A System for Low-Cost Access to Very Large Catalogs

K. Kalpakis[1], M. Riggs[1], M. Pasad[1], and V. Puttagunta[1]

Computer Science & Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250,
{kalpakis,mriggs1,pasad,vputta1}@csee.umbc.edu

J. Behnke

NASA Goddard Space Flight Center, Greenbelt, MD 20771,
jeanne.behnke@gsfc.nasa.gov

Abstract. Many catalogs in astronomy, few old and most of the new ones, contain data for very large number of objects. In order for researchers to conduct their studies it is crucial that they are provided with fast methods to access those catalogs. At the same time, the monetary cost of achieving fast access should not come at the expense of resources that would be better used to support the actual scientific studies.

We demonstrate how to achieve fast access to data on a low cost desktop for a very large catalog using the Informix Object-Relational database system. We report on experimental results from a development of a solution for efficiently indexing the USNO-A2.0 catalog, which has approximately 500 million objects. The solution offers significant performance improvements over some existing methods.

We also describe an extension of Informix that enables users to evaluate their IDL scripts over data stored in Informix using SQL. This extension brings the powerful data-analysis and visualization capabilities of IDL within Informix.

1. Introduction

There are several astronomical catalogs that contain data for very large number of (mostly static) stellar objects (e.g. Tycho–2, GSC–I, etc). In this study, we focus on the USNO–A2.0 catalog. USNO–A2.0 is a catalog of 526,280,881 (i.e. over 500M) stars that presents right ascension and declination (J2000, epoch of the mean of the blue and red plate) and the blue and red magnitude for each star.

While those data can be made available to public and astronomical communities all over the world via the WWW, there are several problems to be dealt with because of the shear bulk of the data. In order to conduct any kind

[1]Supported in part by NASA under contract number NAS5-32337 and cooperative agreement NCC5-315.

of research on such datasets, it is both crucial and critical that they be provided with rapid access mechanism(s). At the same time, monetary costs of achieving fast access should not come at the expense of resources that could be better used to support the actual scientific studies. In other words, low cost solutions that achieve efficient and effective storage management and rapid access to (query processing of) such large catalogs is of vital importance to the research community.

We demonstrate how to achieve fast access to data on a low cost desktop for a very large catalog using the Informix Dynamic Server, an extensible Object-Relational database system (ORDBMS). Further, we illustrate how to enhance the functionality of the system by extending the ORDBMS with a statistical/scientific-computing package, IDL.

2. System Summary Description

We design a low cost system for storing and querying large catalogs (e.g. USNO–A2.0). In designing our system, we wanted to ensure that the following types of spatial queries can be processed efficiently without a significant hardware cost: 1) **Spatial window**, e.g. *Find all stars within a user–defined bounding rectangle.* 2) **Spatial OR–window**, e.g. *Find all stars within at least one of two user–defined bounding rectangles.* 3) **Spatial Multi–join** (catalog correlations). (a) Spatial Chain–join, e.g. *Find all stars and associated data in a "chain–joined" sequence of catalogs*, and (b) Spatial Star–join, e.g. *Find all stars and associated data in a "star–joined" sequence of catalogs.* [1] 4) **Spatial Self–join** (catalog mining), e.g. *Find all stars that are within a specified box centered on each star.*

The spatial window and spatial multi–join queries are very critical and users expect them to be processed online (e.g. response times in the order of couple of seconds). Spatial self–joins over large spatial datasets usually require a very large computation time and we do not consider them for online processing at this time.

Hardware and System Software. The hardware we use is a PC with an AMD Duron 650Mhz processor on a FIC model AZ11 motherboard with an IDE ATA–66 and an Ultra/ATA–100 controller, 256MB of PC100 RAM, and two IBM Ultra/100 7200rpm IDE disks (model DTLA307045). The cost of our hardware platform is well below $2,000. The machine is running the Linux Mandrake version 7.0 Operating system, with the updated Linux Kernel version 2.4.0–test4, and is also running the Informix Dynamic Server 2000 version 9.20 UC–1, configured with our custom datablade (which we call *SimpleShape*, and custom data loading software.

Custom Datablade. SimpleShape provides a 16–byte opaque User Defined Type (UDT) storing 4 small floats, full R–tree index support including bottom–up index building, and full B–tree index support based on 2nd order Hilbert

[1] Given three catalogs A, B, and C, a chain–join joins A with B, and B with C to extract information from all three, while a star–join joins A with B and A with C.

curves. It defines two other UDTs with appropriate I/O functions and constructors: the *SimplePoint* to store the coordinates of 2–D points, and the *SimpleBox* to store the lower left and upper right corner points of a rectangle. Each star in the USNO–A2.0 catalog comes packaged as three integers: RA (right ascension), DEC (declination), and MAG (a coding for the field, and the read and blue magnitudes). We create a table with schema **(coordinates SimplePoint, magnitude Integer)** to store the catalog. The data are stored in the database in their original (compressed) format, while user–defined functions convert them into more readily usable values. The RA and DEC fields are stored in a SimplePoint value. To further simplify access to the data, we define additional utility User Defined Functions (UDFs). An example SQL spatial window query is: SELECT Ra(coords), Dec(coords), Red(mag), Blue(mag), Field(mag) FROM catalog WHERE Within(coords, monetbox(-3.22, 22.45, 1.23, 22.78));.

