divided by the number of stars.

The amount of storage required by the **S** schemas for the data only is about 10 times that required for the **B** schema, while the amount of storage required for creating all the indexes for the **S** schemas is about three times that required for the **B** schemas. The **G** schemas require about 30% extra space for the data and the indexes than the **S** schemas.

In comparing the space efficiency of the user–defined datatypes for the **G** and **S** schemas, we note that they require approximately 420 and 300 bytes respectively, while storing the same data with the built–in relational datatypes in the **B** schemas requires about 30 bytes. The `MyPoint` and `MyBox` UDTs of the Shapes2 datablade are at least 30% more space efficient than the corresponding UDTs of the Geodetic datablade.

In comparing the space efficiency of the R–tree indexes of the Shapes2 and the Geodetic datablade, we note that an R–tree index on a point requires 141 and 186 bytes respectively, while an R–tree index on a box requires 166 and 225 bytes respectively. Creating R–tree indexes using the Geodetic datablade requires over 30% more space than doing so with the Shapes2 datablade. Creating B–tree indexes on integers or floats requires about 15 bytes per attribute per object for all schemas.

To store all the data of the Monet catalog will require approximately 12.6GB, 140GB, and 190.7GB for the **B** , **S** , and **G** schemas. Building B–tree indexes on a total of 8 attributes (integers or floats) will require approximately 56GB. Building an R–tree index on a point attribute will require 65.6GB and 86.6GB with the Shapes2 and Geodetic datablades respectively. Building an R–tree index on a box attribute will require 77.3GB and 104.8GB with the Shapes2 and Geodetic datablades respectively. Hence, storing the Monet catalog, data and indexes, will consume approximately 69GB, 283GB, and 382GB when using none of the datablades, the Shapes2 datablade, or the Geodetic datablade respectively. Note that the compact storage of the Monet catalog by the USNO requires 6.5GB. An implementation using Shapes2 (Geodetic) will require almost 4 (5.5) times the space of an implementation without these datablades.

Is the extra space worth it? We will see in Section 4.3 under which conditions that extra space consumption is beneficial with respect to client elapsed time, number of buffer reads, and number of page reads.

Table 2: Space consumed for the data and for creating all indexes (B–tree indexes on the attributes dec, RA, dec–RA, RA–dec, blue, and red, and if applicable R–tree indexes on the attributes coords and ebox.

Schema	Total bytes per star		Bytes per star for tables		Bytes per star for indexes	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
B	164.93	113.80	26.97	16.60	137.96	97.24
S	710.58	138.76	298.50	12.90	412.08	125.87
G	930.37	139.26	418.11	17.96	512.26	121.31

Table 3: Space consumed for indexes.

Schema	Index on Attributes	Index bytes per star	
		Avg.	Std. Dev.
S	Coords	143.21	27.36
G	Coords	186.10	19.25
S	Ebox	167.94	27.25
G	Ebox	225.24	30.81
B	Dec	22.84	14.96
S	Dec	23.24	18.72
G	Dec	23.24	18.72
B	Dec and RA	29.74	19.02
S	Dec and RA	27.19	16.94
G	Dec and RA	27.19	16.94
B	RA	22.94	14.90
S	RA	23.29	18.70
G	RA	23.29	18.70
B	RA and Dec	29.74	19.02
S	RA and Dec	27.19	16.94
G	RA and Dec	27.19	16.94
B	Blue	16.64	14.61
S	Blue	13.24	8.85
G	Blue	13.24	8.85
B	Red	16.05	14.92
S	Red	12.55	9.15
G	Red	12.55	9.15

4.2 Loading Data and Creating Indexes

The elapsed time to load data into the various schemas is shown in Figure 2. For example, to load 140,000 objects into the **B** schemas takes 71.2 secs. To load the same objects into the **S** schemas takes 232.8 secs. For the **G** schemas the load time is 342.6 secs.

We perform a linear regression at the 95% significance level on the elapsed time to load data into the various schemas. Let the linear regression equation be $t(n) = a_0 + a_1 \cdot n$, and let R^2 be

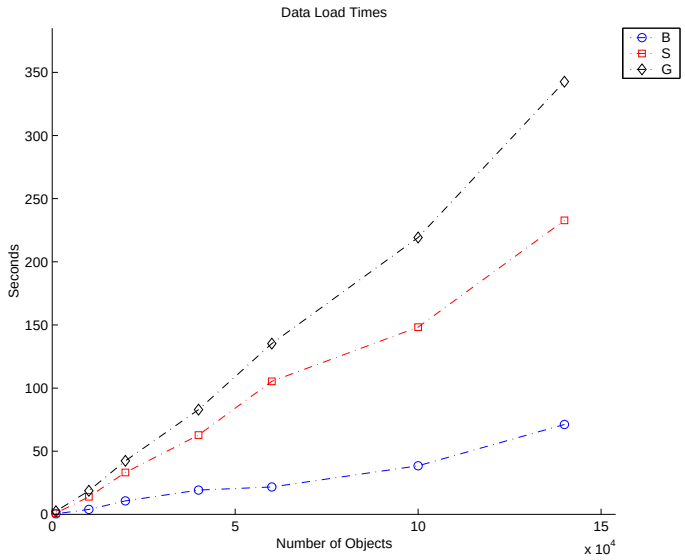


Figure 2: Elapsed time to load data into the the **B** , **S** , and **G** schemas (B, S, G) for various number of objects loaded.

Table 4: Results of linear regression on elapsed time to load data for the various schemas as a function of the number n of objects loaded, $n \geq 1000$. The first column shows the schema type, the second the linear regression equation, the third the 95% confidence interval for the coefficient of n, and the last column shows the estimated average time to load 500 million objects.

Schema	$t(n)$ in secs	C.I.	R^2	$t(5 \cdot 10^8)$ in secs
B	$-1.271 + 0.000471n$	$[0.000364, 0.000578]$	96.24%	235304
S	$-0.786 + 0.001625n$	$[0.001466, 0.001783]$	99.28%	812261
G	$-7.082 + 0.002407n$	$[0.002208, 0.002606]$	99.48%	1203600

the percentage of variability on the data that is explained by the linear regression equation. The regression coefficients, together with 95%–confidence intervals for them, as well as the R^2 for each schema are shown in Table 4. The F–tests of significance at the 95%–level for these regression lines were all significant. Using the computed linear regression equations, the estimated average elapsed time to load 500 million objects is 235304 secs ($\approx$ 2.7 days), 812261 secs ($\approx$ 9.6 days), 1203600 secs ($\approx$ 14.2 days) for the **B** , **S** , and **G** schemas respectively. It takes approximately 3 and 4.8 times more time to load data into the **S** and **G** schemas than into the **B** schemas, respectively. [12]

The performance of creating R–tree indexes is shown in Figure 4. For comparison, we show the performance of creating B–tree indexes for the various schemas in Figure 3, and the performance of altering un–clustered B–tree indexes to clustered indexes in Figure 5.

We perform a linear regression at the 95% significance level on the elapsed time to create R–tree

[12] Recently, using a compact representation similar to the one used in the USNO–A2.0 (Monet) catalog, a load time of 18.5 secs was achieved for 140,000 objects.

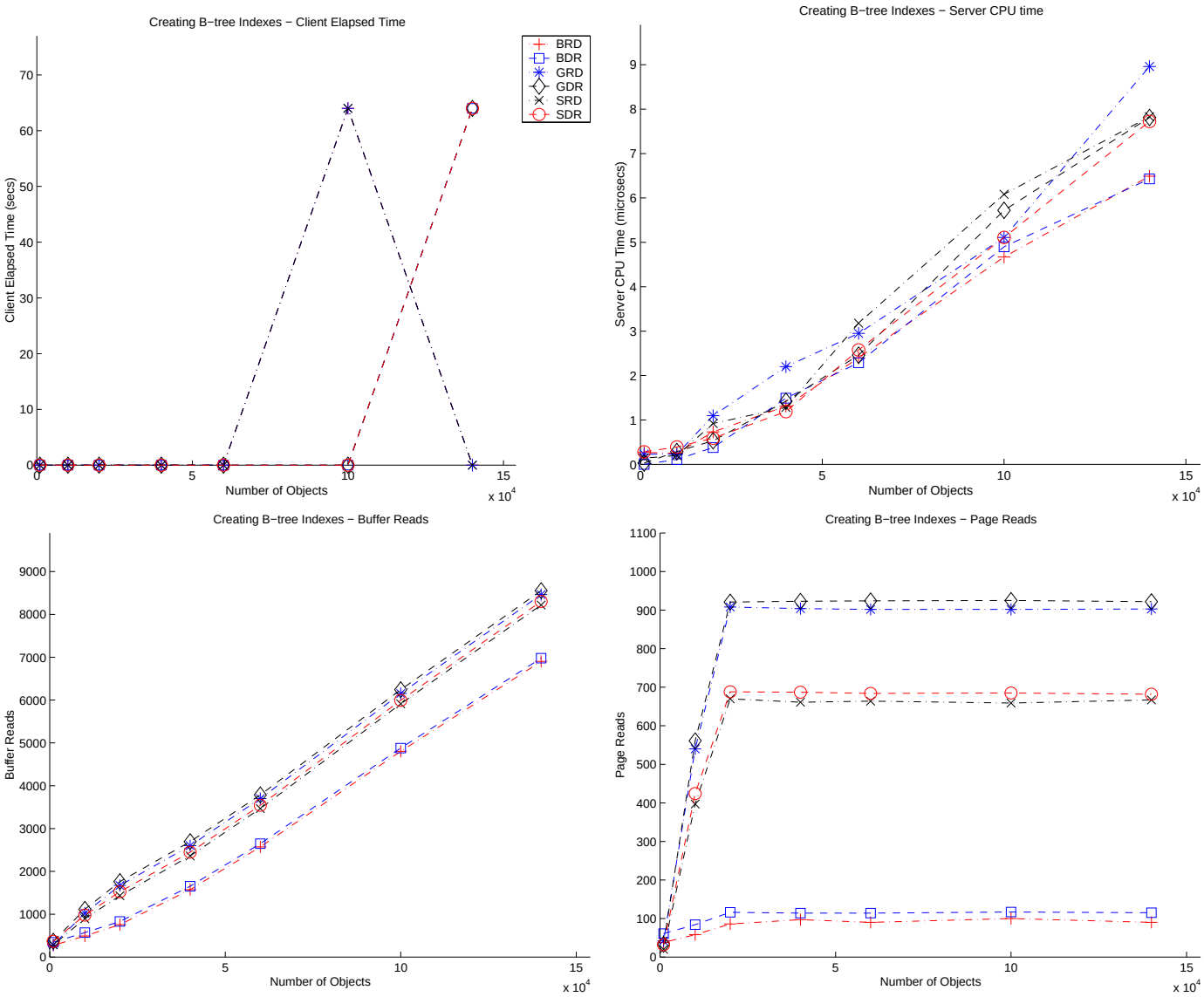


Figure 3: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for creating B–tree indexes on dec–RA (DR) and RA–dec (RD).

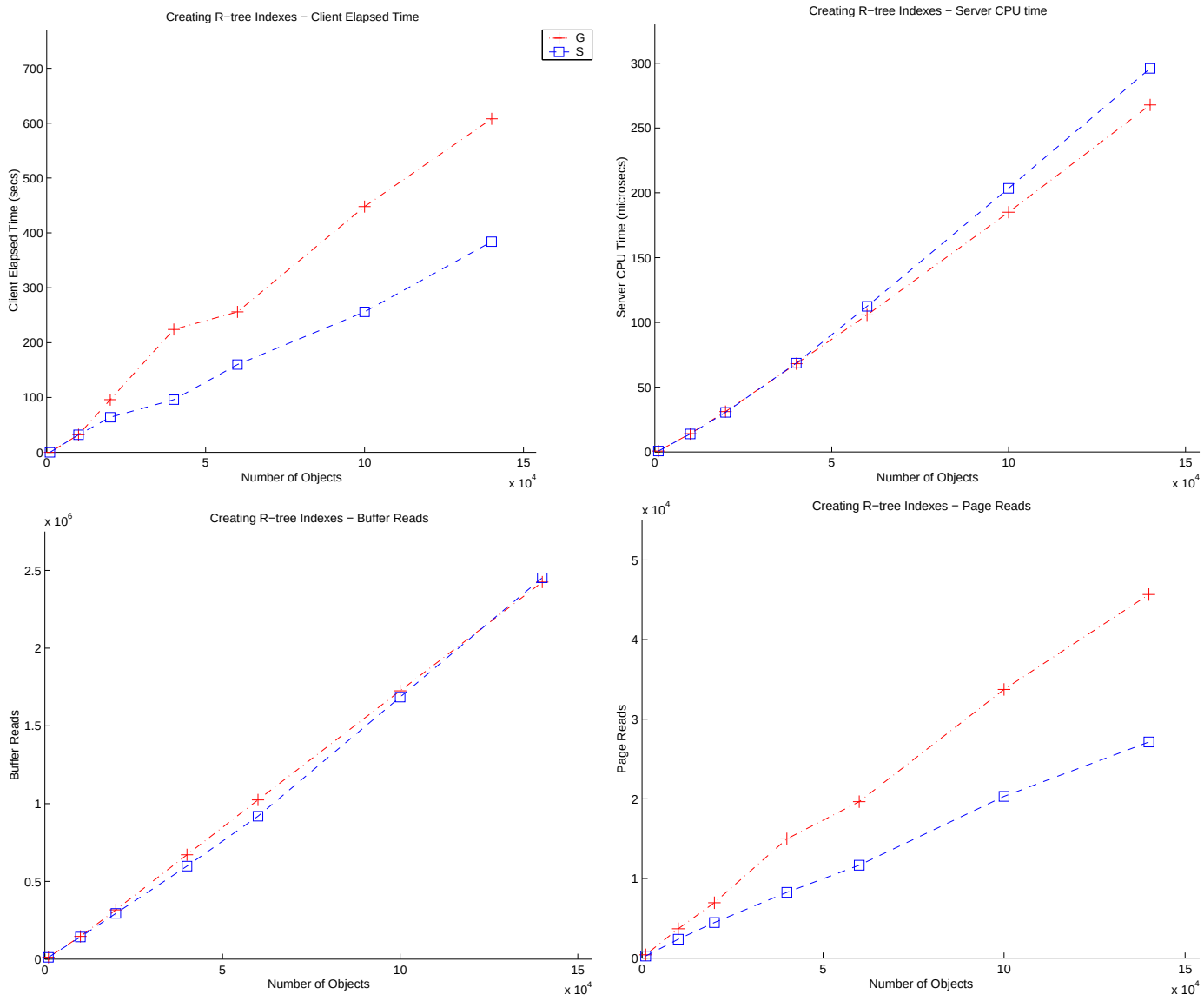


Figure 4: Performance of the Geodetic (**G**) and Shapes2 (**S**) schemas for creating R–tree indexes

.

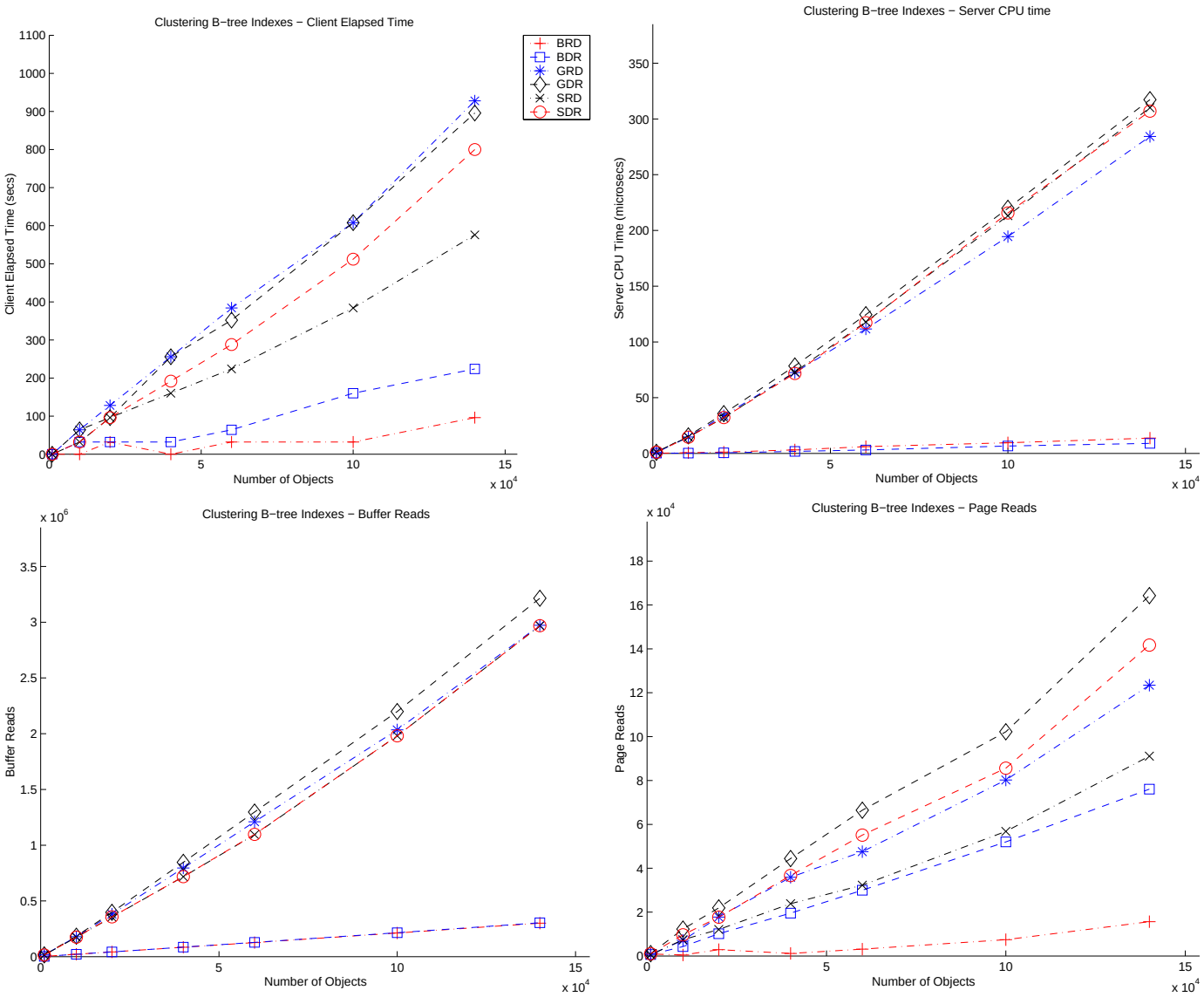


Figure 5: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for clustering B–tree indexes on dec–RA (DR) and RA–dec (RD).

Table 5: Results of linear regression on elapsed time to create R–tree indexes on point data for the **S** and **G** schemas as a function of the number n of objects, $n \geq 1000$. The second column shows the linear regression equation, the third column the 95% confidence interval for the coefficient of n, and the last column the estimated average time to create R–tree indexes on point data fro 500 million stars.

Schema	$t(n)$ in secs	C.I.	R^2	$t(5 \cdot 10^8)$ in secs
S	$-0.0347 + 0.00267451n$	$[0.0024712882, 0.0028777334]$	99.56%	1337255
G	$6.2739 + 0.0043667988n$	$[0.0039013521, 0.0048322456]$	99.15%	2183405

indexes. [13] The regression coefficients, together with 95%–confidence intervals for them, as well as the R^2 are shown in Table 5. The F–tests of significance at the 95%–level for these regression lines were all significant. Using the computed linear regression equations, the estimated average elapsed time to create R–tree indexes for the **G** and **S** schemas 500 million objects are 2183405 secs ($\approx$ 25.3 days) and 1337255 secs ($\approx$ 15.5 days) respectively.

On the other hand the estimated average elapsed times to create clustered B–tree indexes on RA for the **B** , **G** , and **S** schemas for 500 million objects are 488380 secs ($\approx$ 5.7 days), 7827520 secs ($\approx$ 90.6 days), and 4954045 secs ($\approx$ 57.3 days) respectively. Contrast these times with the times to alter a un–clustered B–tree index on dec to clustered which are approximately 21.8, 80.1, and 70 days for each schema respectively. We observe that it takes about 3.3 and 5.4 times more time to create an R–tree index for Shapes2's or Geodetic's point data than to create a clustered B–tree index on built–in data types, respectively.

4.3 Queries

In this section, we discuss the effect of the **B** , **S** , and **G** schemas and of the two choices of clustering fields (dec–RA and RA–dec) on the performance of spatial queries. In summary, for simple spatial window queries with small windows the **B** schemas are almost as fast as the **S** and **G** schemas. However, for spatial join queries (self, chain, and star joins) the **S** and **G** schemas are significantly faster than the **B** schemas in most cases (or as fast in few cases). Further, the choice of clustering fields has significant impact on the performance of **B** schemas for spatial join queries, while the **S** and **G** schemas are less sensitive to the choice of clustering fields.

Before we look into the results of the experiments, we need to present some preliminaries that describe how we "ranked" the schemas and clusterings.

We include simple statistics of the experimental measurement data in Appendix F. We use the data in Appendix F to compute relative quantitative improvements in processing various query types for the three schemas and two choices of clustering fields. In most cases, we compute the relative quantitative improvement by using the average of the corresponding measure for the case of 140,000 stars.

[13] We also performed linear regression for the server CPU time, buffer reads, and page reads for creating R–tree indexes. However, for creating non–clustered B–tree indexes and for altering B–tree indexes to clustered, linear regressions were significant only for the server CPU time and number of buffer reads.

We also rank the **B** , **G** , and **S** schemas and the the two choices of clustering fields dec–RA (DR) and RA–dec (RD) with respect to the client elapsed time, server CPU time, number of buffer reads, and number of page reads, for each table size (number of stars loaded). We call these rankings the *per–table–size rankings*. In addition, using the per–table–size rankings for the schema–clustering fields, we compute overall rankings for the schema–clustering fields with respect to the same measures. We call these rankings the *overall rankings*. For simplicity, we denote the combination of schema–clustering fields with a three letter acronym: **BRD** and **BDR** for the **B** and clustering on RA–dec and dec–RA respectively; **GRD** and **GDR** for the **G** and clustering on RA–dec and dec–RA respectively; **SRD** and **SDR** for the **S** and clustering on RA–dec and dec–RA respectively.

In order to compute the per–table–size rankings, we use the *Fisher's Least Significant Difference* method [2]. This method is based on a t–test of the sample averages for *treatment groups* of equal size. [14] In our case, for each query type, a *treatment group* consists of the set of sample queries of that type that were executed using one of the schema–clustering fields choices (i.e. one of the BRD, BDR, GRD, GDR, SRD, SDR). The result of the Fisher's Least Significant Difference method is a stratification of the six treatment methods BRD, BDR, GRD, GDR, SRD, and SDR. The per–table–size rankings (stratifications) are shown in Tables 10, 11, 12, 13, 14, 15, 16, 17, 18, and 19. Note that whenever the F–test of significance, that is a prerequisite for applying Fisher's method, lead into accepting the null hypothesis (of no difference), the six treatment groups were all ranked the same.

In order to compute the overall rankings, we use the *Wilcoxon Rank Sum* method [1, 2] and the *normalized* per–table–size rankings that were computed using Fisher's method. We normalized the per–table–size rankings by assigning to the treatment groups with the same per–table–size rank k the average order of all the treatments of rank k. For example, if the per–table–size ranking of the six treatment groups BRD, BDR, GRD, GDR, SRD, and SDR are 1, 1, 1, 1, 2, and 3 respectively, then their normalized ranks are (1+2+3+4)/4=2.5, 2.5, 2.5, 2.5, 5, and 6 respectively. The overall ranking of the six treatment groups was obtained by applying the Wilcoxon Rank Sum method on their normalized per–table–size rankings.

The overall rankings should be treated as a qualitative comparison on the the three schemas and two choices of clustering fields. In most cases the overall rankings rank the schemas to the rank that is supported by the maximum number of data points. This sometimes fails to identify trends for larger table sizes and/or capture relative differences of performance. For example, for the number of page reads for *spatial self–join* queries, BDR is ranked together with SRD (even though for tables with 100,000 or more stars, SRD performs at least an order of magnitude more number of page reads on the average than BDR); the per–table–size rankings correctly rank the six treatments with respect to the number of page reads for spatial self–join queries.

Hence, in order to assess the performance of the various schemas and choices of clustering fields, the per–table–size and overall rankings should be looked at together with the figures and basic statistics of the experimental performance measures in Appendix F.

Most of the times the performance of the the **G** and the **S** schemas performed within an order of

[14] Note that a two–sample t–test assumes that the two sample populations are normal with the same variance.

magnitude of each other. We state the constant ratios in the performance of the various schemas. Since we are concerned with large table sizes, the factors provided are for the tables of size 140,000. These factors were computed using the averages of the performance measures for 140,000 stars in Appendix F.

4.3.1 Spatial Window Queries

For spatial window queries, based on Tables 7 and 10, and Figures 12 and 28, and Appendix F, we make the following observations [15]:

- Response time (i.e. Client Elapsed Time) for spatial window queries is within 3 second for all schemas up to 140,000 stars regardless of schema or clustering. In fact, we ran spatial window queries on tables with two million stars, and even the **B** schema executed each of these queries in less than one second. Our other performance measures, however, capture different behavior amongst the different schemas, and clustering schemes.

- All schemas required less than 0.15 microsecs of server CPU time for all schemas up to 140,000 stars.

- **B** schemas performed, on the average, approximately three times more buffer reads than the **G** and **S** schemas. The **G** schemas perform approximately 5% more buffer reads on the average than the **S** schemas. For the **B** , **S** , and **G** schemas, clustering on dec–RA leads into approximately four times, 12%, and -4% more buffer reads on the average than clustering on RA–dec, for 140,000 stars, respectively.

  Clustering of the tables made a significant difference for the **B** schema with respect to the number of buffer reads, The **S** and the **G** seemed less sensitive to the clustering.

  The number of ISAM reads, which is a measure of the efficiency of the index structure, exhibited the same behavior as the number of buffer reads. Hence, the B–tree indexes are less efficient than R–tree indexes for spatial window queries. The fact that with respect to client elapsed time **B** schemas were not statistically different than **G** and **S** schemas is due to their much smaller record size as reflected by the number of page reads below.

- The **S** and **G** schemas made approximately 38% and 85% more page reads on the average than the **B** schemas for 140,000 stars respectively. The smaller record size significantly improved the performance of the **S** schemas with respect to the number of page reads as compared the **G** schemas.

  Regarding the effect of the clustering on the number of page reads, the **B** schemas when clustered on dec–RA leads into approximately twice as many page reads on the average as when clustered on RA–dec. For the **S** schemas clustering on dec–RA leads into approximately

[15] Some of the (average) measures shown in the graphs are not monotonically increasing with respect to the number of stars as one would expect. Though part of such behavior can be attributed to the variance of the corresponding measure, which are shown in Appendix F, we can not fully explain this behavior; we suspect that caching and data and query distribution play a major role in that behavior.

90% more page reads on the average than clustering on RA–dec for 140,000 stars. Clustering did not affect the average number of page reads for the **G** schemas.

- The choice of the clustering field, affected the number of buffer, ISAM, and page reads. Clustering had far greater impact on the the **B** schemas than on the **S** and **G** schemas. Furthermore, clustering had greater impact on the average number of buffer reads than on the average number of page reads. The choice of clustering fields for the three schemas did not affect the client elapsed time.

For spatial Or–Window queries, based on Tables 7 and 11, and Figures 13 and 29, and Appendix F, we make the following observations:

- The client elapsed time and server CPU time is approximately the same as for the single spatial window queries.
- The **B** and **G** schemas perform approximately five times and 8% more buffer reads on the average than the **S** schemas for 140,000 stars respectively. Clustering on dec–RA leads into five times more buffer reads on the average for the **B** schemas than clustering on RA–dec. Clustering on dec–RA leads into approximately 24% more buffer reads for the **S** schemas than clustering on RA–dec for 140,000 stars.
- The **S** and **G** perform approximately 43% and 90% more page reads on the average than the **B** schemas for 140,000 stars, respectively. Clustering on dec–RA, instead of clustering on RA–dec, leads into 190% more page reads on the average for the **B** schemas, while it leads into approximately 97% increase on the average for the **S** schemas (with no effect for the **G** schemas), for 140,000 stars.

Thus, for spatial Or–Window queries we have similar behavior with respect to client elapsed time, server CPU time, number of buffer and page reads, and effect of clustering, as for spatial window queries.

In conclusion, for single spatial window queries with small windows, with respect to response time, the traditional relational schemas and indexes are as good as the extended relational schemas and indexes, but as the size of the relations increases, traditional relations and schemas perform far more buffer and ISAM reads than the extended relational schemas. The significantly larger record size for the schemas that use the Shapes2 and Geodetic datablades did not enable the extended relational schemas to outperform the traditional relational schemas with respect to the number of disk I/O operations, and consequently on the response time. The choice of clustering fields plays a more important role on the performance of the **B** schemas than it does on the performance of the **S** and **G** schemas. **B** schemas seem to be more sensitive to the input data distribution than the **S** and **G** schemas. Given the fact that the data used for the experiments have larger variance on RA than they have on dec, clustering on RA–dec gives better performance than clustering on dec–RA.

4.3.2 Spatial Self–Join Queries

For spatial self–join queries, based on Tables 7 and 12, and Figures 14 and 30, and Appendix F, we make the following observations:

- With respect to client elapsed time, the **S** and **G** significantly outperform the **B** schemas. The response time for the **B** schemas is approximately four times and 45% more, on the average, than that for the **S** and **G** schemas for 140,000 stars, respectively. The **S** schemas outperformed the **G** schemas by approximately three times on the average for 140,000 stars. Computing a self–join for 140,000 stars for the **S** schemas took approximately 33 mins elapsed time. Clustering on dec–RA instead of clustering on RA–dec leads into approximately 5–fold and 2–fold increase on the elapsed time for the **B** and **S** schemas for 140,000 stars. Interestingly enough, clustering on dec–RA instead of clustering on RA–dec, leads into approximately 15% decrease of the client elapsed time on the average for the **G** schemas for 140,000 stars.

  Further, as the number of stars stored increases, the **S** schemas became more competitive than the **G** and **B** schemas with respect to the client elapsed time.

- With respect to the server CPU time for 140,000 stars, the **S** schema (with clustering on dec–RA) outperformed the **B** and **G** by approximately 21 times and 20% on the average. However, regarding the effect of clustering, clustering on dec–RA instead of clustering on RA–dec lead into approximately a 4–fold reduction, a 27% increase, and a 5–fold increase, of the server CPU time on the average for the **S** , **G** , and **B** schemas. The **S** schemas for 140,000 stars took approximately 410 microseconds server CPU time on the average. This behavior of the server CPU time for the three schemas and clustering is not consistent with the client elapsed time, and number of buffer and page reads.

- With respect to the number of buffer reads, for 140,000 stars, the **B** and **G** schemas perform approximately 20 times and 25% more buffer reads on the average than the **S** schemas. Further, clustering on dec–RA instead of RA–dec, leads into approximately a 5–fold increase, a 5% reduction, and a 10% increase on the average number of buffer reads for the **B** , **G** , and **S** schemas respectively.

- With respect to the number of page reads, the **B** schemas significantly outperform the **S** and **G** schemas. For 140,000 stars, the **S** and **G** schemas perform approximately 40 times and 200 times more page reads on the average than the **B** schemas. Clustering on dec–RA instead of RA–dec, for 140,000 stars, leads into approximately a 20% increase, a 18% decrease, and a 4–fold increase on the average number of page reads for the **B** , **G** , and **S** schemas respectively. The effect of clustering fields on the number of page reads correlates positively with its effect on the number of buffer reads and client elapsed time. However, it does not correlate positively with it effect on the server CPU time for the **G** and **S** schemas for 140,000 stars.

  Even though the **B** schemas outperformed the **G** and **S** schemas by at least an order of magnitude with respect to the number of page reads, the **B** schemas turned out to be four times slower than the **S** schemas for 140,000 stars. The much larger number of buffer reads for the **B** schemas than for the **S** and **G** significantly worsened the performance of the **B**

schemas. This, despite the fact that the record size for the **B** schemas is significantly smaller than that for the **S** and **G** schemas.

Note that the storage for a **B** schema for 140,000 stars is about 2,000 pages, which is not large and a large portion of it can fit in memory; moreover, since there is only one table involved in the join, many of the needed pages will be in memory. Despite the significant space overhead of the **S** and **G** schemas, due to the gross inefficiency of the B–tree indices with respect to that of R–tree indexes for spatial query processing, the **B** schemas are significantly slower compared to the **S** and **G** schemas for spatial self–join queries.

In conclusion, for single spatial self–join queries, even for small relations, using R–tree indexes is significantly beneficial and out-weights any storage overhead associated with such an approach. Also, the simple Shapes2 datablade outperformed the more sophisticated Geodetic datablade primarily due to the smaller record size.

4.3.3 Spatial Window Chain–Join Queries

For spatial window chain–join queries, based on Tables 8, 13, 14, 15, and 16, and Figures 15, 16, 17, 18, 22, 24, 26, 31, 32, 33, and 34, and Appendix F, we make the following observations:

- Spatial chain–join queries with small windows can be computed within approximately 3 and 6 secs average elapsed time for both the **S** and **G** schemas for 2–and–3–chain–joins and 4–chain–joins respectively, for up to 140,000 stars. For 5–chain–joins, the average elapsed time is approximately 32 and 38 secs for the **S** and **G** schemas for 140,000 stars. For the **B** schemas, clustered on RA–dec, and 140,000 stars the average elapsed times are approximately 3 secs for 2–and–3–chain–joins, 20 secs for 4–chain–joins, and 80 secs for 5–chain–joins. That is, the **S** and **G** schemas outperformed the **B** schemas by approximately a factor of 2 for 4,5–chain–join queries with respect to elapsed time.

  Clustering on dec–RA instead of RA–dec, leads into an approximately 6–fold increase on the average elapsed time for 3,4,5–chain–join queries for the **B** schemas and 140,000 stars. Clustering did not have any statistically significant impact on the elapsed time for the **S** and **G** schemas.

- With respect to server CPU time, for 140,000 stars, the **S** and **G** schemas outperformed the **B** schemas by approximately a factor of 3 for chain–join queries. Clustering on dec–RA instead of RA–dec, leads into an approximately 6–fold increase on the average for the **B** schemas, and it leads into approximately no more than 45% increase on the average for the **S** and **G** schemas.

- The **B** schemas perform approximately 10 times more buffer reads on the average than the **S** and **G** schemas for chain–join–queries. Clustering on dec–RA instead of RA–dec, leads into approximately a 6–fold increase on the average number of buffer reads for the **B** schemas, and approximately no more than 40% increase on the average number of buffer reads for the **S** schemas. Clustering did not have a statistically significant impact on the number of buffer reads for the **G** schemas.

- The **S** schemas perform approximately 2.5 times more page reads on the average than the **B** schemas for chain–joins. The **G** schemas perform approximately 35% more page reads on the average than the **S** schemas. Clustering on dec–RA instead of RA–dec, leads into approximately a 9–fold increase and a 3–fold increase on the average number of page reads for the **B** and **S** schemas respectively, and it seems to have a negligible impact on the average number of page reads for the **G** schemas.

In conclusion, for spatial k–chain–join queries, the **S** and **G** schemas provide approximately half the average response time than the traditional **B** schemas. Using the Shapes2 datablade, we can process spatial k–chain–join queries with small spatial windows within approximately 32 secs elapsed time for relations with 140,000 objects for $k = 2, 3, 4$ and 5. Furthermore, the choice of clustering fields for the B–tree indexes has a significant impact on the performance of spatial chain–join queries, while it has significantly lesser impact when R–tree indexes are used.

4.3.4 Spatial Window Star–Join Queries

For spatial window star–join queries, based on Tables 9, 17, 18, and 19, and Figures 19, 20, 21, 23, 25, 27, 35, 36, and 37, and Appendix F, we make the following observations:

- With respect to the elapsed time, the **B** schemas take approximately 3 times more on the average than the **S** and **G** schemas for star–joins for 140,000 stars. On the average, it takes approximately 1 secs, 6 secs, and 32 secs to compute 3–star–joins, 4–star–joins, and 5–star–joins for 140,000 stars for the **S** and **G** schemas. Clustering on dec–RA instead of RA–dec, leads into approximately a 6–fold increase on the elapsed time for the **B** schemas. Statistically, clustering does not have significant effect on the elapsed time for the **S** and **G** schemas.
- The **B** schemas require approximately 3 times more average server CPU time than the **S** and **G** schemas. There does not seem to be significant difference between the **S** and **G** schemas with respect to the average server CPU time. Clustering on dec–RA instead of RA–dec, leads into a 6 times more average server CPU time for the **B** schemas, while it does not seem to significantly affect the server CPU time for the **S** and **G** schemas.
- The **S** schemas perform approximately 2 times more page reads on the average than the **B** schemas. The **G** schemas perform approximately 30% more page reads on the average than the **S** schemas. Clustering on dec–RA instead of RA–dec, leads into approximately a 8–fold increase and a 130% increase on the average number of page reads for the **B** and **S** schemas for 140,000 stars respectively, while there does seem to affect the average number of page reads for the **G** schemas.

In conclusion, for spatial k–star–join queries, it is beneficial, from the point of view of response times, to use the Shapes2 datablade. Using the Shapes2 datablade, we can process spatial 5–star–join queries with small spatial windows within approximately 32 secs elapsed time for relations with 140,000 objects. Further, the choice of clustering fields for the B–tree indexes has a significant

impact on the performance of spatial star–join queries, while it has significantly lesser impact when R–tree indexes are used.

5 Fractal Analysis of Spatial Window Queries

While doing spatial searches using an R–tree, we want to minimize the number of nodes of the R–tree that need to be looked into. example, consider the spatial window queries. Recall that each node of an R–tree contains a number of pairs of the form `(<rectangle>, <pointer>)`, where `<rectangle>` is the minimum bounding rectangle (MBR) for all the objects stored in the subtree of nodes specified by `<pointer>`. For the leaf nodes, the `<pointer>` points to the location of an object (or a disk page that contains a set of objects) [6]. When, searching for objects contained within the query window, one needs to recursively search (lookup) all the subtrees `(<pointer>)` whose MBR overlaps with the query window. As a result, due to this "bifurcation", more than one path from the root to a leaf may need to be traversed, and thus increase the number of R–tree nodes that will be looked up. In general, if the height of the R–tree is h and N is the number of leaf nodes that have an MBR that overlaps with query window, then the number of nodes that need to be looked up is in the order of hN.

Faloutsos and Kamel [5] show how to estimate the average number of nodes of an R–tree that will be accessed for window queries using the fractal dimension of the data. The fractal dimension of a data set can be computed by using a boxcount–plot [5] as follows. Partition the input space into a uniform grid with square cells with edge r, and let $N(r)$ be the number of cells that have at least one data point. Plot $\log(N(r))$ versus $\log(r)$. This plot should be line. The negative of the slope of this line is an estimate of the fractal dimension of the dataset. Assume that the input space has been normalized to a unit square. Consider now an R–tree for n data points. Let C_{leaf} be the average number of MBRs per page for the leafs of the R–tree, and let $C_{internal}$ be the average number of MBRs per disk page for the non-leaf nodes of the R–tree. Let n_j be the number of nodes at level $j = 0, 1, 2, \ldots, h$ of the R–tree, where h is the height of the R–tree, with the root being at level 0. Note that $n_h = \lceil n/C_{leaf} \rceil$, and that

$$n_{j-1} = \lceil n_j/C_{internal} \rceil,$$

for $0 \leq j < h$. Faloutsos and Kamel [5] assume that the MBRs of the R–tree are tight square–like MBRs roughly of the same size. Let σ_j be the edge of MBRs at level $j = 0, 1, \ldots, h$. They show that $n_j = 1/(\sigma_j)^d$, where d is the fractal dimension of the data. Further, for a window query Q with window of size (extend) $q_x \times q_y$, they show that the average number of pages read at level $j = 0, 1, \ldots, h$ of the R–tree is

$$n_j(\sigma_j + q_x)(\sigma_j + q_y), \tag{1}$$

which implies that the average number of page reads is

$$\sum_{j=0}^{h} n_j(\sigma_j + q_x)(\sigma_j + q_y) + A, \tag{2}$$

where A is the number of page reads that need to be performed in order to retrieve the actual records (points) within the query window.

The estimated average number N_Q of data points within the query Q can be obtained by $n_h(\sigma_h + q_x)(\sigma_h + q_y)C_{leaf}$.

Further, when the index is clustered we take $A = \lceil N_Q/C_S \rceil \approx n_h$, otherwise we take $A = N_Q$, where C_S is the number of data points that fit in a page.

For the **G** schema, since we have pages of size 2KB, and the average number of bytes per star is 418.11, while the R–tree index uses 190 bytes on the average, we estimate $C_{leaf} \approx 5$ and $C_{internal} \approx 10$. For the **S** , the average number of bytes per star is 298.50, while the R–tree index uses 141 bytes on the average, we estimate $C_{leaf} \approx 7$ and $C_{internal} \approx 15$.

We also consider a proposed *lightweight* spatial datablade that supports R–tree indexes on point and rectangle data. The purpose behind such a lightweight datablade is to provide all the benefits of R–trees for spatial queries while having much smaller space overhead. In particular, point data in this lightweight datablade will take 16 bytes (2 reals), while rectangle data will take 32 bytes (4 reals). That lightweight datablade will have $C_{internal} \approx 25$, for 2KB pages since R–tree nodes in Informix have a fixed space overhead of 48 bytes per MBR. Furthermore, we assume that that lightweight datablade can cluster point/rectangle data according to their order in the leafs of an R–tree index, in which case we refer to it as a lightweight datablade with clustering.

We use the results of Faloutsos and Kamel [5] for our data by estimating the fractal dimensions of the data used in the experiments as well as for the complete Monet catalog. In order to estimate the fractal dimension of a dataset, we followed the approach in [5], which works as follows. Normalize the points to be in the unit square. For each value $0 < r < 1$, fit a uniform grid on the square and count the number $N(r)$ of non–empty grid cells. The plot of $\log(N(r)$ versus $\log(r)$ is known as the boxcount plot for a dataset. The boxcount plot for a dataset with fractal dimension d is a line with slope $-d$. To estimate d from a dataset perform linear regression analysis of the $\log(N(r))$ versus $\log(r)$. The results of this analysis are shown in Figures 6, 7, and 8 and in Table 6.

We perform a comparison between the estimated average number of page reads and the actual average number of reads for spatial window queries for the **S** and **G** schemas. Recall, that each query window had an extend of 0.066666×0.066666. Since the data points need to be normalized to the unit square for this analysis, we need the extend of the data with 1000, 10000, 20000, 40000, 60000, 100000, 140000, 1992564 (number of stars in the zone0000.cat file), and 526280881 (for the complete Monet catalog).

The results of our comparisons are shown in Figures 9, 10, and 11.

In Figure 9.(a) we show the average number of page reads for spatial window queries for: (a)

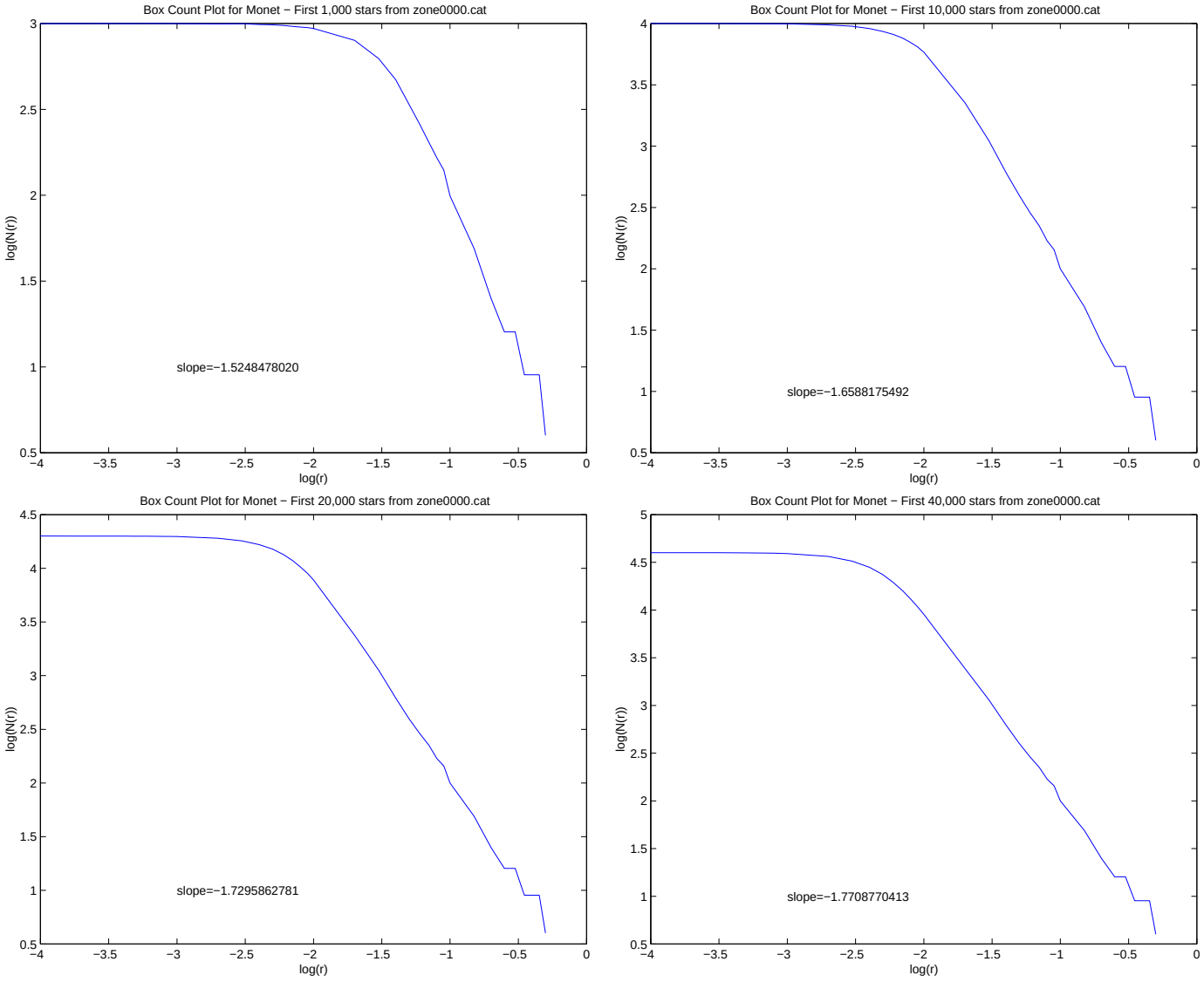


Figure 6: Fractal dimension of data used in experiments. Here, r is the edge–length of a square box (cell), and $N(r)$ is the number of non-empty cells of the uniform grid imposed on the data. The fractal dimension is the negative of the slope.

