

The Terabyte Analysis Machine Project

The Distance Machine: Performance Report

James Annis, Gabriele Garzoglio, Chris Stoughton

Abstract: The Terabyte Analysis Machine is a research project in the fields of Knowledge and Distributed Intelligence (KDI) and Large Distributed Archives in Astronomy and Particle Physics (ALDAP). This report focuses on one of the studies in progress: the Distance Machine Framework. Many analysis applications in Astronomy and Particle Physics need a transparent access to large database-managed dataset. The Distance Machine Framework provides facilities to interface these analysis engines to the Objectivity Database System, allowing the user to easily include application specific indexing and partitioning algorithms. ANN is a useful example of the use of this framework to implement an Approximate Nearest Neighbors engine. With its client server architecture, ANN interfaces the Sloan Digital Sky Survey catalog database to different kind of astronomical analysis programs. The performance of this system is thoroughly discussed.

Table of Content

Overview.....	1
Performance tests.....	2
Building and searching the tree	2
The re-clustering module.....	4
The client/server architecture.....	4
Future developments.....	4

Overview

The goal of the Terabyte Analysis Machine (TAM) project is to explore and integrate emerging and stable technologies into an efficient computing environment for physicists and astronomers working data intensive scientific analysis.

The Terabyte Analysis Machine is a Linux cluster of currently 5 machines, connected to a RAID disk via fibre channel. Using this cluster, several studies are in progress in the areas of Knowledge and Distributed Intelligence (KDI) and Large Distributed Archives in Astronomy and Particle Physics (ALDAP). The studies include:

- using the Grid Data Management Pilot (GDMP)¹ to transfer the Sloan Digital Sky Survey² database (SX³) among collaborating institutions;

- testing performance and reliability of the Global File System⁴ for concurrent and hi rate accesses to the shared RAID disk and, in particular, its interaction with the Objectivity Database System⁵;
- testing farms of IDE disks to provide direct access to a several terabytes single node at low cost, for non I/O bound applications.
- developing the Distance Machine framework.

Particle physics and, increasingly, astronomical analysis programs are data intensive: accessing the data in a homogeneous and efficient way is critical in these applications. The Distance Machine framework allows specific analysis programs to transparently access large databases, making best use of sophisticated specific algorithms for indexing and partitioning the database.

At the current stage of development, the Distance Machine is a library, implemented in C++, which uses Objectivity as database system. To guarantee the scientist flexibility in the choice of data analysis language, a Corba interface based client/server architecture is provided to the user. The transparent access to the large database embodied dataset is granted by embedding the objectivity reference to each data object within a wrapper, a wrapper whose standard array operators are overloaded. A vector of wrappers can be effectively treated as a vector of pointers to memory: this approach not only allows almost immediate extension of most existing standalone analysis applications to large Objectivity managed datasets, but also enables new applications to use the re-indexing and re-partitioning features of the framework to increase the efficiency of the data access.

As a useful example of the use of the framework, we have modified and extended an existing library, ANN⁶ (Approximate Nearest Neighbors), which provides methods to find in memory the k^{th} nearest neighbors to a query point, in a defined parameter space. Efficient nearest neighbor (NN) searching algorithms are an area of active research in the computer science community (there are no exact solutions for dimensions greater than 10 that are faster than a brute force scan), play a central role in SPH numerical calculations of structure formation, and are the core of a large of astronomical analysis, clustering analysis, including searches for clusters of galaxies and for resolved dwarf galaxies. It is worth noting that the central algorithm of ANN is the kd-tree, which is also the base algorithm for the new fast N-point correlation functions codes⁷: the Distance Machine framework could be straightforwardly extended to incorporate these powerful codes.

Using the Distance Machine framework, ANN has the ability to efficiently provide set of NN to any object of the Sloan Digital Sky Survey database, which in 4 years is expected to have a catalog of 500 Gigabytes.

As a note, The NN engine could provide a test bed for the GriPhyN vision of virtual dataset connected via a computing grid: the Distance Machine would provide a specialized service.

This report focuses on the performance of this NN engine: ANN/Objectivity.

Performance tests

The Terabyte Analysis Machine is a cluster of (currently) 5 dual Pentium III 650 MHz computers, each with 1Gbyte of RAM, 70Gbytes of local IDE disk, and a fibre channel connection to a terabyte-scale hardware RAID box. This system has been design to allow 1 Gigabyte/sec aggregate throughput I/O rates for sequential data scans⁸.