Custom Loader. Due to a bug in the regular table load command for the PC platform, the lack of an Infomix high performance loader for Linux, and in order to avoid time–consuming string conversions during loading, we develop a custom high–performance loader. Our loader is built using the Virtual Table Interface (VTI) of Informix which maps a binary file into a database table. An example of its use is: CREATE TABLE monet(ra INT, dec INT, mag INT) USING vti_load(file='/tmp/zone0000.cat');.

IDL Extension for the Informix Server. In order to enable powerful data–mining applications to be prototyped quickly we also develop an extension to Informix that enables users to apply their IDL scripts to data stored in the database within the DBMS. The IDL extension to Informix allows users to tap the data analysis and visualization capabilities of IDL through Informix and SQL. Our implementation is based on UDFs and the RPC mechanism. We define two UDFs, the **idl_exec** that evaluates a user provided IDL script for each qualifying tuple, and the **idl_agg** which evaluates an aggregation defined via three IDL scripts over groups of tuples specified by standard SQL expressions. We also define utility and casting UDFs to access data returned by IDL and to pass data to IDL from the database server. For example, the following SQL computes three cluster centers of an $m \times n$ array of data stored in a database table using the IDL CLUST_WTS function, and then returns them as a virtual table of cluster centers:

```
SELECT toList( idl_agg( ROW(ra,dec), ROW( "count=-1", "count=count+1&
if count eq 0 then y=[$ 1, $ 2] else y = [[y]], [$ 1, $ 2]]", "w=CLUST_WTS(y,
N_CLUSTERS=3); $ RETURN w" ) ) ) :: list( ROW(x float, b float)
NOT NULL) FROM monet;
```

3. Experiments

We compare the performance of our SimpleShape datablade and customized loader for processing the four types of queries with two other datablades for Informix, the Geodetic and the Shapes2 datablades, as well as with the traditional relational database approach (e.g. no user–defined indexes, no first–class citizen UDTs and UDFs, and no user–defined loaders). For each one of these four

approaches, we create a database schema, thus having the Relational, Geodetic, Shapes2, and SimpleShape schemas, to store USNO–A2.0 data. The experiments for the SimpleShape schema were conducted on the PC platform described earlier, while the others were conducted on a Sun UltraSparc 60 with 512MB of RAM and two 8GB and two 4GB SCSI disks. In all the experiments, we use random query windows of size 0.0666×0.0666 decimal degrees. We can see from Table 1 that the SimpleShape datablade and custom loader improve upon the loading and indexing times with respect to the other two datablades by over 5 times, while they reduce the amount of disk space used by over 3 times. Further, they both improve substantially over the traditional relational approaches in terms of loading and indexing spatial data. Moreover, as we can see from Table 2, our custom datablade reduces the number of page reads for spatial window and multi–join queries substantially over the traditional relational approach, and by over two times with respect to the Geodetic and Shapes2 datablades. For Spatial self–joins, the SimpleShape is dramatically better than the Geodetic and Shapes2 datablades, while is competitive with relational approaches (with respect to page reads; the SimpleShape gives substantially better performance on response time and page reads over relational methods as the number of stars increases). For more detailed experimental results and analysis please refer to (Kalpakis et al. 2000) or see `http://www.csee.umbc.edu/~kalpakis/monet` for updated information.

Table 1. Loading and Indexing all 500M USNO–A2.0 Stars.

Schema	Loading Time[a]	Indexing Time[a]		Table Size	Index Size	
		B–tree	R–tree		B–tree	R–tree
Relational[b]	3 days (46 secs)	6 days		13GB	56GB	
Geodetic[b]	14 days (234 secs)	90 days	25 days (448 secs)	190GB	66GB	104GB
Shapes[b]	10 days (162 secs)	57 days	15 days (256 secs)	140GB	86GB	77GB
SimpleShape	1 day (27 secs)		1 day (8 secs)	15GB	14GB	20GB

[a]The times in parentheses are measurements for 100K stars.

[b]Estimate for 500M stars based on regression from measurements for up to 140K stars.

Table 2. Performance for various queries and schemas for 60K stars.

Query	Number of Page Reads				Elapsed Time
	Relational	Geodetic	Shapes2	SimpleShape	SimpleShape
Spatial OR–window	22	35	44	14	0.163 secs
Spatial Self–Join	4440	1005023	60443	6775	3072.000 secs
2–Chain Spatial Join	23	49	87	19	2.377 secs
3–Star Spatial Join	24	50	43	12	9.532 secs

References

Kalpakis, K., Behnke, J., Pasad, M., & Riggs, M. 2000, *Performance of Spatial Queries in Object–Relational Database Systems*, NASA/CESDIS Technical Report TR-00-226.