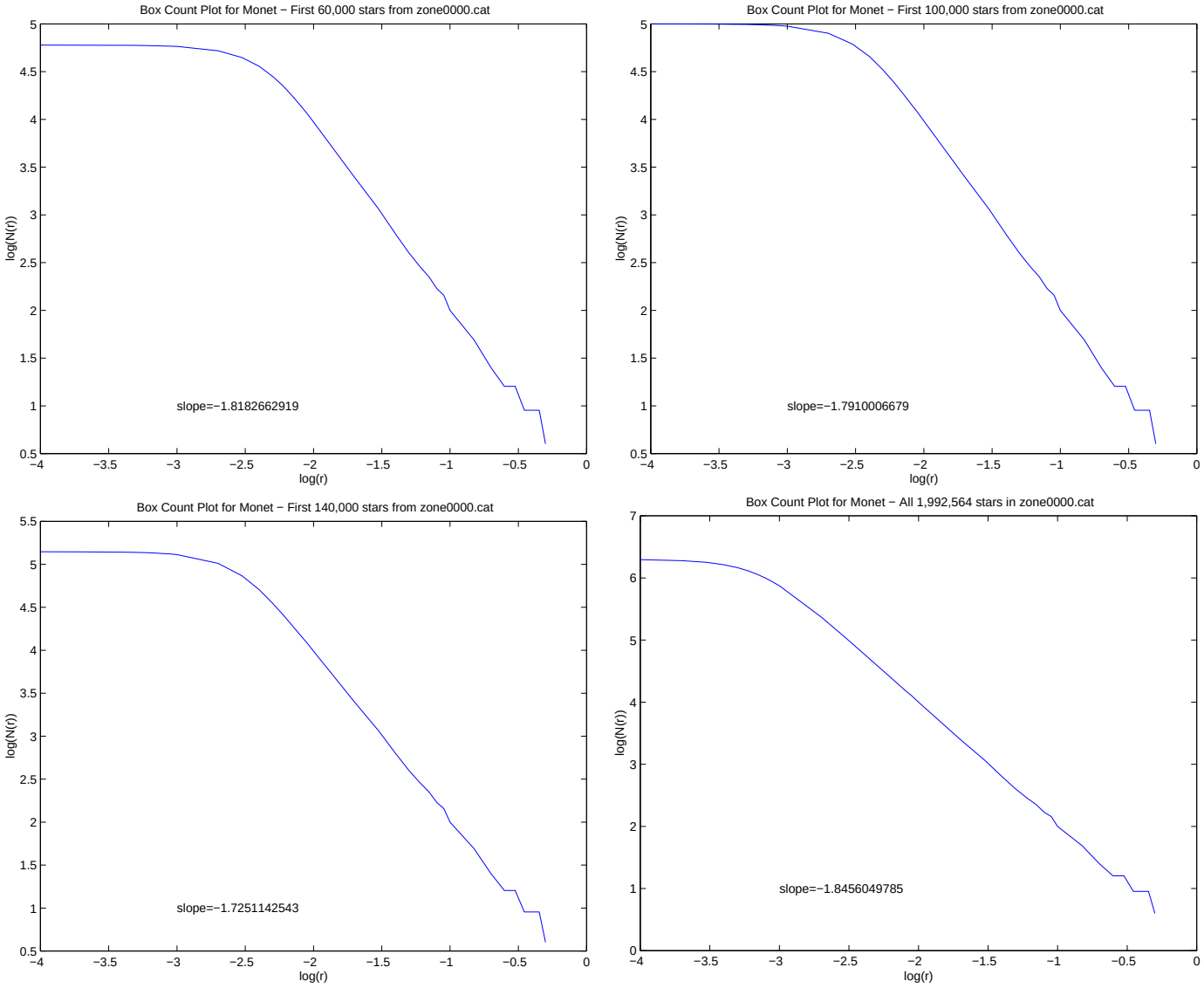


Figure 7: Fractal dimension of data used in experiments. Here, r is the edge–length of a square box (cell), and $N(r)$ is the number of non-empty cells of the uniform grid imposed on the data. The fractal dimension is the negative of the slope.

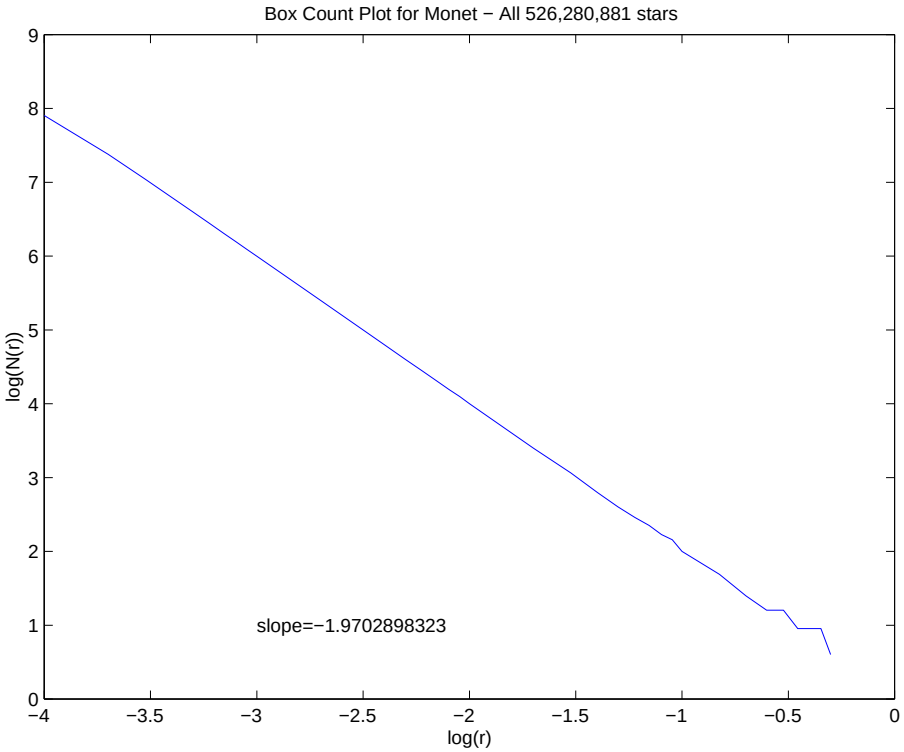
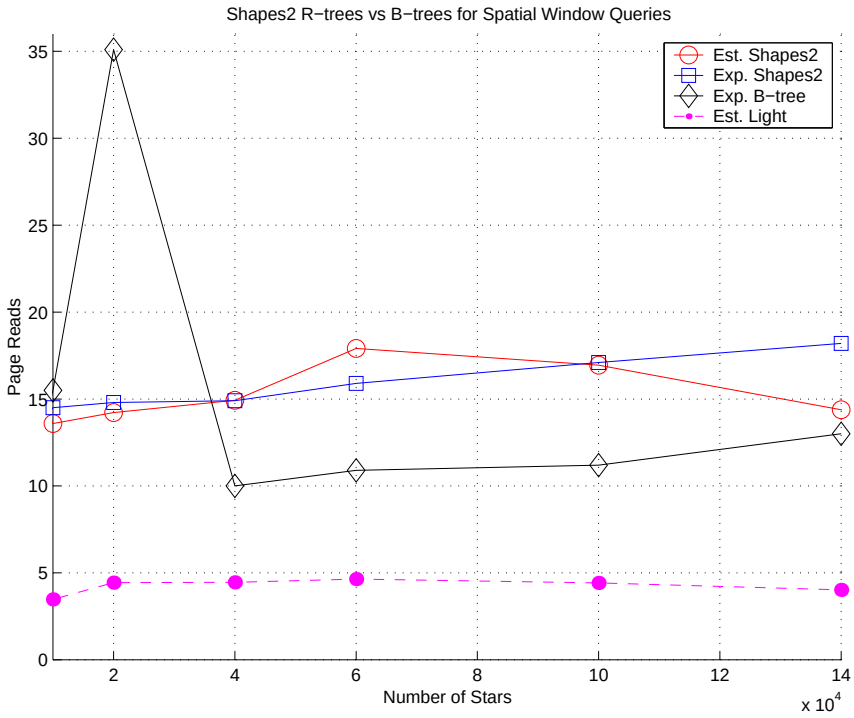
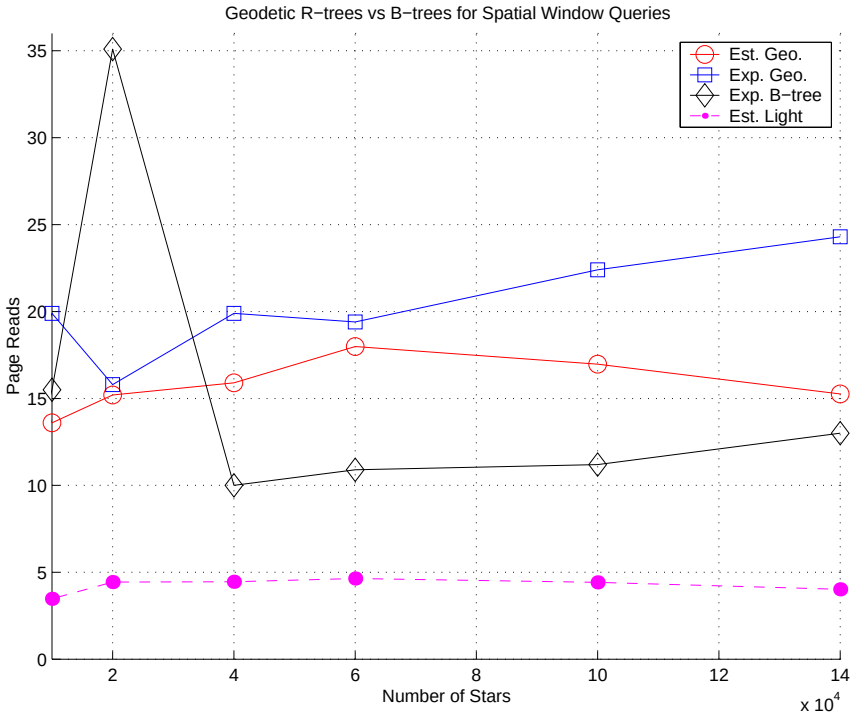


Figure 8: Fractal dimension of the USNO–A2.0 (Monet) Catalog. Here, r is the edge–length of a square box (cell), and $N(r)$ is the number of non-empty cells of the uniform grid imposed on the data. The fractal dimension is the negative of the slope.



(a)



(b)

Figure 9: Estimating, validating with experimental results, and comparing the number of page reads for R–tree and B–tree indexes for spatial window queries with window of size 0.0666×0.0666 decimal degrees. The capacity of the R–tree nodes is 10, 15, and 25 for the Geodetic, Shapes2, and a proposed light–weight datablade respectively.

Table 6: Fractal dimension and window extends (in decimal degrees) for various data sizes.

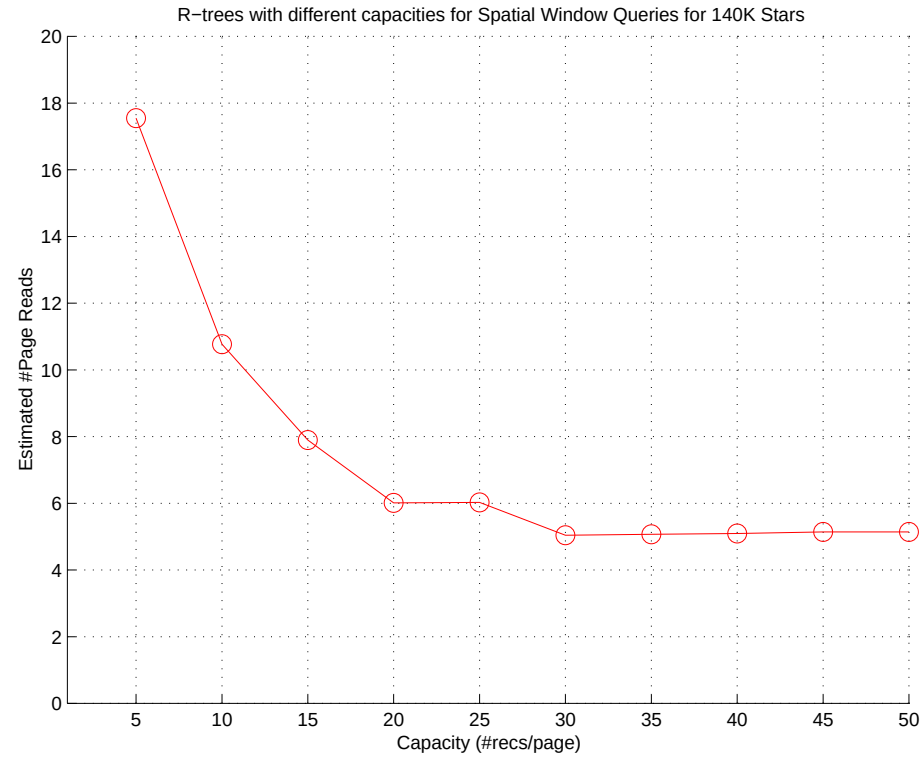
Dataset size	Fractal Dimension	RA Extend	Dec Extend
1000	1.5248	0.2700	7.1981
10000	1.6588	2.5561	7.4317
20000	1.7296	5.1059	7.4642
40000	1.7709	10.1768	7.4642
60000	1.8183	15.0765	7.4705
100000	1.7910	25.1253	7.4836
140000	1.7251	35.4130	7.4836
1992564	1.8456	359.9999	7.4928
526280881	1.9703	360.0000	180.0000

the **S** schema (as measured experimentally), (b) the estimated average number of page reads for R–tree node capacity (fan–out) of 15 (that corresponds to Shapes2) and no clustering for the R–tree index, (c) the (experimental) average number of page reads for the **B** schemas, and (d) the estimated average number of page reads for a proposed lightweight spatial datablade with clustering. We observe that the experimental and estimated average number of page reads for the **S** schema are in close agreement, hence validating the formula for the average number of page reads for **S** . Further, as shown, a lightweight datablade with clustering, in addition to reducing the space overhead for storing the data and an R–tree index, it can reduce the average number of page reads by approximately a factor of three with respect to the **S** and **B** schemas.

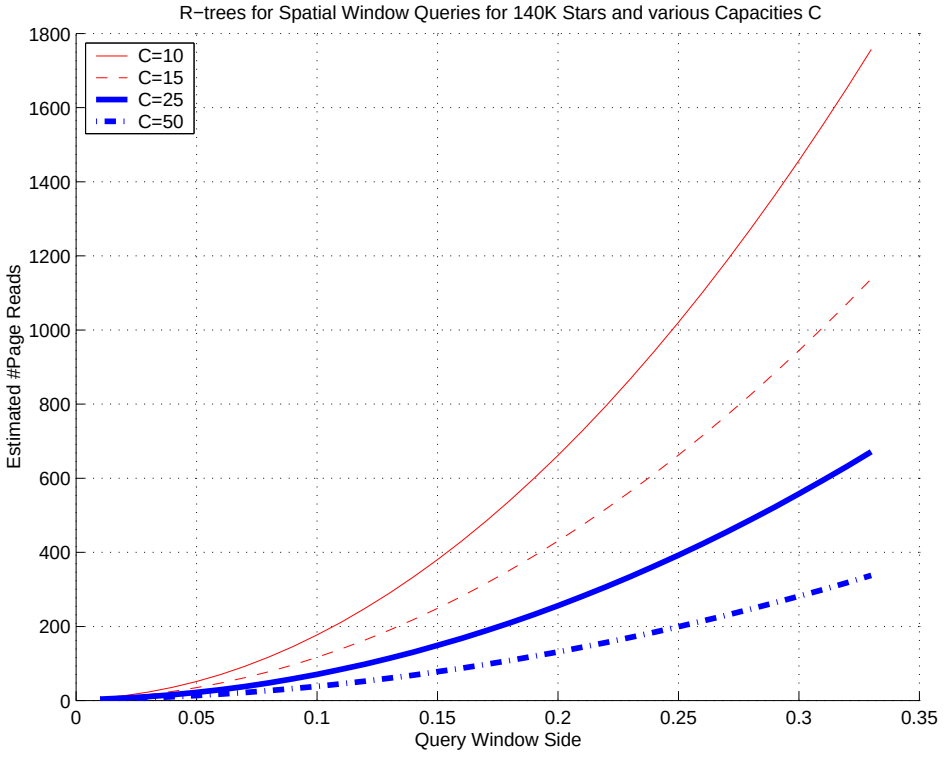
Figure 9.(b) is similar to Figure 9.(a), except that we show the experimental and estimated average number of page reads for **G** schemas (with no clustering for the R–tree index) instead of **S** schemas. It is interesting to note that the estimated average number of page reads is diverging from the experimentally determined average number of page reads for **G** schemas.

In Figure 10.(a) we show the effect of R–tree node capacity (fan–out) on the estimated average number of page reads for spatial window queries with window extend of 0.06666×0.06666 decimal degrees. As, we see from this figure, the largest marginal improvement on the number of page reads can be realized by increasing the R–tree node capacity from about 5 to about 20. Thus, our target value of capacity of 25 for the lightweight datablade with clustering is ideal. In Figure 10.(b), we show the effect of R–tree node capacity on the average number of page reads for spatial window queries as the window extend increases. Note that as the window extend is increasing the estimated average number of page reads is increasing much slower for larger R–tree node capacities. For example, for window extends of 30%, the lightweight datablade is expected to perform about half the average number of page reads that the Shapes2 will do for spatial window queries.

In Figure 11, we show the estimated number of page reads for spatial window queries with window extend 0.0666×0.0666 decimal degrees for the Shapes2, Geodetic, and the lightweight datablade, for various number of stars. For example, for 500 million stars, the Shapes2 and Geodetic datablades are expected to perform approximately 100 page reads (since the estimated number of stars within the query window is 80 and Shapes2 and Geodetic do not support clustering), while the lightweight datablade will perform approximately 15 page reads. Furthermore, the lack of ability to cluster records based on how they are stored at the leafs of an R–tree index, has a significant impact on the number of page reads. Thus, the largest improvement on the average number of page reads



(a)



(b)

Figure 10: Estimated number of page reads for R–trees for spatial window queries, for various R–tree node fan–out (capacities) and normalized in $[0, 1]^2$ query window sizes for the first 140K stars from zone0000.cat.

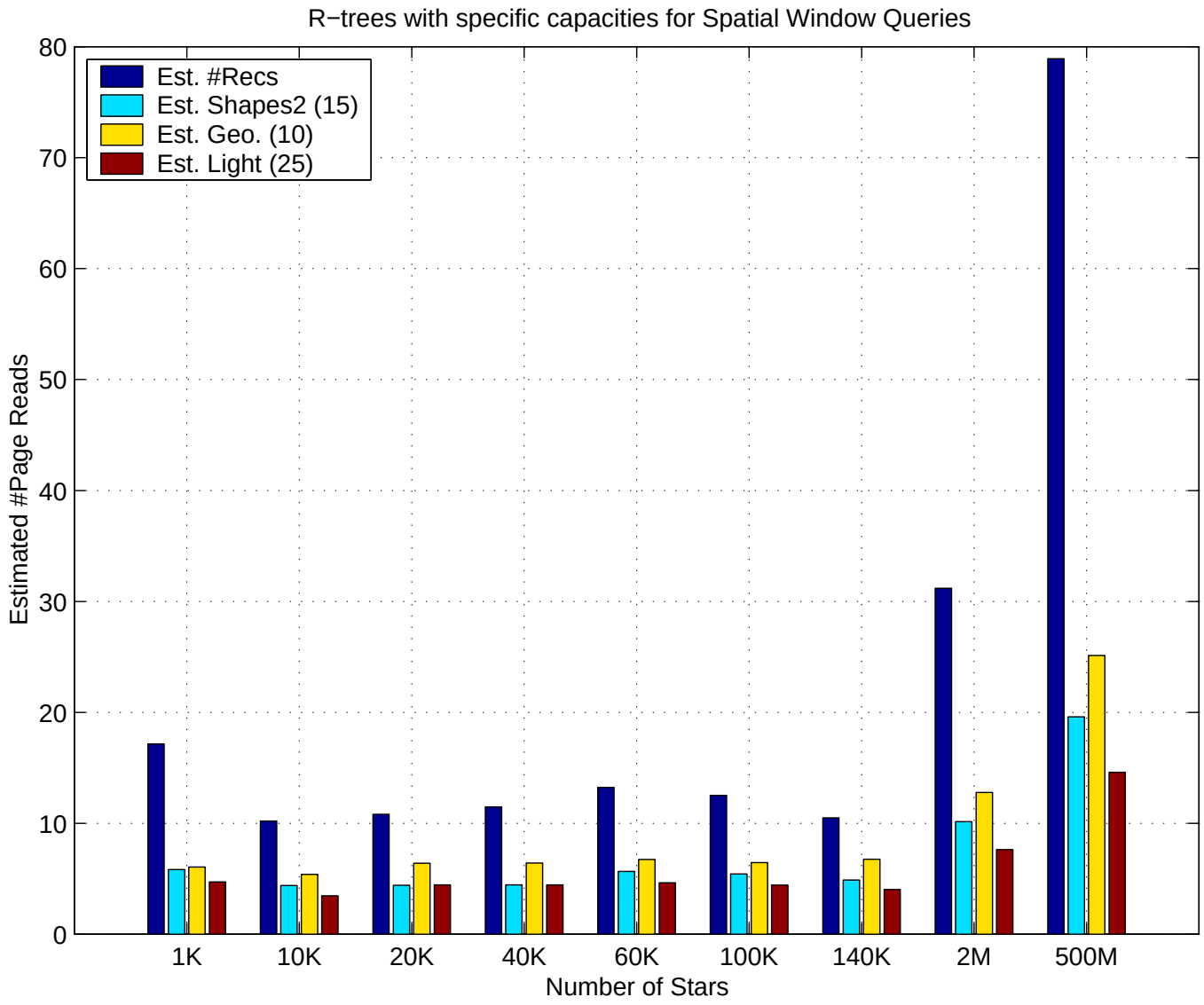


Figure 11: Estimated number of page reads for R–trees for spatial window queries, with window size 0.0666×0.0666 decimal degrees, for the Shapes2 and Geodetic datablades and a proposed light–weight datablade. It also shows the average number of stars within a random query window of size 0.0666×0.0666. It is assumed that records are clustered based on the leaves of the R–trees; note that clustered R–tree indexes for the Shapes2 and Geodetic are not supported by Informix DS/UDO 9.14. The capacity of the R–tree nodes is shown in ().

is expected to come from the ability to cluster records according to their storage in an R–tree index. The improvement due to larger R–tree node capacity will have a non–negligible impact (approximately 25%) on the number of page reads, but it will have a much larger impact on the total storage for the data by reducing the average record size from approximately 300 bytes to approximately 60 bytes, a five–fold reduction.

6 Extending the Database Server: Example Functions

In order to improve the performance of query processing and reduce the amount and complexity of code that needs to be developed and maintained, it is usually beneficial to introduce new data types as first–class types in database systems. This new data types usually define not only the structure of the information but also the behavior of objects of those types (e.g. methods in the O–O nomenclature). By first–class types, we mean that objects of those new data types can be used in the same manner and everywhere else where built–in data types (e.g. integers, character strings etc) can be used.

Informix allows to define new data types either as *row data types*, structures that consists of built–in data types which are easily accessible by the users, or as *opaque data types*, sequence of bytes that must be accessed via well-defined accessor methods. The decision of using row or opaque data types is based on the size of the objects and accessibility we want to provide. This decision however can have a significant impact on the performance of query processing (e.g. space overhead and processing overhead). For example, the following SQL statement creates a row data point for points:

```
CREATE ROW TYPE Point(x FLOAT, y FLOAT);
```

Then, we can declare relations that use this type and query them:

```
CREATE TABLE monet(location Point, red Float, blue Float, field Int);
SELECT location.x FROM monet WHERE location.y > 45.0;
```

Creating new opaque data types requires the development of appropriate constructor, destructor, and accessor functions. Further, for either row or opaque data types, in order to specify their behavior, we need to develop methods that are implemented as functions, usually external functions. Furthermore, even when we do not introduce new data types, we may want to introduce new behaviors for existing built–int datatypes (e.g. new aggregate functions). For this reason, in this section, we focus on the development of external functions to extend a database server.

External functions that execute on the database server with excellent performance are the heart of database extension. Even though, stored SQL procedures and functions can be used to extend a database server, these functions are quite limited in their capabilities, due to limitations imposed by the SQL dialect of the database server, and have considerable processing overhead (that depends on the quality of the database server's SQL compiler).

To use memory efficiently, external functions are stored in shared libraries that are dynamically loaded into the database server only when needed. Once the functions are loaded into the server they may be used anywhere built-in functions can used in SQL statements. Since external functions execute at the server, where the data are available, this may lead in reduced amount of data that needs to be sent to the client.

All external functions must follow a strict set of standards and guidelines, otherwise the database server may experience unexpected results. Among those guidelines, the guidelines related to memory management and parameter passing are the two most important ones.

Memory allocation and use.
All memory must be allocated and de-allocated by the appropriate DataBlade API calls. Memory can be allocated once per function call or once per SQL statement execution. Note that allocating memory per SQL statement execution, allows an external function to maintain state between function calls in the context of that SQL statement execution. Memory allocation per SQL statement execution is accomplished by using the Informix structure `MI_FPARAM`, that holds state information between function calls within a single SQL statement execution. For example, consider an external function `count()` that simply returns the number of calls to it during an SQL statement execution. This function, needs to initialize a counter at its first call, and increment that counter for each subsequent call. The signature for `count()`, in the C language, might look like

```
int count(int dummy, MI_FPARAM *fparam)
```

This function may be used in an SQL statement as in the following example:

```
SELECT count(null) FROM monet;
```

A single `MI_FPARAM` structure will be created and initialized once for this SQL statement and passed to every call to `count()`. The `MI_FPARAM` structure holds various pieces of information about the calling arguments, return types, and internal state status. The `count()` function can maintain and access its state within the execution context of this SQL statement through the `MI_FPARAM` structure. For detailed examples, refer to sections 6.2.1, 6.2.2, and 6.2.3.

Data Type Mapping and Parameter Passing.
The next important guideline deals with the mapping of the database data types to and from the data types of the implementation language of the external functions, often the C language, as well as to the methods for parameter passing (e.g. pass–by–value or pass–by–reference). Every data type in the database has a corresponding data type in C. All data types of length 4 bytes or smaller may be passed to the external function by value (e.g. integers, small integers). Larger data types are passed by reference (e.g. varying length characters, double precision, user defined types).

External functions can be classified as *basic*, *iterative*, *aggregate*, and *datablade module* functions.

Basic external functions are the building blocks for iterative, aggregate, and datablade module

functions. Basic functions take a number of input parameters, return a single result value, perform rather simple computations, and can either be stand–alone or support functions in other categories. See section 6.1 for an example.

Iterative functions are a generalization of basic functions that can return more than one value. Iterative functions are controlled through the `MI_FPARAM` structure and must handle three different states. The first state is used for initialization. No output is shown to the user during the first state. The function enters the second state, the *iterative* state, for the repeated calls to itself during the execution of an SQL statement. An SQL statement that uses an iterative function, will call this function as many times as there are values to be returned. When those values are exhausted, the function completes its execution in the iterative state. Each call to the function in the iterative state returns a value. Upon completing its execution in the iterative state, it indicates this fact by using the `MI_FPARAM` structure. Note that in order to avoid an infinite loop, the programmer must make sure that the function indicates when it completed execution. At the next function call, the function enters the *final* or *end* state. In the final state, the function does some housekeeping (e.g. deallocating memory). The return value from this state is not shown to the user.

Aggregate functions are a group of basic functions that work together to compute a summary of their input. An aggregate functions usually consists of four basic functions: an *initialization* function, an *iterator* function, a *combinator* function, and a *finalization* function. The initialization function initializes appropriate data structures for the aggregation. For example, for an `average` aggregate function, the initialization function could allocate two variables, a count and a running sum, and initialize them to zero. The iterator function updates for each input tuple the structures set up during initialization. For the `average` aggregate function above, it will update the count and running sum variables. The combinator function is used when the aggregate is executed in "parallel" on more than one fragments of the input. One call of the combine function will "merge" the data structures for two fragments of the input. For the `average` aggregate, it will add the counts and running sums of the two fragments. The finalization function performs any steps that are needed upon completion of the input in order to compute the final value of the aggregate to be returned. In addition, it will perform any necessary housekeeping (e.g. deallocating memory). For the `average` aggregate, the finalization function will divide the running sum by the count, deallocate the memory of the two variables, and return the result in the appropriate format and type. For a detailed example of an aggregate external function see section 6.2.3.

DataBlade module functions are functions that are packaged together with user defined types into a module. These functions can be of any of the previous types (i.e. basic, iterator, or aggregate). See the functions in the Shapes2 and Geodetic datablades for an example.

6.1 Basic External Functions

In this section, we describe a simple external C function that takes as input two numbers and returns a number that consists of an interleaving of the bits of the two input numbers. Since the Informix Dynamic Server allows for the creation of indexes on the return values of functions, one can use the function described in this section in order to process spatial queries.

The `BitInterleave` function generates a bit interleaving pattern from two numbers x and y. The bits cycle from x first then y. The input numbers x and y are expected to range from -180 to 180 and have a decimal precision of 10^-6. The internal structure of a float is hard to manipulate so the input numbers are multiplied by 10^6 and stored as integers. A mask value is generated so that the Most Significant Bit (MSB) is set. A loop takes the the input values and looks at each bit according to the mask value and adds that bit to the result. The mask is then shifted to the left (towards the LSB) and the loop continues. The int8 structure is split into two unsigned long integers along with a sign value. When the int8 structure is returned to the database the second integer is output before the first. Imagine one big number. All the bits starting from the MSB from the second integer get placed into the big number. Then all the bits from the first are placed after the second. Now the MSB of the second integer matches the MSB of the big number and the LSB of the first integer matches the LSB of the big number. The reason the outer loop is because one int8 is really two C integers.

The `BitInterleave` function creates a liner ordering of the input points known as the *Morton order* [7, 8]. The crucial observation here is that all the points (x, y) that are contained in a window $((x_l, y_l), (x_u, y_u))$ have `BitInterleave`(x, y) contained in the range `BitInterleave`(x_l, y_l) through `BitInterleave`(x_u, y_u). Then, one can build a B–tree index (or a variant) to evaluate a spatial window query. Tropf and Herzog [10], and White [11] show how to evaluate a spatial window query for a relation with n points in a k–dimensional space in time $O(k \log n + A)$, where A is the number of points in the window (see also [8]). Faloutsos [3, 4] conjectures that by using the Gray codes of x and y in the bit–interleaving an improvement of at most 50% can be obtained. He shows such an improvement for *partial match queries*. Note that by considering only a fixed number of most–significant bits from x and y, one can obtain a *uniform grid partition* [8] with rectangular cells. Thus, it would be possible to create a clustered B–tree index on the bit–interleaving of x and y as an approach to improve on the number of page reads for an R–tree index of those points.

```
#include <mi.h>

#define SIGN(X)  ((X) < 0 ? (-1) : (1))
#define VALUE(X) (*(X))
#define ABS(X)   ((X) < 0 ? (-(X)) : (X))
#define SCALE    1000000

mi_int8* BitInterleave(mi_double_precision* X, mi_double_precision* Y) {
  mi_unsigned_integer Xint, Yint, mask = 0;
  mi_double_precision Xval, Yval;
  mi_int8* result = 0;
  mi_integer i = 0, j = 0;

  result = (mi_int8*) mi_alloc (sizeof(mi_int8));

  if (result == NULL) {
    mi_db_error_raise (NULL, MI_EXCEPTION, "BitInterleave: Out of Memory");
```

```
  }

  Xval = VALUE(X);
  Yval = VALUE(Y);

  result->data[0] = 0;
  result->data[1] = 0;
  result->sign = SIGN(Xval) * SIGN(Yval);

  Xint = ABS(Xval) * SCALE;
  Yint = ABS(Yval) * SCALE;

  /* Prepare the bit mark */
  mask = ~mask;
  mask = mask >> 1;
  mask = ~mask;

  /* Start our loop twice for both integers in the int8 */
  for (i=1; i >= 0; i--) {
    for (j = sizeof(mi_unsigned_integer)*4; j>0; j--) {
      if (mask & Xint)
        result->data[i]++;
      result->data[i] *= 2; /* Shift left */
      if (mask & Yint)
        result->data[i]++;
      if (j != 1)
        result->data[i] *= 2; /* Shift left */
      mask = mask >> 1;
    }
  }
  return (result);
}
```

Compile the external C function `BitInterleave` by including the following directory

```
${Informix}/incl/public
```

and append it to `/monet/bits.so`. Declare a corresponding SQL function for the external C functions as follows:

```
CREATE FUNCTION BitInterleave (float, float)
        RETURNING int8
```

```
EXTERNAL NAME "/monet/bits(BitInterleave)"
LANGUAGE c;
```

An example that uses the bit interleaving function is:

SELECT BitInterleave(x,y) FROM test;

where the schema of `test` is `test(x Float, y Float)`.

6.2 Datablade Functions

In this section we describe external C functions for the Geodetic and Shapes2 datablades. These functions are meant as utility functions for simplifying SQL spatial queries.

We have written a `MakeGeoBox` external C function for the Geodetic datablade that takes as input a point and two numbers W and H and returns a box (of type `GeoBox` centered at the input point with width W and height H. The `MakeGeoBox` function is quite useful for dynamically creating "uncertainty" boxes around stars when specifying spatial selections and joins. Unfortunately, when executing this function on relations with more than 100,000 objects, the database server runs out of memory. Despite all our efforts so far, and contacts with the Informix technical support, we did not manage to avoid this memory problem.

We have also written similar external functions for the Shapes2 datablade: a `MakeShapes2Box` similar to the MakeGeoBox function for the Geodetic datablade, and two other functions `GetPointX` and `GetPointY` that take as input a point (of type `MyPoint` in the Shapes2) and return the x and y coordinates of that point as floats. Unfortunately, we encountered the same difficulties with memory with the Shapes2 as we did with the Geodetic datablade.

6.2.1 External Functions for the Geodetic Datablade

We provide an external C function for creating a box, for the Geodetic datablade, of type GeoBox, centered on a point.

```
#include <mi.h>
#include <geodetic.h>

/* Construct and return a GeoBox with width 2w and height 2h
   centered on the input GeoPoint */

mi_lvarchar* MakeGeoBox (mi_lvarchar* pointIn,
                         mi_double_precision* w,
```

```
                            mi_double_precision* h,
                            MI_FPARAM* fparam) {

  GeoObjectData* objData;
  GeoBoxData* boxData;
  GeoObjectData* tempObj;
  GeoPointData* tempPoint;
  mi_lvarchar* retVal;
  mi_integer objSize = 0;

  tempObj = (GeoObjectData*) mi_get_vardata (pointIn);
  tempPoint = (GeoPointData*) tempObj->data;

  objSize = sizeof (GeoObjectData) + sizeof (GeoBoxData) - 1;
  objData = (GeoObjectData*) mi_zalloc (objSize); /* Fills with zeros */

  objData->version = GEO_OBJECT_VERSION;
  objData->tag = GeoData_Box;
  objData->alt_range.rtype = GeoRange_Any;
  objData->time_range.rtype = GeoRange_Any;

  boxData = (GeoBoxData*) objData->data;
  boxData->sw.lat = tempPoint->lat - 2*(*h);
  boxData->sw.lon = tempPoint->lon - 2*(*w);
  boxData->ne.lat = tempPoint->lat + 2*(*h);
  boxData->ne.lon = tempPoint->lon + 2*(*w);

  retVal = (mi_lvarchar*) mi_new_var (objSize);
  mi_set_vardata (retVal, (mi_char*) objData);

  return (retVal);
}
```

We compile the `MakeGeoBox` external C function using the following include directories

```
${Informix}/incl/public
${Informix}/extend/geodetic.2.11.UC3/client/solaris/incl
```

and append it to `/monet/Geodetic.so`. We declare a corresponding SQL function for the external C function `MakeGeoBox` as follows:

```
CREATE FUNCTION MakeGeoBox (GeoObject, float, float)
        RETURNING GeoBox
```

```
        EXTERNAL NAME "/monet/Geodetic.so(MakeGeoBox)"
        LANGUAGE c;
```

The SQL function `MakeGeoBox` can be used as in the following two examples:

(a) `SELECT MakeGeoBox(shape::lvarchar,1,2) FROM geotest;` and
(b) `EXECUTE FUNCTION MakeGeoBox(GeoPoint(GeoCoords(5,4), null, null)::lvarchar, 1,2);`.

6.2.2 External Functions for the Shapes2 Datablade

We describe two external C functions that return the X and Y values from a `MyPoint` structure. A pointer to the binary data is extracted from the `LVARCHAR` and mapped into a `Shape` structure. The same memory that was passed in gets returned, so no memory allocation is needed.

```
#include <ShapeTypes.h>
#include <mi.h>

mi_double_precision* GetMyPointX(mi_lvarchar* pointIn, MI_FPARAM* fparam) {

  Shape* tempObj;
  PointData* tempPoint;

  tempObj = (Shape*) mi_get_vardata_align (pointIn, SHAPES_ALIGN);
  tempPoint = (PointData*) tempObj->data;

  if (tempObj->hdr.tag != PointT) {
    mi_db_error_raise (NULL, MI_EXCEPTION, "GetMyPointX: Not a Point");
  }
  return (&tempPoint->x);
}

mi_double_precision* GetMyPointY(mi_lvarchar* pointIn, MI_FPARAM* fparam) {
  Shape* tempObj;
  PointData* tempPoint;

  tempObj = (Shape*) mi_get_vardata_align (pointIn, SHAPES_ALIGN);
  tempPoint = (PointData*) tempObj->data;

  if (tempObj->hdr.tag != PointT) {
    mi_db_error_raise (NULL, MI_EXCEPTION, "GetMyPointY: Not a Point");
  }
```

```
  return (&tempPoint->y);
}
```

We compile these two external C functions, `GetMyPointX` and `GetMyPointX`, using the following include directories `${Informix}/incl/public` and `${Informix}/extend/ShapesTemp/Shapes2.1.0/src` and append to `/monet/Shapes2.so`. We declare corresponding SQL functions for the `GetMyPointX` and `GetMyPointX` functions:

```
CREATE FUNCTION GetMyPointX (MyShape) returning float
        WITH (NOT VARIANT)
        EXTERNAL NAME "/monet/Shapes2.so(GetMyPointX)"
        LANGUAGE c;

CREATE FUNCTION GetMyPointY (MyShape) returning float
        WITH (NOT VARIANT)
        EXTERNAL NAME "/monet/Shapes2.so(GetMyPointY)"
        LANGUAGE c;
```

An example usage of these SQL functions is:

```
SELECT GetMyPointX(shape), GetMyPointY(shape) FROM shapestest;
```

We also describe the following external C function that creates a box, for the Shapes2 datablade, of type `MyBox`, centered on a point.

```
#include <ShapeTypes.h>
#include <mi.h>

mi_double_precision* GetMyPointX (mi_lvarchar*, MI_FPARAM*);
mi_double_precision* GetMyPointY (mi_lvarchar*, MI_FPARAM*);

mi_lvarchar* MakeShapes2Box (mi_lvarchar* pointIn,
     mi_double_precision* w,
     mi_double_precision* h,
     MI_FPARAM* fparam) {

  Shape* boxOut = NULL;
  BoxData* boxData;
```

```
mi_lvarchar* stringPtr = NULL;
mi_integer varchar_len = 0;
mi_double_precision *xIn = GetMyPointX (pointIn, fparam);
mi_double_precision *yIn = GetMyPointY (pointIn, fparam);

mi_tracelevel_set ("__myErrors__ 25");

/* Is this our first time around ? If not get our pointer to stringPtr */
stringPtr = mi_fp_funcstate (fparam);

if (stringPtr == (mi_lvarchar*) NULL) {

  /* Set our memory size */
  varchar_len = (sizeof (ShapeHdr) + sizeof (BoxData)) * 2;
  varchar_len = MI_ALIGNBYTES (varchar_len, SHAPES_ALIGN);

  /* Change memory duration to last through multiple calls and allocate memory.
     Database will deallocate memory when sql command is finished */

  mi_switch_mem_duration(PER_COMMAND);
  stringPtr = (mi_lvarchar*) mi_new_var (varchar_len);

 DPRINTF ("__myErrors__", 20, ("MakeShapes2Box: Allocating %d bytes of memory\n",
          sizeof (stringPtr)));

  mi_switch_mem_duration(PER_FUNCTION);

  if (stringPtr == (mi_lvarchar*) NULL) {
    mi_db_error_raise (NULL, MI_EXCEPTION, "MakeShapes2Box: Out of Memory.");
  }

  /* Save our memory pointer for next time */
  mi_fp_setfuncstate (fparam, (void*) stringPtr);
}

/* get the shape pointer out of the string */
boxOut = (Shape*) mi_get_vardata_align (stringPtr, SHAPES_ALIGN);

/* get the BoxData pointer out of the shape */
boxData = (BoxData*) boxOut->data;

/* Filling data */
boxData->ll.x = *xIn - (*w)/2;
boxData->ll.y = *yIn - (*h)/2;
boxData->ur.x = *xIn + (*w)/2;
```

```
  boxData->ur.y = *yIn + (*h)/2;

  /* Filling in the header */
  boxOut->hdr.xmin = boxData->ll.x;
  boxOut->hdr.ymin = boxData->ll.y;
  boxOut->hdr.xmax = boxData->ur.x;
  boxOut->hdr.ymax = boxData->ur.y;
  boxOut->hdr.tag = BoxT;

  return (stringPtr);
}
```

The `MakeShapes2Box` function creates a bounding box based on a point and a width W and height H. The input point is of type `MyPoint` and the input width and height are of type `float`. The input point will lie in the center of the bounding box and the box will have a width W and height H. This function is to be used with the Shapes2 datablade and will return an object of type `MyBox`. A special technique is used within the function to save memory and time when the function is called multiple times within a single SQL statement. All `MyBox` structures consume the same amount of memory and therefore can use the same memory space over and over. When this function gets called multiple times, the database passes the same `MI_FPARAM` structure to every call within a single SQL statement. When the function is called for the first time, it allocates memory for the `MyBox` return value and then stores the memory location in the `MI_FPARAM` structure for the next call. If the function gets called more than once within a single SQL statement, it allocates memory once.

We compile the `MakeShapes2Box` external C function using the following include directories

```
${Informix}/incl/public
${Informix}/extend/ShapesTemp/Shapes2.1.0/src
```

and append it to `/monet/Shapes2.so`. Then, we declare the corresponding SQL function for the `MakeShapes2Box` C external function as follows:

```
CREATE FUNCTION MakeShapes2Box (MyShape, float, float)
        RETURNING MyBox
        WITH (NOT VARIANT)
        EXTERNAL NAME "/monet/Shapes2.so(MakeShapes2Box)"
        LANGUAGE c;
```

An example usage of this SQL function is:

```
SELECT MakeShapes2Box(shape::lvarchar,1,2) FROM shapestest;
```

6.2.3 Aggregate External Functions

The functions in this section work together to make an aggregate function returning multiple values. The example function in this section will return the largest K integers from a column of tuples where K is passed as input from the SQL statement. This example could be easily modified to return the smallest K items or use any other method of comparison. Items other than integers may be used, but must be referenced by pointers and dynamically allocated. This example implements a sorted array (truncated to be of size K) to keep track of the largest K items by using a variation of insertion sort.

In order to implement an aggregate function, one needs to implement a number of support functions: (a) a function that initializes the data structures for the aggregate, (b) an iterator function that is executed for qualifying element together with a combinator function that updates the data structures for the aggregate function, and (c) a finalization function that is executed upon the completion of the input elements, which in addition to housekeeping, it may do additional computations to find the value of the aggregate function (e.g. for an average aggregate the finalizer may divide the running sum by the number of qualifying elements). The code below describes an implementation of these functions for the `topK` aggregate function of this section.

```
#include <mi.h>
#include <string.h>

#define value_t  mi_integer

typedef struct {
  int K;
  int n;
  value_t *data;
} array_t;

void InsertElement(array_t *v, value_t value) {

  if ( v->n == v->K && value < v->data[v->K-1] )
    return; /* value is smaller than the K values already present */
  i = v->n-1;

  /* Shift the smaller elements to the left by 1 position */
  while ( i >=0  && v->data[i] < value ) {
    if ( i + 1 < v->K )
      v->data[i+1] = v->data[i];
  }
  v->data[i] = value;
  if ( v->n < K )
    v->n++;
```

```
}

array_t* TopKInit(value_t dummy, mi_integer K, MI_FPARAM* fparam) {
  array_t *v;

  v = (array_t*) mi_dalloc (sizeof(array_t), PER_COMMAND);

  if ( v == NULL )
    mi_db_error_raise (NULL, MI_EXCEPTION, "TopKInit: Out of Memory");

  if (mi_fp_argisnull (fparam, 0))
    ; /* Error if arg1 is NULL */
  if (K <= 0)
    K = 1;

  v->K = K;
  v->n = 0;
  v->data = (value_t*) mi_dalloc (sizeof(value_t)*K, PER_COMMAND);
  if ( v->data == NULL )
    mi_db_error_raise (NULL, MI_EXCEPTION, "TopKInit: Out of Memory");
  memset(v->data, 0, sizeof(value_t)*K);
  return (v);
}

array_t* TopKIter(array_t *v, value_t value) {
  InsertElement(v, value);
  return(v);
}

array_t* TopKCombine(array_t *u, array_t *v) {
  int i;
  for(i=0; i<v->n; i++)
   InsertElement(u, v->data[i]);

  mi_free(v->data);
  mi_free(v);

  return(u);
}

MI_COLLECTION* TopKFinal (array_t *v) {

  MI_COLLECTION *collection;
  MI_COLL_DESC  *collection_desc;
  MI_TYPEID     *typeId;
```

```
  MI_CURSOR_ACTION action;
  INDEX i;

  /* This is the type of the collection. All sql types are valid, but it
   * should remain a list, set, or multiset */

  typeId = mi_typestring_to_id (NULL, "list(int not null)");
  if (typeId == NULL) {
    mi_db_error_raise (NULL, MI_EXCEPTION, "TopKFinal: Not a valid type");
  }

  collection = mi_collection_create (NULL, typeId);
  if (collection == NULL)
    mi_db_error_raise (NULL, MI_EXCEPTION, "TopKFinal: Cannot create a collection");

  collection_desc = mi_collection_open (NULL, collection);

  action = MI_CURSOR_ABSOLUTE;

  for(i=0; i<v->n; i++)
    mi_collection_insert(NULL, collection_desc, (MI_DATUM) v->data[i], action, i);

  mi_collection_close (NULL, collection_desc);
  mi_free (v->data);
  mi_free (v);

  return (collection);
}
```

We compile these external C functions by including the following directory `${Informix}/incl/public` and append them to `/monet/topK.so`. Declare a corresponding SQL function for the external C functions as follows:

```
CREATE FUNCTION TopKInit (integer, integer)
RETURNING pointer
WITH (handlesnulls)
EXTERNAL NAME "/monet/topK.so(TopKInit)"
LANGUAGE c;

CREATE FUNCTION TopKIter (pointer, integer)
        RETURNING pointer
WITH (handlesnulls)
        EXTERNAL NAME "/monet/topK.so(TopKIter)"
```

```
        LANGUAGE c;

CREATE FUNCTION TopKCombine (pointer, pointer)
        RETURNING pointer
WITH (handlesnulls)
        EXTERNAL NAME "/monet/topK.so(TopKCombine)"
        LANGUAGE c;

CREATE FUNCTION TopKFinal (pointer)
        RETURNING list(int not null)
WITH (handlesnulls)
        EXTERNAL NAME "/monet/topK.so(TopKFinal)"
        LANGUAGE c;

CREATE AGGREGATE TopK WITH(
INIT = TopKInit,
ITER = TopKIter,
COMBINE = TopKCombine,
FINAL = TopKFinal,
handlesnulls);
```

An example that uses the aggregate function `topK` is:

```
SELECT topK(x, 5) FROM test;
```

where the schema of `test` is `test(x Int)`.

7 Development Environment

Using user–defined types and functions has a substantial impact on the amount and complexity of application code that has to be written. For example, referring to Sections 2.4.5 and 2.4.6, we see that expressing spatial 3–chain–join and 3–star–join queries using the **B** schemas uses seven rather complex conditions in the where–clause of the select statement, while for the **G** and **S** schemas it uses 3 intuitive conditions in the where–clause of the select statement.

We also make the following observations regarding the overall application development environment that was available to us:

- The lack of sufficient instrumentation for performance measurement per query, that includes timing with high resolution, complicated the development. Informix provides performance measurement tools per table and per session, but not per query for a number of measures, but no sufficient timing information (with the exception of server CPU times). We feel that an `onperf` utility that provides measurements per query as well as per every major database

element (e.g. table, tablespace, etc) that includes high–resolution timing measurements will be quite useful.

- There seem to be certain limitations on the maximum number of aggregates in the select list of SQL queries that are not documented. The performance measurement data are spread over tables `sysvpproff` and `syssesprof` of the `sysmaster` database. To obtain the performance data we had to join the two tables on the table name, and session id. We needed the `avg, min, max, stddev and variance` functions for each of the (approximately) 20 measures listed in section 2.6, i.e. a total of approximately 100 aggregate functions. It would have been easiest to run just one SQL query on the `sysmaster` database to get the required information. Unfortunately, Informix would hang every time we tried to run such a query. We found out the hard way that lot of aggregate functions in the same select statement with multiple join conditions are not well supported by Informix. It would have been quite useful if there was documentation that stated the maximum number of aggregates supported for queries involving joins.

- It is not clear why creation of R–tree indexes on non-logged databases is not permitted. It would have been useful if that requirement was stated much clearly and much earlier.

- Insufficient object–oriented features:

  – Informix allows definition of distinct data types that are based on some existing type, e.g.:

    ```
    CREATE DISTINCT TYPE Celsius AS INTEGER;
    ```

    Informix defines explicit casts between a distinct type and the corresponding base type. Hence, every time the cast needs to be used explicitly, which increases the amount and complexity of application code that is needed.

    ```
    CREATE TABLE temperature(day INTEGER, temp celsius);
    INSERT INTO TABLE temperature VALUES (1, 32::celsius);
    ```

    Also, while comparing an attribute of type `celsius` to an attribute or value of `integer` type, an explicit type cast would be needed `SELECT * FROM temperature WHERE temp::integer > 45;` In a true object–oriented environment the cast is context dependent and is implicit in similar circumstances.

  – Constraints on named row types can not be defined directly. (see *Informix Guide to SQL: Syntax*, Vol. 1, Version 9.1, pg. 1–199). Named row types is the simplest method for creating user–defined–types. Constraints need to be defined in the table that uses the named row type. When the same named row type in used in several tables, it leads to data integrity problems.

  – User–defined–Types (UDTs) and User–defined–functions (UDFs) are not *first–class citizens* in Informix.

- There was not sufficient documentation of advanced features (e.g. writing datablades, indexes, UDFs, etc). For example, developing aggregate UDFs was not sufficiently documented for the version we are using.

- It is unclear why the database server is running out of memory when using certain UDFs for large tables (with at least 100,000 rows) even though those UDFs are using the proper flags to reuse memory.
- To better understand the behavior, advantages, and disadvantages of R–tree indexes, it is useful to look at the internal information stored in R–tree indexes. Informix's `sysindices` table of the `sysmaster` database provides internal information for B–tree indexes, such as number of leaves, levels of the index, key values stored at index nodes, etc. Unfortunately, similar information for the R–tree indexes of the Shapes2 and Geodetic datablades is not available.

8 Conclusions

We investigated experimentally the efficiency of certain object–relational features of the Informix Universal Server for processing spatial queries for large datasets and compared it with the features in traditional relational systems. We also developed a number of user–defined–functions for spatial queries. From our experiments, we draw the following conclusions:

- B–tree indexes can be adequate for spatial window queries with small windows.
- R–tree indexes outperform B–tree indexes for spatial join queries.
- B–tree indexes are more sensitive to the distribution and clustering of the data than R–tree indexes. The choice of clustering fields has a greater impact on the performance of B–tree indexes than on R–tree indexes.
- Creating R–tree indexes is not significantly more time–consuming than creating B–tree indexes.
- Clustering and small space requirement are significant advantages for B–tree indexes compared with R–tree indexes.
- The average number of page reads, as a performance measurement, is an important but not always a good predictor of the response time for spatial queries. The number of buffer reads can have a significant impact on the response time of spatial queries (e.g. spatial join queries). One should look at both the number of page reads and the number of buffer reads when investigating the performance of spatial queries. An excessive number of buffer reads, as compared with the number of page reads, will degrade query response times.
- A datablade that allows clustering for R–tree indexes will provide significantly improved performance.
- A "light–weight" datablade, with smaller space overhead than the Shapes2 and Geodetic datablades, that implements the R–tree operators will significantly reduce the storage cost for spatial data and will have a noticeable effect on the performance of spatial queries.

- Allowing the creation of optimized R–tree indexes for static spatial data is desirable, since in many catalog applications such as Monet, the stored data change infrequently.
- Building a "general" spatial datablade, which is bound to be "heavy–weight" in terms of space overhead since it tries to handle a large variety of spatial objects, is not appropriate for all applications. Instead, a skeletonized and easily configurable–modifiable spatial datablade with small default space overhead will be more appropriate.
- In the absence of R–trees or clustering for R–trees, mapping spatial 2–D objects on the line (e.g. mapping 2–D points to 1–D points on a line via the Morton order) is helpful for spatial queries. [ongoing work]
- The approach by Faloutsos and Kamel [5] for predicting the average number of page reads by R–tree indexes for spatial window queries seems to be effective for the Shapes2 datablade, but not for the Geodetic datablade.
- True object–relational Database Management Systems should treat user–defined access methods, operator classes, and data types as *first–class citizens*. Unfortunately, this is not always the case with the Informix Universal Server.
- Using object–relational features can reduce the amount and complexity of application code and improve its comprehensibility.

Regarding the Monet catalog, since it is expected that not only spatial window queries but also spatial join queries will be extensively used, we conclude that it is beneficial to manage it using the UDTs, UDFs, and R–tree indexes of the Shapes2 datablade rather the traditional relational features and B–tree indexes. The extra storage requirements for the solution that uses the Shapes2 features pay off in significantly smaller response times for spatial join queries. Moreover, we conclude that a light–weight datablade with clustered R–tree indexes will significantly improve on the performance of spatial query processing for the Monet catalog and other catalogs to be joined with it.

Regarding future work, we plan to experiment with some additional user–defined functions for querying and indexing spatial data, and develop a lightweight datablade for point and box data that can be used for the Monet catalog application. We also plan to develop additional user–defined–functions for aggregating, clustering, sampling, classification, etc, possible by using a statistical analysis package like S–plus. We will focus on spatial and non-spatial time–series data mining.

Acknowledgments

We would like to thank Dr. Tom McGlynn, NASA GSFC, Code 668 for his comments and suggestions.

References

[1] J. W. Barnes. *Statistical Analysis for Engineers and Scientists: A Computer–Based Approach.* McGraw–Hill, New York, NY, 1994.

[2] S. Dowdy and S. Wearden. *Statistics for Research.* John Wiley & Sons, New York, NY, 1983.

[3] C. Faloutsos. Multiattribute Hashing Using Gray Codes. In *Proceedigs of the ACM SIGMOD Conference*, pages 227–238, 1986.

[4] C. Faloutsos. Gray Codes for Partial Match and Range Queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, 1988.

[5] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R–trees Using the Concept of Fractal Dimension. In *Proceedings of the ACM SIGMOD Conference*, pages 4–13, 1994.

[6] A. Guttman. R–Trees A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, 1984.

[7] G. M. Morton. A Computer Oriented Geodetic Database and a New Technique in File Sequencing. IBM Ltd., Ottawa, Canada, 1966.

[8] Hanan Samet. *Geographic Information Systems (GIS): A Database Management (DBMS) Perspective.* Continuing Education In Engineering, University Extension, University of California, Berkeley, 1998.

[9] M. Stonebraker, P. Brown, and D. Moore. *Object–Relational DBMSs: Tracking the Next Great Wave.* Morgan Kaufmann, San Fransisco, CA, 1999.

[10] H. Tropf and H. Herzog. Multidimensional Range Search in Dynamically Balanced Trees. *Angewandte Informatik*, 23(2):71–77, 1981.

[11] M. White. N–Trees: Large Ordered Indexes for Multi–Dimensional Space. Statistical Research Division, U.S. Bureau of Census, Washington, DC, 1982.

A The Monet Catalog

This appendix makes several points about the format of the Monet catalog.

1. To avoid negative values, the Monet catalog uses South Polar Distance (SPD) instead of declination. SPD ranges from 0 degrees to 180 degrees, and is given by *SPD=dec + 90 degrees.*

2. The sky is broken up into 24 zones of SPD, each 7.5 degrees wide. The Monet catalog consists of 24 files, one for each zone of SPD. These files are labeled zoneXXXX.cat, where XXXX=10 times the SPD (0,75,150,...,1725).

3. Each record consists of three 32-bit signed integers. Byte order is big endian. The format is **RA — DEC — MAG** where **RA** and **DEC** both take a full 32 bit integer. The third 32-bit integer, representing **MAG** (magnitude), has the following (decimal) format: SQFFFBBBRRR. The significance of these digits is as follows:

   S Sign. The sign is "-" if this star is also in the ACT catalog, and "+" otherwise.

   Q Flag. The flag is 1 if the magnitudes are suspected to be in error, and 0 if they should be accurate.

   FFF Field. The first Palomar Observatory Sky Survey (POSS-I) established a canonical decomposition of the sky into 1431 fields. In the North, fields start at 1 in the North Pole and at 825 at -20 degrees. In the South, fields start at 1 in the South Pole and end at 606 at -20 degrees. To save space, only the field number is given, without an indication of North or South. All records in the first 11 catalog files with field number less than 606 are measured from the South. Otherwise, they are measured from the North.

   BBB Blue. This value is ten times the blue magnitude. Only values between 0 and 250 are valid.

   RRR Red. This value is ten times the red magnitude. Only values between 0 and 250 are valid.

4. To store coordinates as integers, the compilers of the Monet catalog made the following conversions:

   - **RA**=(RA in decimal hours)*15*3600*100
   - **DEC**=((DEC in Decimal degrees) +90)*3600*100

5. Our datablades store latitude and longitude in decimal degrees and hours respectively, so we convert back as follows:

   - (long in decimal degrees) = RA/(15*3600*100)
   - (lat in decimal degrees) = dec/((3600*100)-90)

B The Datablades

This appendix describes the two datablades that were used in these experiments: the *Informix Geodetic Datablade* and the *Shapes2 Datablade.*

B.1 The Informix Geodetic Datablade

The Geodetic Datablade was designed for use in geodesy. For our application, we used only a limited fraction of datablade's functionality. In particular, we had no use for the *altitude range* or *time range* attributes of the `GeoObject` datatype. The components we used were:

`GeoLatitude.` measured in decimal degrees from -90 through +90, inclusive.

`GeoLongitude.` measured in decimal degrees from -180 to +180, inclusive.

`GeoObject.` `GeoObject` is the supertype for spatio-temporal objects, and represents the elements common to all spatio-temporal objects: `altitude range` and `time range`. We had no use for either of these attributes, and always set their values to NULL.

`GeoPoint.` A `GeoPoint` is a `GeoObject` with the external representation `((lat,long), altrange, timerange)`, where lat and long represent `GeoLatitude` and `GeoLongitude` respectively.

`GeoBox.` A `GeoBox` is a `GeoObject` with the external representation `((latS, longW), (latN, longE), altrange, timerange)`, where `(latS, longW)` represents the southwest corner of the `GeoBox` and `(latN, longE)` represents the northeast corner of the `GeoBox`.

Inside. The Inside function compares two `GeoObject`s, and returns TRUE if the first object is contained in the second, and FALSE otherwise. We used this function to determine if a specified `GeoPoint` was within a specified `GeoBox`.

MakeGeoBox. Create a `GeoBox` object. See Section 6.2.1.

B.2 The Shapes2 DataBlade

The Shapes2 DataBlade is a sample datablade, which demonstrates how a type hierarchy is implemented using opaque types and implicit casts. It is no longer supported by Informix.

`MyShape.` MyShape is the supertype for all UDTs in the Shapes2 datablade. Every UDT is stored internally as a `MyShape`.

`MyPoint.` MyPoint is a `MyShape` with the external representation `point(x, y)`.

`MyBox.` `MyBox` is a `MyShape` with the external representation `box(xS, yW, xN, yE)`.

Within. This function is semantically equivalent to the Geodetic Datablade's Inside function.

MakeShapes2Box. Create a `MyBox` object. See Section 6.2.2.

C Usage Notes of Programs and Commands

C.1 Data Loading

We have written the C program `LoadFile` that reads a binary encoded file from the Monet catalog, and formats the data so that they can be loaded into database tables using the Informix `LOAD` command. Since the Monet catalog is encoded in binary format, our first step is to read the binary files and convert them to ASCII. The mechanisms of this conversion are described in Appendix A. The program, `LoadFile`, that performs the conversion also formats the data for loading into an Informix table. The usage of `LoadFile` is as follows:

```
LoadFile <catalog filename> <output filename> <mode> <no.of recs>
<RecordType> <clustering>
```

where `<mode>` is `a` (for append) or `w` (for write), `<recordType>` is `1` (for **G** schema), `2` (for **B** schema), `3` (for **S** schema), `<clustering>` is `1` (for unclustered schema) or `2` (for clustered **S** and **G** schemas).

The Informix `LOAD` command is used to load the formatted data into the tables. The usage of `LOAD` is as follows:

```
LOAD FROM <data filename> DELIMITER '|' INSERT INTO <table name>
```

This command is run from the ESQL-C program `sqlplus`, which records performance measures.

We generate sample spatial window and spatial self–join queries using our `gen_queries` program. The usage of `gen_queries` is:

```
gen_queries <output filename> <base tablename> <table size> <option>,
```

where `<option>` is `1` (for **G** schemas), `2` (for **B** schemas), and `3` (for **S** schemas).

We compute statistics for the measurements obtained for the queries using the C program `analyze` we developed. The `analyze` program computes the the minimum, maximum, average, and standard deviation for each measurement for each query type and each schema and clustering. It also outputs appropriate commands for generating a variety of plots using Matlab. Further, it creates LaTeXfiles tabulating the statistical information above. The usage of this program is

```
analyze <option> <stats filename> <measures files PREFIX> | <tables file
PREFIX> <index>
```

where `<option>` is `1` (for printing measures files for Matlab), `2` (for printing measures table files for LaTeX), and `3` (for both); `<index>` is the type of index, e.g. Clustered B–tree, Unclustered R–tree etc. For example,

```
analyze 3 measures.stats ClustBtree ClustBtree "Clustered Btree"
analyze 3 measures.stats unclustRtree unclustRtree "Unclustered Btree"
```

D Informix Configuration Parameters

The configuration parameters for the Informix server used for the experiments, obtained with the `onstat -c` Informix utility, are as follows:

```
INFORMIX-Universal Server Version 9.14.UC5   -- On-Line -- Up 00:03:25 -- 100864 Kbytes

Configuration File: /usr/local/Informix/etc/onconfig.jeanne
#**************************************************************************
#
#    INFORMIX SOFTWARE, INC.
#
#  Title: onconfig.std
#  Description: INFORMIX-Universal Server Configuration Parameters
#
#**************************************************************************

# Root Dbspace Configuration

ROOTNAME        rootdbs         # Root dbspace name
ROOTPATH        /isc2/INFORMIX/rootdbs # Path for device containing root dbspace
ROOTOFFSET      0               # Offset of root dbspace into device (Kbytes)
ROOTSIZE        100000          # Size of root dbspace (Kbytes)

# Disk Mirroring Configuration Parameters

MIRROR          0               # Mirroring flag (Yes = 1, No = 0)
MIRRORPATH                      # Path for device containing mirrored root
MIRROROFFSET    0               # Offset into mirrored device (Kbytes)

# Physical Log Configuration

PHYSDBS         rootdbs         # Location (dbspace) of physical log
PHYSFILE        1000            # Physical log file size (Kbytes)

# Logical Log Configuration

LOGFILES        13              # Number of logical log files
LOGSIZE         2000            # Logical log size (Kbytes)

# Diagnostics

MSGPATH         /usr/local/Informix/online.log # System message log file path
```

```
CONSOLE         /dev/console    # System console message path
ALARMPROGRAM    /usr/local/Informix/etc/log_full.sh # Alarm program path

# System Archive Tape Device

TAPEDEV         /dev/null       # Tape device path
TAPEBLK         16              # Tape block size (Kbytes)
TAPESIZE        10240           # Maximum amount of data to put on tape (Kbytes)

# Log Archive Tape Device

LTAPEDEV        /dev/null       # Log tape device path
LTAPEBLK        16
LTAPESIZE       10240           # Max amount of data to put on log tape (Kbytes)

# Optical

STAGEBLOB                       # INFORMIX-OnLine/Universal Server staging area

# System Configuration

SERVERNUM       0               # Unique id corresponding to a OnLine instance
DBSERVERNAME    jeanne          # Name of default database server
DBSERVERALIASES                 # List of alternate dbservernames
DEADLOCK_TIMEOUT 60              # Max time to wait of lock in distributed env.
RESIDENT        0               # Forced residency flag (Yes = 1, No = 0)

MULTIPROCESSOR  1               # 0 for single-processor, 1 for multi-processor
NUMCPUVPS       2               # Number of user (cpu) vps
SINGLE_CPU_VP   0               # If non-zero, limit number of cpu vps to one

NOAGE           0               # Process aging
AFF_SPROC       0               # Affinity start processor
AFF_NPROCS      0               # Affinity number of processors

# Shared Memory Parameters

LOCKS           2000            # Maximum number of locks
BUFFERS         200             # Maximum number of shared buffers
NUMAIOVPS       4               # Number of IO vps
PHYSBUFF        32              # Physical log buffer size (Kbytes)
LOGBUFF         32              # Logical log buffer size (Kbytes)
LOGSMAX         100             # Maximum number of logical log files
CLEANERS        1               # Number of buffer cleaner processes
SHMBASE         0xa000000         # Shared memory base address
```

```
SHMVIRTSIZE     100000           # initial virtual shared memory segment size
SHMADD          8192            # Size of new shared memory segments (Kbytes)
SHMTOTAL        0               # Total shared memory (Kbytes). 0=>unlimited
CKPTINTVL       3000            # Check point interval (in sec)
LRUS            4               # Number of LRU queues
LRU_MAX_DIRTY   80              # LRU percent dirty begin cleaning limit
LRU_MIN_DIRTY   70              # LRU percent dirty end cleaning limit
LTXHWM          50              # Long transaction high water mark percentage
LTXEHWM         60              # Long transaction high water mark (exclusive)
TXTIMEOUT       0x12c             # Transaction timeout (in sec)
STACKSIZE       32              # Stack size (Kbytes)

# System Page Size
# BUFFSIZE - OnLine no longer supports this configuration parameter.
#            To determine the page size used by OnLine on your platform
#            see the last line of output from the command, 'onstat -b'.


# Recovery Variables
# OFF_RECVRY_THREADS:
# Number of parallel worker threads during fast recovery or an offline restore.
# ON_RECVRY_THREADS:
# Number of parallel worker threads during an online restore.

OFF_RECVRY_THREADS 10              # Default number of offline worker threads
ON_RECVRY_THREADS 1               # Default number of online worker threads

# Data Replication Variables
# DRAUTO: 0 manual, 1 retain type, 2 reverse type
DRAUTO          0               # DR automatic switchover
DRINTERVAL      30              # DR max time between DR buffer flushes (in sec)
DRTIMEOUT       30              # DR network timeout (in sec)
DRLOSTFOUND     /usr/informix/etc/dr.lostfound # DR lost+found file path

# Backup/Restore variables
BAR_ACT_LOG     /tmp/bar_act.log
BAR_MAX_BACKUP  0
BAR_RETRY       1
BAR_NB_XPORT_COUNT 10
BAR_XFER_BUF_SIZE 31

# Read Ahead Variables
RA_PAGES                        # Number of pages to attempt to read ahead
RA_THRESHOLD                    # Number of pages left before next group
```

```
# DBSPACETEMP:
# OnLine equivalent of DBTEMP for SE. This is the list of dbspaces
# that the OnLine SQL Engine will use to create temp tables etc.
# If specified it must be a colon separated list of dbspaces that exist
# when the OnLine system is brought online.  If not specified, or if
# all dbspaces specified are invalid, various ad hoc queries will create
# temporary files in /tmp instead.

DBSPACETEMP     tempdbs1,tmpspace2,tmpspace3,tmpdbspace4 # Default temp dbspaces

# DUMP*:
# The following parameters control the type of diagnostics information which
# is preserved when an unanticipated error condition (assertion failure) occurs
# during OnLine operations.
# For DUMPSHMEM, DUMPGCORE and DUMPCORE 1 means Yes, 0 means No.

DUMPDIR         /tmp            # Preserve diagnostics in this directory
DUMPSHMEM       1               # Dump a copy of shared memory
DUMPGCORE       0               # Dump a core image using 'gcore'
DUMPCORE        0               # Dump a core image (Warning:this aborts OnLine)
DUMPCNT         1               # Number of shared memory or gcore dumps for
# a single user's session

FILLFACTOR      90              # Fill factor for building indexes

# method for OnLine to use when determining current time
USEOSTIME       0               # 0: use internal time(fast), 1: get time from OS(slow)

# Parallel Database Queries (pdq)
MAX_PDQPRIORITY 100             # Maximum allowed pdqpriority
DS_MAX_QUERIES                  # Maximum number of decision support queries
DS_TOTAL_MEMORY                 # Decision support memory (Kbytes)
DS_MAX_SCANS    1048576         # Maximum number of decision support scans
DATASKIP        off             # List of dbspaces to skip

# OPTCOMPIND
# 0 => Nested loop joins will be preferred (where
#      possible) over sortmerge joins and hash joins.
# 1 => If the transaction isolation mode is not
#      "repeatable read", optimizer behaves as in (2)
#      below.  Otherwise it behaves as in (0) above.
# 2 => Use costs regardless of the transaction isolation
#      mode.  Nested loop joins are not necessarily
#      preferred.  Optimizer bases its decision purely
#      on costs.
```

```
OPTCOMPIND      2               # To hint the optimizer

ONDBSPACEDOWN   2               # Dbspace down option: 0 = CONTINUE, 1 = ABORT, 2 = WAIT
LBU_PRESERVE    0               # Preserve last log for log backup
OPCACHEMAX      0               # Maximum optical cache size (Kbytes)

# HETERO_COMMIT (Gateway participation in distributed transactions)
# 1 => Heterogeneous Commit is enabled
# 0 (or any other value) => Heterogeneous Commit is disabled
HETERO_COMMIT   0

SBSPACENAME                     # Default smartblob space name

JVPON           0               # Boot the Java vp (Yes = 1, No = 0 or anything but 1)

BLOCKTIMEOUT    3600            # Default timeout for system block
SYSALARMPROGRAM /usr/local/Informix/etc/evidence.sh # System Alarm program path
```

E Rankings of Approaches and Their Performance Measure Plots

Table 7: Overall ranking of different schemas and clusterings for all table sizes for individual spatial queries. For each query type, the overall rankings are computed by running the Wilcoxon Run Sum Method for the rankings per table size, which are computed by Fisher's method.

Spatial Window

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2		BDR	BRD	SRD
3			BDR	GDR
4				GRD
5				SDR
6				BDR

Spatial Or–Window

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	GDR GRD SDR	SRD
2		BDR	SRD	BRD
3			BRD	BDR
4			BDR	GDR GRD
5				SDR

Spatial Self–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	SRD	GRD	SDR	BRD
2	SDR	SDR	SRD	BDR SRD
3	GDR	GDR	GDR	GDR SDR
4	GRD	SRD	GRD	GRD
5	BRD	BDR BRD	BRD	
6	BDR		BDR	

Table 8: Overall ranking of different schemas and clusterings for all table sizes for spatial chain–join queries (cont. from Table 7).

Spatial 2–Chain–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2		BDR BRD	BRD	SRD
3			BDR	GRD
4				GDR
5				BDR
6				SDR

Spatial 3—Chain–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	BRD GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BDR	BRD	BRD	SRD
3		BDR	BDR	GDR
4				GRD
5				BDR
6				SDR

Spatial 4–Chain–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BRD	BRD	BRD	SRD
3	BDR	BDR	BDR	GDR
4				GRD
5				SDR
6				BDR

Spatial 5–Chain–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BRD	BRD	BRD	GRD
3	BDR	BDR	BDR	SRD
4				GDR
5				SDR
6				BDR

Table 9: Overall ranking of different schemas and clusterings for all table sizes for spatial star–join queries (cont. from Table 8).

Spatial 3–Star–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	BRD GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BDR	BRD	BRD	SRD
3		BDR	BDR	BDR
4				GDR
5				GRD
6				SDR

Spatial 4–Star–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BRD	BRD	BRD	BDR
3	BDR	BDR	BDR	GDR SRD
4				GRD
5				SDR

Spatial 5–Star–Join

Rank	Client Elapsed Time	Server CPU time	Buffer Reads	Page Reads
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD
2	BRD	BRD	BRD	GDR GRD
3	BDR	BDR	BDR	BDR
4				SRD
5				SDR

Table 10: Spatial Window queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2							BDR

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD
2		BDR	BDR BRD	BRD	BRD	BRD	BRD
3		BRD		BDR	BDR	BDR	BDR

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BRD	BDR GDR GRD SDR SRD	GDR GRD SDR SRD	BRD	BRD	BRD	BRD
2	BDR SDR SRD	BRD	BDR BRD	BDR SRD	GDR SRD	SRD	SRD
3	GRD			GDR GRD SDR	BDR GRD SDR	GDR GRD	GDR GRD
4	GDR					BDR SDR	BDR SDR

Table 11: Spatial Or–Window queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2					BDR	BDR	BDR

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD
2	BDR SRD	BDR	BDR BRD	BRD	BRD	BRD	BRD
3	BRD	BRD		BDR	BDR	BDR	BDR

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BRD	BDR SRD	GRD SRD	BRD	BRD	BRD	BRD
2	BDR	GDR SDR	GDR SDR	BDR SRD	SRD	SRD	SRD
3	SDR SRD	GRD	BDR	GDR SDR	GDR GRD	GDR GRD	GDR GRD
4	GRD	BRD	BRD	GRD	BDR SDR	BDR SDR	BDR SDR
5	GDR						

Table 12: Spatial Self–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR SDR SRD	GDR SDR SRD	SRD	SRD	SRD	SRD
2	BDR BRD	GRD	GRD	GDR SDR	SDR	SDR	SDR
3		BDR BRD	BRD	GRD	GDR	GDR	GDR
4			BDR	BRD	GRD	GRD	GRD
5				BDR	BRD	BRD	BRD
6					BDR	BDR	BDR

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GRD	BDR	GRD	GDR GRD	GRD SDR	GRD SDR
2	BDR BRD	SDR	GRD	SDR	SDR SRD	GDR	GDR
3		SRD	SDR	GDR	BRD	SRD	SRD
4		GDR	GDR SRD	SRD	BDR	BRD	BRD
5		BRD	BRD	BRD		BDR	BDR
6		BDR		BDR			

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR	SDR	SDR	SDR	SRD	SRD	SRD
2	SDR SRD	GDR	SRD	SRD	SDR	SDR	SDR
3	GRD	SRD	GDR	GDR	GDR	GDR	GDR
4	BRD	GRD	GRD	GRD	GRD	GRD	GRD
5	BDR	BRD	BRD	BRD	BRD	BRD	BRD
6		BDR	BDR	BDR	BDR	BDR	BDR

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BRD	BRD	BDR BRD	BDR BRD	BRD	BRD
2		GDR	SRD	SRD	SRD	BDR	BDR
3		SDR	GDR	GDR	SDR	SRD	SRD
4		SRD	SDR	SDR	GDR	SDR	SDR
5		BDR	GRD	GRD	GRD	GDR	GDR
6		GRD	BDR			GRD	GRD

Table 13: Spatial 2–Chain–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	BDR	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	GDR GRD SDR SRD	BDR	BRD	BRD	BRD	BDR
3		BRD	BRD	BDR	BDR	BDR	

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BRD
3				BDR	BDR	BDR	BDR

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD	BRD GDR GRD SDR SRD	BRD GRD	BRD	BRD	BRD	BRD SRD
2	GDR GRD SDR	BDR	GDR SDR SRD	BDR	BDR SRD	SRD	GDR GRD
3	SRD		BDR	SRD	GDR GRD	GDR GRD	SDR
4				GDR GRD SDR	SDR	BDR	BDR
5						SDR	

Table 14: Spatial 3—Chain–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2		BDR				BDR	BDR

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	BDR	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	GDR GRD SDR SRD	BDR BRD	BRD	BRD	BRD	BDR
3		BRD		BDR	BDR	BDR	

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BDR
3				BDR	BDR	BDR	

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD SRD
2	GDR SDR	BDR	BDR	BDR	BDR SRD	SRD	GDR GRD
3	GRD			SRD	GDR GRD	GDR GRD	BDR SDR
4	SRD			GDR GRD SDR	SDR	BDR	
5						SDR	

Table 15: Spatial 4–Chain–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2		BDR BRD		BDR	BDR	BDR	BDR

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BDR
3				BDR	BDR	BDR	

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR SDR	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD GDR GRD SRD
2	GRD	BDR	BDR	BDR	BDR SRD	GDR GRD SRD	SDR
3	SRD			SDR SRD	GDR GRD SDR	BDR SDR	BDR
4				GDR GRD			

Table 16: Spatial 5–Chain–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BDR	BDR
3				BDR	BDR		

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD GDR GRD SDR SRD
2	SRD		BDR	BDR	BDR GRD SRD	GDR GRD SRD	BDR
3				GRD SDR SRD	GDR SDR	BDR SDR	
4				GDR			

Table 17: Spatial 3–Star–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2						BDR	BDR

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BDR
3				BDR	BDR	BDR	

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BDR
3				BDR	BDR	BDR	

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD	BDR BRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD SRD
2	GDR SDR	GDR SDR	BDR	BDR	BDR SRD	SRD	GDR GRD
3	GRD	GRD		SRD	GDR GRD	GDR GRD	SDR
4	SRD	SRD		GDR GRD SDR	SDR	BDR	BDR
5						SDR	

Table 18: Spatial 4–Star–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2				BRD	BDR	BDR	BDR
3				BDR			

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BRD	BDR
3				BDR	BDR	BDR	

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR	BDR BRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD GDR GRD SRD
2	GRD SDR	GDR GRD SDR	BDR	BDR	BDR SRD	SRD	SDR
3	SRD	SRD		GRD SDR SRD	GDR GRD SDR	GDR GRD	BDR
4				GDR		BDR SDR	

Table 19: Spatial 5–Star–Join queries. Ranking of different schemas and clusterings with respect to **client elapsed time, server CPU time, buffer and page reads.** for various number of stars (table sizes), with lower ranks being better. The rankings are computed using Fisher's method.

Client Elapsed Time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Server CPU time

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	BDR BRD GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD		BRD	BDR	BDR	BDR
3				BDR			

Buffer Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	GDR GRD SDR SRD	BRD GDR GRD SDR SRD	BRD GDR GRD SDR SRD
2	BDR BRD	BDR BRD	BDR BRD	BRD	BRD	BDR	BDR
3				BDR	BDR		

Page Reads

Rank	Number of Stars						
	1,000	10,000	20,000	40,000	60,000	100,000	140,000
1	BDR BRD GDR	BDR BRD	BRD GDR GRD SDR SRD	BRD	BRD	BRD	BRD GDR GRD SRD
2	GRD SDR	GDR GRD SDR	BDR	BDR	BDR GRD SRD	SRD	SDR
3	SRD	SRD		GDR GRD SDR SRD	GDR SDR	GDR GRD	BDR
4						BDR SDR	

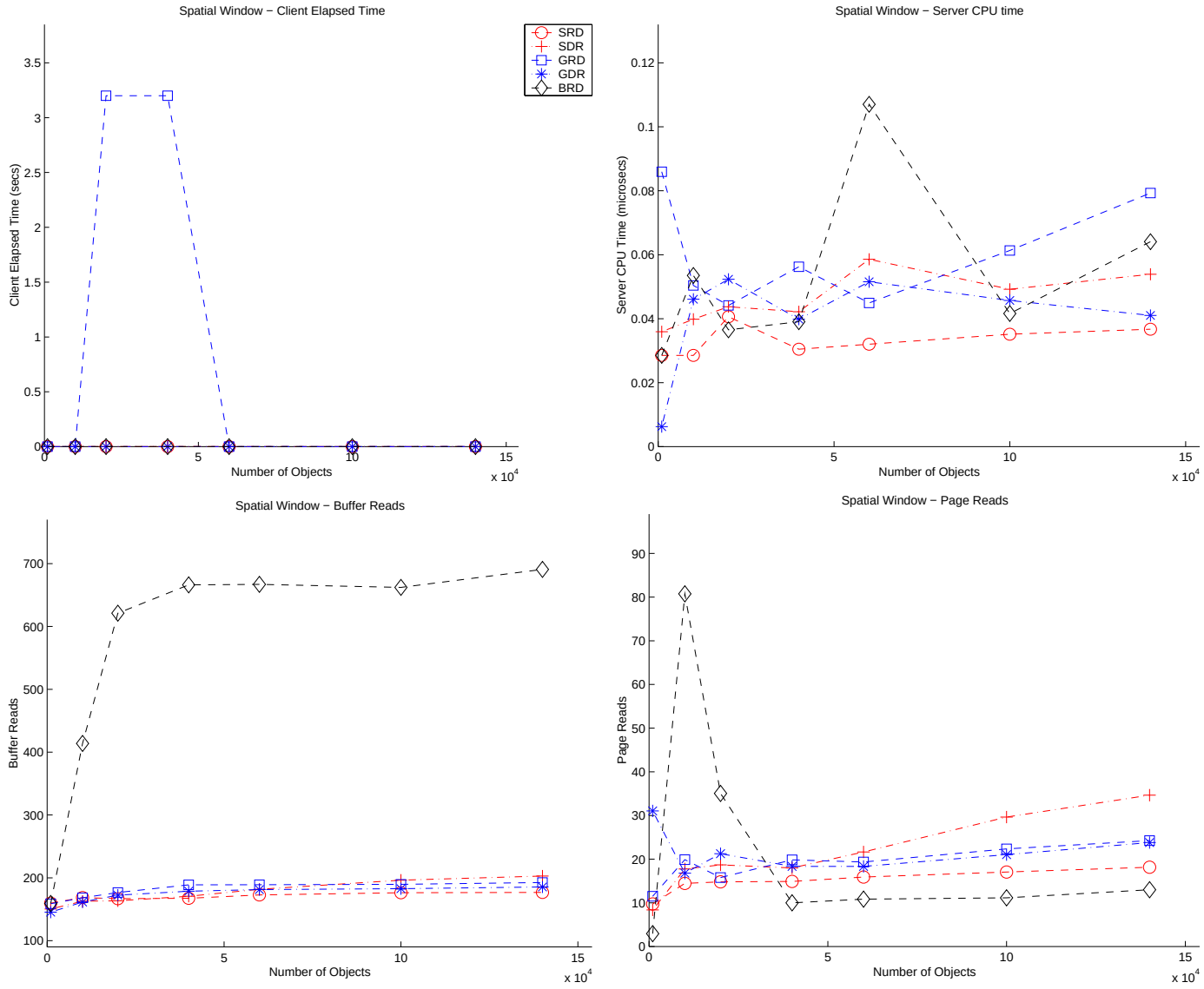


Figure 12: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Window queries and clustering on dec–RA and RA–dec.

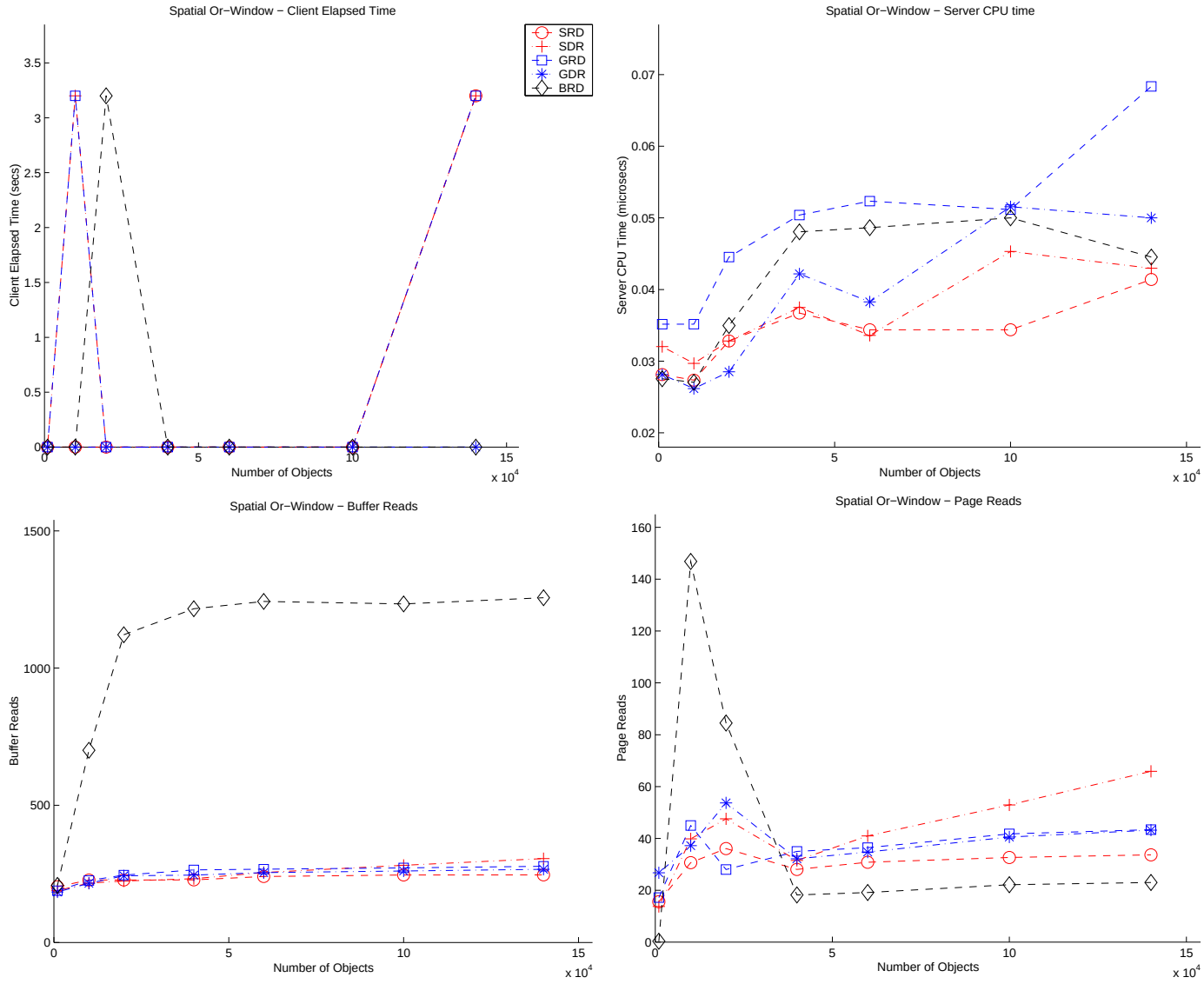


Figure 13: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Or-Window queries and clustering on dec–RA and RA–dec.

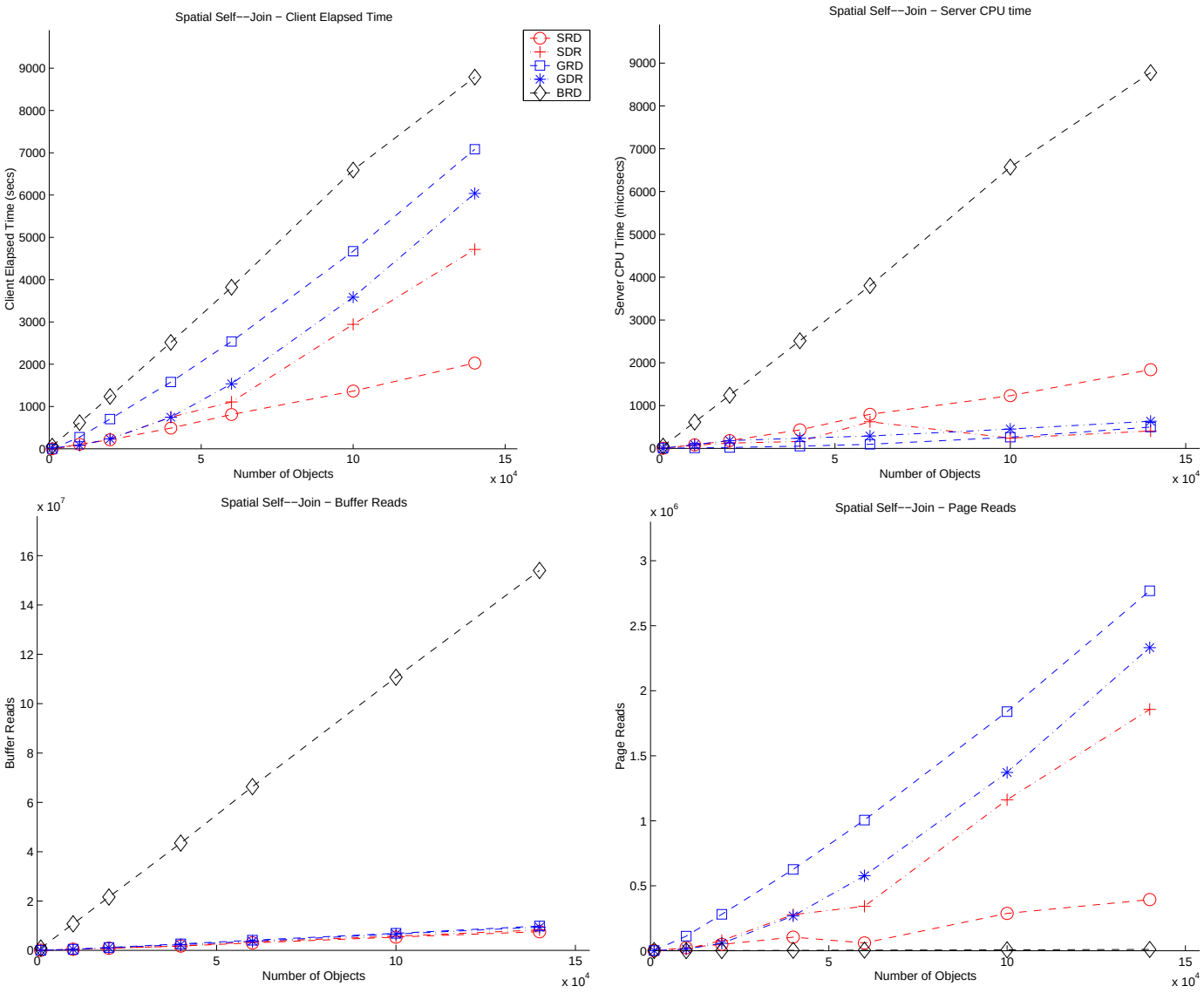


Figure 14: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Self–Join queries and clustering on dec–RA and RA–dec.

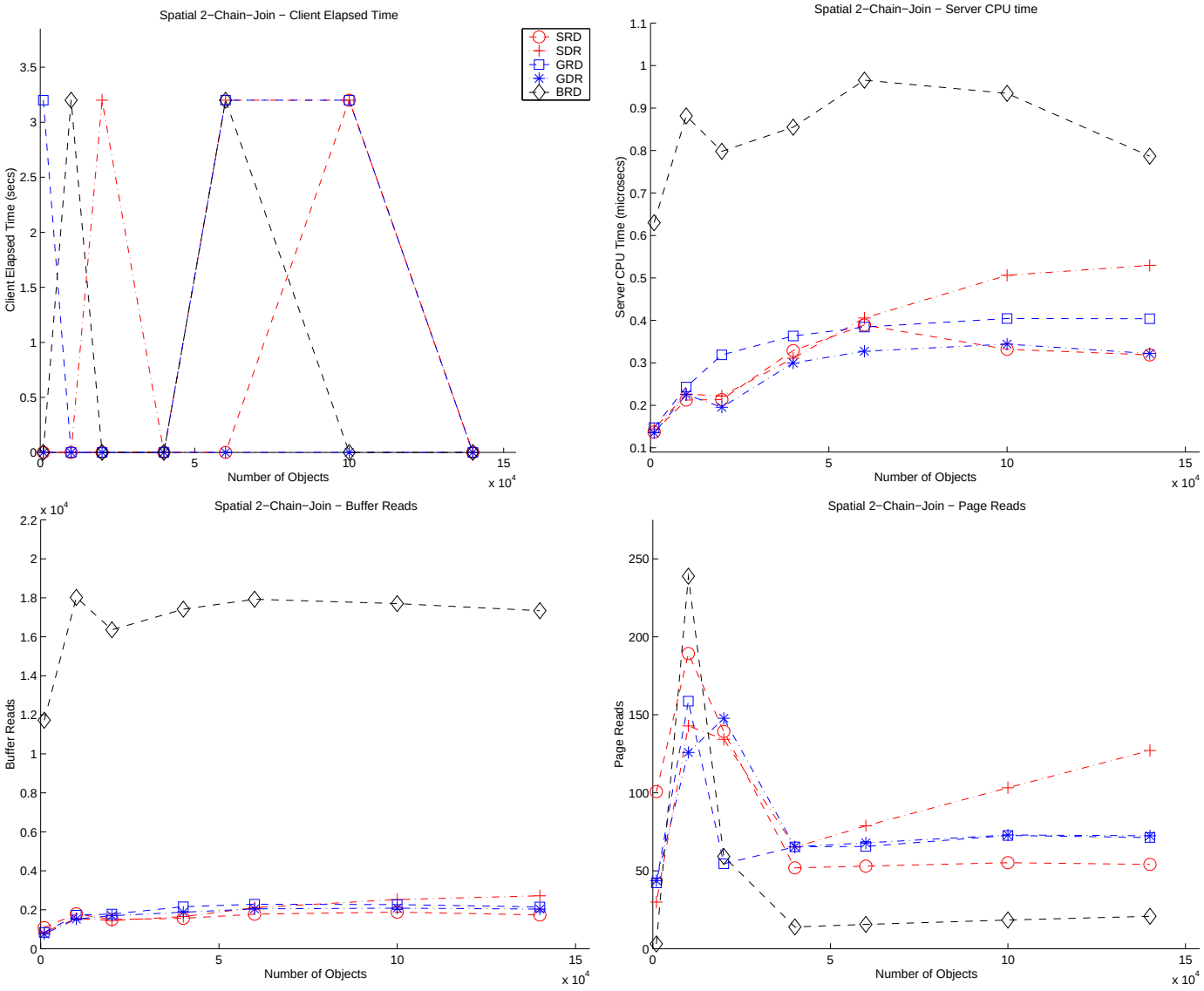


Figure 15: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 2-Chain-Join queries and clustering on dec–RA and RA–dec.

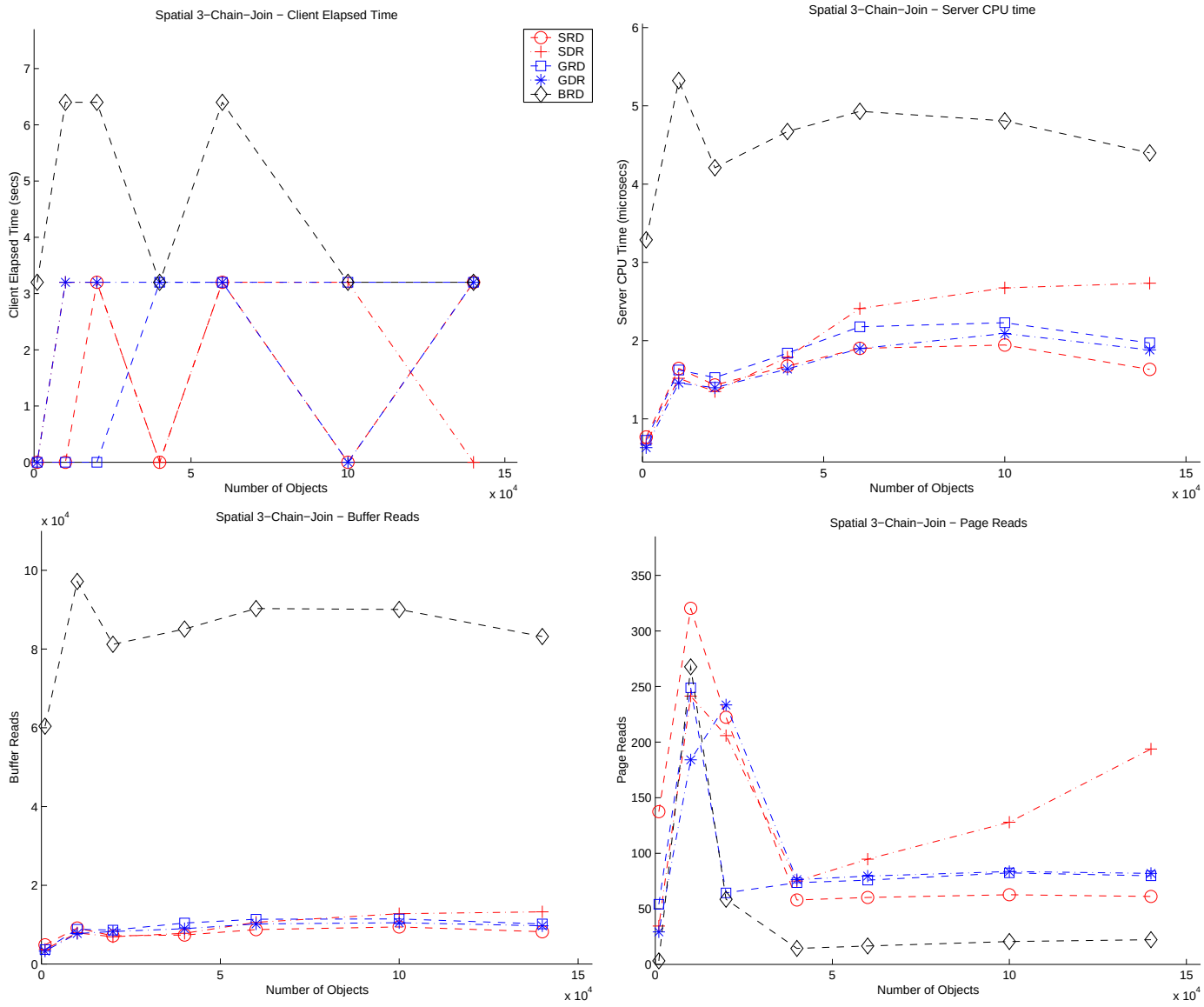


Figure 16: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 3-Chain-Join queries and clustering on dec–RA and RA–dec.

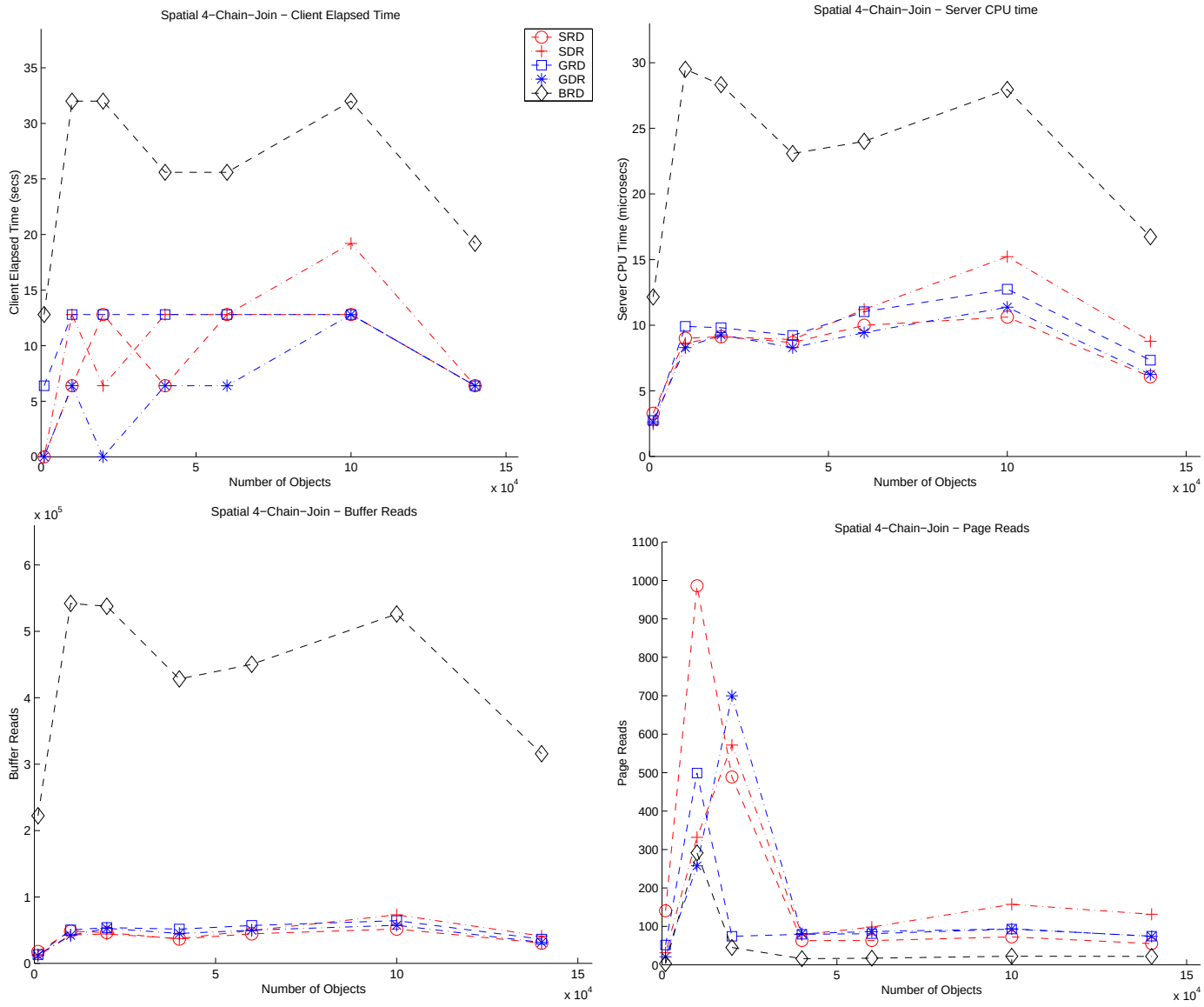


Figure 17: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 4-Chain-Join queries and clustering on dec–RA and RA–dec.

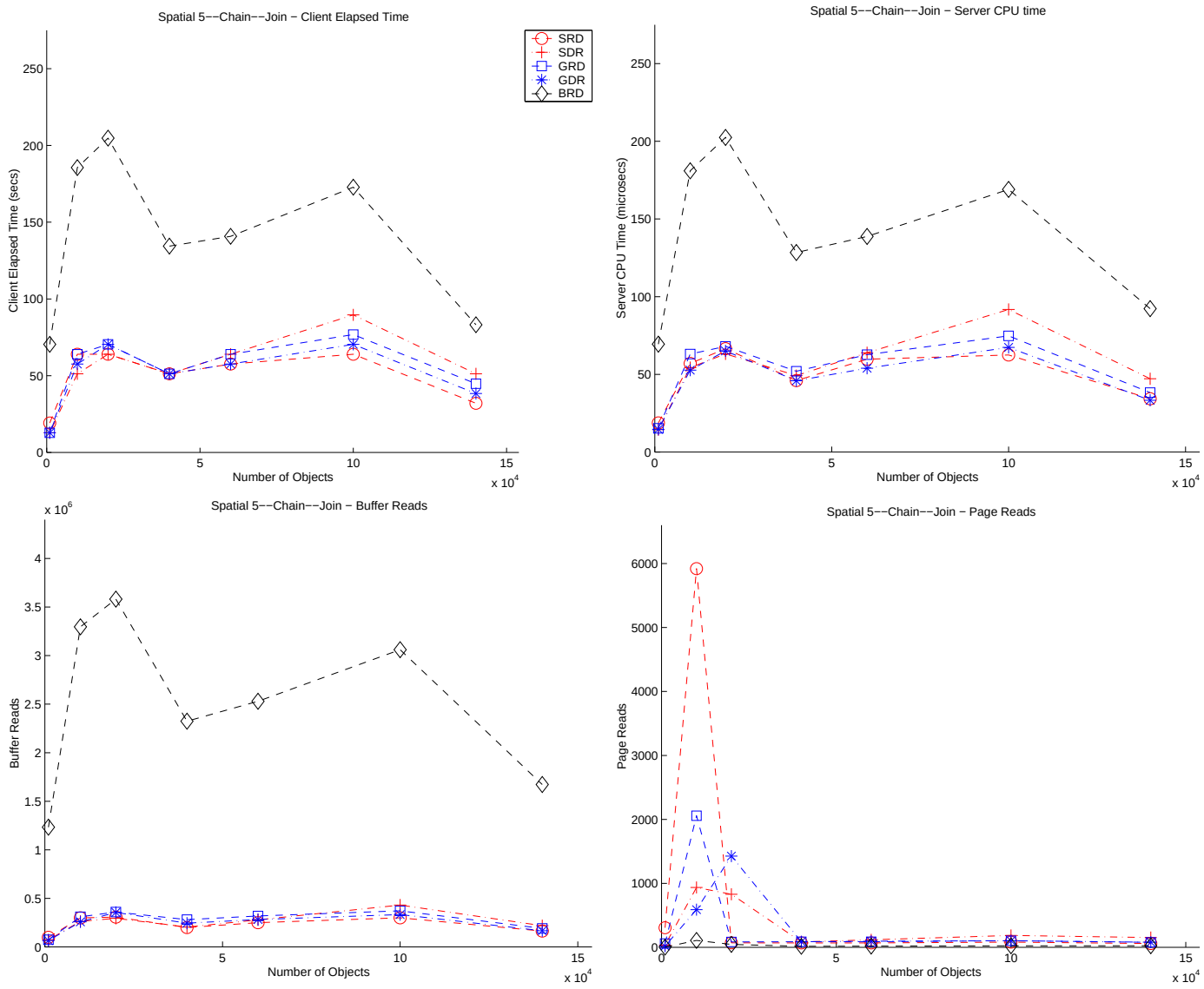


Figure 18: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 5–Chain–Join queries and clustering on dec–RA and RA–dec.

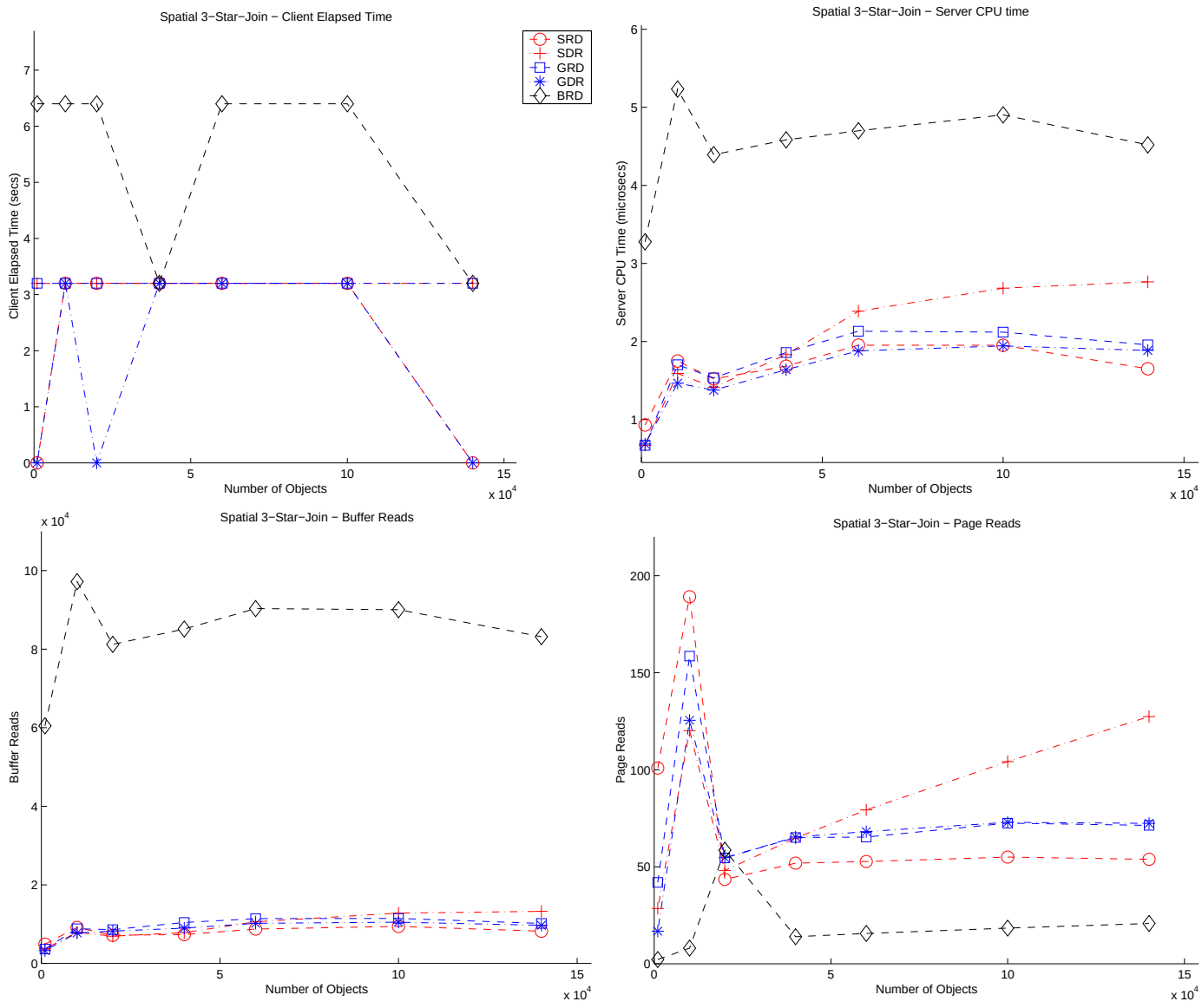


Figure 19: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 3-Star-Join queries and clustering on dec–RA and RA–dec.

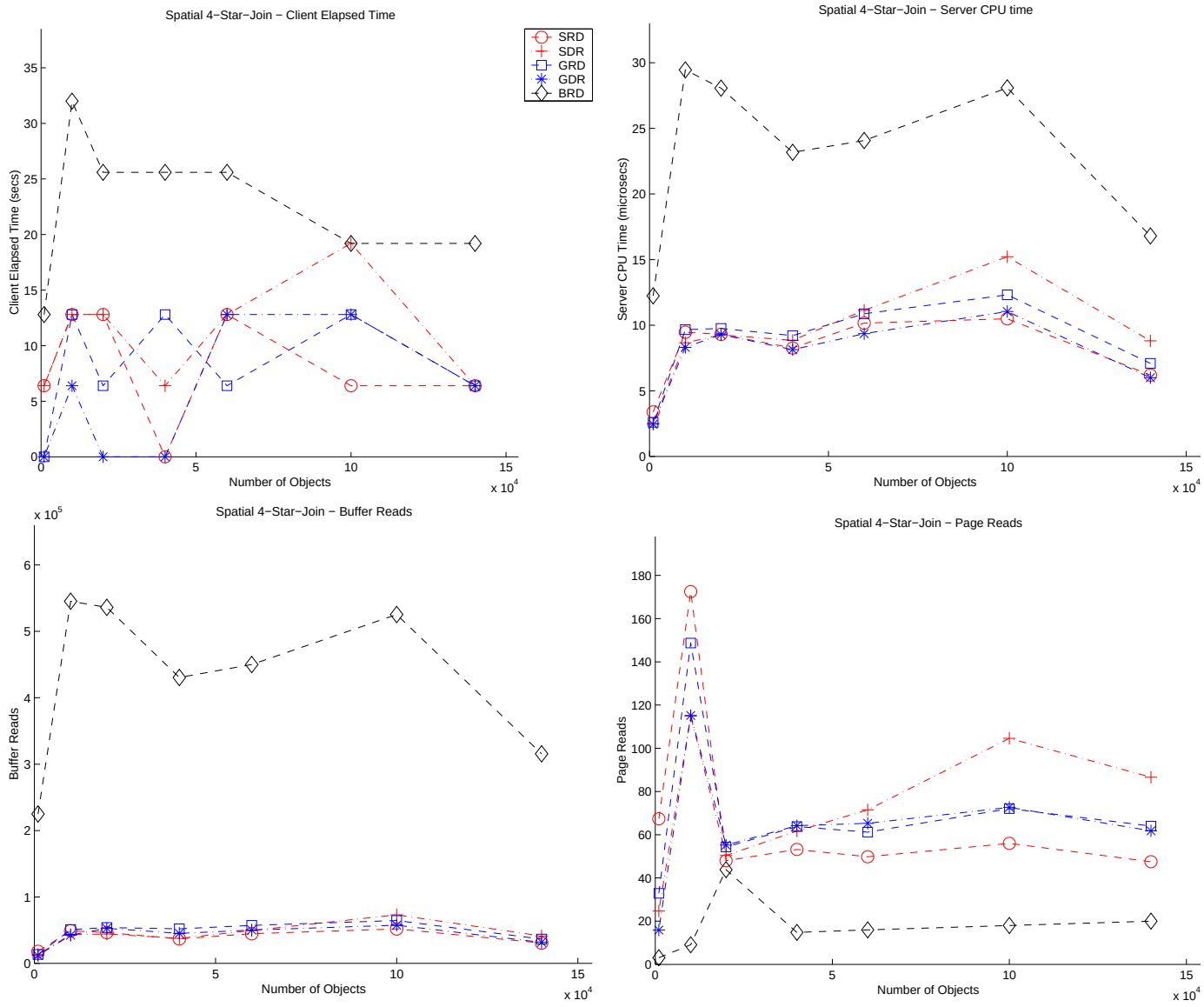


Figure 20: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 4-Star-Join queries and clustering on dec–RA and RA–dec.

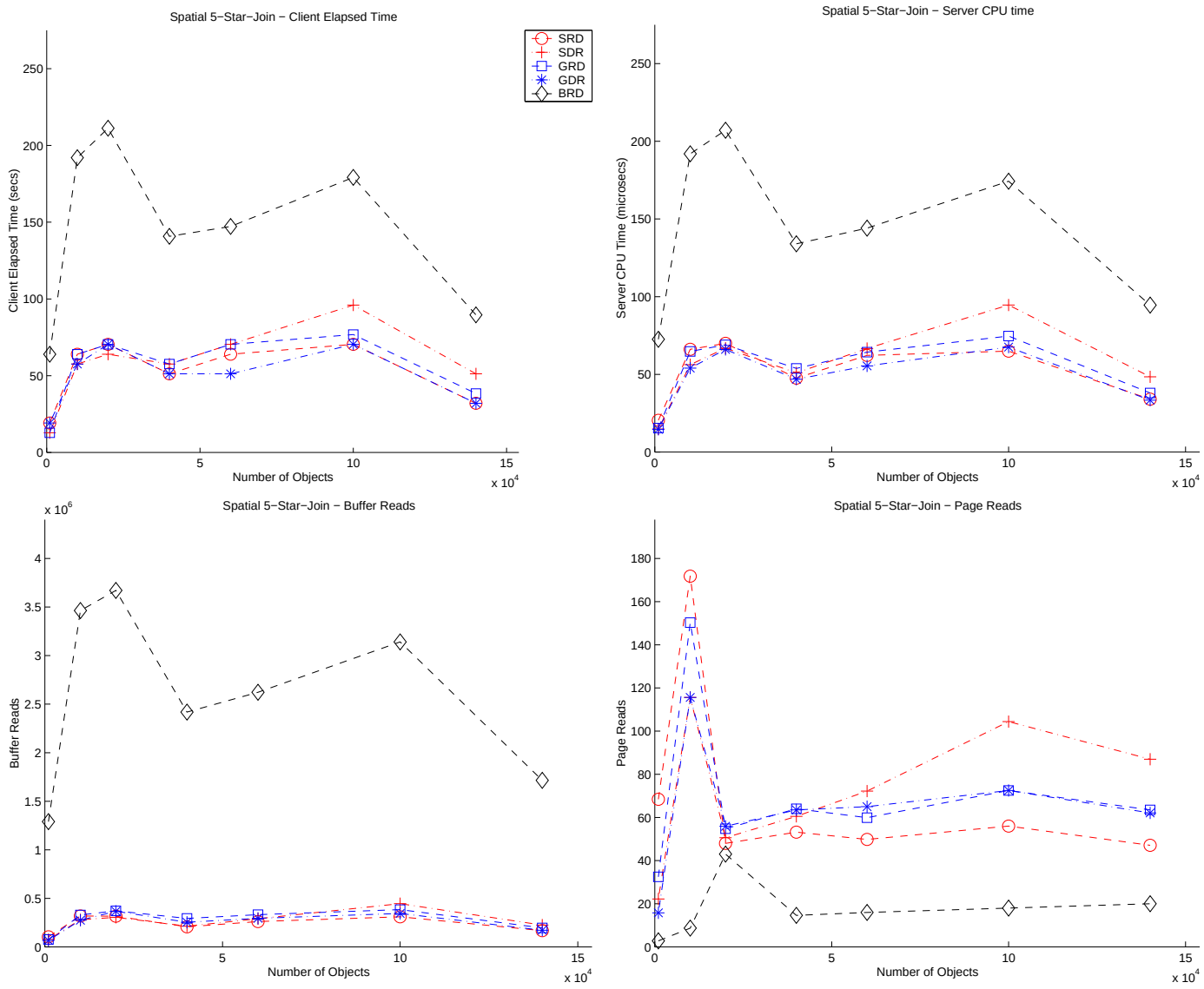


Figure 21: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 5-Star-Join queries and clustering on dec–RA and RA–dec.

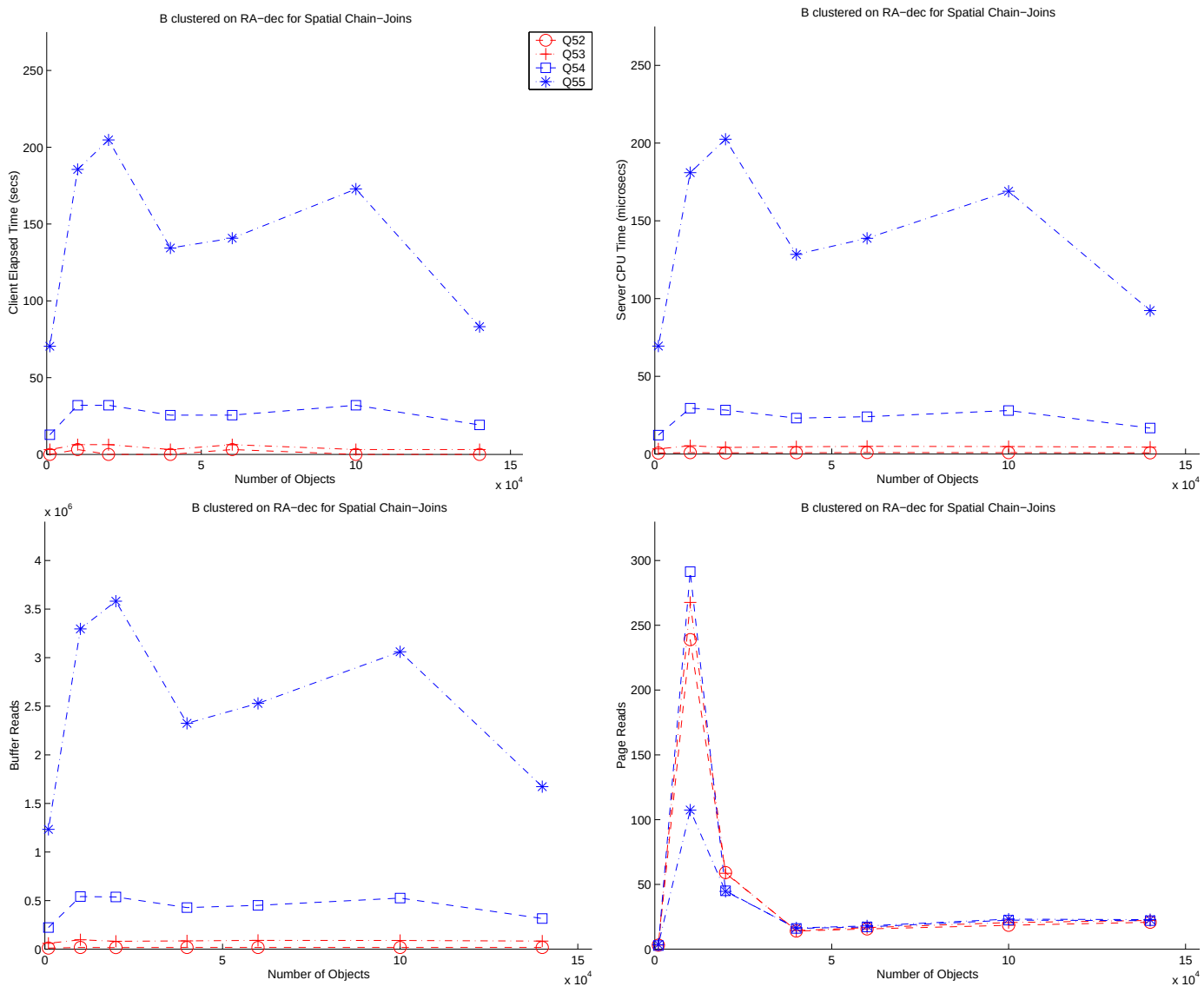


Figure 22: Performance of Relational (**B**) schemas clustered on RA–dec for Spatial Chain–Join queries.

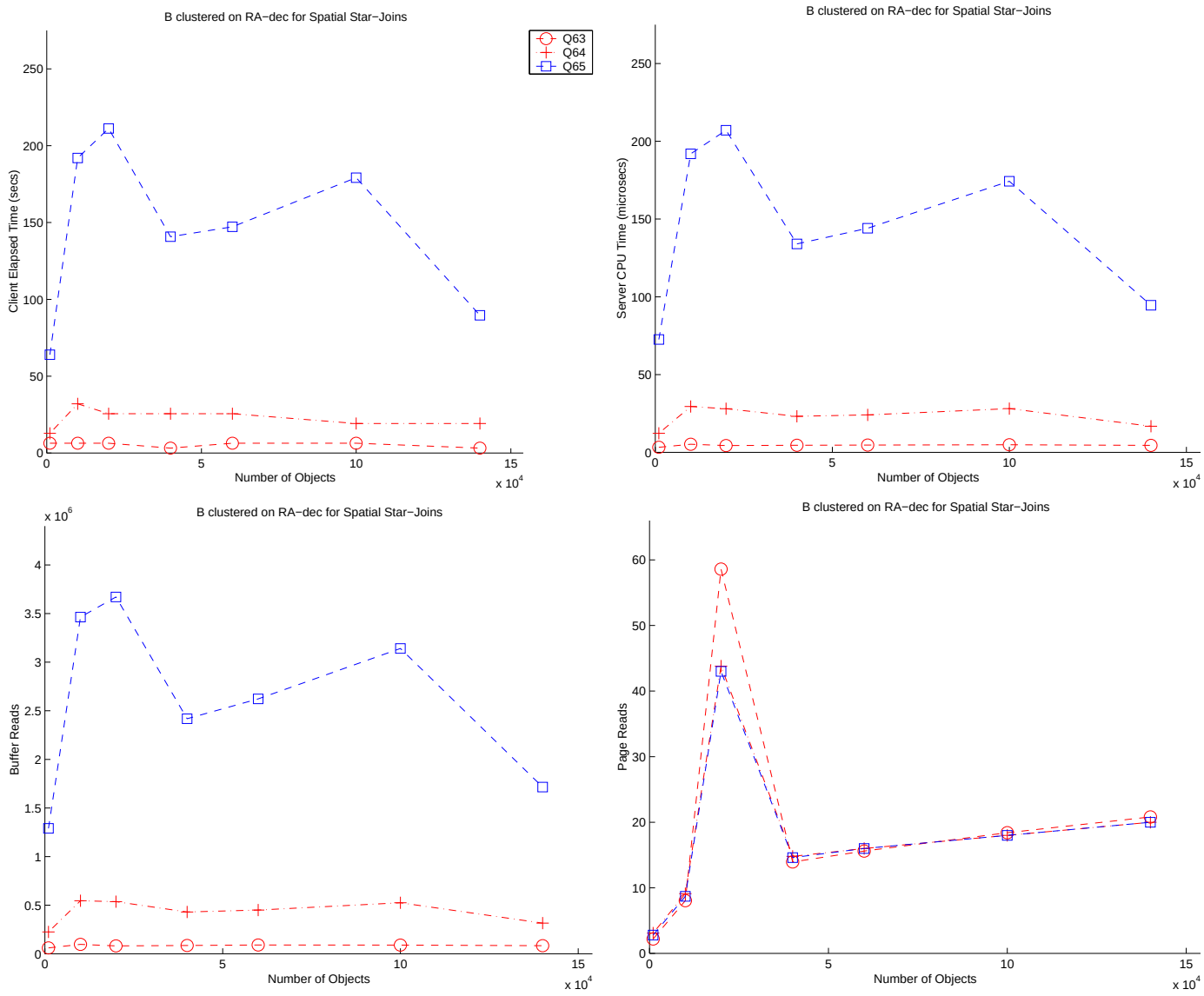


Figure 23: Performance of Relational (**B**) schemas clustered on RA–dec for Spatial Star–Join queries.

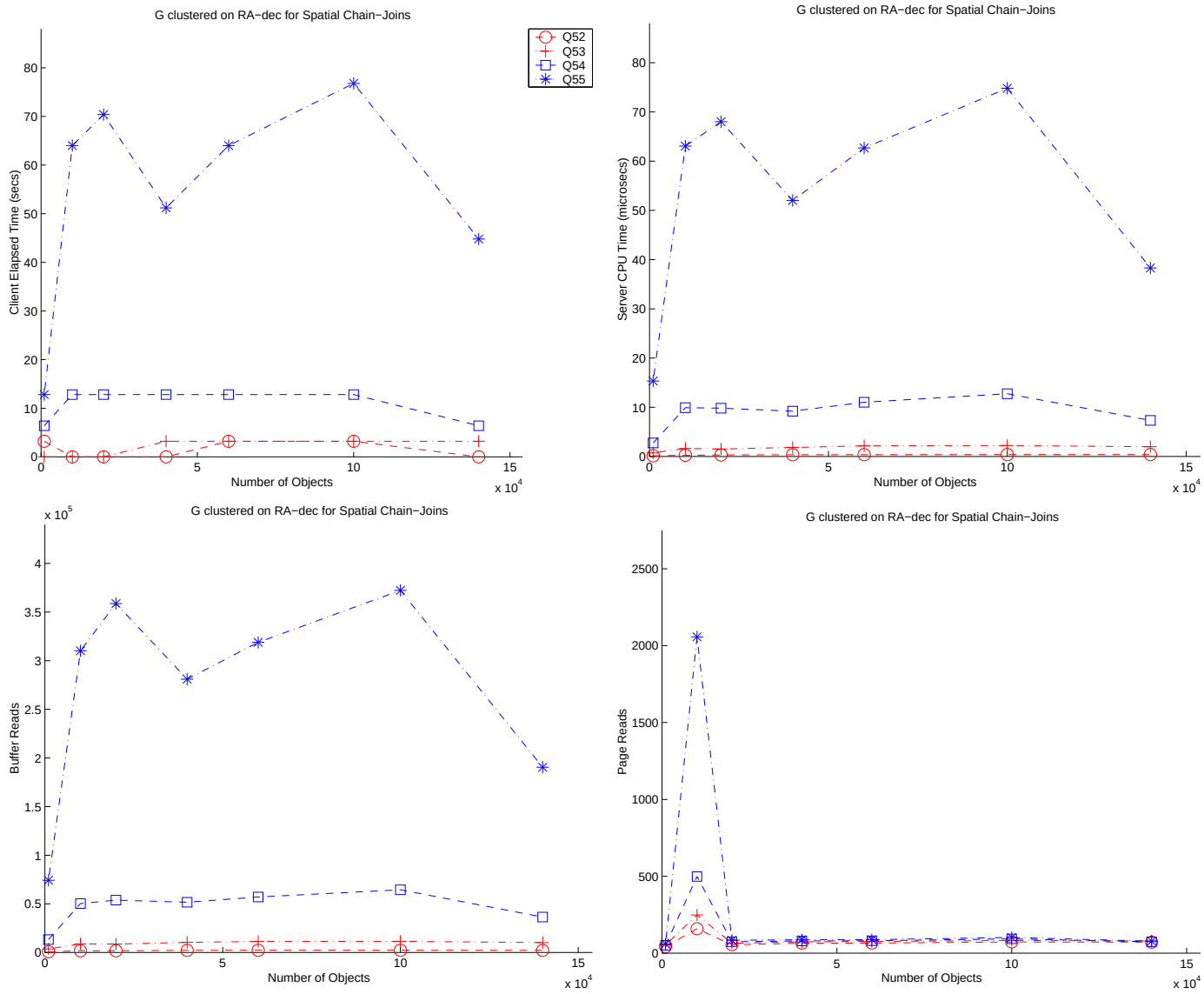


Figure 24: Performance of Geodetic (**G**) schemas clustered on RA–dec for Spatial Chain–Join queries.

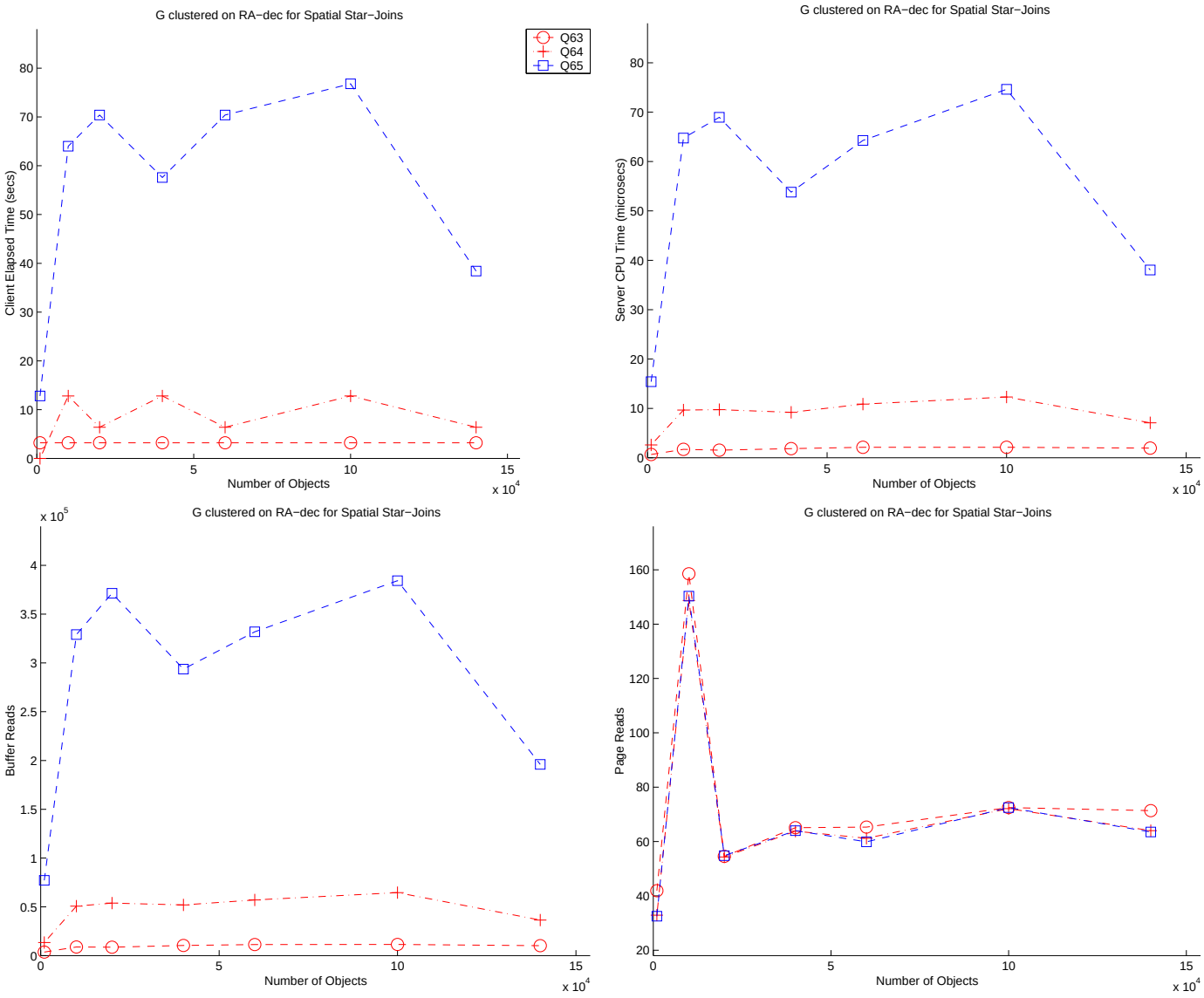


Figure 25: Performance of Geodetic (**G**) schemas clustered on RA–dec for Spatial Star–Join queries.

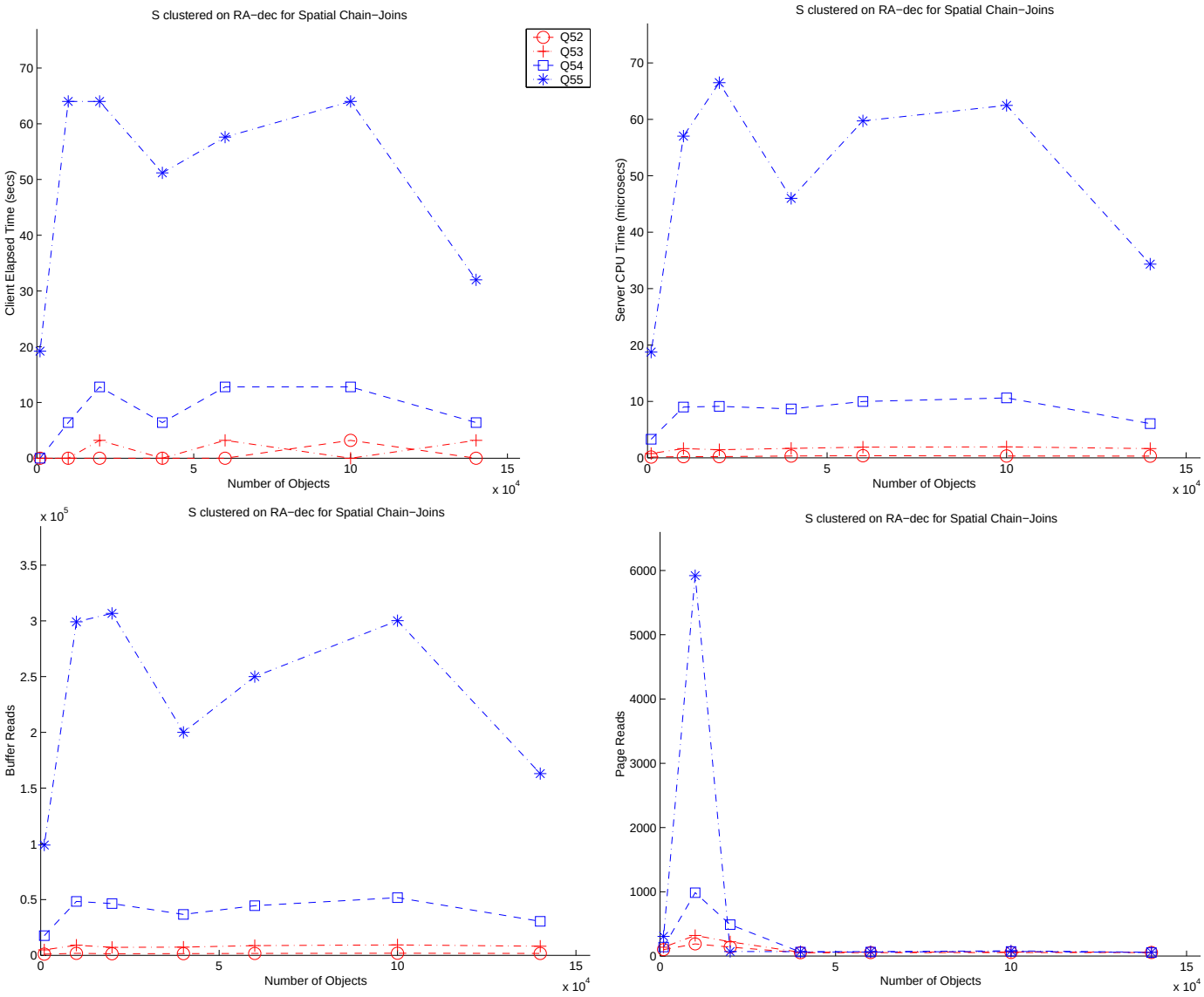


Figure 26: Performance of Shapes2 (**S**) schemas clustered on RA–dec for Spatial Chain–Join queries.

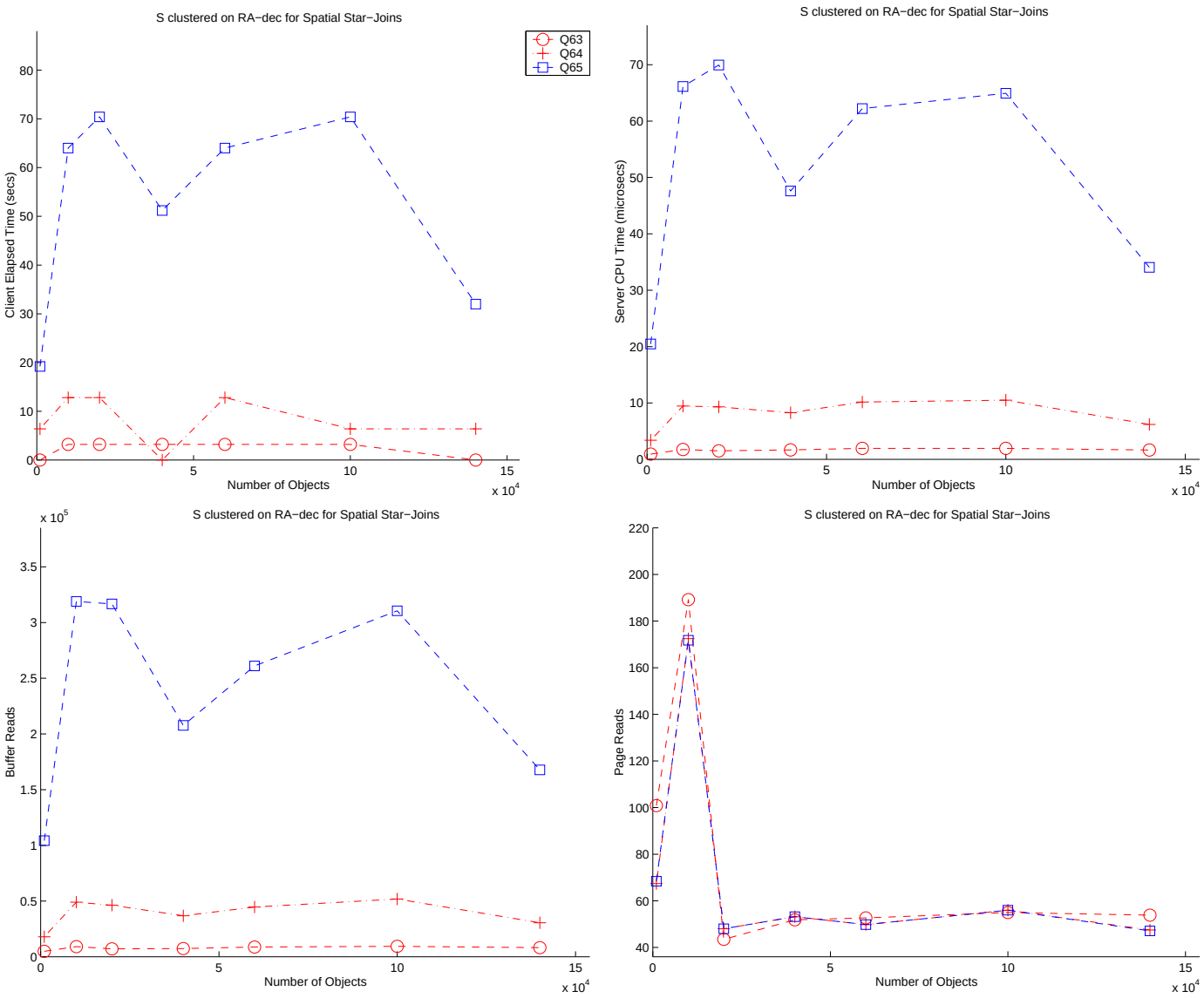


Figure 27: Performance of Shapes2 (**S**) schemas clustered on RA–dec for Spatial Star–Join queries.

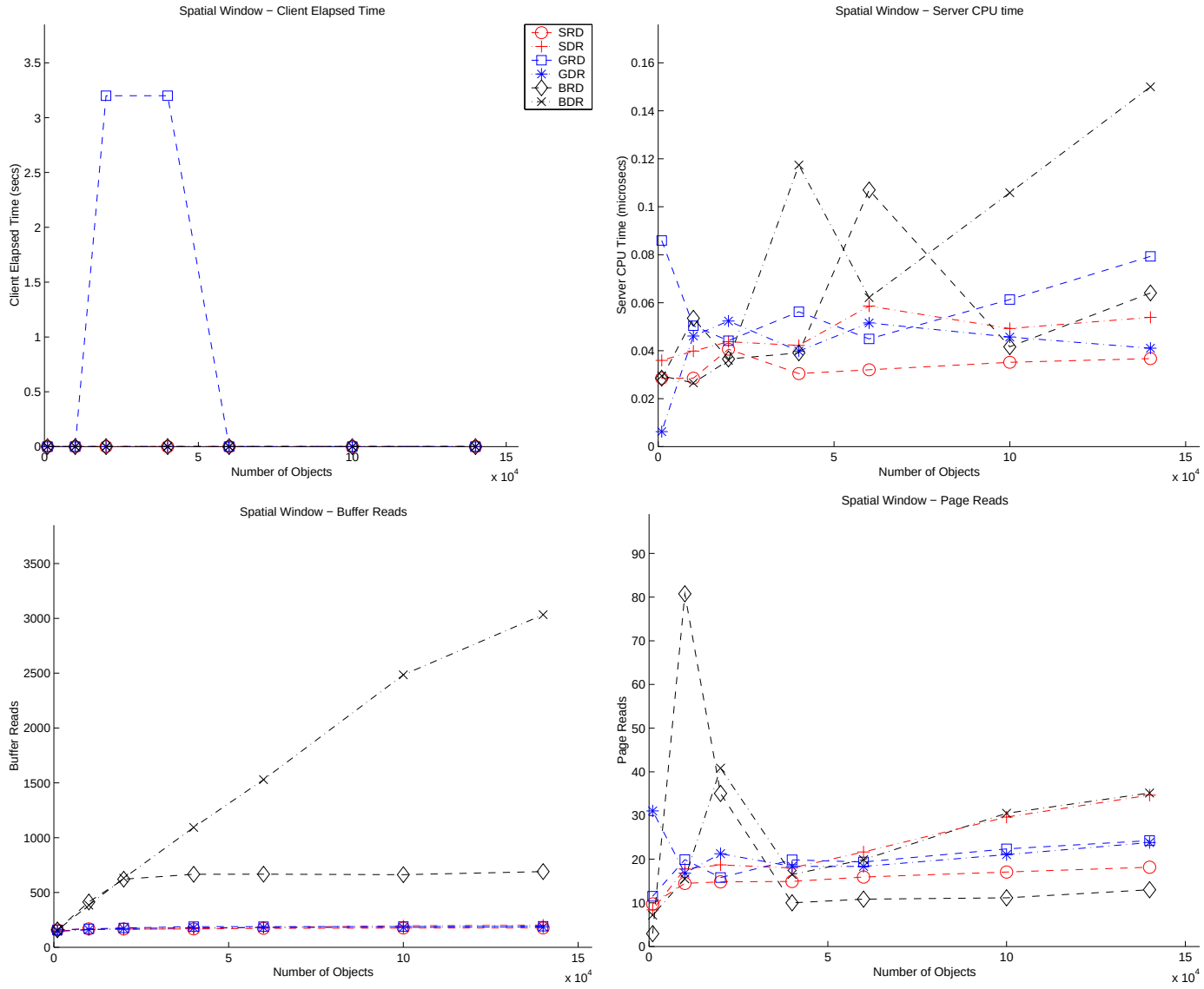


Figure 28: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Window queries and clustering on dec–RA and RA–dec.

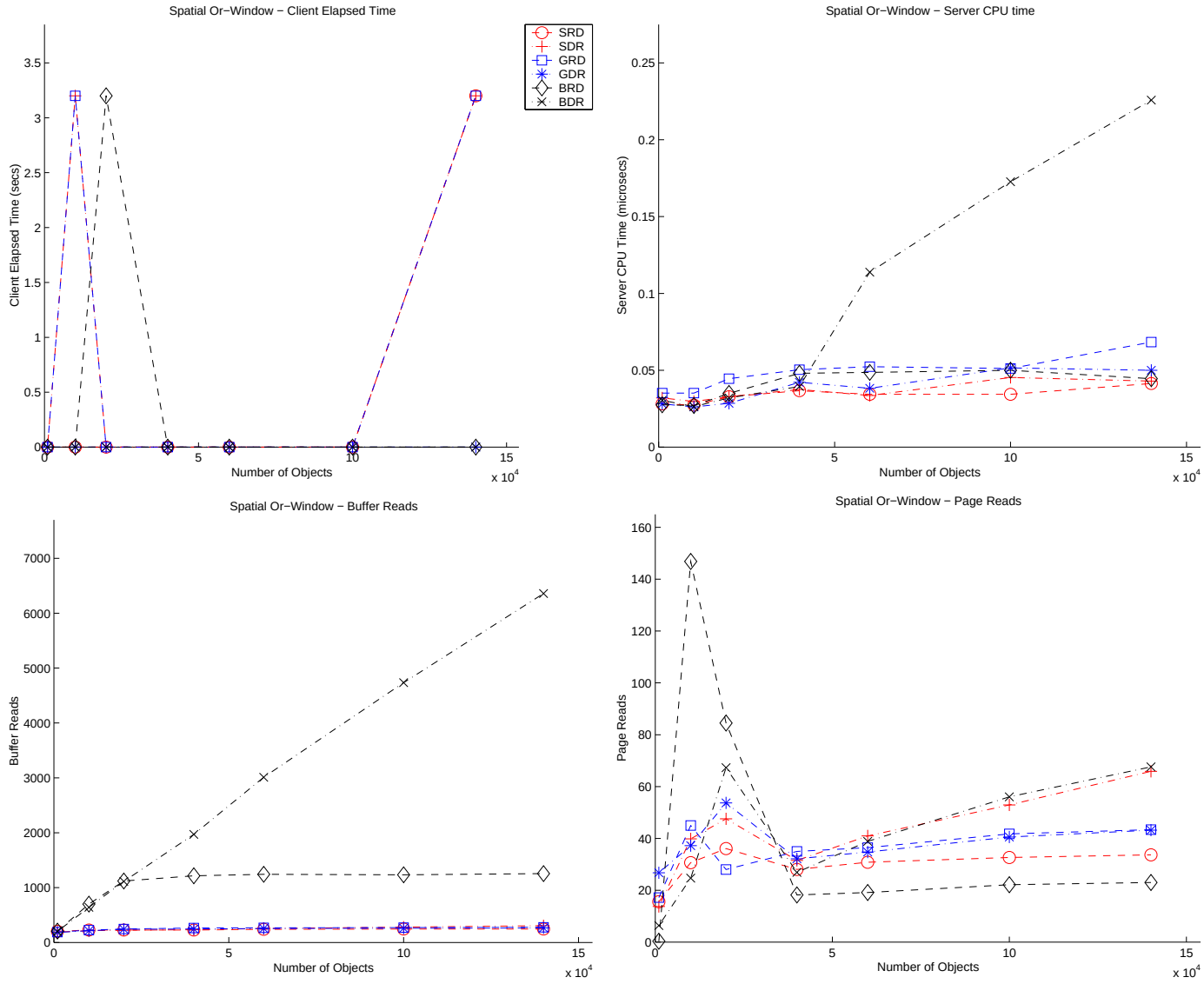


Figure 29: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Or-Window queries and clustering on dec–RA and RA–dec.

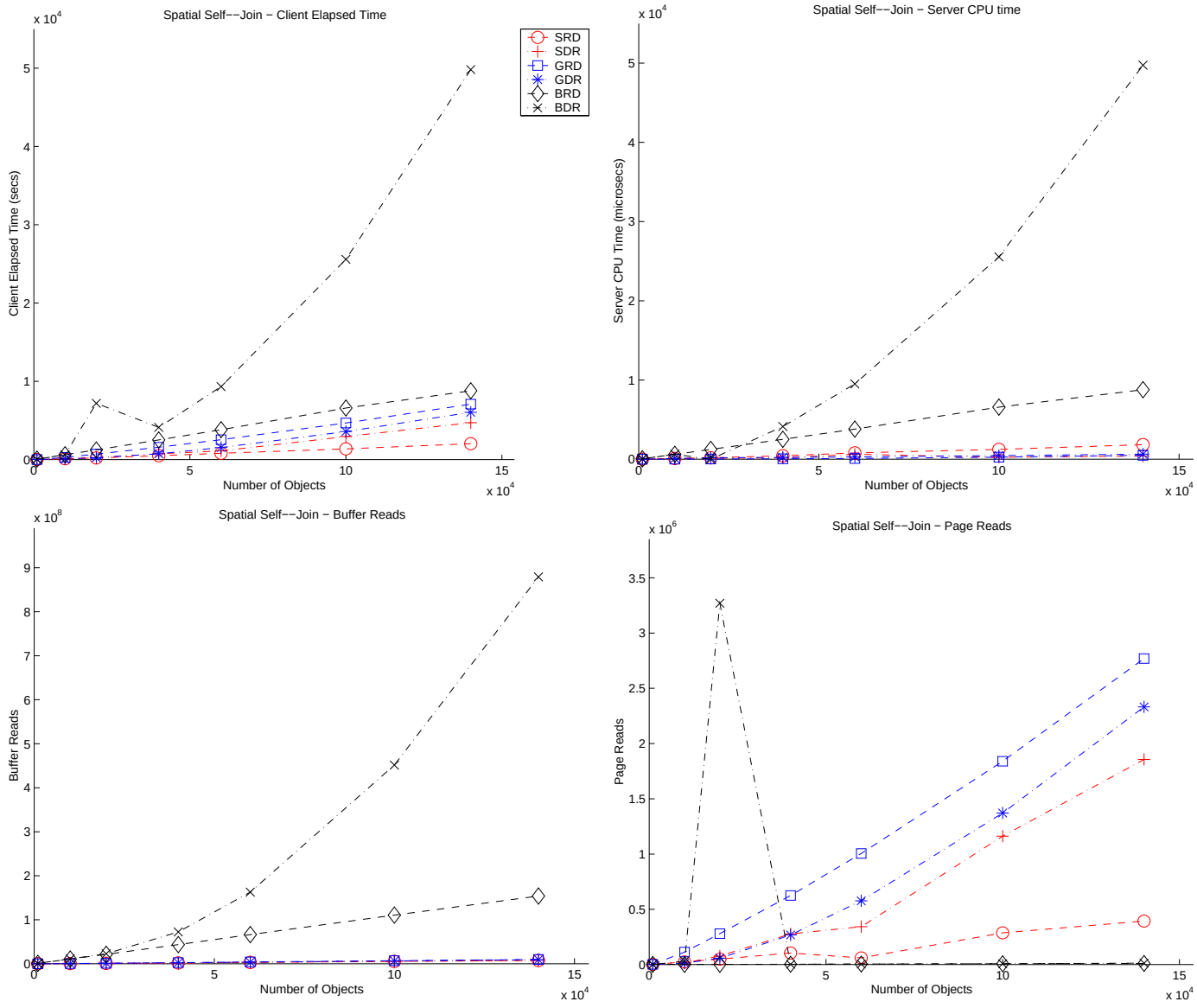


Figure 30: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial Self–Join queries and clustering on dec–RA and RA–dec.

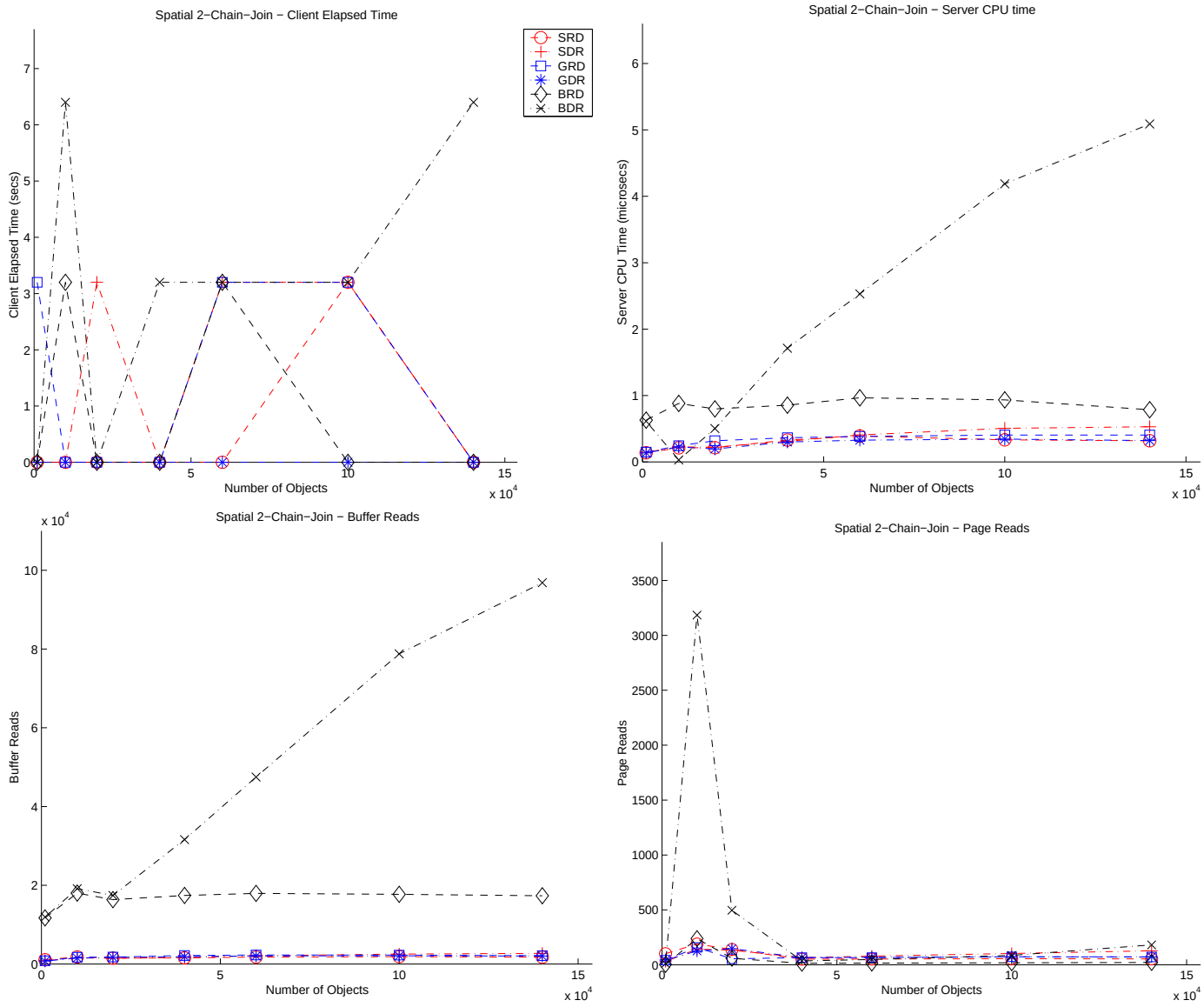


Figure 31: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 2-Chain-Join queries and clustering on dec–RA and RA–dec.

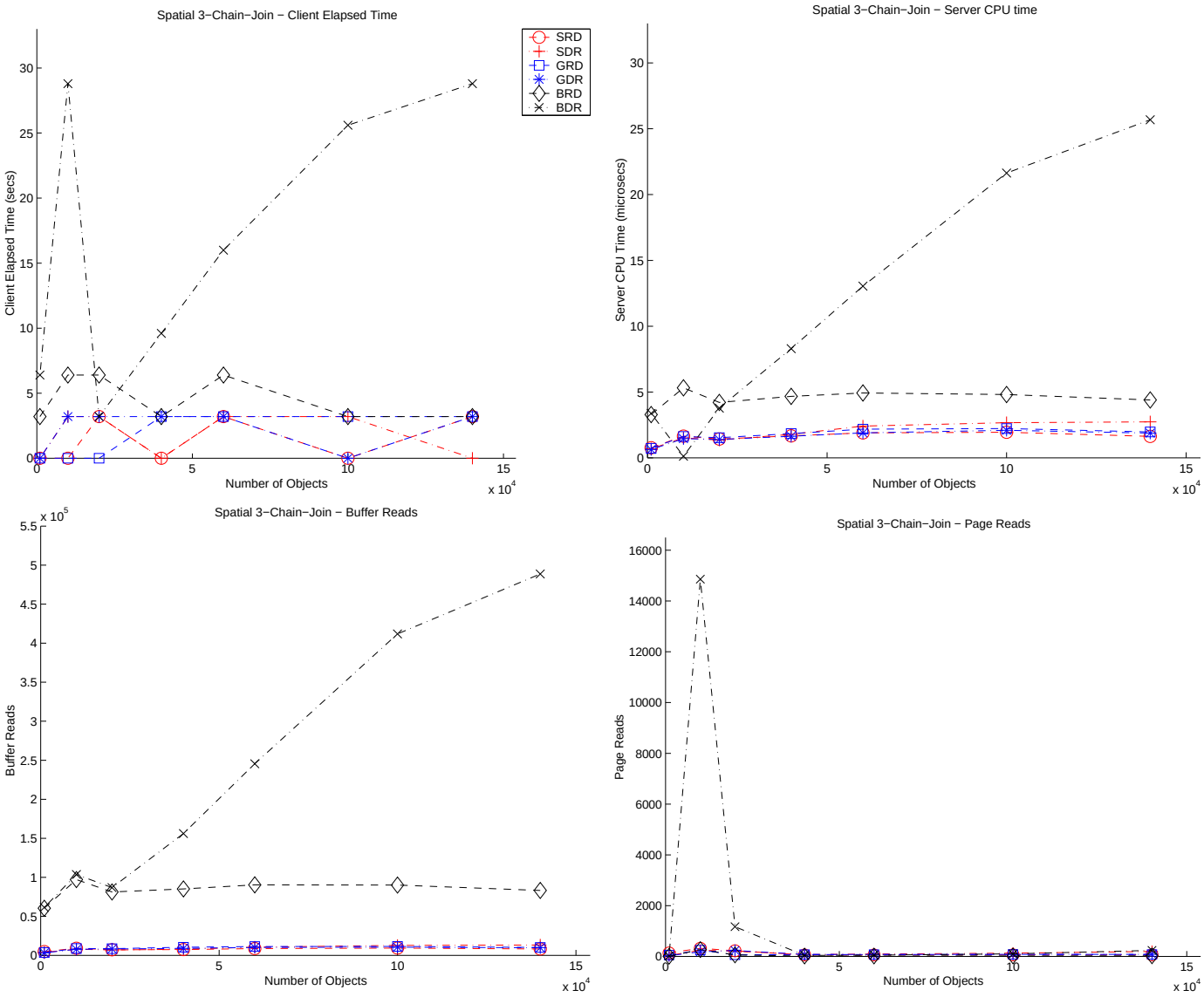


Figure 32: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 3-Chain-Join queries and clustering on dec–RA and RA–dec.

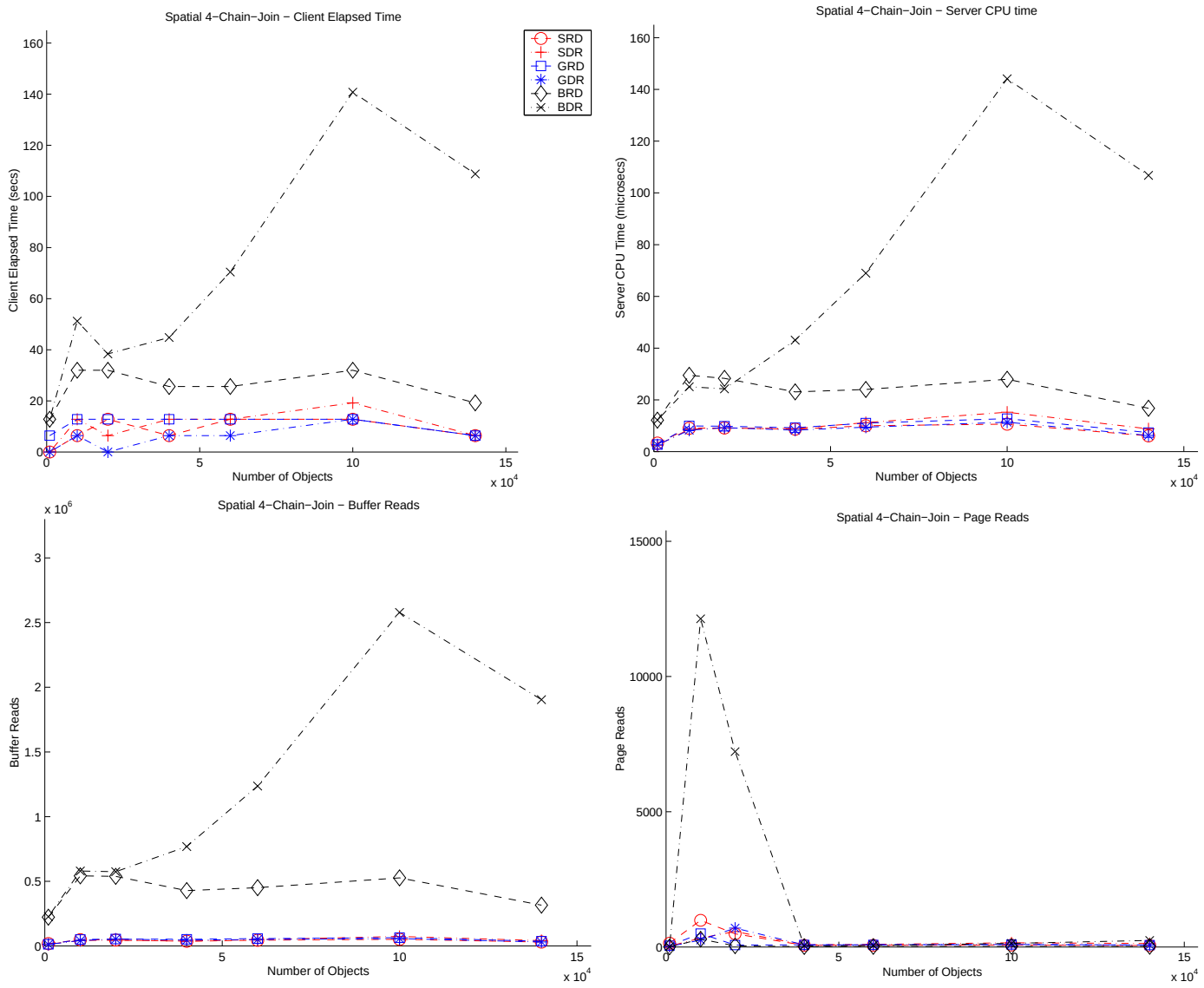


Figure 33: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 4-Chain-Join queries and clustering on dec–RA and RA–dec.

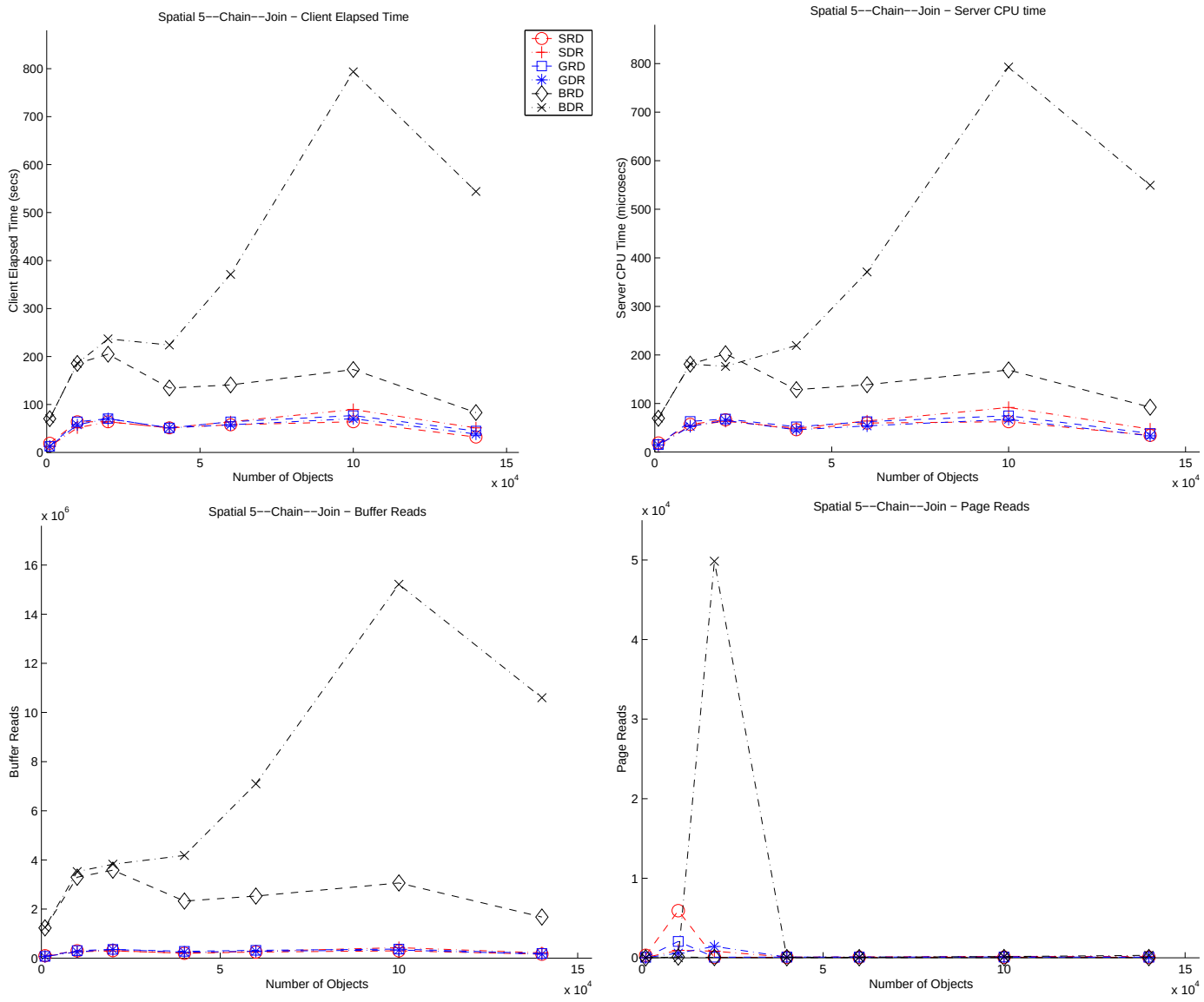


Figure 34: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 5–Chain–Join queries and clustering on dec–RA and RA–dec.

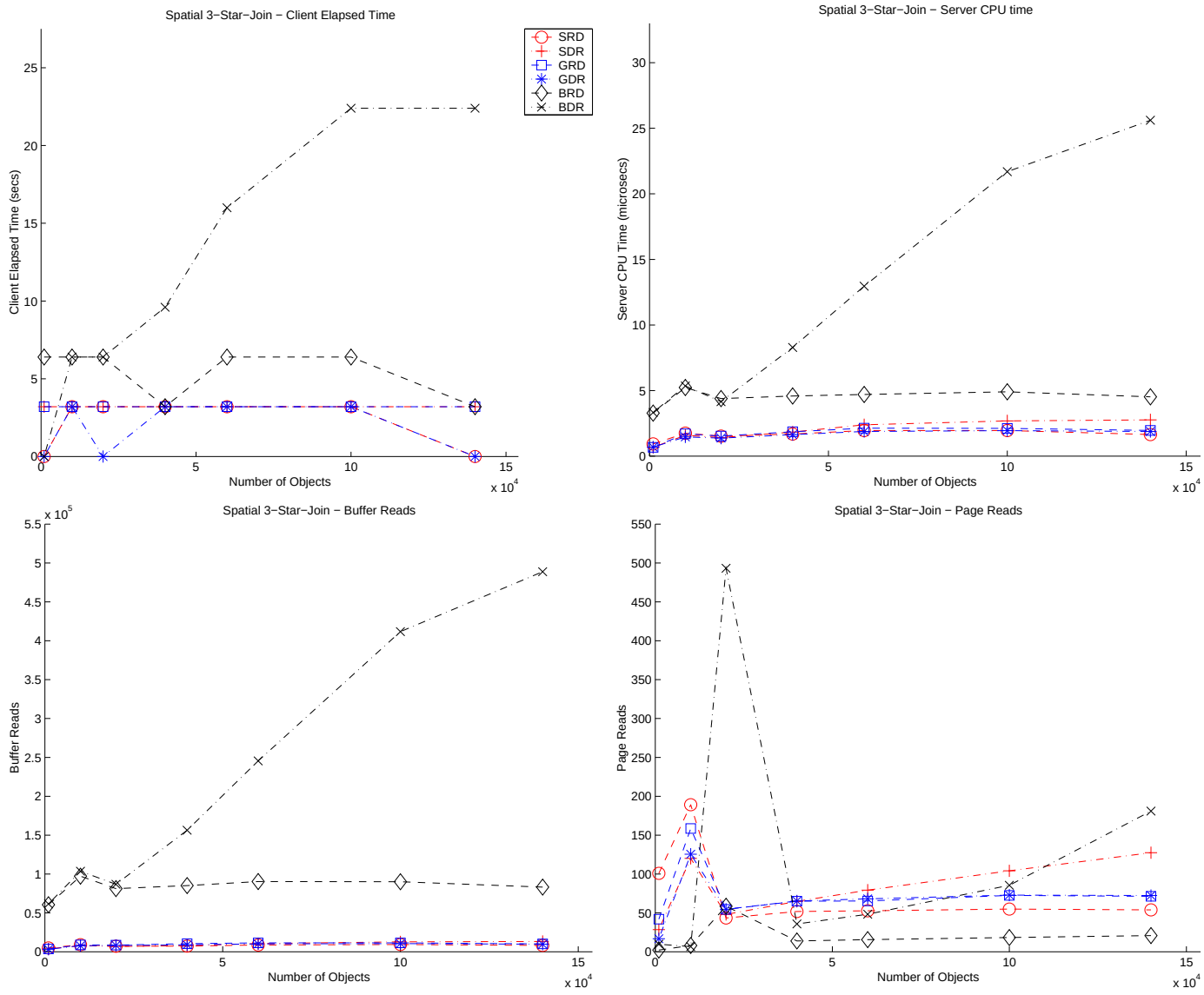


Figure 35: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 3-Star-Join queries and clustering on dec–RA and RA–dec.

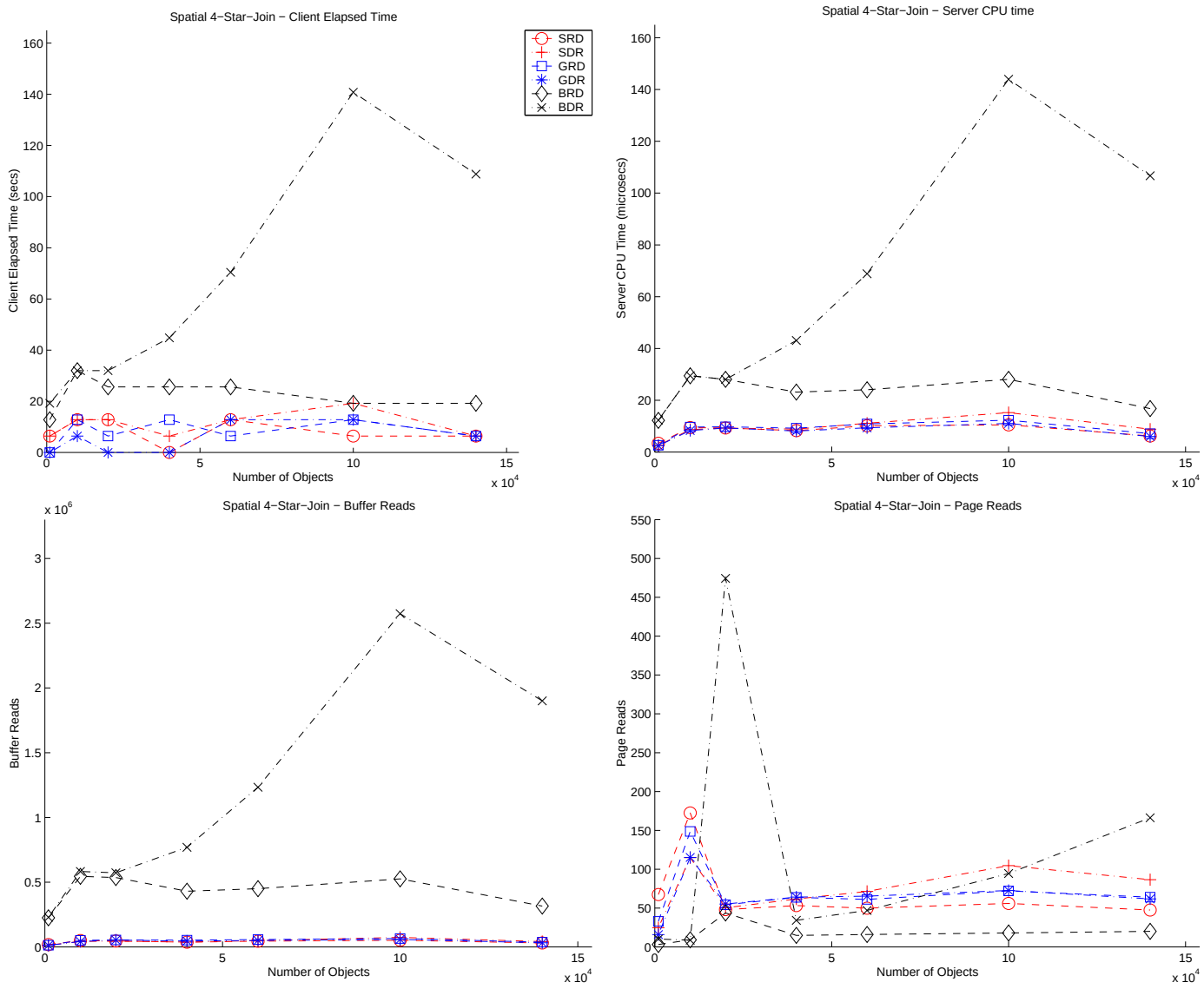


Figure 36: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 4-Star-Join queries and clustering on dec–RA and RA–dec.

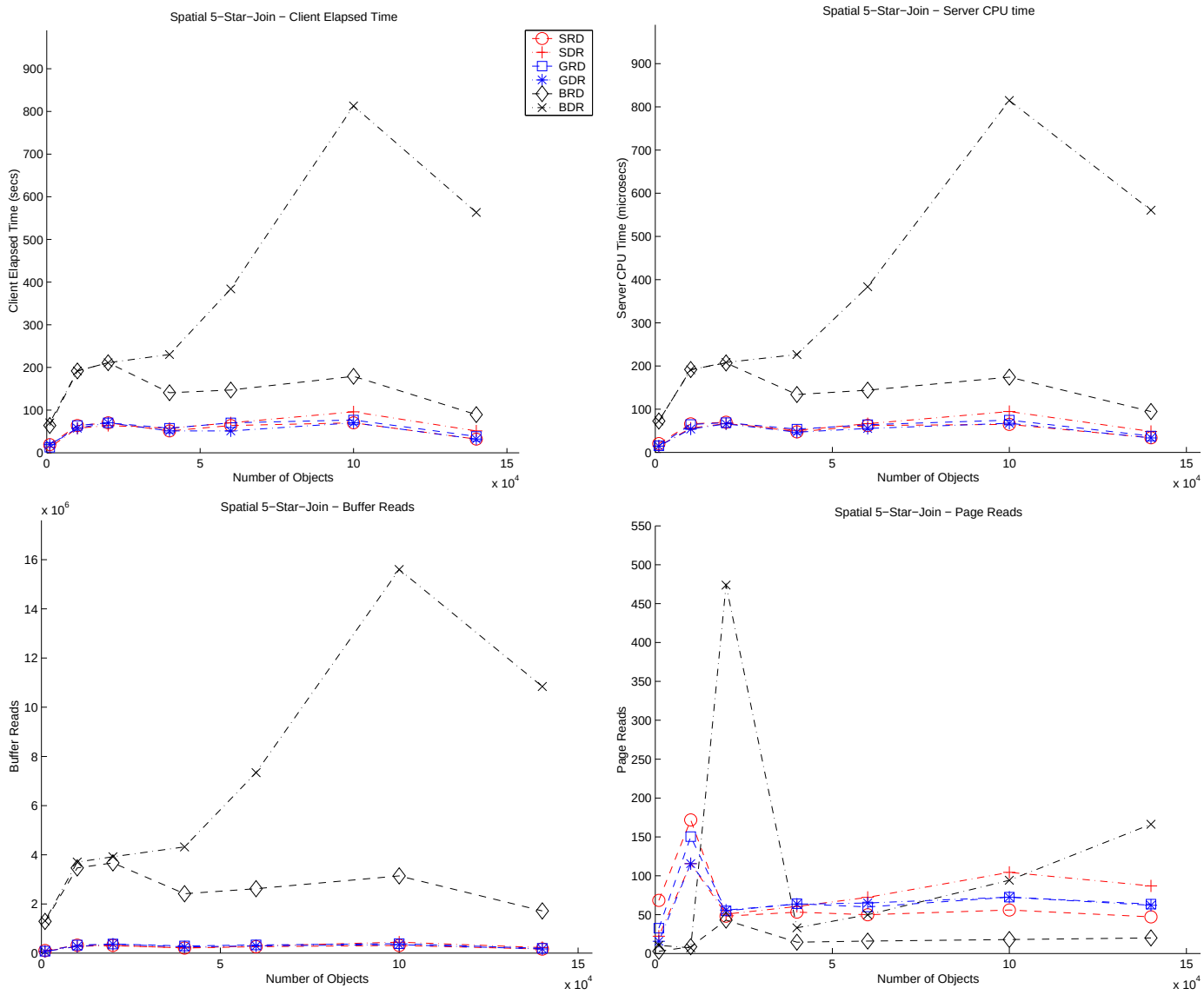


Figure 37: Performance of the Relational (**B**), Geodetic (**G**), and Shapes2 (**S**) schemas for Spatial 5-Star-Join queries and clustering on dec–RA and RA–dec.

F Summary and Statistics of Performance Measurement Data

Performance of Clustering on B-trees for the B, G, and S schemas

Schema	No. Stars	Count	Elapsed secs	Server CPU usecs	ISAM Reads	Buffer Reads	Page Reads
B	1000	2	0.000	0.099	1069.5	4762.5	909.5
G	1000	2	32.000	2.273	3978.0	29996.0	1890.0
S	1000	2	32.000	1.504	4158.5	28717.0	1688.0
B	10000	2	32.000	0.855	10053.0	42102.0	4048.5
G	10000	2	64.000	30.250	30966.0	345430.0	16750.0
S	10000	2	64.000	28.953	31182.0	333013.5	12484.5
B	20000	2	32.000	1.594	20053.0	83609.5	13006.0
G	20000	2	224.000	69.336	60966.0	745961.5	31380.0
S	20000	2	160.000	63.145	61182.0	681952.5	27767.0
B	40000	2	96.000	3.886	40053.0	166669.0	23093.0
G	40000	2	512.000	151.910	120966.0	1574949.5	67945.0
S	40000	2	352.000	144.820	121182.0	1375062.0	50505.0
B	60000	2	96.000	7.176	60053.0	250351.0	34636.5
G	60000	2	704.000	236.953	180966.0	2407547.5	98992.5
S	60000	2	512.000	237.664	181182.0	2104358.5	75052.0
B	100000	2	192.000	12.295	100053.0	418138.0	59154.5
G	100000	2	1280.000	416.602	300966.0	4066753.0	160752.0
S	100000	2	896.000	423.711	301182.0	3814643.5	120204.5
B	140000	2	224.000	18.115	140053.0	585721.5	78688.5
G	140000	2	1696.000	600.680	420966.0	5726494.5	226325.0
S	140000	2	1376.000	616.824	421182.0	5525795.0	178343.5

Performance of the B, G, and S schemas for various queries and table sizes

Query Type	Count	Schema	Elapsed secs	Server CPU usecs	ISAM Reads	Buffer Reads	Page Reads
Q1 - 1000	20	BDR	0.000	0.029	45.0	155.9	7.2
	20	BRD	0.000	0.029	45.0	159.2	3.0
	20	GDR	0.000	0.006	34.9	145.3	31.1
	20	GRD	0.000	0.086	40.4	158.8	11.5
	20	SDR	0.000	0.036	35.3	150.4	8.4
	20	SRD	0.000	0.029	35.3	159.7	9.8
Q1 - 10000	20	BDR	0.000	0.027	156.2	384.7	15.5
	20	BRD	0.000	0.054	156.2	413.8	80.8
	20	GDR	0.000	0.046	35.0	161.4	16.8
	20	GRD	0.000	0.050	35.0	168.2	19.9
	20	SDR	0.000	0.040	35.0	162.8	17.8
	20	SRD	0.000	0.029	35.0	168.5	14.5
Q1 - 20000	20	BDR	0.000	0.036	265.3	630.2	40.9
	20	BRD	0.000	0.037	265.3	621.2	35.1
	20	GDR	0.000	0.052	35.1	172.5	21.3
	20	GRD	3.200	0.044	35.1	176.7	15.8
	20	SDR	0.000	0.044	35.1	163.5	18.7
	20	SRD	0.000	0.041	35.1	166.8	14.8
Q1 - 40000	20	BDR	0.000	0.117	503.0	1093.7	16.5
	20	BRD	0.000	0.039	293.8	666.3	10.0
	20	GDR	0.000	0.040	35.1	178.8	18.4
	20	GRD	3.200	0.056	35.1	188.8	19.9
	20	SDR	0.000	0.042	35.0	170.7	18.0
	20	SRD	0.000	0.030	35.0	167.4	14.9
Q1 - 60000	20	BDR	0.000	0.062	713.6	1529.4	19.9
	20	BRD	0.000	0.107	292.7	667.1	10.9
	20	GDR	0.000	0.052	35.1	181.2	18.4
	20	GRD	0.000	0.045	35.1	189.0	19.4
	20	SDR	0.000	0.059	35.1	181.8	21.7
	20	SRD	0.000	0.032	35.1	172.8	15.9
Q1 - 100000	20	BDR	0.000	0.106	1180.6	2485.0	30.5
	20	BRD	0.000	0.042	292.5	662.1	11.2
	20	GDR	0.000	0.046	35.2	182.7	21.1
	20	GRD	0.000	0.061	35.2	189.6	22.4
	20	SDR	0.000	0.049	35.2	196.3	29.7
	20	SRD	0.000	0.035	35.2	176.2	17.1
Q1 - 140000	20	BDR	0.000	0.150	1448.2	3033.2	35.2
	20	BRD	0.000	0.064	303.9	690.9	13.0
	20	GDR	0.000	0.041	35.5	185.5	23.8
	20	GRD	0.000	0.079	35.5	192.5	24.3

	20 SDR	0.000	0.054	35.5	202.8	34.7
	20 SRD	0.000	0.037	35.5	176.8	18.2
Q2 - 1000	3 BDR	42.667	52.187	437089.7	919252.3	25.7
	3 BRD	64.000	53.500	437080.7	907889.0	1631.0
	3 GDR	21.333	5.510	4544.0	26412.0	1079.0
	3 GRD	0.000	5.083	4545.0	29149.0	1662.7
	3 SDR	0.000	4.406	4603.3	27237.7	936.0
	3 SRD	0.000	4.208	4546.7	27331.3	830.0
Q2 - 10000	3 BDR	618.667	623.050	5173201.7	11245321.7	31328.3
	3 BRD	618.667	614.021	5173190.3	10744385.3	848.7
	3 GDR	85.333	90.083	48629.0	429053.0	12933.3
	3 GRD	277.333	11.219	48629.3	459099.3	112773.0
	3 SDR	85.333	73.865	48601.0	398836.7	14460.3
	3 SRD	106.667	84.781	48584.3	443199.7	23422.3
Q2 - 20000	3 BDR	7189.333	3.383	10400807.7	22616227.0	3269357.7
	3 BRD	1237.333	1242.135	10400796.3	21604593.3	1535.0
	3 GDR	234.667	183.125	96962.0	1091954.0	57184.3
	3 GRD	704.000	20.333	96962.3	1180249.7	280160.7
	3 SDR	234.667	124.990	96889.0	838664.0	77732.3
	3 SRD	213.333	183.521	96872.3	883367.7	48863.3
Q2 - 40000	3 BDR	4117.333	4125.370	35024583.7	72318871.0	2994.0
	3 BRD	2517.333	2514.167	20946114.3	43502835.3	2982.0
	3 GDR	746.667	238.656	194594.0	2349136.0	268077.0
	3 GRD	1578.667	53.344	194594.3	2636033.7	625331.7
	3 SDR	746.667	162.750	194443.0	1754491.3	278036.7
	3 SRD	490.667	432.938	194426.3	1766313.7	104440.3
Q2 - 60000	3 BDR	9322.667	9499.396	79022515.7	162914632.0	4564.7
	3 BRD	3818.667	3803.781	31846980.3	66368407.3	4440.3
	3 GDR	1536.000	289.708	294296.0	3715424.0	576829.3
	3 GRD	2538.667	95.052	294296.3	4119699.7	1005023.3
	3 SDR	1109.333	622.240	294069.0	3441114.0	342323.7
	3 SRD	810.667	795.552	294052.3	3060131.7	60443.3
Q2 - 100000	3 BDR	25578.667	25542.682	219611519.7	451416615.7	7688.7
	3 BRD	6592.000	6569.948	53113224.3	110684084.7	7384.3
	3 GDR	3584.000	452.458	491454.0	6642590.0	1371447.0
	3 GRD	4672.000	262.917	491454.3	6917961.7	1839736.3
	3 SDR	2944.000	241.313	491057.0	5730658.7	1161742.7
	3 SRD	1365.333	1233.208	491040.3	5390411.7	287260.7
Q2 - 140000	3 BDR	49813.333	49736.383	428239515.7	879193633.0	12335.7
	3 BRD	8789.333	8777.448	73900002.3	154025032.3	10332.0
	3 GDR	6037.333	634.573	683663.0	9428507.0	2331740.0
	3 GRD	7082.667	498.927	683663.3	9883210.7	2768815.0
	3 SDR	4714.667	407.615	683097.0	8288146.0	1856346.3
	3 SRD	2026.667	1840.167	683080.0	7562354.0	393461.0

Q4 - 1000	20 BDR	0.000	0.030	58.8	201.6	6.4
	20 BRD	0.000	0.028	58.8	208.2	0.4
	20 GDR	0.000	0.028	39.5	184.0	26.8
	20 GRD	0.000	0.035	39.5	189.0	17.2
	20 SDR	0.000	0.032	39.5	189.4	13.8
	20 SRD	0.000	0.028	39.5	203.6	15.6
Q4 - 10000	20 BDR	0.000	0.026	273.3	642.0	24.7
	20 BRD	0.000	0.027	273.3	700.5	146.9
	20 GDR	0.000	0.026	39.4	215.7	37.2
	20 GRD	3.200	0.035	39.4	225.9	45.0
	20 SDR	3.200	0.030	39.4	216.6	39.9
	20 SRD	0.000	0.027	39.4	227.9	30.7
Q4 - 20000	20 BDR	0.000	0.032	488.0	1113.5	67.3
	20 BRD	3.200	0.035	488.0	1121.6	84.5
	20 GDR	0.000	0.029	40.3	243.3	53.7
	20 GRD	0.000	0.045	40.3	245.8	28.0
	20 SDR	0.000	0.033	40.3	224.3	47.6
	20 SRD	0.000	0.033	40.3	227.7	36.1
Q4 - 40000	20 BDR	0.000	0.039	922.3	1970.0	26.9
	20 BRD	0.000	0.048	554.3	1216.5	18.2
	20 GDR	0.000	0.042	39.8	245.9	32.2
	20 GRD	0.000	0.050	39.8	264.7	35.0
	20 SDR	0.000	0.038	39.7	232.6	31.5
	20 SRD	0.000	0.037	39.7	228.2	28.1
Q4 - 60000	20 BDR	0.000	0.114	1426.8	3010.5	38.9
	20 BRD	0.000	0.049	565.4	1243.1	19.2
	20 GDR	0.000	0.038	39.9	255.0	34.8
	20 GRD	0.000	0.052	39.9	267.1	36.5
	20 SDR	0.000	0.034	39.9	255.3	41.0
	20 SRD	0.000	0.034	39.9	240.8	30.8
Q4 - 100000	20 BDR	0.000	0.173	2269.3	4734.6	56.1
	20 BRD	0.000	0.050	560.2	1233.8	22.2
	20 GDR	0.000	0.052	40.0	260.6	40.6
	20 GRD	0.000	0.051	40.0	271.9	41.8
	20 SDR	0.000	0.045	40.0	281.5	52.9
	20 SRD	0.000	0.034	40.0	245.9	32.7
Q4 - 140000	20 BDR	0.000	0.226	3061.9	6358.2	67.6
	20 BRD	0.000	0.045	569.8	1256.6	23.0
	20 GDR	0.000	0.050	39.6	266.9	43.2
	20 GRD	3.200	0.068	39.6	277.6	43.4
	20 SDR	3.200	0.043	39.6	305.7	65.9
	20 SRD	3.200	0.041	39.6	246.4	33.7
Q52 - 1000	20 BDR	0.000	0.613	5548.5	11927.1	10.5

	20 BRD	0.000	0.630	5548.5	11720.5	3.1
	20 GDR	0.000	0.135	149.0	762.3	43.7
	20 GRD	3.200	0.147	149.0	840.3	42.3
	20 SDR	0.000	0.145	148.9	850.9	30.0
	20 SRD	0.000	0.137	148.9	1083.9	100.8
Q52 - 10000	20 BDR	6.400	0.029	8519.2	19145.4	3184.8
	20 BRD	3.200	0.882	8519.2	18022.9	238.9
	20 GDR	0.000	0.225	203.0	1527.5	125.9
	20 GRD	0.000	0.243	203.0	1723.4	158.8
	20 SDR	0.000	0.227	203.0	1558.7	142.9
	20 SRD	0.000	0.213	203.0	1792.0	189.3
Q52 - 20000	20 BDR	0.000	0.503	7744.1	17420.1	497.0
	20 BRD	0.000	0.798	7744.1	16361.8	59.2
	20 GDR	0.000	0.195	176.6	1699.9	147.8
	20 GRD	0.000	0.319	176.6	1778.5	54.6
	20 SDR	3.200	0.222	176.5	1445.4	134.3
	20 SRD	0.000	0.213	176.5	1497.1	139.4
Q52 - 40000	20 BDR	3.200	1.713	15188.1	31595.3	35.2
	20 BRD	0.000	0.855	8262.9	17419.2	14.0
	20 GDR	0.000	0.300	180.9	1875.3	65.4
	20 GRD	0.000	0.363	180.9	2157.6	65.4
	20 SDR	0.000	0.311	180.8	1650.7	65.2
	20 SRD	0.000	0.329	180.8	1562.2	51.9
Q52 - 60000	20 BDR	3.200	2.530	22886.9	47484.9	48.2
	20 BRD	3.200	0.966	8441.7	17928.8	15.6
	20 GDR	0.000	0.327	190.7	2044.3	67.9
	20 GRD	3.200	0.384	190.7	2286.3	65.6
	20 SDR	3.200	0.405	190.6	2110.5	78.9
	20 SRD	0.000	0.389	190.6	1773.6	53.0
Q52 - 100000	20 BDR	3.200	4.189	38178.8	78775.8	85.5
	20 BRD	0.000	0.935	8332.5	17705.2	18.4
	20 GDR	0.000	0.344	191.5	2082.7	73.0
	20 GRD	3.200	0.404	191.5	2267.4	72.7
	20 SDR	3.200	0.506	191.3	2523.2	103.3
	20 SRD	3.200	0.332	191.3	1877.8	55.2
Q52 - 140000	20 BDR	6.400	5.089	47034.6	96862.0	181.0
	20 BRD	0.000	0.787	8172.7	17340.2	20.8
	20 GDR	0.000	0.322	173.6	2036.4	72.5
	20 GRD	0.000	0.404	173.6	2147.6	71.4
	20 SDR	0.000	0.530	173.3	2720.4	127.2
	20 SRD	0.000	0.319	173.3	1736.0	54.1
Q53 - 1000	20 BDR	6.400	3.373	28715.0	61495.0	12.4
	20 BRD	3.200	3.287	28715.0	60395.2	3.2
	20 GDR	0.000	0.633	631.4	3275.2	29.3

	20 GRD	0.000	0.726	631.4	3671.0	54.1
	20 SDR	0.000	0.684	631.1	3727.8	34.5
	20 SRD	0.000	0.769	631.1	4881.1	137.5
Q53 - 10000	20 BDR	28.800	0.129	46115.1	103628.4	14860.3
	20 BRD	6.400	5.322	46115.1	97168.6	267.7
	20 GDR	3.200	1.458	982.3	7760.2	184.1
	20 GRD	0.000	1.624	982.3	8820.0	248.9
	20 SDR	3.200	1.519	981.5	7915.5	241.4
	20 SRD	0.000	1.645	981.5	9136.2	320.4
Q53 - 20000	20 BDR	3.200	3.783	38498.8	86774.2	1171.6
	20 BRD	6.400	4.209	38498.8	81199.1	58.6
	20 GDR	3.200	1.398	811.7	8240.5	233.5
	20 GRD	0.000	1.528	811.7	8605.5	64.3
	20 SDR	3.200	1.350	809.9	6914.0	205.9
	20 SRD	3.200	1.433	809.9	7158.7	222.4
Q53 - 40000	20 BDR	9.600	8.294	75061.7	156046.0	40.5
	20 BRD	3.200	4.671	40394.8	85092.0	14.4
	20 GDR	3.200	1.637	787.7	8991.6	76.3
	20 GRD	3.200	1.840	787.7	10410.6	73.6
	20 SDR	0.000	1.788	786.4	7820.2	75.0
	20 SRD	0.000	1.675	786.4	7361.3	58.0
Q53 - 60000	20 BDR	16.000	13.045	118418.3	245552.2	52.3
	20 BRD	6.400	4.929	42515.4	90301.0	16.6
	20 GDR	3.200	1.898	864.7	10192.4	79.5
	20 GRD	3.200	2.177	864.7	11389.6	75.9
	20 SDR	3.200	2.410	863.8	10603.4	94.7
	20 SRD	3.200	1.901	863.8	8741.3	60.3
Q53 - 100000	20 BDR	25.600	21.634	199696.9	411906.9	103.3
	20 BRD	3.200	4.808	42400.3	90063.9	20.6
	20 GDR	0.000	2.091	882.8	10495.9	83.6
	20 GRD	3.200	2.229	882.8	11457.4	82.4
	20 SDR	3.200	2.674	880.9	12792.4	128.0
	20 SRD	0.000	1.944	880.9	9417.5	62.7
Q53 - 140000	20 BDR	28.800	25.679	237414.8	488745.9	237.4
	20 BRD	3.200	4.398	39210.5	83180.6	22.2
	20 GDR	3.200	1.881	750.1	9715.5	82.0
	20 GRD	3.200	1.970	750.1	10180.7	79.7
	20 SDR	0.000	2.734	747.1	13248.1	193.9
	20 SRD	3.200	1.633	747.1	8200.6	61.1
Q54 - 1000	10 BDR	12.800	12.190	106263.2	226302.3	10.2
	10 BRD	12.800	12.156	106263.2	222353.6	2.8
	10 GDR	0.000	2.632	2429.1	11731.7	20.7
	10 GRD	6.400	2.745	2429.1	13304.1	51.6
	10 SDR	0.000	2.480	2429.1	13082.7	31.3

	10 SRD	0.000	3.291	2429.1	17708.5	140.6
Q54 - 10000	10 BDR	51.200	25.077	257364.1	579284.7	12135.9
	10 BRD	32.000	29.504	257364.1	542029.6	291.4
	10 GDR	6.400	8.295	5818.9	42113.5	257.4
	10 GRD	12.800	9.909	5818.9	50258.7	498.8
	10 SDR	12.800	8.625	5818.9	42773.1	331.7
	10 SRD	6.400	9.005	5818.9	48379.1	986.3
Q54 - 20000	10 BDR	38.400	24.233	255327.6	574398.2	7224.6
	10 BRD	32.000	28.331	255327.6	537799.8	45.1
	10 GDR	0.000	9.260	6282.3	52807.9	699.9
	10 GRD	12.800	9.807	6282.3	53861.3	74.1
	10 SDR	6.400	9.134	6282.2	44234.0	571.8
	10 SRD	12.800	9.122	6282.2	46438.2	488.7
Q54 - 40000	10 BDR	44.800	43.069	369510.5	768280.7	43.5
	10 BRD	25.600	23.083	203473.6	428313.4	16.0
	10 GDR	6.400	8.291	4117.4	44920.4	81.2
	10 GRD	12.800	9.213	4117.4	51627.6	79.5
	10 SDR	12.800	8.902	4115.5	37344.1	78.6
	10 SRD	6.400	8.675	4115.5	36725.9	63.0
Q54 - 60000	10 BDR	70.400	68.970	596224.6	1236132.6	56.1
	10 BRD	25.600	24.009	211997.0	450419.8	17.2
	10 GDR	6.400	9.439	4537.0	50030.2	86.4
	10 GRD	12.800	11.028	4537.0	57050.6	81.0
	10 SDR	12.800	11.206	4537.0	49490.2	98.1
	10 SRD	12.800	9.989	4537.0	44530.2	63.1
Q54 - 100000	10 BDR	140.800	144.112	1250809.1	2578560.5	133.9
	10 BRD	32.000	27.959	247560.5	526104.5	22.4
	10 GDR	12.800	11.370	5571.3	57277.1	93.6
	10 GRD	12.800	12.748	5571.3	64429.1	93.0
	10 SDR	19.200	15.220	5564.1	73031.5	157.2
	10 SRD	12.800	10.623	5564.1	51729.7	72.6
Q54 - 140000	10 BDR	108.800	106.784	925216.4	1904610.9	245.6
	10 BRD	19.200	16.742	148673.1	315748.8	22.0
	10 GDR	6.400	6.250	2956.2	31629.4	74.0
	10 GRD	6.400	7.334	2956.2	36474.2	74.4
	10 SDR	6.400	8.783	2955.5	41151.3	131.6
	10 SRD	6.400	6.075	2955.5	30545.7	55.1
Q55 - 1000	10 BDR	70.400	69.927	589608.0	1254346.6	11.3
	10 BRD	70.400	69.463	589608.0	1232585.4	3.5
	10 GDR	12.800	14.664	14316.1	65194.1	19.3
	10 GRD	12.800	15.320	14316.1	74203.9	59.3
	10 SDR	12.800	14.427	14316.1	73088.9	33.7
	10 SRD	19.200	18.759	14316.1	98925.9	306.1

Q55 - 10000	10 BDR	185.600	181.381	1566044.9	3523304.2	8.1
	10 BRD	185.600	180.985	1566044.9	3296610.9	107.4
	10 GDR	57.600	52.724	37703.9	258742.5	589.3
	10 GRD	64.000	63.065	37703.9	310258.9	2057.4
	10 SDR	51.200	54.347	37703.9	263174.3	935.6
	10 SRD	64.000	57.017	37703.9	299118.1	5919.7
Q55 - 20000	10 BDR	236.800	176.339	1701062.2	3823120.9	49844.8
	10 BRD	204.800	202.466	1701062.2	3581219.9	44.6
	10 GDR	70.400	65.115	46403.4	354943.8	1426.1
	10 GRD	70.400	68.031	46403.4	358810.4	84.2
	10 SDR	64.000	63.225	46402.3	294859.9	828.8
	10 SRD	64.000	66.498	46402.3	306737.3	70.4
Q55 - 40000	10 BDR	224.000	219.519	2012730.7	4184247.3	48.1
	10 BRD	134.400	128.433	1104690.2	2324773.8	16.4
	10 GDR	51.200	45.874	23563.5	245032.9	87.9
	10 GRD	51.200	52.023	23563.5	280953.3	86.9
	10 SDR	51.200	49.219	23556.8	204160.4	84.8
	10 SRD	51.200	45.997	23556.8	200037.4	68.2
Q55 - 60000	10 BDR	371.200	370.909	3425688.6	7099736.7	62.7
	10 BRD	140.800	138.782	1191010.3	2529615.1	18.0
	10 GDR	57.600	53.972	26699.4	281919.0	95.9
	10 GRD	64.000	62.684	26699.4	318919.0	87.6
	10 SDR	64.000	63.906	26699.4	279109.8	117.1
	10 SRD	57.600	59.722	26699.4	250024.4	69.0
Q55 - 100000	10 BDR	793.600	792.902	7382032.7	15216199.8	159.3
	10 BRD	172.800	169.023	1440585.0	3061139.3	23.2
	10 GDR	70.400	67.371	34378.5	332863.7	102.7
	10 GRD	76.800	74.807	34378.5	372285.1	102.4
	10 SDR	89.600	91.761	34333.2	430976.2	185.6
	10 SRD	64.000	62.483	34333.2	300123.4	80.7
Q55 - 140000	10 BDR	544.000	549.419	5152113.4	10602962.0	313.5
	10 BRD	83.200	92.290	787516.8	1672742.5	22.8
	10 GDR	38.400	33.341	16711.2	166731.8	79.8
	10 GRD	44.800	38.303	16711.2	190508.4	78.3
	10 SDR	51.200	47.202	16708.0	218301.4	152.7
	10 SRD	32.000	34.331	16708.0	162985.2	58.7
Q63 - 1000	20 BDR	0.000	3.330	28754.3	61578.4	9.7
	20 BRD	6.400	3.277	28754.3	60478.4	2.2
	20 GDR	0.000	0.686	630.6	3275.6	16.8
	20 GRD	3.200	0.670	630.6	3670.3	41.9
	20 SDR	3.200	0.671	630.3	3727.5	28.5
	20 SRD	0.000	0.931	630.3	4888.4	100.9
Q63 - 10000	20 BDR	6.400	5.347	46129.6	103669.8	6.6
	20 BRD	6.400	5.234	46129.6	97201.2	8.1

	20 GDR	3.200	1.466	982.6	7759.3	125.5
	20 GRD	3.200	1.702	982.6	8814.6	158.6
	20 SDR	3.200	1.591	982.2	7918.7	120.2
	20 SRD	3.200	1.750	982.2	9147.1	189.2
Q63 - 20000	20 BDR	6.400	4.158	38512.3	86828.6	493.3
	20 BRD	6.400	4.393	38512.3	81243.1	58.6
	20 GDR	0.000	1.377	805.9	8232.9	54.7
	20 GRD	3.200	1.535	805.9	8598.7	54.6
	20 SDR	3.200	1.413	804.1	6915.0	48.1
	20 SRD	3.200	1.520	804.1	7148.8	43.5
Q63 - 40000	20 BDR	9.600	8.284	75186.7	156312.7	35.8
	20 BRD	3.200	4.584	40406.4	85126.5	14.0
	20 GDR	3.200	1.639	788.9	8990.8	65.4
	20 GRD	3.200	1.859	788.9	10421.7	65.1
	20 SDR	3.200	1.833	787.5	7829.4	64.8
	20 SRD	3.200	1.684	787.5	7354.0	51.9
Q63 - 60000	20 BDR	16.000	12.961	118380.1	245470.1	48.3
	20 BRD	6.400	4.699	42538.4	90339.4	15.6
	20 GDR	3.200	1.880	867.0	10199.5	68.1
	20 GRD	3.200	2.132	867.0	11392.8	65.3
	20 SDR	3.200	2.388	866.1	10602.3	79.3
	20 SRD	3.200	1.953	866.1	8750.1	52.7
Q63 - 100000	20 BDR	22.400	21.692	199625.4	411759.4	85.5
	20 BRD	6.400	4.904	42400.6	90067.7	18.4
	20 GDR	3.200	1.944	881.9	10494.9	72.9
	20 GRD	3.200	2.120	881.9	11457.8	72.5
	20 SDR	3.200	2.684	880.0	12798.1	104.2
	20 SRD	3.200	1.952	880.0	9422.6	55.0
Q63 - 140000	20 BDR	22.400	25.611	237426.3	488758.9	181.0
	20 BRD	3.200	4.519	39219.6	83183.5	20.8
	20 GDR	0.000	1.886	748.8	9715.2	72.4
	20 GRD	3.200	1.957	748.8	10181.5	71.4
	20 SDR	3.200	2.764	746.0	13255.4	127.4
	20 SRD	0.000	1.651	746.0	8192.9	53.9
Q64 - 1000	10 BDR	19.200	12.295	107393.9	228895.8	10.2
	10 BRD	12.800	12.236	107393.9	224914.2	3.1
	10 GDR	0.000	2.455	2504.1	11832.7	15.9
	10 GRD	0.000	2.600	2504.1	13385.1	32.9
	10 SDR	6.400	2.533	2504.1	13217.7	24.7
	10 SRD	6.400	3.386	2504.1	17964.7	67.4
Q64 - 10000	10 BDR	32.000	29.472	258942.3	582705.2	8.7
	10 BRD	32.000	29.457	258942.3	545269.2	9.1
	10 GDR	6.400	8.313	6095.8	42523.8	115.2
	10 GRD	12.800	9.659	6095.8	50739.6	148.7

	10 SDR	12.800	8.659	6095.8	43245.8	114.9
	10 SRD	12.800	9.464	6095.8	48993.8	172.5
Q64 - 20000	10 BDR	32.000	28.041	254421.2	572730.0	474.4
	10 BRD	25.600	28.063	254421.2	536185.4	43.8
	10 GDR	0.000	9.288	6421.1	52776.9	55.2
	10 GRD	6.400	9.752	6421.1	53833.7	54.3
	10 SDR	12.800	9.263	6421.1	44238.1	50.4
	10 SRD	12.800	9.306	6421.1	46330.3	48.0
Q64 - 40000	10 BDR	44.800	43.087	369981.7	769337.6	34.3
	10 BRD	25.600	23.173	204494.8	430464.9	14.8
	10 GDR	0.000	8.142	4259.8	45082.2	64.2
	10 GRD	12.800	9.210	4259.8	52008.2	63.8
	10 SDR	6.400	8.850	4257.3	37608.3	61.7
	10 SRD	0.000	8.267	4257.3	36834.5	53.2
Q64 - 60000	10 BDR	70.400	68.857	595009.6	1233502.2	47.0
	10 BRD	25.600	24.076	211786.7	450042.9	16.0
	10 GDR	12.800	9.355	4691.6	50190.4	65.3
	10 GRD	6.400	10.871	4691.6	57102.0	61.2
	10 SDR	12.800	11.141	4691.6	49507.0	71.5
	10 SRD	12.800	10.164	4691.6	44675.4	49.8
Q64 - 100000	10 BDR	140.800	143.943	1248322.4	2573423.3	94.5
	10 BRD	19.200	28.088	247246.2	525325.3	18.0
	10 GDR	12.800	11.046	5712.4	57476.2	72.7
	10 GRD	12.800	12.320	5712.4	64561.0	72.1
	10 SDR	19.200	15.219	5707.9	73153.1	104.6
	10 SRD	6.400	10.492	5707.9	51894.9	56.0
Q64 - 140000	10 BDR	108.800	106.695	923091.4	1900194.2	166.2
	10 BRD	19.200	16.804	148618.9	315556.4	20.0
	10 GDR	6.400	6.002	3022.2	31639.6	61.8
	10 GRD	6.400	7.089	3022.2	36494.2	64.0
	10 SDR	6.400	8.809	3022.2	41236.4	86.6
	10 SRD	6.400	6.203	3022.2	30550.0	47.5
Q65 - 1000	10 BDR	70.400	72.982	616576.7	1312829.2	10.5
	10 BRD	64.000	72.595	616576.7	1290272.7	2.8
	10 GDR	19.200	14.635	15647.9	68098.1	15.8
	10 GRD	12.800	15.418	15647.9	77203.5	32.5
	10 SDR	12.800	14.819	15647.9	76479.7	22.2
	10 SRD	19.200	20.459	15647.9	104267.1	68.4
Q65 - 10000	10 BDR	192.000	191.268	1645761.0	3703074.5	7.3
	10 BRD	192.000	191.964	1645761.0	3463965.9	8.7
	10 GDR	57.600	54.035	42981.5	274713.7	115.6
	10 GRD	64.000	64.747	42981.5	329119.7	150.3
	10 SDR	57.600	56.219	42981.5	280067.9	115.6
	10 SRD	64.000	66.131	42981.5	318946.3	171.8

Q65 - 20000	10 BDR	211.200	208.279	1742530.5	3918084.5	473.8
	10 BRD	211.200	207.143	1742530.5	3671520.5	43.0
	10 GDR	70.400	66.080	51072.5	367385.5	55.9
	10 GRD	70.400	68.972	51072.5	371363.9	54.8
	10 SDR	64.000	66.830	51072.5	305018.3	50.6
	10 SRD	70.400	69.941	51072.5	316776.7	48.0
Q65 - 40000	10 BDR	230.400	226.581	2078501.1	4321648.5	33.1
	10 BRD	140.800	134.013	1149151.7	2417759.3	14.6
	10 GDR	51.200	46.859	25970.5	254457.7	63.5
	10 GRD	57.600	53.798	25970.5	293680.1	64.0
	10 SDR	57.600	50.995	25961.5	213250.1	60.5
	10 SRD	51.200	47.598	25961.5	207688.5	53.2
Q65 - 60000	10 BDR	384.000	383.663	3541842.4	7339275.6	49.9
	10 BRD	147.200	144.106	1233993.5	2622037.4	16.0
	10 GDR	51.200	55.457	29788.4	294503.4	65.0
	10 GRD	70.400	64.277	29788.4	331908.6	59.9
	10 SDR	70.400	66.653	29788.4	290393.8	72.2
	10 SRD	64.000	62.220	29788.4	261111.6	49.8
Q65 - 100000	10 BDR	812.800	814.730	7569339.3	15601961.7	94.3
	10 BRD	179.200	174.302	1478420.1	3140411.8	18.0
	10 GDR	70.400	67.578	37335.2	344613.4	72.5
	10 GRD	76.800	74.631	37335.2	384260.6	72.5
	10 SDR	96.000	94.630	37313.2	445210.8	104.4
	10 SRD	70.400	64.938	37313.2	310591.2	56.0
Q65 - 140000	10 BDR	563.200	560.786	5268063.5	10841504.6	166.3
	10 BRD	89.600	94.556	808019.0	1715642.4	20.0
	10 GDR	32.000	33.420	18022.9	171505.1	62.1
	10 GRD	38.400	38.032	18022.9	195955.7	63.5
	10 SDR	51.200	48.447	18022.9	224956.9	87.0
	10 SRD	32.000	34.056	18022.9	167763.3	47.1

Elapsed times of Spatial Queries for the B, G, and S Schemas

Query Type	No. Stars	Minimum	Maximum	Average	Std. Dev.
Q1 - BDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q1 - BRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q1 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q1 - GRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	3.2	14.3
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q1 - SDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q1 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0

Q2 - BDR	1000	0.0	64.0	42.7	37.0
	10000	576.0	640.0	618.7	37.0
	20000	7168.0	7232.0	7189.3	37.0
	40000	4096.0	4160.0	4117.3	37.0
	60000	9216.0	9472.0	9322.7	133.2
	100000	25472.0	25728.0	25578.7	133.2
	140000	49664.0	50112.0	49813.3	258.7
Q2 - BRD	1000	64.0	64.0	64.0	0.0
	10000	576.0	640.0	618.7	37.0
	20000	1216.0	1280.0	1237.3	37.0
	40000	2496.0	2560.0	2517.3	37.0
	60000	3776.0	3840.0	3818.7	37.0
	100000	6528.0	6656.0	6592.0	64.0
	140000	8768.0	8832.0	8789.3	37.0
Q2 - GDR	1000	0.0	64.0	21.3	37.0
	10000	64.0	128.0	85.3	37.0
	20000	192.0	256.0	234.7	37.0
	40000	704.0	768.0	746.7	37.0
	60000	1536.0	1536.0	1536.0	0.0
	100000	3584.0	3584.0	3584.0	0.0
	140000	6016.0	6080.0	6037.3	37.0
Q2 - GRD	1000	0.0	0.0	0.0	0.0
	10000	256.0	320.0	277.3	37.0
	20000	704.0	704.0	704.0	0.0
	40000	1536.0	1600.0	1578.7	37.0
	60000	2496.0	2560.0	2538.7	37.0
	100000	4672.0	4672.0	4672.0	0.0
	140000	7040.0	7104.0	7082.7	37.0
Q2 - SDR	1000	0.0	0.0	0.0	0.0
	10000	64.0	128.0	85.3	37.0
	20000	192.0	256.0	234.7	37.0
	40000	704.0	768.0	746.7	37.0
	60000	1088.0	1152.0	1109.3	37.0
	100000	2944.0	2944.0	2944.0	0.0
	140000	4672.0	4736.0	4714.7	37.0
Q2 - SRD	1000	0.0	0.0	0.0	0.0
	10000	64.0	128.0	106.7	37.0
	20000	192.0	256.0	213.3	37.0
	40000	448.0	512.0	490.7	37.0
	60000	768.0	832.0	810.7	37.0
	100000	1344.0	1408.0	1365.3	37.0
	140000	1984.0	2048.0	2026.7	37.0
Q4 - BDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0

	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q4 - BRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	64.0	3.2	14.3
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q4 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q4 - GRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	64.0	3.2	14.3
Q4 - SDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	64.0	3.2	14.3
Q4 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	64.0	3.2	14.3
Q52 - BDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	6.4	19.7
	20000	0.0	0.0	0.0	0.0
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3

	100000	0.0	64.0	3.2	14.3
	140000	0.0	64.0	6.4	19.7
Q52 - BRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	3.2	14.3
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q52 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	0.0	0.0	0.0
	140000	0.0	0.0	0.0	0.0
Q52 - GRD	1000	0.0	64.0	3.2	14.3
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q52 - SDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	64.0	3.2	14.3
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q52 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	0.0	0.0	0.0
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q53 - BDR	1000	0.0	64.0	6.4	19.7
	10000	0.0	64.0	28.8	32.7
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	9.6	23.4
	60000	0.0	64.0	16.0	28.4
	100000	0.0	64.0	25.6	32.2
	140000	0.0	64.0	28.8	32.7

Q53 - BRD	1000	0.0	64.0	3.2	14.3
	10000	0.0	64.0	6.4	19.7
	20000	0.0	64.0	6.4	19.7
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	6.4	19.7
	100000	0.0	64.0	3.2	14.3
	140000	0.0	64.0	3.2	14.3
Q53 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	0.0	0.0	0.0
	140000	0.0	64.0	3.2	14.3
Q53 - GRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	0.0	0.0	0.0
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	64.0	3.2	14.3
Q53 - SDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	64.0	3.2	14.3
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q53 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	0.0	0.0	0.0
	20000	0.0	64.0	3.2	14.3
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	3.2	14.3
	100000	0.0	0.0	0.0	0.0
	140000	0.0	64.0	3.2	14.3
Q54 - BDR	1000	0.0	64.0	12.8	27.0
	10000	0.0	128.0	51.2	50.5
	20000	0.0	128.0	38.4	44.7
	40000	0.0	128.0	44.8	43.2
	60000	0.0	128.0	70.4	47.2
	100000	0.0	256.0	140.8	84.3
	140000	0.0	320.0	108.8	124.6
Q54 - BRD	1000	0.0	64.0	12.8	27.0
	10000	0.0	64.0	32.0	33.7
	20000	0.0	128.0	32.0	45.3

	40000	0.0	64.0	25.6	33.0
	60000	0.0	64.0	25.6	33.0
	100000	0.0	64.0	32.0	33.7
	140000	0.0	64.0	19.2	30.9
Q54 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	6.4	20.2
	20000	0.0	0.0	0.0	0.0
	40000	0.0	64.0	6.4	20.2
	60000	0.0	64.0	6.4	20.2
	100000	0.0	64.0	12.8	27.0
	140000	0.0	64.0	6.4	20.2
Q54 - GRD	1000	0.0	64.0	6.4	20.2
	10000	0.0	64.0	12.8	27.0
	20000	0.0	64.0	12.8	27.0
	40000	0.0	64.0	12.8	27.0
	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	12.8	27.0
	140000	0.0	64.0	6.4	20.2
Q54 - SDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	12.8	27.0
	20000	0.0	64.0	6.4	20.2
	40000	0.0	64.0	12.8	27.0
	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	19.2	30.9
	140000	0.0	64.0	6.4	20.2
Q54 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	6.4	20.2
	20000	0.0	64.0	12.8	27.0
	40000	0.0	64.0	6.4	20.2
	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	12.8	27.0
	140000	0.0	64.0	6.4	20.2
Q55 - BDR	1000	0.0	192.0	70.4	82.3
	10000	0.0	576.0	185.6	169.2
	20000	0.0	1344.0	236.8	395.7
	40000	0.0	448.0	224.0	148.6
	60000	0.0	896.0	371.2	298.4
	100000	128.0	1536.0	793.6	492.2
	140000	0.0	1856.0	544.0	710.3
Q55 - BRD	1000	0.0	192.0	70.4	76.6
	10000	0.0	512.0	185.6	158.1
	20000	0.0	1152.0	204.8	339.7
	40000	0.0	320.0	134.4	97.5
	60000	0.0	320.0	140.8	103.6
	100000	64.0	320.0	172.8	95.6

	140000	0.0	256.0	83.2	109.0
Q55 - GDR	1000	0.0	64.0	12.8	27.0
	10000	0.0	192.0	57.6	63.6
	20000	0.0	384.0	70.4	114.7
	40000	0.0	128.0	51.2	50.5
	60000	0.0	128.0	57.6	36.3
	100000	0.0	128.0	70.4	47.2
	140000	0.0	128.0	38.4	54.0
Q55 - GRD	1000	0.0	64.0	12.8	27.0
	10000	0.0	192.0	64.0	67.5
	20000	0.0	384.0	70.4	114.7
	40000	0.0	192.0	51.2	66.1
	60000	0.0	128.0	64.0	52.3
	100000	0.0	128.0	76.8	40.5
	140000	0.0	128.0	44.8	52.7
Q55 - SDR	1000	0.0	64.0	12.8	27.0
	10000	0.0	128.0	51.2	50.5
	20000	0.0	320.0	64.0	100.1
	40000	0.0	128.0	51.2	40.5
	60000	0.0	128.0	64.0	52.3
	100000	0.0	192.0	89.6	61.8
	140000	0.0	192.0	51.2	66.1
Q55 - SRD	1000	0.0	64.0	19.2	30.9
	10000	0.0	256.0	64.0	79.8
	20000	0.0	384.0	64.0	120.7
	40000	0.0	128.0	51.2	40.5
	60000	0.0	128.0	57.6	47.2
	100000	0.0	128.0	64.0	52.3
	140000	0.0	128.0	32.0	54.4
Q63 - BDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	6.4	19.7
	20000	0.0	64.0	6.4	19.7
	40000	0.0	64.0	9.6	23.4
	60000	0.0	64.0	16.0	28.4
	100000	0.0	64.0	22.4	31.3
	140000	0.0	64.0	22.4	31.3
Q63 - BRD	1000	0.0	64.0	6.4	19.7
	10000	0.0	64.0	6.4	19.7
	20000	0.0	64.0	6.4	19.7
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	6.4	19.7
	100000	0.0	64.0	6.4	19.7
	140000	0.0	64.0	3.2	14.3
Q63 - GDR	1000	0.0	0.0	0.0	0.0

	10000	0.0	64.0	3.2	14.3
	20000	0.0	0.0	0.0	0.0
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q63 - GRD	1000	0.0	64.0	3.2	14.3
	10000	0.0	64.0	3.2	14.3
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	64.0	3.2	14.3
Q63 - SDR	1000	0.0	64.0	3.2	14.3
	10000	0.0	64.0	3.2	14.3
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	64.0	3.2	14.3
Q63 - SRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	3.2	14.3
	20000	0.0	64.0	3.2	14.3
	40000	0.0	64.0	3.2	14.3
	60000	0.0	64.0	3.2	14.3
	100000	0.0	64.0	3.2	14.3
	140000	0.0	0.0	0.0	0.0
Q64 - BDR	1000	0.0	64.0	19.2	30.9
	10000	0.0	128.0	32.0	45.3
	20000	0.0	128.0	32.0	45.3
	40000	0.0	64.0	44.8	30.9
	60000	0.0	128.0	70.4	56.0
	100000	64.0	256.0	140.8	78.7
	140000	0.0	320.0	108.8	135.1
Q64 - BRD	1000	0.0	64.0	12.8	27.0
	10000	0.0	64.0	32.0	33.7
	20000	0.0	128.0	25.6	44.7
	40000	0.0	64.0	25.6	33.0
	60000	0.0	64.0	25.6	33.0
	100000	0.0	64.0	19.2	30.9
	140000	0.0	64.0	19.2	30.9
Q64 - GDR	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	6.4	20.2
	20000	0.0	0.0	0.0	0.0
	40000	0.0	0.0	0.0	0.0

	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	12.8	27.0
	140000	0.0	64.0	6.4	20.2
Q64 - GRD	1000	0.0	0.0	0.0	0.0
	10000	0.0	64.0	12.8	27.0
	20000	0.0	64.0	6.4	20.2
	40000	0.0	64.0	12.8	27.0
	60000	0.0	64.0	6.4	20.2
	100000	0.0	64.0	12.8	27.0
	140000	0.0	64.0	6.4	20.2
Q64 - SDR	1000	0.0	64.0	6.4	20.2
	10000	0.0	64.0	12.8	27.0
	20000	0.0	64.0	12.8	27.0
	40000	0.0	64.0	6.4	20.2
	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	19.2	30.9
	140000	0.0	64.0	6.4	20.2
Q64 - SRD	1000	0.0	64.0	6.4	20.2
	10000	0.0	64.0	12.8	27.0
	20000	0.0	64.0	12.8	27.0
	40000	0.0	0.0	0.0	0.0
	60000	0.0	64.0	12.8	27.0
	100000	0.0	64.0	6.4	20.2
	140000	0.0	64.0	6.4	20.2
Q65 - BDR	1000	0.0	192.0	70.4	82.3
	10000	0.0	576.0	192.0	175.9
	20000	0.0	1152.0	211.2	336.0
	40000	0.0	448.0	230.4	165.8
	60000	0.0	960.0	384.0	326.3
	100000	128.0	1600.0	812.8	529.5
	140000	0.0	1856.0	563.2	708.1
Q65 - BRD	1000	0.0	192.0	64.0	67.5
	10000	0.0	576.0	192.0	175.9
	20000	0.0	1216.0	211.2	359.6
	40000	0.0	320.0	140.8	99.1
	60000	0.0	320.0	147.2	104.7
	100000	64.0	384.0	179.2	99.1
	140000	0.0	320.0	89.6	109.6
Q65 - GDR	1000	0.0	64.0	19.2	30.9
	10000	0.0	192.0	57.6	63.6
	20000	0.0	384.0	70.4	114.7
	40000	0.0	128.0	51.2	40.5
	60000	0.0	128.0	51.2	40.5
	100000	0.0	128.0	70.4	47.2
	140000	0.0	64.0	32.0	33.7

Q65 - GRD	1000	0.0	64.0	12.8	27.0
	10000	0.0	192.0	64.0	60.3
	20000	0.0	384.0	70.4	114.7
	40000	0.0	128.0	57.6	47.2
	60000	0.0	128.0	70.4	47.2
	100000	64.0	128.0	76.8	27.0
	140000	0.0	128.0	38.4	44.7
Q65 - SDR	1000	0.0	64.0	12.8	27.0
	10000	0.0	192.0	57.6	56.0
	20000	0.0	384.0	64.0	116.8
	40000	0.0	128.0	57.6	47.2
	60000	0.0	128.0	70.4	56.0
	100000	0.0	192.0	96.0	69.1
	140000	0.0	128.0	51.2	58.8
Q65 - SRD	1000	0.0	64.0	19.2	30.9
	10000	0.0	192.0	64.0	60.3
	20000	0.0	384.0	70.4	114.7
	40000	0.0	128.0	51.2	40.5
	60000	0.0	128.0	64.0	42.7
	100000	0.0	128.0	70.4	47.2
	140000	0.0	64.0	32.0	33.7

Server CPU times in microsecs of Spatial Queries for the B, G, and S Schemas

Query Type	No. Stars	Minimum	Maximum	Average	Std. Dev.
Q1 - BDR	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.1	0.0	0.1
	40000	0.0	1.3	0.1	0.3
	60000	0.0	0.2	0.1	0.1
	100000	0.0	0.6	0.1	0.2
	140000	0.0	0.5	0.2	0.2
Q1 - BRD	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.8	0.1	0.2
	20000	0.0	0.1	0.0	0.1
	40000	0.0	0.1	0.0	0.1
	60000	0.0	1.4	0.1	0.3
	100000	0.0	0.1	0.0	0.1
	140000	0.0	0.6	0.1	0.1
Q1 - GDR	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.2	0.0	0.1
	20000	0.0	0.4	0.1	0.1
	40000	0.0	0.1	0.0	0.0
	60000	0.0	0.4	0.1	0.1
	100000	0.0	0.1	0.0	0.1
	140000	0.0	0.2	0.0	0.1
Q1 - GRD	1000	0.0	1.0	0.1	0.2
	10000	0.0	0.3	0.1	0.1
	20000	0.0	0.2	0.0	0.1
	40000	0.0	0.3	0.1	0.1
	60000	0.0	0.2	0.0	0.1
	100000	0.0	0.3	0.1	0.1
	140000	0.0	0.6	0.1	0.1
Q1 - SDR	1000	0.0	0.2	0.0	0.1
	10000	0.0	0.2	0.0	0.1
	20000	0.0	0.3	0.0	0.1
	40000	0.0	0.2	0.0	0.1
	60000	0.0	0.5	0.1	0.1
	100000	0.0	0.2	0.0	0.1
	140000	0.0	0.2	0.1	0.1
Q1 - SRD	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.2	0.0	0.1
	40000	0.0	0.1	0.0	0.0
	60000	0.0	0.1	0.0	0.1
	100000	0.0	0.1	0.0	0.1
	140000	0.0	0.2	0.0	0.1

Q2 - BDR	1000	50.9	53.0	52.2	1.2
	10000	621.2	625.5	623.0	2.2
	20000	3.1	3.6	3.4	0.2
	40000	4118.3	4137.8	4125.4	10.8
	60000	9222.9	10004.0	9499.4	437.7
	100000	25419.5	25699.8	25542.7	143.2
	140000	49606.3	49948.6	49736.4	185.3
Q2 - BRD	1000	51.5	56.1	53.5	2.3
	10000	610.7	616.0	614.0	2.9
	20000	1237.8	1245.6	1242.1	4.0
	40000	2499.8	2527.5	2514.2	13.9
	60000	3790.9	3821.9	3803.8	16.1
	100000	6536.0	6593.6	6569.9	30.1
	140000	8735.2	8800.4	8777.4	36.7
Q2 - GDR	1000	4.8	6.2	5.5	0.7
	10000	89.6	90.4	90.1	0.5
	20000	181.0	185.7	183.1	2.3
	40000	232.3	244.2	238.7	6.0
	60000	284.3	299.6	289.7	8.6
	100000	442.7	467.4	452.5	13.2
	140000	628.0	642.0	634.6	7.0
Q2 - GRD	1000	4.1	6.9	5.1	1.6
	10000	10.3	12.9	11.2	1.5
	20000	19.7	21.6	20.3	1.1
	40000	50.8	58.3	53.3	4.3
	60000	91.8	100.7	95.1	4.9
	100000	258.4	270.7	262.9	6.8
	140000	490.6	504.7	498.9	7.4
Q2 - SDR	1000	3.7	4.9	4.4	0.7
	10000	71.8	76.3	73.9	2.3
	20000	123.9	126.8	125.0	1.6
	40000	160.4	164.3	162.8	2.1
	60000	615.9	632.9	622.2	9.3
	100000	238.3	247.2	241.3	5.1
	140000	398.9	422.9	407.6	13.3
Q2 - SRD	1000	2.8	5.0	4.2	1.2
	10000	83.4	85.9	84.8	1.3
	20000	182.3	185.6	183.5	1.8
	40000	428.1	436.5	432.9	4.4
	60000	792.6	797.3	795.6	2.6
	100000	1223.0	1238.7	1233.2	8.8
	140000	1832.6	1851.9	1840.2	10.3
Q4 - BDR	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0

	20000	0.0	0.1	0.0	0.0
	40000	0.0	0.2	0.0	0.1
	60000	0.0	0.4	0.1	0.2
	100000	0.0	0.6	0.2	0.2
	140000	0.0	0.8	0.2	0.3
Q4 - BRD	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.1	0.0	0.1
	40000	0.0	0.2	0.0	0.1
	60000	0.0	0.2	0.0	0.1
	100000	0.0	0.2	0.1	0.1
	140000	0.0	0.2	0.0	0.1
Q4 - GDR	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.1	0.0	0.0
	40000	0.0	0.1	0.0	0.1
	60000	0.0	0.1	0.0	0.1
	100000	0.0	0.2	0.1	0.1
	140000	0.0	0.2	0.1	0.1
Q4 - GRD	1000	0.0	0.1	0.0	0.1
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.2	0.0	0.1
	40000	0.0	0.2	0.1	0.1
	60000	0.0	0.2	0.1	0.1
	100000	0.0	0.2	0.1	0.1
	140000	0.0	0.3	0.1	0.1
Q4 - SDR	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.1	0.0	0.0
	40000	0.0	0.1	0.0	0.1
	60000	0.0	0.1	0.0	0.0
	100000	0.0	0.2	0.0	0.1
	140000	0.0	0.2	0.0	0.1
Q4 - SRD	1000	0.0	0.1	0.0	0.0
	10000	0.0	0.1	0.0	0.0
	20000	0.0	0.1	0.0	0.0
	40000	0.0	0.2	0.0	0.1
	60000	0.0	0.2	0.0	0.1
	100000	0.0	0.1	0.0	0.1
	140000	0.0	0.2	0.0	0.1
Q52 - BDR	1000	0.0	1.5	0.6	0.6
	10000	0.0	0.0	0.0	0.0
	20000	0.0	1.3	0.5	0.3
	40000	0.0	3.3	1.7	0.9
	60000	0.0	5.3	2.5	1.7

	100000	0.0	9.5	4.2	2.3
	140000	0.0	11.1	5.1	3.2
Q52 - BRD	1000	0.0	1.6	0.6	0.6
	10000	0.0	1.8	0.9	0.6
	20000	0.0	2.9	0.8	0.8
	40000	0.0	1.9	0.9	0.7
	60000	0.0	2.1	1.0	0.6
	100000	0.0	1.9	0.9	0.7
	140000	0.0	1.6	0.8	0.6
Q52 - GDR	1000	0.0	0.5	0.1	0.1
	10000	0.0	0.7	0.2	0.2
	20000	0.0	0.6	0.2	0.2
	40000	0.0	0.8	0.3	0.3
	60000	0.0	0.8	0.3	0.3
	100000	0.0	1.5	0.3	0.5
	140000	0.0	1.0	0.3	0.4
Q52 - GRD	1000	0.0	0.5	0.1	0.1
	10000	0.0	0.6	0.2	0.2
	20000	0.0	0.8	0.3	0.3
	40000	0.0	1.2	0.4	0.5
	60000	0.0	1.0	0.4	0.3
	100000	0.0	1.1	0.4	0.4
	140000	0.0	0.9	0.4	0.3
Q52 - SDR	1000	0.0	0.4	0.1	0.1
	10000	0.0	0.7	0.2	0.3
	20000	0.0	0.8	0.2	0.3
	40000	0.0	1.2	0.3	0.4
	60000	0.0	1.4	0.4	0.5
	100000	0.0	1.0	0.5	0.3
	140000	0.0	1.3	0.5	0.5
Q52 - SRD	1000	0.0	0.5	0.1	0.2
	10000	0.0	0.7	0.2	0.3
	20000	0.0	0.8	0.2	0.3
	40000	0.0	1.1	0.3	0.5
	60000	0.0	1.2	0.4	0.5
	100000	0.0	1.1	0.3	0.4
	140000	0.0	0.9	0.3	0.3
Q53 - BDR	1000	0.0	6.6	3.4	2.2
	10000	0.1	0.2	0.1	0.0
	20000	0.7	13.1	3.8	2.7
	40000	0.1	15.7	8.3	4.6
	60000	1.5	27.4	13.0	7.0
	100000	3.0	38.1	21.6	11.8
	140000	2.7	52.7	25.7	18.6

Q53 - BRD	1000	0.0	7.9	3.3	2.1
	10000	0.8	10.2	5.3	2.6
	20000	0.7	16.9	4.2	3.4
	40000	1.9	7.7	4.7	2.0
	60000	0.9	10.4	4.9	2.4
	100000	0.0	9.2	4.8	2.5
	140000	0.0	8.3	4.4	2.7
Q53 - GDR	1000	0.0	1.9	0.6	0.6
	10000	0.0	3.1	1.5	0.8
	20000	0.0	3.9	1.4	0.9
	40000	0.2	2.9	1.6	0.6
	60000	0.6	3.7	1.9	0.8
	100000	0.5	3.1	2.1	0.6
	140000	0.0	5.1	1.9	1.3
Q53 - GRD	1000	0.0	1.7	0.7	0.5
	10000	0.0	2.9	1.6	0.7
	20000	0.0	4.0	1.5	0.9
	40000	0.0	4.4	1.8	1.1
	60000	1.0	4.9	2.2	1.0
	100000	0.4	3.9	2.2	0.9
	140000	0.1	3.9	2.0	1.0
Q53 - SDR	1000	0.0	1.6	0.7	0.4
	10000	0.0	3.0	1.5	0.8
	20000	0.0	3.4	1.4	0.9
	40000	0.0	4.0	1.8	1.1
	60000	0.4	6.3	2.4	1.4
	100000	0.0	6.3	2.7	1.6
	140000	0.0	6.4	2.7	2.0
Q53 - SRD	1000	0.0	2.0	0.8	0.8
	10000	0.0	3.6	1.6	0.8
	20000	0.0	3.8	1.4	0.9
	40000	0.0	3.0	1.7	0.8
	60000	0.0	4.6	1.9	1.0
	100000	0.2	3.6	1.9	0.9
	140000	0.4	3.3	1.6	0.9
Q54 - BDR	1000	0.1	31.5	12.2	11.6
	10000	0.4	71.9	25.1	22.5
	20000	1.9	96.1	24.2	26.7
	40000	4.5	76.8	43.1	25.5
	60000	4.8	148.9	69.0	48.0
	100000	37.1	241.8	144.1	75.4
	140000	4.1	322.1	106.8	124.8
Q54 - BRD	1000	0.2	32.7	12.2	11.2
	10000	4.6	73.5	29.5	20.4
	20000	2.4	117.2	28.3	32.5

	40000	4.2	49.2	23.1	13.5
	60000	4.6	43.9	24.0	13.5
	100000	9.5	46.2	28.0	12.0
	140000	1.4	43.9	16.7	14.8
Q54 - GDR	1000	0.0	6.8	2.6	2.3
	10000	2.3	19.1	8.3	5.6
	20000	0.0	41.9	9.3	11.9
	40000	1.3	17.0	8.3	5.1
	60000	1.2	19.9	9.4	6.1
	100000	3.5	19.5	11.4	5.2
	140000	1.5	14.5	6.3	4.9
Q54 - GRD	1000	0.0	6.6	2.7	2.5
	10000	1.0	25.5	9.9	7.2
	20000	0.8	40.6	9.8	11.2
	40000	1.3	22.7	9.2	6.3
	60000	2.6	23.5	11.0	6.5
	100000	2.8	18.7	12.7	5.0
	140000	1.7	18.8	7.3	6.3
Q54 - SDR	1000	0.0	6.2	2.5	2.3
	10000	1.3	20.4	8.6	5.9
	20000	0.0	38.5	9.1	11.1
	40000	1.7	20.3	8.9	5.8
	60000	2.4	23.3	11.2	7.3
	100000	3.4	26.2	15.2	7.8
	140000	0.0	31.8	8.8	9.7
Q54 - SRD	1000	0.0	7.6	3.3	2.9
	10000	0.0	23.2	9.0	6.8
	20000	0.0	37.4	9.1	10.7
	40000	1.0	16.8	8.7	5.8
	60000	2.6	20.3	10.0	5.7
	100000	2.5	17.1	10.6	4.6
	140000	0.9	15.5	6.1	5.3
Q55 - BDR	1000	2.9	198.8	69.9	71.5
	10000	7.0	522.8	181.4	153.6
	20000	7.6	948.3	176.3	277.8
	40000	13.4	436.5	219.5	156.6
	60000	7.7	892.4	370.9	298.9
	100000	158.9	1508.8	792.9	497.9
	140000	13.2	1824.4	549.4	707.5
Q55 - BRD	1000	0.6	199.7	69.5	71.5
	10000	4.7	522.8	181.0	153.5
	20000	8.3	1128.9	202.5	331.6
	40000	13.2	296.8	128.4	94.8
	60000	9.2	304.2	138.8	96.7
	100000	56.2	325.7	169.0	93.2

	140000	6.2	294.2	92.3	103.0
Q55 - GDR	1000	0.0	41.4	14.7	15.3
	10000	2.1	155.5	52.7	46.5
	20000	2.7	372.7	65.1	109.8
	40000	5.2	108.7	45.9	34.2
	60000	3.6	123.3	54.0	38.2
	100000	18.7	109.9	67.4	35.8
	140000	2.3	98.1	33.3	35.3
Q55 - GRD	1000	0.0	42.6	15.3	16.2
	10000	1.9	197.4	63.1	58.7
	20000	3.8	373.6	68.0	109.5
	40000	5.1	136.8	52.0	41.4
	60000	3.4	147.3	62.7	43.0
	100000	21.2	118.4	74.8	34.5
	140000	3.2	111.4	38.3	39.1
Q55 - SDR	1000	0.4	45.8	14.4	15.8
	10000	1.1	162.8	54.3	47.7
	20000	1.3	341.7	63.2	100.0
	40000	4.5	131.1	49.2	40.0
	60000	3.8	141.6	63.9	46.7
	100000	16.4	169.2	91.8	56.2
	140000	4.7	166.8	47.2	56.4
Q55 - SRD	1000	0.7	55.3	18.8	20.2
	10000	2.0	185.5	57.0	53.1
	20000	3.6	358.5	66.5	105.0
	40000	3.9	94.7	46.0	33.4
	60000	5.3	140.5	59.7	43.1
	100000	18.9	111.3	62.5	31.5
	140000	2.3	102.4	34.3	37.0
Q63 - BDR	1000	0.5	6.5	3.3	2.1
	10000	1.3	9.1	5.3	2.4
	20000	0.0	15.8	4.2	3.3
	40000	1.0	16.8	8.3	4.4
	60000	2.0	28.5	13.0	7.5
	100000	0.0	41.1	21.7	12.5
	140000	1.3	55.8	25.6	18.8
Q63 - BRD	1000	0.0	6.8	3.3	2.1
	10000	0.8	11.2	5.2	2.9
	20000	1.2	16.8	4.4	3.5
	40000	0.9	7.8	4.6	2.2
	60000	0.5	11.6	4.7	2.9
	100000	1.5	8.9	4.9	2.6
	140000	0.0	9.9	4.5	2.7
Q63 - GDR	1000	0.0	1.4	0.7	0.4

	10000	0.4	3.3	1.5	0.8
	20000	0.0	4.5	1.4	0.9
	40000	0.0	3.2	1.6	0.8
	60000	0.0	4.8	1.9	1.1
	100000	0.0	4.5	1.9	1.3
	140000	0.0	4.8	1.9	1.3
Q63 - GRD	1000	0.0	1.4	0.7	0.4
	10000	0.0	4.5	1.7	1.0
	20000	0.0	4.6	1.5	1.0
	40000	0.3	3.4	1.9	0.9
	60000	0.8	3.5	2.1	0.8
	100000	0.2	3.8	2.1	0.9
	140000	0.1	3.8	2.0	0.9
Q63 - SDR	1000	0.0	1.9	0.7	0.6
	10000	0.0	3.8	1.6	0.9
	20000	0.0	3.7	1.4	1.0
	40000	0.0	4.4	1.8	0.9
	60000	0.0	5.5	2.4	1.8
	100000	0.0	7.7	2.7	1.9
	140000	0.0	6.3	2.8	1.9
Q63 - SRD	1000	0.0	2.1	0.9	0.8
	10000	0.0	4.8	1.8	1.3
	20000	0.0	3.9	1.5	1.0
	40000	0.0	3.0	1.7	0.8
	60000	0.0	4.7	2.0	1.3
	100000	0.7	2.9	2.0	0.7
	140000	0.0	3.4	1.7	1.0
Q64 - BDR	1000	0.7	31.4	12.3	11.2
	10000	2.2	73.5	29.5	20.1
	20000	0.6	122.2	28.0	34.4
	40000	1.6	79.9	43.1	26.5
	60000	4.5	148.6	68.9	47.7
	100000	35.6	244.2	143.9	75.5
	140000	5.4	321.6	106.7	124.0
Q64 - BRD	1000	0.0	29.7	12.2	10.7
	10000	1.9	73.7	29.5	20.3
	20000	2.5	118.6	28.1	33.1
	40000	4.3	49.3	23.2	13.8
	60000	3.7	46.9	24.1	13.4
	100000	12.1	46.2	28.1	11.8
	140000	3.9	46.8	16.8	14.9
Q64 - GDR	1000	0.0	5.6	2.5	2.2
	10000	1.1	20.4	8.3	6.0
	20000	0.5	42.8	9.3	12.2
	40000	1.3	16.8	8.1	5.1

	60000	0.0	19.9	9.4	6.1
	100000	2.7	17.4	11.0	5.0
	140000	1.8	15.9	6.0	5.1
Q64 - GRD	1000	0.0	6.9	2.6	2.4
	10000	0.9	24.6	9.7	7.2
	20000	0.8	40.9	9.8	11.4
	40000	1.1	21.2	9.2	6.3
	60000	1.1	21.6	10.9	6.4
	100000	2.8	18.3	12.3	4.6
	140000	1.0	18.0	7.1	6.0
Q64 - SDR	1000	0.0	6.3	2.5	2.4
	10000	1.1	20.6	8.7	5.7
	20000	0.2	38.5	9.3	10.9
	40000	1.8	23.2	8.9	6.7
	60000	2.5	21.5	11.1	7.2
	100000	3.0	27.4	15.2	8.1
	140000	1.5	30.7	8.8	9.5
Q64 - SRD	1000	0.0	8.5	3.4	3.0
	10000	0.0	27.4	9.5	7.7
	20000	0.6	38.0	9.3	10.6
	40000	2.3	14.2	8.3	4.3
	60000	2.7	20.5	10.2	6.2
	100000	3.8	15.7	10.5	4.0
	140000	1.2	16.0	6.2	5.3
Q65 - BDR	1000	0.3	197.8	73.0	73.5
	10000	7.2	558.8	191.3	163.6
	20000	8.1	1181.6	208.3	347.7
	40000	14.1	456.6	226.6	162.3
	60000	7.5	912.6	383.7	305.3
	100000	161.2	1589.9	814.7	514.3
	140000	13.6	1884.7	560.8	721.2
Q65 - BRD	1000	1.3	196.4	72.6	72.8
	10000	8.9	566.2	192.0	165.3
	20000	8.6	1173.3	207.1	345.1
	40000	13.3	318.9	134.0	99.9
	60000	9.6	313.4	144.1	98.7
	100000	56.3	343.6	174.3	98.1
	140000	7.9	301.4	94.6	105.0
Q65 - GDR	1000	1.1	39.8	14.6	15.0
	10000	1.1	162.9	54.0	48.9
	20000	2.4	387.3	66.1	114.5
	40000	3.1	113.0	46.9	35.3
	60000	2.2	124.6	55.5	38.7
	100000	18.5	111.5	67.6	35.8
	140000	4.5	98.1	33.4	34.7

Q65 - GRD	1000	0.0	41.1	15.4	15.7
	10000	2.1	206.4	64.7	61.0
	20000	3.2	384.8	69.0	112.9
	40000	4.8	144.8	53.8	43.7
	60000	4.6	149.5	64.3	43.0
	100000	19.6	122.4	74.6	34.5
	140000	1.8	112.8	38.0	38.8
Q65 - SDR	1000	0.5	42.5	14.8	15.6
	10000	2.4	171.8	56.2	50.4
	20000	3.5	374.5	66.8	110.3
	40000	5.2	138.8	51.0	41.8
	60000	4.8	145.7	66.7	47.3
	100000	18.8	176.0	94.6	57.6
	140000	3.4	169.9	48.4	58.3
Q65 - SRD	1000	0.0	56.0	20.5	21.7
	10000	1.8	236.1	66.1	67.1
	20000	3.6	392.4	69.9	115.5
	40000	4.9	101.2	47.6	35.5
	60000	2.5	146.4	62.2	44.8
	100000	18.8	121.1	64.9	33.7
	140000	3.2	102.6	34.1	37.0

ISAM Reads of Spatial Queries for the B, G, and S Schemas

Query Type	No. Stars	Minimum	Maximum	Average	Std. Dev.
Q1 - BDR	1000	36.0	57.0	45.0	5.6
	10000	85.0	204.0	156.2	30.4
	20000	156.0	309.0	265.3	46.3
	40000	269.0	670.0	503.0	108.5
	60000	289.0	912.0	713.6	157.7
	100000	377.0	1579.0	1180.6	331.4
	140000	398.0	2114.0	1448.2	564.2
Q1 - BRD	1000	36.0	57.0	45.0	5.6
	10000	85.0	204.0	156.2	30.4
	20000	156.0	309.0	265.3	46.3
	40000	251.0	336.0	293.8	24.1
	60000	262.0	319.0	292.7	16.5
	100000	258.0	327.0	292.5	17.4
	140000	277.0	334.0	303.9	16.8
Q1 - GDR	1000	32.0	40.0	34.9	2.5
	10000	32.0	42.0	35.0	2.3
	20000	32.0	41.0	35.1	2.5
	40000	33.0	38.0	35.1	1.4
	60000	33.0	40.0	35.1	1.7
	100000	33.0	38.0	35.2	1.3
	140000	32.0	39.0	35.5	2.0
Q1 - GRD	1000	32.0	144.0	40.4	24.5
	10000	32.0	42.0	35.0	2.3
	20000	32.0	41.0	35.1	2.5
	40000	33.0	38.0	35.1	1.4
	60000	33.0	40.0	35.1	1.7
	100000	33.0	38.0	35.2	1.3
	140000	32.0	39.0	35.5	2.0
Q1 - SDR	1000	32.0	42.0	35.3	2.9
	10000	32.0	42.0	35.0	2.3
	20000	32.0	41.0	35.1	2.5
	40000	33.0	38.0	35.0	1.3
	60000	33.0	40.0	35.1	1.7
	100000	33.0	38.0	35.2	1.3
	140000	32.0	39.0	35.5	2.0
Q1 - SRD	1000	32.0	42.0	35.3	2.9
	10000	32.0	42.0	35.0	2.3
	20000	32.0	41.0	35.1	2.5
	40000	33.0	38.0	35.0	1.3
	60000	33.0	40.0	35.1	1.7
	100000	33.0	38.0	35.2	1.3
	140000	32.0	39.0	35.5	2.0

Q2 - BDR	1000	437072.0	437125.0	437089.7	30.6
	10000	5173184.0	5173237.0	5173201.7	30.6
	20000	10400790.0	10400843.0	10400807.7	30.6
	40000	35024566.0	35024619.0	35024583.7	30.6
	60000	79022498.0	79022551.0	79022515.7	30.6
	100000	219611502.0	219611555.0	219611519.7	30.6
	140000	428239498.0	428239551.0	428239515.7	30.6
Q2 - BRD	1000	437072.0	437098.0	437080.7	15.0
	10000	5173184.0	5173203.0	5173190.3	11.0
	20000	10400790.0	10400809.0	10400796.3	11.0
	40000	20946108.0	20946127.0	20946114.3	11.0
	60000	31846974.0	31846993.0	31846980.3	11.0
	100000	53113218.0	53113237.0	53113224.3	11.0
	140000	73899996.0	73900015.0	73900002.3	11.0
Q2 - GDR	1000	4544.0	4544.0	4544.0	0.0
	10000	48629.0	48629.0	48629.0	0.0
	20000	96962.0	96962.0	96962.0	0.0
	40000	194594.0	194594.0	194594.0	0.0
	60000	294296.0	294296.0	294296.0	0.0
	100000	491454.0	491454.0	491454.0	0.0
	140000	683663.0	683663.0	683663.0	0.0
Q2 - GRD	1000	4544.0	4547.0	4545.0	1.7
	10000	48629.0	48630.0	48629.3	0.6
	20000	96962.0	96963.0	96962.3	0.6
	40000	194594.0	194595.0	194594.3	0.6
	60000	294296.0	294297.0	294296.3	0.6
	100000	491454.0	491455.0	491454.3	0.6
	140000	683663.0	683664.0	683663.3	0.6
Q2 - SDR	1000	4546.0	4718.0	4603.3	99.3
	10000	48584.0	48635.0	48601.0	29.4
	20000	96872.0	96923.0	96889.0	29.4
	40000	194426.0	194477.0	194443.0	29.4
	60000	294052.0	294103.0	294069.0	29.4
	100000	491040.0	491091.0	491057.0	29.4
	140000	683080.0	683131.0	683097.0	29.4
Q2 - SRD	1000	4546.0	4548.0	4546.7	1.2
	10000	48584.0	48585.0	48584.3	0.6
	20000	96872.0	96873.0	96872.3	0.6
	40000	194426.0	194427.0	194426.3	0.6
	60000	294052.0	294053.0	294052.3	0.6
	100000	491040.0	491041.0	491040.3	0.6
	140000	683080.0	683080.0	683080.0	0.0
Q4 - BDR	1000	47.0	74.0	58.8	6.9
	10000	175.0	375.0	273.3	62.1

	20000	366.0	575.0	488.0	60.1
	40000	470.0	1158.0	922.3	172.7
	60000	929.0	1675.0	1426.8	187.5
	100000	1742.0	2910.0	2269.3	343.6
	140000	1604.0	4081.0	3061.9	819.4
Q4 - BRD	1000	47.0	74.0	58.8	6.9
	10000	175.0	375.0	273.3	62.1
	20000	366.0	575.0	488.0	60.1
	40000	485.0	624.0	554.3	30.8
	60000	508.0	605.0	565.4	26.2
	100000	490.0	597.0	560.2	27.8
	140000	504.0	632.0	569.8	32.6
Q4 - GDR	1000	34.0	49.0	39.5	3.7
	10000	35.0	48.0	39.4	3.2
	20000	35.0	47.0	40.3	3.0
	40000	36.0	44.0	39.8	1.9
	60000	36.0	44.0	39.9	1.9
	100000	37.0	43.0	40.0	1.7
	140000	36.0	45.0	39.6	2.6
Q4 - GRD	1000	34.0	49.0	39.5	3.7
	10000	35.0	48.0	39.4	3.2
	20000	35.0	47.0	40.3	3.0
	40000	36.0	44.0	39.8	1.9
	60000	36.0	44.0	39.9	1.9
	100000	37.0	43.0	40.0	1.7
	140000	36.0	45.0	39.6	2.6
Q4 - SDR	1000	34.0	49.0	39.5	3.7
	10000	35.0	48.0	39.4	3.2
	20000	35.0	47.0	40.3	3.0
	40000	36.0	44.0	39.7	1.9
	60000	36.0	44.0	39.9	1.9
	100000	37.0	43.0	40.0	1.7
	140000	36.0	45.0	39.6	2.6
Q4 - SRD	1000	34.0	49.0	39.5	3.7
	10000	35.0	48.0	39.4	3.2
	20000	35.0	47.0	40.3	3.0
	40000	36.0	44.0	39.7	1.9
	60000	36.0	44.0	39.9	1.9
	100000	37.0	43.0	40.0	1.7
	140000	36.0	45.0	39.6	2.6
Q52 - BDR	1000	914.0	9499.0	5548.5	2327.5
	10000	3529.0	13295.0	8519.2	2682.2
	20000	2933.0	17002.0	7744.1	3025.8
	40000	4199.0	22625.0	15188.1	5490.5
	60000	4550.0	40060.0	22886.9	9586.5

	100000	4425.0	62242.0	38178.8	16911.9
	140000	6121.0	87890.0	47034.6	28128.3
Q52 - BRD	1000	914.0	9499.0	5548.5	2327.5
	10000	3529.0	13295.0	8519.2	2682.2
	20000	2933.0	17002.0	7744.1	3025.8
	40000	4215.0	11731.0	8262.9	2016.5
	60000	4428.0	12507.0	8441.7	2346.7
	100000	3345.0	11746.0	8332.5	2359.7
	140000	4050.0	12536.0	8172.7	2814.1
Q52 - GDR	1000	43.0	274.0	149.0	64.1
	10000	66.0	374.0	203.0	80.1
	20000	63.0	507.0	176.6	93.5
	40000	78.0	278.0	180.9	57.8
	60000	79.0	324.0	190.7	67.6
	100000	63.0	291.0	191.5	67.6
	140000	67.0	285.0	173.6	76.6
Q52 - GRD	1000	43.0	274.0	149.0	64.1
	10000	66.0	374.0	203.0	80.1
	20000	63.0	507.0	176.6	93.5
	40000	78.0	278.0	180.9	57.8
	60000	79.0	324.0	190.7	67.6
	100000	63.0	291.0	191.5	67.6
	140000	67.0	285.0	173.6	76.6
Q52 - SDR	1000	43.0	274.0	148.9	64.1
	10000	66.0	374.0	203.0	80.1
	20000	63.0	507.0	176.5	93.4
	40000	78.0	277.0	180.8	57.7
	60000	79.0	322.0	190.6	67.3
	100000	63.0	291.0	191.3	67.4
	140000	67.0	285.0	173.3	76.5
Q52 - SRD	1000	43.0	274.0	148.9	64.1
	10000	66.0	374.0	203.0	80.1
	20000	63.0	507.0	176.5	93.4
	40000	78.0	277.0	180.8	57.7
	60000	79.0	322.0	190.6	67.3
	100000	63.0	291.0	191.3	67.4
	140000	67.0	285.0	173.3	76.5
Q53 - BDR	1000	2383.0	58886.0	28715.0	15743.9
	10000	9669.0	91908.0	46115.1	21930.9
	20000	8152.0	133069.0	38498.8	26145.9
	40000	12985.0	135765.0	75061.7	36051.5
	60000	12607.0	242082.0	118418.3	65746.9
	100000	11512.0	364591.0	199696.9	111054.0
	140000	16542.0	491158.0	237414.8	172576.6

Q53 - BRD	1000	2383.0	58886.0	28715.0	15743.9
	10000	9669.0	91908.0	46115.1	21930.9
	20000	8152.0	133069.0	38498.8	26145.9
	40000	12852.0	68783.0	40394.8	16223.2
	60000	12631.0	75875.0	42515.4	18061.7
	100000	8694.0	68866.0	42400.3	17911.6
	140000	10228.0	74970.0	39210.5	21313.5
Q53 - GDR	1000	57.0	1488.0	631.4	405.9
	10000	115.0	2359.0	982.3	612.8
	20000	117.0	3918.0	811.7	803.1
	40000	175.0	1502.0	787.7	402.9
	60000	158.0	1781.0	864.7	456.2
	100000	112.0	1556.0	882.8	473.6
	140000	121.0	1566.0	750.1	500.3
Q53 - GRD	1000	57.0	1488.0	631.4	405.9
	10000	115.0	2359.0	982.3	612.8
	20000	117.0	3918.0	811.7	803.1
	40000	175.0	1502.0	787.7	402.9
	60000	158.0	1781.0	864.7	456.2
	100000	112.0	1556.0	882.8	473.6
	140000	121.0	1566.0	750.1	500.3
Q53 - SDR	1000	57.0	1488.0	631.1	405.9
	10000	115.0	2359.0	981.5	612.2
	20000	117.0	3918.0	809.9	801.9
	40000	175.0	1502.0	786.4	402.2
	60000	158.0	1763.0	863.8	454.3
	100000	112.0	1556.0	880.9	471.0
	140000	120.0	1566.0	747.1	499.6
Q53 - SRD	1000	57.0	1488.0	631.1	405.9
	10000	115.0	2359.0	981.5	612.2
	20000	117.0	3918.0	809.9	801.9
	40000	175.0	1502.0	786.4	402.2
	60000	158.0	1763.0	863.8	454.3
	100000	112.0	1556.0	880.9	471.0
	140000	120.0	1566.0	747.1	499.6
Q54 - BDR	1000	5050.0	264946.0	106263.2	94575.2
	10000	23023.0	616155.0	257364.1	174079.2
	20000	27877.0	1083983.0	255327.6	302658.3
	40000	38636.0	657916.0	369510.5	217576.6
	60000	32312.0	1289265.0	596224.6	419671.2
	100000	314738.0	2085271.0	1250809.1	649229.9
	140000	44098.0	2744840.0	925216.4	1075070.4
Q54 - BRD	1000	5050.0	264946.0	106263.2	94575.2
	10000	23023.0	616155.0	257364.1	174079.2
	20000	27877.0	1083983.0	255327.6	302658.3

	40000	37839.0	405048.0	203473.6	116656.0
	60000	32292.0	411351.0	211997.0	122024.7
	100000	102660.0	397805.0	247560.5	101330.8
	140000	24229.0	394552.0	148673.1	132839.7
Q54 - GDR	1000	83.0	6613.0	2429.1	2418.5
	10000	231.0	16694.0	5818.9	4838.9
	20000	507.0	33670.0	6282.3	9824.4
	40000	471.0	9030.0	4117.4	2832.4
	60000	362.0	9696.0	4537.0	3014.2
	100000	1941.0	9956.0	5571.3	2894.9
	140000	259.0	8739.0	2956.2	3154.9
Q54 - GRD	1000	83.0	6613.0	2429.1	2418.5
	10000	231.0	16694.0	5818.9	4838.9
	20000	507.0	33670.0	6282.3	9824.4
	40000	471.0	9030.0	4117.4	2832.4
	60000	362.0	9696.0	4537.0	3014.2
	100000	1941.0	9956.0	5571.3	2894.9
	140000	259.0	8739.0	2956.2	3154.9
Q54 - SDR	1000	83.0	6613.0	2429.1	2418.5
	10000	231.0	16694.0	5818.9	4838.9
	20000	507.0	33670.0	6282.2	9824.4
	40000	471.0	9011.0	4115.5	2828.7
	60000	362.0	9696.0	4537.0	3014.2
	100000	1941.0	9956.0	5564.1	2886.9
	140000	252.0	8739.0	2955.5	3155.6
Q54 - SRD	1000	83.0	6613.0	2429.1	2418.5
	10000	231.0	16694.0	5818.9	4838.9
	20000	507.0	33670.0	6282.2	9824.4
	40000	471.0	9011.0	4115.5	2828.7
	60000	362.0	9696.0	4537.0	3014.2
	100000	1941.0	9956.0	5564.1	2886.9
	140000	252.0	8739.0	2955.5	3155.6
Q55 - BDR	1000	10113.0	1652861.0	589608.0	597024.1
	10000	55051.0	4616352.0	1566044.9	1344366.0
	20000	86232.0	9393225.0	1701062.2	2754275.2
	40000	117597.0	3998939.0	2012730.7	1427757.8
	60000	83761.0	8203487.0	3425688.6	2763309.7
	100000	1508534.0	14307876.0	7382032.7	4666423.0
	140000	126325.0	17248010.0	5152113.4	6663484.4
Q55 - BRD	1000	10113.0	1652861.0	589608.0	597024.1
	10000	55051.0	4616352.0	1566044.9	1344366.0
	20000	86232.0	9393225.0	1701062.2	2754275.2
	40000	114457.0	2542328.0	1104690.2	803115.3
	60000	83466.0	2617045.0	1191010.3	819185.4
	100000	498527.0	2733497.0	1440585.0	773576.3

	140000	59605.0	2480557.0	787516.8	863489.6
Q55 - GDR	1000	133.0	42976.0	14316.1	15871.8
	10000	523.0	128928.0	37703.9	38382.0
	20000	1628.0	303889.0	46403.4	91426.5
	40000	1404.0	59189.0	23563.5	19654.3
	60000	909.0	63008.0	26699.4	20614.0
	100000	9414.0	72062.0	34378.5	22312.2
	140000	635.0	55726.0	16711.2	20474.3
Q55 - GRD	1000	133.0	42976.0	14316.1	15871.8
	10000	523.0	128928.0	37703.9	38382.0
	20000	1628.0	303889.0	46403.4	91426.5
	40000	1404.0	59189.0	23563.5	19654.3
	60000	909.0	63008.0	26699.4	20614.0
	100000	9414.0	72062.0	34378.5	22312.2
	140000	635.0	55726.0	16711.2	20474.3
Q55 - SDR	1000	133.0	42976.0	14316.1	15871.8
	10000	523.0	128928.0	37703.9	38382.0
	20000	1628.0	303889.0	46402.3	91427.0
	40000	1404.0	59122.0	23556.8	19640.8
	60000	909.0	63008.0	26699.4	20614.0
	100000	9414.0	72062.0	34333.2	22266.5
	140000	603.0	55726.0	16708.0	20477.1
Q55 - SRD	1000	133.0	42976.0	14316.1	15871.8
	10000	523.0	128928.0	37703.9	38382.0
	20000	1628.0	303889.0	46402.3	91427.0
	40000	1404.0	59122.0	23556.8	19640.8
	60000	909.0	63008.0	26699.4	20614.0
	100000	9414.0	72062.0	34333.2	22266.5
	140000	603.0	55726.0	16708.0	20477.1
Q63 - BDR	1000	2391.0	59080.0	28754.3	15760.1
	10000	9667.0	92115.0	46129.6	21990.6
	20000	8218.0	133390.0	38512.3	26194.8
	40000	12915.0	135653.0	75186.7	36204.6
	60000	12617.0	242162.0	118380.1	65718.8
	100000	11502.0	364809.0	199625.4	111066.3
	140000	16559.0	490307.0	237426.3	172552.8
Q63 - BRD	1000	2391.0	59080.0	28754.3	15760.1
	10000	9667.0	92115.0	46129.6	21990.6
	20000	8218.0	133390.0	38512.3	26194.8
	40000	12795.0	68762.0	40406.4	16217.7
	60000	12623.0	75705.0	42538.4	18098.4
	100000	8696.0	68905.0	42400.6	17912.2
	140000	10231.0	74964.0	39219.6	21327.2
Q63 - GDR	1000	57.0	1471.0	630.6	402.6

	10000	115.0	2349.0	982.6	612.7
	20000	115.0	3923.0	805.9	803.8
	40000	176.0	1495.0	788.9	404.3
	60000	158.0	1815.0	867.0	459.9
	100000	113.0	1561.0	881.9	471.7
	140000	121.0	1563.0	748.8	499.2
Q63 - GRD	1000	57.0	1471.0	630.6	402.6
	10000	115.0	2349.0	982.6	612.7
	20000	115.0	3923.0	805.9	803.8
	40000	176.0	1495.0	788.9	404.3
	60000	158.0	1815.0	867.0	459.9
	100000	113.0	1561.0	881.9	471.7
	140000	121.0	1563.0	748.8	499.2
Q63 - SDR	1000	57.0	1471.0	630.3	402.6
	10000	115.0	2349.0	982.2	612.3
	20000	115.0	3923.0	804.1	802.6
	40000	176.0	1492.0	787.5	403.4
	60000	158.0	1797.0	866.1	458.0
	100000	113.0	1561.0	880.0	469.1
	140000	121.0	1563.0	746.0	498.3
Q63 - SRD	1000	57.0	1471.0	630.3	402.6
	10000	115.0	2349.0	982.2	612.3
	20000	115.0	3923.0	804.1	802.6
	40000	176.0	1492.0	787.5	403.4
	60000	158.0	1797.0	866.1	458.0
	100000	113.0	1561.0	880.0	469.1
	140000	121.0	1563.0	746.0	498.3
Q64 - BDR	1000	5074.0	262190.0	107393.9	94898.5
	10000	23050.0	623783.0	258942.3	175249.5
	20000	27394.0	1088284.0	254421.2	303900.6
	40000	38571.0	659215.0	369981.7	218111.8
	60000	32346.0	1286458.0	595009.6	416962.1
	100000	314915.0	2086515.0	1248322.4	646027.1
	140000	43811.0	2752806.0	923091.4	1071449.5
Q64 - BRD	1000	5074.0	262190.0	107393.9	94898.5
	10000	23050.0	623783.0	258942.3	175249.5
	20000	27394.0	1088284.0	254421.2	303900.6
	40000	37905.0	411553.0	204494.8	117726.3
	60000	32236.0	410423.0	211786.7	121194.9
	100000	102821.0	397859.0	247246.2	100920.9
	140000	24506.0	396339.0	148618.9	132761.1
Q64 - GDR	1000	83.0	6521.0	2504.1	2464.7
	10000	234.0	17696.0	6095.8	5116.2
	20000	518.0	35007.0	6421.1	10228.7
	40000	486.0	9556.0	4259.8	2966.7

	60000	365.0	9867.0	4691.6	3064.6
	100000	1975.0	10449.0	5712.4	2981.7
	140000	259.0	9019.0	3022.2	3221.0
Q64 - GRD	1000	83.0	6521.0	2504.1	2464.7
	10000	234.0	17696.0	6095.8	5116.2
	20000	518.0	35007.0	6421.1	10228.7
	40000	486.0	9556.0	4259.8	2966.7
	60000	365.0	9867.0	4691.6	3064.6
	100000	1975.0	10449.0	5712.4	2981.7
	140000	259.0	9019.0	3022.2	3221.0
Q64 - SDR	1000	83.0	6521.0	2504.1	2464.7
	10000	234.0	17696.0	6095.8	5116.2
	20000	518.0	35007.0	6421.1	10228.7
	40000	486.0	9531.0	4257.3	2961.7
	60000	365.0	9867.0	4691.6	3064.6
	100000	1975.0	10449.0	5707.9	2977.0
	140000	259.0	9019.0	3022.2	3221.0
Q64 - SRD	1000	83.0	6521.0	2504.1	2464.7
	10000	234.0	17696.0	6095.8	5116.2
	20000	518.0	35007.0	6421.1	10228.7
	40000	486.0	9531.0	4257.3	2961.7
	60000	365.0	9867.0	4691.6	3064.6
	100000	1975.0	10449.0	5707.9	2977.0
	140000	259.0	9019.0	3022.2	3221.0
Q65 - BDR	1000	10169.0	1643375.0	616576.7	614831.2
	10000	56009.0	4919569.0	1645761.0	1429235.7
	20000	85612.0	9781216.0	1742530.5	2872115.2
	40000	120205.0	4112154.0	2078501.1	1480699.7
	60000	84597.0	8382456.0	3541842.4	2818500.1
	100000	1538659.0	15021426.0	7569339.3	4810282.7
	140000	126159.0	17829008.0	5268063.5	6805601.9
Q65 - BRD	1000	10169.0	1643375.0	616576.7	614831.2
	10000	56009.0	4919569.0	1645761.0	1429235.7
	20000	85612.0	9781216.0	1742530.5	2872115.2
	40000	117747.0	2723662.0	1149151.7	846178.2
	60000	84005.0	2669487.0	1233993.5	833724.7
	100000	509754.0	2864882.0	1478420.1	800057.0
	140000	61503.0	2569950.0	808019.0	885827.8
Q65 - GDR	1000	133.0	43464.0	15647.9	17025.3
	10000	547.0	149442.0	42981.5	44769.8
	20000	1806.0	344459.0	51072.5	103941.9
	40000	1520.0	68121.0	25970.5	22162.6
	60000	936.0	67354.0	29788.4	22233.1
	100000	9968.0	83669.0	37335.2	24985.4
	140000	637.0	60811.0	18022.9	22044.7

Q65 - GRD	1000	133.0	43464.0	15647.9	17025.3
	10000	547.0	149442.0	42981.5	44769.8
	20000	1806.0	344459.0	51072.5	103941.9
	40000	1520.0	68121.0	25970.5	22162.6
	60000	936.0	67354.0	29788.4	22233.1
	100000	9968.0	83669.0	37335.2	24985.4
	140000	637.0	60811.0	18022.9	22044.7
Q65 - SDR	1000	133.0	43464.0	15647.9	17025.3
	10000	547.0	149442.0	42981.5	44769.8
	20000	1806.0	344459.0	51072.5	103941.9
	40000	1520.0	68031.0	25961.5	22143.6
	60000	936.0	67354.0	29788.4	22233.1
	100000	9968.0	83669.0	37313.2	24967.0
	140000	637.0	60811.0	18022.9	22044.7
Q65 - SRD	1000	133.0	43464.0	15647.9	17025.3
	10000	547.0	149442.0	42981.5	44769.8
	20000	1806.0	344459.0	51072.5	103941.9
	40000	1520.0	68031.0	25961.5	22143.6
	60000	936.0	67354.0	29788.4	22233.1
	100000	9968.0	83669.0	37313.2	24967.0
	140000	637.0	60811.0	18022.9	22044.7

Buffer Reads of Spatial Queries for the B, G, and S Schemas

Query Type	No. Stars	Minimum	Maximum	Average	Std. Dev.
Q1 - BDR	1000	135.0	196.0	155.9	14.0
	10000	237.0	486.0	384.7	65.0
	20000	386.0	751.0	630.2	108.7
	40000	609.0	1425.0	1093.7	222.9
	60000	660.0	1937.0	1529.4	324.1
	100000	842.0	3298.0	2485.0	676.6
	140000	884.0	4417.0	3033.2	1154.8
Q1 - BRD	1000	136.0	198.0	159.2	14.5
	10000	263.0	507.0	413.8	62.9
	20000	413.0	715.0	621.2	84.8
	40000	567.0	757.0	666.3	51.4
	60000	593.0	727.0	667.1	35.0
	100000	585.0	743.0	662.1	39.4
	140000	623.0	757.0	690.9	37.7
Q1 - GDR	1000	141.0	151.0	145.3	3.1
	10000	151.0	175.0	161.4	5.7
	20000	160.0	186.0	172.5	6.5
	40000	163.0	190.0	178.8	8.2
	60000	166.0	193.0	181.2	6.4
	100000	170.0	202.0	182.7	8.5
	140000	169.0	208.0	185.5	10.6
Q1 - GRD	1000	143.0	371.0	158.8	50.1
	10000	155.0	183.0	168.2	7.4
	20000	164.0	190.0	176.7	7.5
	40000	171.0	208.0	188.8	9.3
	60000	174.0	200.0	189.0	8.1
	100000	171.0	207.0	189.6	10.7
	140000	164.0	211.0	192.5	11.1
Q1 - SDR	1000	141.0	189.0	150.4	10.4
	10000	155.0	175.0	162.8	5.0
	20000	154.0	180.0	163.5	6.1
	40000	157.0	188.0	170.7	9.8
	60000	164.0	208.0	181.8	10.9
	100000	169.0	229.0	196.3	15.8
	140000	172.0	252.0	202.8	25.6
Q1 - SRD	1000	141.0	191.0	159.7	11.3
	10000	157.0	176.0	168.5	5.2
	20000	156.0	175.0	166.8	5.2
	40000	160.0	179.0	167.4	5.2
	60000	162.0	182.0	172.8	6.1
	100000	166.0	186.0	176.2	6.4
	140000	167.0	190.0	176.8	5.0

Q2 - BDR	1000	919187.0	919383.0	919252.3	113.2
	10000	11245257.0	11245451.0	11245321.7	112.0
	20000	22616162.0	22616357.0	22616227.0	112.6
	40000	72318806.0	72319001.0	72318871.0	112.6
	60000	162914569.0	162914758.0	162914632.0	109.1
	100000	451416551.0	451416745.0	451416615.7	112.0
	140000	879193568.0	879193763.0	879193633.0	112.6
Q2 - BRD	1000	907858.0	907951.0	907889.0	53.7
	10000	10744365.0	10744426.0	10744385.3	35.2
	20000	21604574.0	21604632.0	21604593.3	33.5
	40000	43502815.0	43502876.0	43502835.3	35.2
	60000	66368387.0	66368448.0	66368407.3	35.2
	100000	110684066.0	110684122.0	110684084.7	32.3
	140000	154025012.0	154025073.0	154025032.3	35.2
Q2 - GDR	1000	26412.0	26412.0	26412.0	0.0
	10000	429053.0	429053.0	429053.0	0.0
	20000	1091954.0	1091954.0	1091954.0	0.0
	40000	2349136.0	2349136.0	2349136.0	0.0
	60000	3715424.0	3715424.0	3715424.0	0.0
	100000	6642590.0	6642590.0	6642590.0	0.0
	140000	9428507.0	9428507.0	9428507.0	0.0
Q2 - GRD	1000	29144.0	29159.0	29149.0	8.7
	10000	459097.0	459104.0	459099.3	4.0
	20000	1180248.0	1180253.0	1180249.7	2.9
	40000	2636032.0	2636037.0	2636033.7	2.9
	60000	4119698.0	4119703.0	4119699.7	2.9
	100000	6917960.0	6917965.0	6917961.7	2.9
	140000	9883209.0	9883214.0	9883210.7	2.9
Q2 - SDR	1000	27034.0	27645.0	27237.7	352.8
	10000	398776.0	398958.0	398836.7	105.1
	20000	838602.0	838788.0	838664.0	107.4
	40000	1754432.0	1754610.0	1754491.3	102.8
	60000	3441054.0	3441234.0	3441114.0	103.9
	100000	5730600.0	5730776.0	5730658.7	101.6
	140000	8288086.0	8288266.0	8288146.0	103.9
Q2 - SRD	1000	27328.0	27338.0	27331.3	5.8
	10000	443198.0	443203.0	443199.7	2.9
	20000	883366.0	883371.0	883367.7	2.9
	40000	1766312.0	1766317.0	1766313.7	2.9
	60000	3060130.0	3060135.0	3060131.7	2.9
	100000	5390410.0	5390415.0	5390411.7	2.9
	140000	7562354.0	7562354.0	7562354.0	0.0
Q4 - BDR	1000	174.0	232.0	201.6	15.5
	10000	443.0	843.0	642.0	125.6

	20000	854.0	1272.0	1113.5	133.7
	40000	1040.0	2441.0	1970.0	354.2
	60000	1987.0	3517.0	3010.5	385.7
	100000	3660.0	6042.0	4734.6	703.4
	140000	3378.0	8445.0	6358.2	1674.9
Q4 - BRD	1000	177.0	242.0	208.2	17.7
	10000	490.0	910.0	700.5	131.2
	20000	858.0	1276.0	1121.6	117.1
	40000	1054.0	1356.0	1216.5	66.5
	60000	1128.0	1338.0	1243.1	53.5
	100000	1072.0	1322.0	1233.8	58.6
	140000	1124.0	1389.0	1256.6	65.4
Q4 - GDR	1000	170.0	193.0	184.0	5.4
	10000	197.0	234.0	215.7	7.6
	20000	223.0	264.0	243.3	10.6
	40000	222.0	267.0	245.9	12.8
	60000	235.0	278.0	255.0	11.3
	100000	235.0	289.0	260.6	12.5
	140000	233.0	295.0	266.9	15.0
Q4 - GRD	1000	176.0	201.0	189.0	6.4
	10000	205.0	242.0	225.9	10.7
	20000	229.0	268.0	245.8	11.1
	40000	237.0	302.0	264.7	16.2
	60000	239.0	294.0	267.1	14.6
	100000	255.0	295.0	271.9	11.4
	140000	245.0	310.0	277.6	15.9
Q4 - SDR	1000	174.0	207.0	189.4	8.1
	10000	196.0	237.0	216.6	9.7
	20000	204.0	240.0	224.3	10.0
	40000	210.0	254.0	232.6	12.2
	60000	227.0	292.0	255.3	17.2
	100000	241.0	313.0	281.5	20.3
	140000	245.0	356.0	305.7	34.8
Q4 - SRD	1000	174.0	229.0	203.6	15.4
	10000	209.0	244.0	227.9	9.3
	20000	214.0	240.0	227.7	7.5
	40000	217.0	244.0	228.2	8.6
	60000	226.0	260.0	240.8	9.1
	100000	234.0	268.0	245.9	8.1
	140000	232.0	263.0	246.4	9.0
Q52 - BDR	1000	1992.0	20424.0	11927.1	4997.4
	10000	7946.0	29868.0	19145.4	6034.9
	20000	6633.0	38130.0	17420.1	6806.4
	40000	8855.0	46963.0	31595.3	11339.8
	60000	9656.0	82945.0	47484.9	19775.0

	100000	9343.0	128224.0	78775.8	34766.8
	140000	12879.0	180701.0	96862.0	57747.3
Q52 - BRD	1000	1963.0	20050.0	11720.5	4905.8
	10000	7538.0	28055.0	18022.9	5656.1
	20000	6294.0	35792.0	16361.8	6362.7
	40000	8905.0	24709.0	17419.2	4240.4
	60000	9364.0	26561.0	17928.8	4993.6
	100000	7118.0	24868.0	17705.2	5020.1
	140000	8644.0	26513.0	17340.2	5961.6
Q52 - GDR	1000	242.0	1295.0	762.3	284.0
	10000	625.0	2535.0	1527.5	494.7
	20000	700.0	3566.0	1699.9	645.2
	40000	863.0	2683.0	1875.3	496.3
	60000	986.0	3151.0	2044.3	575.0
	100000	790.0	3484.0	2082.7	683.2
	140000	902.0	3380.0	2036.4	822.4
Q52 - GRD	1000	254.0	1451.0	840.3	319.1
	10000	685.0	2797.0	1723.4	573.2
	20000	732.0	3470.0	1778.5	615.3
	40000	1083.0	3587.0	2157.6	594.6
	60000	1180.0	3435.0	2286.3	624.3
	100000	840.0	3568.0	2267.4	685.6
	140000	1032.0	3470.0	2147.6	752.0
Q52 - SDR	1000	242.0	1465.0	850.9	333.2
	10000	633.0	2661.0	1558.7	505.5
	20000	582.0	2940.0	1445.4	560.7
	40000	697.0	2570.0	1650.7	464.6
	60000	932.0	4471.0	2110.5	818.0
	100000	754.0	4822.0	2523.2	973.0
	140000	960.0	5378.0	2720.4	1448.7
Q52 - SRD	1000	254.0	1863.0	1083.9	449.1
	10000	714.0	3015.0	1792.0	553.8
	20000	620.0	2922.0	1497.1	521.2
	40000	751.0	2293.0	1562.2	367.8
	60000	921.0	2627.0	1773.6	515.1
	100000	672.0	2944.0	1877.8	601.9
	140000	844.0	3018.0	1736.0	632.4
Q53 - BDR	1000	5101.0	126502.0	61495.0	33805.3
	10000	21507.0	205984.0	103628.4	49319.8
	20000	18378.0	298395.0	86774.2	58684.7
	40000	27339.0	281705.0	156046.0	74666.5
	60000	26734.0	501181.0	245552.2	135820.1
	100000	24305.0	751033.0	411906.9	228543.8
	140000	34804.0	1010504.0	488745.9	354650.2

Q53 - BRD	1000	5016.0	124127.0	60395.2	33179.6
	10000	20357.0	193197.0	97168.6	46152.2
	20000	17283.0	279855.0	81199.1	55015.8
	40000	27073.0	145206.0	85092.0	34175.5
	60000	26920.0	161322.0	90301.0	38388.0
	100000	18471.0	145711.0	90063.9	38106.9
	140000	21847.0	158555.0	83180.6	45167.0
Q53 - GDR	1000	404.0	7041.0	3275.2	1835.4
	10000	1452.0	16896.0	7760.2	3971.0
	20000	1748.0	27897.0	8240.5	5556.1
	40000	2566.0	15474.0	8991.6	3718.2
	60000	2659.0	19596.0	10192.4	4446.6
	100000	1993.0	20153.0	10495.9	4839.5
	140000	2404.0	19779.0	9715.5	5771.7
Q53 - GRD	1000	430.0	7997.0	3671.0	2087.5
	10000	1582.0	18620.0	8820.0	4595.1
	20000	1844.0	27305.0	8605.5	5419.6
	40000	3236.0	20632.0	10410.6	4448.6
	60000	3245.0	21094.0	11389.6	4672.5
	100000	2157.0	20759.0	11457.4	4881.3
	140000	2780.0	20611.0	10180.7	5524.3
Q53 - SDR	1000	402.0	8221.0	3727.8	2142.0
	10000	1478.0	17588.0	7915.5	4045.0
	20000	1418.0	22665.0	6914.0	4620.2
	40000	2014.0	14583.0	7820.2	3367.1
	60000	2497.0	27322.0	10603.4	5801.1
	100000	1903.0	27878.0	12792.4	6585.1
	140000	2419.0	29839.0	13248.1	9281.3
Q53 - SRD	1000	416.0	10321.0	4881.1	2816.4
	10000	1828.0	20006.0	9136.2	4576.5
	20000	1518.0	22599.0	7158.7	4516.4
	40000	2166.0	12313.0	7361.3	2892.3
	60000	2527.0	15910.0	8741.3	3775.4
	100000	1681.0	16993.0	9417.5	4257.5
	140000	2089.0	16551.0	8200.6	4560.1
Q54 - BDR	1000	10750.0	568919.0	226302.3	201627.3
	10000	50993.0	1396841.0	579284.7	392899.4
	20000	62712.0	2431312.0	574398.2	678344.5
	40000	81290.0	1365810.0	768280.7	451678.0
	60000	68493.0	2668775.0	1236132.6	867503.4
	100000	651913.0	4296654.0	2578560.5	1335776.9
	140000	92811.0	5645162.0	1904610.9	2209727.7
Q54 - BRD	1000	10561.0	558537.0	222353.6	198098.9
	10000	48276.0	1299558.0	542029.6	366584.8
	20000	58381.0	2279402.0	537799.8	636277.6

	40000	79629.0	853505.0	428313.4	245433.6
	60000	68906.0	873932.0	450419.8	259195.7
	100000	217966.0	839985.0	526104.5	215286.3
	140000	51776.0	834294.0	315748.8	281420.2
Q54 - GDR	1000	692.0	30190.0	11731.7	10648.2
	10000	3252.0	101281.0	42113.5	29560.2
	20000	6640.0	229711.0	52807.9	64380.1
	40000	7524.0	90833.0	44920.4	26744.8
	60000	6753.0	100953.0	50030.2	29328.5
	100000	21584.0	90039.0	57277.1	23155.0
	140000	5710.0	80024.0	31629.4	27712.2
Q54 - GRD	1000	744.0	32566.0	13304.1	11943.9
	10000	3518.0	131761.0	50258.7	38197.9
	20000	7978.0	225493.0	53861.3	62638.5
	40000	9498.0	119661.0	51627.6	32379.7
	60000	8267.0	115045.0	57050.6	32252.4
	100000	23284.0	91601.0	64429.1	22508.2
	140000	6694.0	90426.0	36474.2	30511.5
Q54 - SDR	1000	686.0	34562.0	13082.7	12087.6
	10000	3334.0	107129.0	42773.1	30325.3
	20000	4954.0	185311.0	44234.0	52135.7
	40000	5822.0	85258.0	37344.1	24111.5
	60000	6301.0	96211.0	49490.2	30896.0
	100000	18322.0	130041.0	73031.5	37494.5
	140000	5637.0	137237.0	41151.3	42600.5
Q54 - SRD	1000	704.0	46212.0	17708.5	16586.1
	10000	4106.0	134265.0	48379.1	36466.1
	20000	6294.0	185657.0	46438.2	51474.6
	40000	6270.0	69927.0	36725.9	21017.7
	60000	6377.0	90897.0	44530.2	26864.2
	100000	21586.0	76775.0	51729.7	19570.0
	140000	4923.0	76018.0	30545.7	27227.3
Q55 - BDR	1000	21479.0	3547213.0	1254346.6	1271760.3
	10000	121662.0	10470473.0	3523304.2	3037038.0
	20000	194173.0	21077275.0	3823120.9	6177778.9
	40000	247340.0	8302754.0	4184247.3	2966415.1
	60000	177527.0	16981265.0	7099736.7	5713615.6
	100000	3124335.0	29481502.0	15216199.8	9605464.0
	140000	265998.0	35473556.0	10602962.0	##########
Q55 - BRD	1000	21090.0	3482710.0	1232585.4	1249600.2
	10000	115211.0	9736831.0	3296610.9	2831201.4
	20000	180506.0	19753347.0	3581219.9	5791139.5
	40000	240771.0	5358092.0	2324773.8	1689168.2
	60000	178267.0	5558598.0	2529615.1	1738924.9
	100000	1057805.0	5766984.0	3061139.3	1640954.2

	140000	127379.0	5243491.0	1672742.5	1830035.6
Q55 - GDR	1000	1230.0	189283.0	65194.1	67988.6
	10000	7558.0	758751.0	258742.5	227272.5
	20000	21005.0	2006604.0	354943.8	590084.0
	40000	22803.0	569402.0	245032.9	180366.5
	60000	17444.0	642185.0	281919.0	196877.6
	100000	103499.0	551133.0	332863.7	172133.5
	140000	14208.0	503183.0	166731.8	179468.5
Q55 - GRD	1000	1332.0	204367.0	74203.9	76377.9
	10000	8140.0	987463.0	310258.9	291322.8
	20000	25443.0	1972306.0	358810.4	577332.1
	40000	28815.0	741308.0	280953.3	219379.4
	60000	21284.0	734873.0	318919.0	214769.9
	100000	111621.0	617889.0	372285.1	173725.6
	140000	16784.0	568003.0	190508.4	197363.6
Q55 - SDR	1000	1216.0	218475.0	73088.9	77431.0
	10000	7790.0	807889.0	263174.3	235821.5
	20000	15573.0	1610030.0	294859.9	472973.9
	40000	17501.0	533289.0	204160.4	163050.5
	60000	16184.0	612789.0	279109.8	203511.9
	100000	87681.0	816182.0	430976.2	270339.8
	140000	13656.0	765098.0	218301.4	260188.6
Q55 - SRD	1000	1242.0	287501.0	98925.9	104402.6
	10000	9538.0	1014513.0	299118.1	286922.4
	20000	19735.0	1622456.0	306737.3	473424.5
	40000	18861.0	429719.0	200037.4	144955.5
	60000	16378.0	582693.0	250024.4	178311.9
	100000	103539.0	534073.0	300123.4	150957.1
	140000	12096.0	479697.0	162985.2	177160.9
Q63 - BDR	1000	5117.0	126783.0	61578.4	33833.7
	10000	21495.0	206954.0	103669.8	49571.1
	20000	18602.0	299230.0	86828.6	58816.7
	40000	27212.0	281434.0	156312.7	74973.1
	60000	26751.0	501325.0	245470.1	135763.7
	100000	24289.0	751484.0	411759.4	228564.1
	140000	34846.0	1008750.0	488758.9	354589.2
Q63 - BRD	1000	5032.0	124415.0	60478.4	33208.9
	10000	20349.0	193801.0	97201.2	46323.3
	20000	17467.0	280641.0	81243.1	55143.2
	40000	26995.0	145130.0	85126.5	34153.3
	60000	26900.0	161030.0	90339.4	38455.8
	100000	18491.0	145861.0	90067.7	38113.3
	140000	21857.0	158563.0	83183.5	45195.9
Q63 - GDR	1000	404.0	7026.0	3275.6	1832.6

	10000	1446.0	16868.0	7759.3	3972.3
	20000	1750.0	27956.0	8232.9	5566.7
	40000	2543.0	15448.0	8990.8	3714.8
	60000	2659.0	19644.0	10199.5	4450.7
	100000	1992.0	20216.0	10494.9	4848.3
	140000	2403.0	19770.0	9715.2	5771.9
Q63 - GRD	1000	430.0	7976.0	3670.3	2082.0
	10000	1576.0	18629.0	8814.6	4593.9
	20000	1828.0	27312.0	8598.7	5424.7
	40000	3243.0	20660.0	10421.7	4455.9
	60000	3245.0	21118.0	11392.8	4663.9
	100000	2158.0	20808.0	11457.8	4890.1
	140000	2777.0	20584.0	10181.5	5521.7
Q63 - SDR	1000	404.0	8158.0	3727.5	2133.8
	10000	1478.0	17568.0	7918.7	4043.7
	20000	1418.0	22680.0	6915.0	4623.8
	40000	2015.0	14652.0	7829.4	3378.3
	60000	2497.0	27466.0	10602.3	5816.7
	100000	1902.0	28064.0	12798.1	6608.7
	140000	2418.0	29832.0	13255.4	9289.6
Q63 - SRD	1000	416.0	10332.0	4888.4	2818.9
	10000	1830.0	20148.0	9147.1	4596.1
	20000	1514.0	22620.0	7148.8	4518.5
	40000	2173.0	12283.0	7354.0	2883.2
	60000	2533.0	15922.0	8750.1	3783.6
	100000	1684.0	16988.0	9422.6	4255.3
	140000	2090.0	16560.0	8192.9	4557.8
Q64 - BDR	1000	10798.0	565255.0	228895.8	202637.1
	10000	50962.0	1418809.0	582705.2	396768.1
	20000	61702.0	2441886.0	572730.0	681276.2
	40000	81281.0	1368003.0	769337.6	452771.5
	60000	68540.0	2662973.0	1233502.2	861934.1
	100000	652233.0	4299000.0	2573423.3	1329053.2
	140000	92229.0	5661521.0	1900194.2	2202187.0
Q64 - BRD	1000	10609.0	554919.0	224914.2	199105.5
	10000	48290.0	1317298.0	545269.2	369530.2
	20000	57399.0	2290086.0	536185.4	639354.0
	40000	79929.0	866773.0	430464.9	247558.7
	60000	68802.0	871320.0	450042.9	257222.7
	100000	218323.0	839369.0	525325.3	214402.6
	140000	52386.0	838372.0	315556.4	281260.2
Q64 - GDR	1000	695.0	29867.0	11832.7	10661.0
	10000	3242.0	103018.0	42523.8	29904.3
	20000	6614.0	232007.0	52776.9	65079.2
	40000	7500.0	91954.0	45082.2	26937.0

	60000	6753.0	100933.0	50190.4	29195.9
	100000	21663.0	89741.0	57476.2	23222.5
	140000	5715.0	80507.0	31639.6	27693.2
Q64 - GRD	1000	747.0	31859.0	13385.1	11913.3
	10000	3498.0	133690.0	50739.6	38616.9
	20000	7946.0	227403.0	53833.7	63217.9
	40000	9600.0	121486.0	52008.2	32899.1
	60000	8279.0	114367.0	57102.0	31927.5
	100000	23335.0	91323.0	64561.0	22410.4
	140000	6687.0	90991.0	36494.2	30473.4
Q64 - SDR	1000	695.0	34197.0	13217.7	12119.6
	10000	3346.0	109276.0	43245.8	30762.1
	20000	4924.0	186877.0	44238.1	52572.1
	40000	5874.0	86861.0	37608.3	24511.3
	60000	6315.0	95223.0	49507.0	30663.4
	100000	18407.0	130668.0	73153.1	37511.2
	140000	5701.0	137168.0	41236.4	42646.5
Q64 - SRD	1000	707.0	45823.0	17964.7	16707.4
	10000	4128.0	137038.0	48993.8	37136.6
	20000	6224.0	187375.0	46330.3	51961.5
	40000	6368.0	70214.0	36834.5	21152.9
	60000	6417.0	91111.0	44675.4	26810.1
	100000	21625.0	77755.0	51894.9	19597.4
	140000	4985.0	76613.0	30550.0	27213.9
Q65 - BDR	1000	21591.0	3545531.0	1312829.2	1311771.3
	10000	123377.0	11216658.0	3703074.5	3244666.9
	20000	193417.0	21961822.0	3918084.5	6445952.2
	40000	253436.0	8569146.0	4321648.5	3076786.5
	60000	179193.0	17351071.0	7339275.6	5827497.5
	100000	3186321.0	30947726.0	15601961.7	9900512.5
	140000	265720.0	36668848.0	10841504.6	##########
Q65 - BRD	1000	21202.0	3481602.0	1290272.7	1289248.1
	10000	117031.0	10397105.0	3463965.9	3015556.8
	20000	179442.0	20595691.0	3671520.5	6046949.8
	40000	248199.0	5736031.0	2417759.3	1778760.6
	60000	179534.0	5668099.0	2622037.4	1769335.6
	100000	1081841.0	6033265.0	3140411.8	1695630.5
	140000	131503.0	5433709.0	1715642.4	1877263.2
Q65 - GDR	1000	1246.0	188643.0	68098.1	69982.1
	10000	7668.0	816967.0	274713.7	244726.7
	20000	21631.0	2121766.0	367385.5	625336.1
	40000	23369.0	605328.0	254457.7	189740.0
	60000	17599.0	658891.0	294503.4	201372.0
	100000	106129.0	569754.0	344613.4	179914.0
	140000	14238.0	523700.0	171505.1	184481.6

Q65 - GRD	1000	1348.0	200361.0	77203.5	78343.1
	10000	8198.0	1058249.0	329119.7	312502.1
	20000	26197.0	2085666.0	371363.9	611919.4
	40000	29981.0	790872.0	293680.1	233562.0
	60000	21557.0	750085.0	331908.6	217911.5
	100000	113985.0	652514.0	384260.6	179534.0
	140000	16764.0	590202.0	195955.7	202519.5
Q65 - SDR	1000	1246.0	217205.0	76479.7	79643.3
	10000	7976.0	874669.0	280067.9	255401.1
	20000	16031.0	1700046.0	305018.3	500221.7
	40000	18153.0	570584.0	213250.1	173155.5
	60000	16417.0	621777.0	290393.8	208031.5
	100000	90237.0	858011.0	445210.8	281828.3
	140000	14116.0	779381.0	224956.9	266750.1
Q65 - SRD	1000	1258.0	288071.0	104267.1	108403.8
	10000	9792.0	1097797.0	318946.3	311000.4
	20000	19957.0	1717410.0	316776.7	502180.7
	40000	19771.0	448803.0	207688.5	152048.0
	60000	16701.0	599939.0	261111.6	182784.9
	100000	105753.0	573102.0	310591.2	157993.4
	140000	12512.0	500254.0	167763.3	182399.5

Page Reads of Spatial Queries for the B, G, and S Schemas

Query Type	No. Stars	Minimum	Maximum	Average	Std. Dev.
Q1 - BDR	1000	0.0	35.0	7.2	7.7
	10000	8.0	111.0	15.5	22.6
	20000	0.0	158.0	40.9	41.8
	40000	8.0	35.0	16.5	5.3
	60000	7.0	40.0	19.9	7.3
	100000	15.0	41.0	30.5	6.8
	140000	15.0	59.0	35.2	11.2
Q1 - BRD	1000	0.0	35.0	3.0	8.0
	10000	25.0	96.0	80.8	16.3
	20000	0.0	122.0	35.1	40.5
	40000	0.0	24.0	10.0	6.4
	60000	0.0	26.0	10.9	5.2
	100000	0.0	25.0	11.2	5.3
	140000	0.0	36.0	13.0	6.7
Q1 - GDR	1000	28.0	36.0	31.1	2.2
	10000	11.0	30.0	16.8	4.3
	20000	11.0	36.0	21.3	6.4
	40000	13.0	26.0	18.4	4.0
	60000	12.0	29.0	18.4	4.7
	100000	11.0	29.0	21.1	4.8
	140000	15.0	41.0	23.8	6.1
Q1 - GRD	1000	2.0	72.0	11.5	14.5
	10000	13.0	29.0	19.9	4.4
	20000	2.0	35.0	15.8	7.1
	40000	13.0	33.0	19.9	5.0
	60000	11.0	34.0	19.4	4.9
	100000	9.0	31.0	22.4	5.3
	140000	14.0	37.0	24.3	5.3
Q1 - SDR	1000	3.0	21.0	8.4	3.8
	10000	12.0	36.0	17.8	5.4
	20000	11.0	35.0	18.7	5.9
	40000	10.0	28.0	18.0	4.6
	60000	11.0	31.0	21.7	5.4
	100000	19.0	44.0	29.7	7.1
	140000	18.0	59.0	34.7	13.2
Q1 - SRD	1000	1.0	28.0	9.8	6.2
	10000	10.0	24.0	14.5	3.7
	20000	5.0	29.0	14.8	7.3
	40000	8.0	26.0	14.9	4.4
	60000	9.0	31.0	15.9	5.6
	100000	8.0	31.0	17.1	5.1
	140000	12.0	36.0	18.2	5.9

Q2 - BDR	1000	0.0	77.0	25.7	44.5
	10000	31272.0	31441.0	31328.3	97.6
	20000	3269340.0	3269393.0	3269357.7	30.6
	40000	2977.0	3028.0	2994.0	29.4
	60000	4480.0	4712.0	4564.7	128.1
	100000	7676.0	7714.0	7688.7	21.9
	140000	12215.0	12396.0	12335.7	104.5
Q2 - BRD	1000	3.0	4850.0	1631.0	2787.8
	10000	844.0	858.0	848.7	8.1
	20000	1505.0	1595.0	1535.0	52.0
	40000	2977.0	2992.0	2982.0	8.7
	60000	4433.0	4455.0	4440.3	12.7
	100000	7377.0	7399.0	7384.3	12.7
	140000	10324.0	10348.0	10332.0	13.9
Q2 - GDR	1000	1076.0	1085.0	1079.0	5.2
	10000	12932.0	12936.0	12933.3	2.3
	20000	57137.0	57208.0	57184.3	41.0
	40000	267979.0	268153.0	268077.0	89.1
	60000	576668.0	577152.0	576829.3	279.4
	100000	1371397.0	1371472.0	1371447.0	43.3
	140000	2331732.0	2331756.0	2331740.0	13.9
Q2 - GRD	1000	1645.0	1692.0	1662.7	25.6
	10000	112760.0	112799.0	112773.0	22.5
	20000	280096.0	280202.0	280160.7	56.7
	40000	625220.0	625466.0	625331.7	124.6
	60000	1004992.0	1005086.0	1005023.3	54.3
	100000	1839720.0	1839769.0	1839736.3	28.3
	140000	2768781.0	2768832.0	2768815.0	29.4
Q2 - SDR	1000	900.0	1008.0	936.0	62.4
	10000	14363.0	14621.0	14460.3	140.2
	20000	77720.0	77757.0	77732.3	21.4
	40000	277974.0	278068.0	278036.7	54.3
	60000	342275.0	342348.0	342323.7	42.1
	100000	1161528.0	1162172.0	1161742.7	371.8
	140000	1856052.0	1856555.0	1856346.3	262.2
Q2 - SRD	1000	817.0	847.0	830.0	15.4
	10000	23408.0	23437.0	23422.3	14.5
	20000	48818.0	48932.0	48863.3	60.5
	40000	104393.0	104464.0	104440.3	41.0
	60000	60420.0	60490.0	60443.3	40.4
	100000	287230.0	287276.0	287260.7	26.6
	140000	393240.0	393903.0	393461.0	382.8
Q4 - BDR	1000	0.0	20.0	6.4	6.9
	10000	12.0	99.0	24.7	18.0

	20000	7.0	151.0	67.3	44.3
	40000	10.0	35.0	26.9	7.2
	60000	22.0	48.0	38.9	6.4
	100000	48.0	68.0	56.1	6.8
	140000	46.0	93.0	67.6	15.4
Q4 - BRD	1000	0.0	5.0	0.4	1.1
	10000	74.0	171.0	146.9	25.6
	20000	6.0	149.0	84.5	48.0
	40000	8.0	32.0	18.2	6.7
	60000	8.0	28.0	19.2	5.9
	100000	8.0	32.0	22.2	6.8
	140000	12.0	36.0	23.0	6.9
Q4 - GDR	1000	15.0	40.0	26.8	5.6
	10000	23.0	54.0	37.2	7.4
	20000	29.0	65.0	53.7	8.2
	40000	23.0	41.0	32.2	5.0
	60000	25.0	44.0	34.8	5.6
	100000	30.0	49.0	40.6	5.0
	140000	31.0	57.0	43.2	6.9
Q4 - GRD	1000	8.0	29.0	17.2	5.5
	10000	28.0	61.0	45.0	9.5
	20000	16.0	37.0	28.0	5.6
	40000	26.0	45.0	35.0	5.1
	60000	28.0	45.0	36.5	4.7
	100000	33.0	48.0	41.8	5.0
	140000	23.0	60.0	43.4	8.6
Q4 - SDR	1000	3.0	23.0	13.8	4.6
	10000	27.0	50.0	39.9	6.7
	20000	34.0	60.0	47.6	6.8
	40000	21.0	44.0	31.5	6.9
	60000	26.0	60.0	41.0	9.0
	100000	37.0	70.0	52.9	9.9
	140000	37.0	101.0	65.9	18.1
Q4 - SRD	1000	3.0	30.0	15.6	7.4
	10000	19.0	43.0	30.7	6.8
	20000	22.0	52.0	36.1	9.1
	40000	14.0	39.0	28.1	6.2
	60000	19.0	39.0	30.8	5.4
	100000	26.0	41.0	32.7	4.0
	140000	15.0	48.0	33.7	7.4
Q52 - BDR	1000	4.0	31.0	10.5	5.4
	10000	1245.0	4960.0	3184.8	986.7
	20000	253.0	1014.0	497.0	158.2
	40000	14.0	48.0	35.2	10.3
	60000	6.0	66.0	48.2	17.9

	100000	32.0	121.0	85.5	26.8
	140000	33.0	495.0	181.0	139.0
Q52 - BRD	1000	0.0	19.0	3.1	4.4
	10000	16.0	350.0	238.9	75.0
	20000	0.0	188.0	59.2	79.8
	40000	0.0	32.0	14.0	10.7
	60000	0.0	28.0	15.6	8.4
	100000	0.0	28.0	18.4	8.1
	140000	0.0	28.0	20.8	7.3
Q52 - GDR	1000	17.0	101.0	43.7	22.2
	10000	34.0	264.0	125.9	58.1
	20000	51.0	360.0	147.8	69.9
	40000	42.0	87.0	65.4	12.5
	60000	45.0	97.0	67.9	14.4
	100000	41.0	101.0	73.0	14.1
	140000	40.0	108.0	72.5	19.8
Q52 - GRD	1000	10.0	80.0	42.3	18.5
	10000	38.0	365.0	158.8	80.9
	20000	25.0	96.0	54.6	18.9
	40000	42.0	95.0	65.4	12.6
	60000	43.0	99.0	65.6	17.5
	100000	47.0	100.0	72.7	12.5
	140000	38.0	105.0	71.4	16.6
Q52 - SDR	1000	7.0	54.0	30.0	13.7
	10000	35.0	333.0	142.9	70.2
	20000	46.0	348.0	134.3	75.4
	40000	34.0	90.0	65.2	15.4
	60000	39.0	167.0	78.9	30.4
	100000	33.0	186.0	103.3	37.8
	140000	51.0	266.0	127.2	65.7
Q52 - SRD	1000	11.0	224.0	100.8	58.1
	10000	46.0	354.0	189.3	73.6
	20000	51.0	347.0	139.4	70.5
	40000	25.0	72.0	51.9	12.6
	60000	31.0	81.0	53.0	15.9
	100000	31.0	78.0	55.2	12.8
	140000	22.0	77.0	54.1	17.0
Q53 - BDR	1000	4.0	22.0	12.4	4.6
	10000	2883.0	29634.0	14860.3	6984.5
	20000	218.0	4537.0	1171.6	868.5
	40000	18.0	56.0	40.5	12.5
	60000	6.0	79.0	52.3	21.8
	100000	33.0	159.0	103.3	36.7
	140000	35.0	636.0	237.4	195.2

Q53 - BRD	1000	0.0	9.0	3.2	2.7
	10000	19.0	401.0	267.7	88.9
	20000	0.0	184.0	58.6	77.8
	40000	0.0	32.0	14.4	11.2
	60000	0.0	32.0	16.6	10.2
	100000	0.0	32.0	20.6	9.3
	140000	0.0	32.0	22.2	9.5
Q53 - GDR	1000	11.0	76.0	29.3	15.5
	10000	32.0	457.0	184.1	101.7
	20000	57.0	885.0	233.5	176.1
	40000	42.0	106.0	76.3	17.7
	60000	46.0	124.0	79.5	20.9
	100000	38.0	114.0	83.6	19.9
	140000	47.0	123.0	82.0	24.0
Q53 - GRD	1000	9.0	141.0	54.1	33.2
	10000	39.0	640.0	248.9	153.3
	20000	28.0	119.0	64.3	24.2
	40000	42.0	112.0	73.6	16.7
	60000	44.0	119.0	75.9	22.5
	100000	44.0	111.0	82.4	17.7
	140000	43.0	114.0	79.7	20.3
Q53 - SDR	1000	7.0	74.0	34.5	17.2
	10000	39.0	729.0	241.4	164.3
	20000	49.0	783.0	205.9	163.9
	40000	37.0	107.0	75.0	19.9
	60000	40.0	232.0	94.7	43.7
	100000	36.0	321.0	128.0	63.5
	140000	54.0	606.0	193.9	167.8
Q53 - SRD	1000	11.0	335.0	137.5	86.3
	10000	50.0	949.0	320.4	217.2
	20000	65.0	769.0	222.4	153.9
	40000	20.0	84.0	58.0	16.1
	60000	31.0	95.0	60.3	20.6
	100000	31.0	92.0	62.7	17.3
	140000	22.0	85.0	61.1	21.1
Q54 - BDR	1000	1.0	21.0	10.2	5.1
	10000	4.0	61106.0	12135.9	22187.1
	20000	214.0	37497.0	7224.6	10768.8
	40000	18.0	64.0	43.5	14.2
	60000	24.0	86.0	56.1	22.4
	100000	61.0	201.0	133.9	43.2
	140000	39.0	744.0	245.6	278.0
Q54 - BRD	1000	0.0	7.0	2.8	2.7
	10000	19.0	418.0	291.4	130.6
	20000	0.0	184.0	45.1	70.5

	40000	0.0	36.0	16.0	14.7
	60000	4.0	36.0	17.2	11.8
	100000	0.0	36.0	22.4	12.4
	140000	4.0	32.0	22.0	11.2
Q54 - GDR	1000	9.0	43.0	20.7	10.9
	10000	30.0	561.0	257.4	166.2
	20000	61.0	4219.0	699.9	1247.8
	40000	43.0	115.0	81.2	24.1
	60000	46.0	131.0	86.4	28.9
	100000	38.0	113.0	93.6	23.5
	140000	51.0	134.0	74.0	24.9
Q54 - GRD	1000	9.0	184.0	51.6	51.6
	10000	38.0	2012.0	498.8	571.8
	20000	34.0	131.0	74.1	28.5
	40000	41.0	116.0	79.5	23.0
	60000	43.0	124.0	81.0	25.7
	100000	44.0	115.0	93.0	22.0
	140000	55.0	121.0	74.4	20.1
Q54 - SDR	1000	7.0	83.0	31.3	21.8
	10000	39.0	869.0	331.7	286.3
	20000	52.0	2843.0	571.8	837.5
	40000	38.0	116.0	78.6	26.4
	60000	40.0	161.0	98.1	42.3
	100000	37.0	386.0	157.2	97.1
	140000	55.0	526.0	131.6	140.8
Q54 - SRD	1000	9.0	471.0	140.6	143.5
	10000	46.0	6528.0	986.3	1955.8
	20000	43.0	2711.0	488.7	810.3
	40000	21.0	97.0	63.0	23.6
	60000	29.0	110.0	63.1	27.3
	100000	39.0	110.0	72.6	21.0
	140000	22.0	88.0	55.1	24.0
Q55 - BDR	1000	1.0	19.0	11.3	5.8
	10000	4.0	13.0	8.1	2.8
	20000	206.0	326391.0	49844.8	97700.9
	40000	15.0	66.0	48.1	16.0
	60000	24.0	98.0	62.7	21.5
	100000	62.0	278.0	159.3	66.2
	140000	39.0	945.0	313.5	385.1
Q55 - BRD	1000	0.0	8.0	3.5	2.8
	10000	8.0	465.0	107.4	160.1
	20000	0.0	184.0	44.6	70.1
	40000	0.0	40.0	16.4	16.2
	60000	0.0	37.0	18.0	12.5
	100000	0.0	40.0	23.2	13.3

	140000	4.0	36.0	22.8	12.1
Q55 - GDR	1000	9.0	43.0	19.3	10.3
	10000	30.0	2525.0	589.3	746.3
	20000	59.0	13427.0	1426.1	4216.8
	40000	41.0	131.0	87.9	28.4
	60000	46.0	159.0	95.9	36.1
	100000	43.0	128.0	102.7	27.6
	140000	51.0	160.0	79.8	33.1
Q55 - GRD	1000	9.0	246.0	59.3	70.7
	10000	36.0	12915.0	2057.4	3896.9
	20000	39.0	143.0	84.2	31.2
	40000	38.0	126.0	86.9	27.7
	60000	43.0	135.0	87.6	30.0
	100000	48.0	130.0	102.4	25.9
	140000	55.0	137.0	78.3	25.3
Q55 - SDR	1000	7.0	109.0	33.7	29.3
	10000	33.0	4805.0	935.6	1464.7
	20000	52.0	7553.0	828.8	2362.8
	40000	38.0	129.0	84.8	30.8
	60000	40.0	264.0	117.1	66.6
	100000	45.0	516.0	185.6	136.4
	140000	55.0	700.0	152.7	194.7
Q55 - SRD	1000	8.0	1769.0	306.1	533.3
	10000	43.0	50296.0	5919.7	15606.3
	20000	38.0	121.0	70.4	30.9
	40000	24.0	109.0	68.2	26.8
	60000	27.0	126.0	69.0	33.0
	100000	39.0	126.0	80.7	27.4
	140000	21.0	102.0	58.7	28.7
Q63 - BDR	1000	1.0	15.0	9.7	2.8
	10000	1.0	12.0	6.6	2.9
	20000	214.0	1028.0	493.3	163.5
	40000	16.0	49.0	35.8	11.2
	60000	6.0	66.0	48.3	18.0
	100000	31.0	117.0	85.5	25.8
	140000	33.0	495.0	181.0	139.0
Q63 - BRD	1000	0.0	6.0	2.2	2.2
	10000	4.0	16.0	8.1	3.3
	20000	0.0	188.0	58.6	78.7
	40000	0.0	32.0	14.0	10.7
	60000	0.0	28.0	15.6	8.4
	100000	0.0	28.0	18.4	8.1
	140000	0.0	28.0	20.8	7.3
Q63 - GDR	1000	4.0	28.0	16.8	6.1

	10000	33.0	262.0	125.5	58.4
	20000	25.0	100.0	54.7	18.7
	40000	42.0	87.0	65.4	12.7
	60000	45.0	97.0	68.1	13.9
	100000	43.0	101.0	72.9	13.9
	140000	40.0	108.0	72.4	19.8
Q63 - GRD	1000	10.0	82.0	41.9	19.1
	10000	39.0	361.0	158.6	80.1
	20000	25.0	96.0	54.6	18.8
	40000	42.0	94.0	65.1	13.0
	60000	43.0	99.0	65.3	17.9
	100000	47.0	100.0	72.5	12.5
	140000	39.0	105.0	71.4	16.4
Q63 - SDR	1000	7.0	49.0	28.5	13.0
	10000	33.0	280.0	120.2	60.9
	20000	19.0	94.0	48.1	19.5
	40000	34.0	89.0	64.8	15.5
	60000	39.0	169.0	79.3	30.3
	100000	43.0	184.0	104.2	36.8
	140000	51.0	266.0	127.4	65.6
Q63 - SRD	1000	7.0	233.0	100.9	58.8
	10000	43.0	346.0	189.2	72.0
	20000	16.0	85.0	43.5	18.1
	40000	25.0	72.0	51.9	12.7
	60000	31.0	81.0	52.7	15.9
	100000	31.0	78.0	55.0	13.1
	140000	22.0	77.0	53.9	16.8
Q64 - BDR	1000	3.0	19.0	10.2	4.8
	10000	4.0	16.0	8.7	3.7
	20000	207.0	643.0	474.4	128.1
	40000	14.0	49.0	34.3	11.7
	60000	23.0	68.0	47.0	15.8
	100000	55.0	117.0	94.5	19.9
	140000	33.0	496.0	166.2	163.7
Q64 - BRD	1000	0.0	6.0	3.1	2.4
	10000	4.0	16.0	9.1	3.7
	20000	0.0	179.0	43.8	71.4
	40000	0.0	28.0	14.8	11.6
	60000	0.0	28.0	16.0	8.6
	100000	0.0	28.0	18.0	9.5
	140000	0.0	28.0	20.0	9.0
Q64 - GDR	1000	9.0	27.0	15.9	5.8
	10000	32.0	186.0	115.2	53.4
	20000	29.0	100.0	55.2	20.6
	40000	42.0	87.0	64.2	15.4

	60000	45.0	82.0	65.3	13.2
	100000	38.0	85.0	72.7	14.5
	140000	46.0	94.0	61.8	13.9
Q64 - GRD	1000	10.0	66.0	32.9	18.1
	10000	37.0	287.0	148.7	81.0
	20000	28.0	96.0	54.3	19.9
	40000	42.0	91.0	63.8	13.7
	60000	44.0	82.0	61.2	12.6
	100000	44.0	84.0	72.1	11.2
	140000	52.0	88.0	64.0	11.4
Q64 - SDR	1000	7.0	49.0	24.7	13.1
	10000	33.0	177.0	114.9	53.1
	20000	23.0	94.0	50.4	20.7
	40000	34.0	88.0	61.7	17.0
	60000	39.0	118.0	71.5	23.0
	100000	35.0	169.0	104.6	37.1
	140000	51.0	190.0	86.6	40.4
Q64 - SRD	1000	9.0	140.0	67.4	47.7
	10000	43.0	302.0	172.5	79.4
	20000	16.0	85.0	48.0	20.4
	40000	25.0	72.0	53.2	15.8
	60000	31.0	81.0	49.8	17.4
	100000	39.0	75.0	56.0	9.9
	140000	22.0	69.0	47.5	16.3
Q65 - BDR	1000	4.0	18.0	10.5	4.4
	10000	1.0	11.0	7.3	3.4
	20000	218.0	627.0	473.8	122.1
	40000	16.0	49.0	33.1	12.5
	60000	24.0	68.0	49.9	13.4
	100000	55.0	117.0	94.3	19.8
	140000	33.0	495.0	166.3	163.7
Q65 - BRD	1000	0.0	6.0	2.8	2.4
	10000	4.0	14.0	8.7	3.5
	20000	0.0	176.0	43.0	69.9
	40000	0.0	28.0	14.6	11.4
	60000	0.0	28.0	16.0	8.6
	100000	0.0	28.0	18.0	9.5
	140000	0.0	28.0	20.0	9.0
Q65 - GDR	1000	9.0	27.0	15.8	5.8
	10000	33.0	190.0	115.6	55.1
	20000	29.0	100.0	55.9	19.8
	40000	42.0	87.0	63.5	16.2
	60000	45.0	82.0	65.0	12.1
	100000	43.0	85.0	72.5	12.9
	140000	46.0	94.0	62.1	14.1

Q65 - GRD	1000	9.0	68.0	32.5	18.7
	10000	37.0	297.0	150.3	83.0
	20000	28.0	96.0	54.8	19.5
	40000	42.0	91.0	64.0	14.4
	60000	44.0	81.0	59.9	12.5
	100000	48.0	84.0	72.5	10.0
	140000	50.0	88.0	63.5	11.4
Q65 - SDR	1000	7.0	48.0	22.2	12.7
	10000	33.0	173.0	115.6	52.7
	20000	23.0	91.0	50.6	19.4
	40000	34.0	88.0	60.5	17.2
	60000	39.0	118.0	72.2	22.3
	100000	40.0	167.0	104.4	35.3
	140000	51.0	190.0	87.0	40.5
Q65 - SRD	1000	7.0	144.0	68.4	49.5
	10000	41.0	308.0	171.8	81.4
	20000	16.0	85.0	48.0	20.4
	40000	25.0	72.0	53.2	15.8
	60000	31.0	81.0	49.8	17.4
	100000	39.0	75.0	56.0	9.9
	140000	22.0	69.0	47.1	15.8

List of Tables

List of Figures