The Red Hat 7 boxes access the RAID disk via the Global File System, which enables each node to directly access the shared file system without filesystem corruption. As mentioned before, the Distance Machine interfaces the data objects via Objectivity. From our tests, objectivity cannot see disks mounted via GFS as shared.

Building and searching the tree

ANN/Objectivity uses the Distance Machine framework to implement a Nearest Neighbor engine. First, it loads each objectivity data reference into a vector of wrappers whose role it is to overload the standard array operators. For our database configuration (1 container per database, max test database size 200 Megabytes, default ooinit() parameters, object size 250 bytes) this takes 7 μs /object. Then the user provided definition of the parameter space (schema + overloading rule for operator[]), is used to build in memory a tree structure on the data. At each node of the tree (hyper-cubes in the parameter space) summary information like node boundaries are maintained. Pointers to the data wrappers are stored in the leaf nodes.

The maximum number of pointers in a leaf node is called bucket size. The total memory size of the tree structure depends inversely on

this number. During the NN search, the tree is descended to the leaf level; at this level the NN algorithm is the standard $O(N^2)$. The choice of the bucket size is, in fact, a trade off between available memory and search speed. In our case we have used a bucket size of 10 objects.

Fig 1 shows the memory usage after building the tree: this should be considered when configuring ANN to deal with database larger than 10 million objects. Note that we have set the Objectivity client page cache to 4 Mbytes.

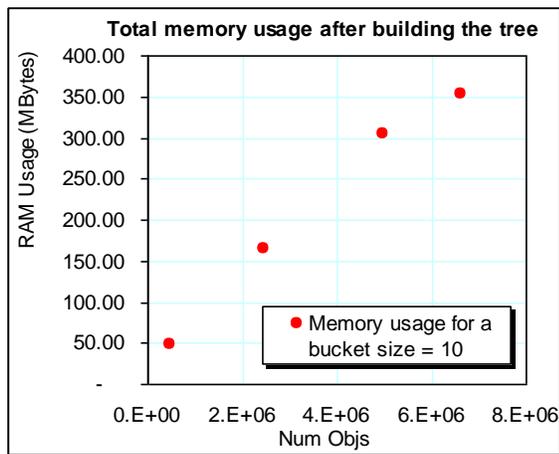


Fig 1: Memory usage for various database size after building the tree.

The tree structure can be saved in a text format for future retrieval: for our system, the retrieval time is of the order of 10 μ s/object, while building the tree is a CPU intensive operation of the order of fraction of ms/object (see Fig.2). The data agree with the expected $O(n \log(n))$ complexity of the algorithm with a squared correlation coefficient (R^2) of 0.98 .

The search for NN to a query point is made extremely quick using the summary meta-data maintained in the nodes of the tree. Fig.3 shows a comparison between the standard $O(n^2)$ algorithm and the tree based algorithms that ANN implements¹. From the figure is

¹ ANN can build two kinds of trees: kd and bd (box decomposition) tree. The former divides the space recursively splitting the sample along a single dimension; the latter, in addition to this, divides the space in concentric boxes: the bd-tree is optimal when

also clear that for our problem the different possible tree-based algorithms are equivalent. (These tests were run on an Origin 2000.)

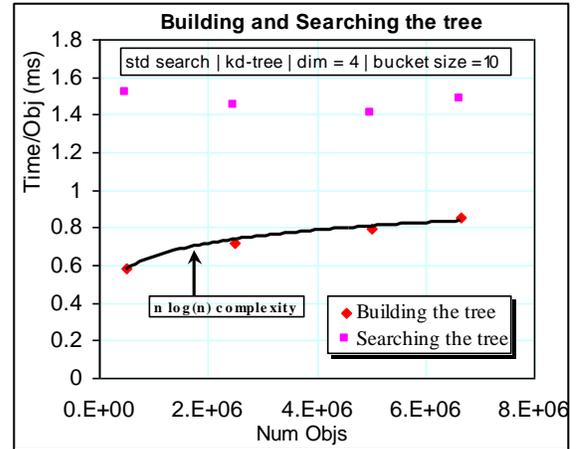


Fig 2: Time per object to build and search a kd-tree for different database sizes.

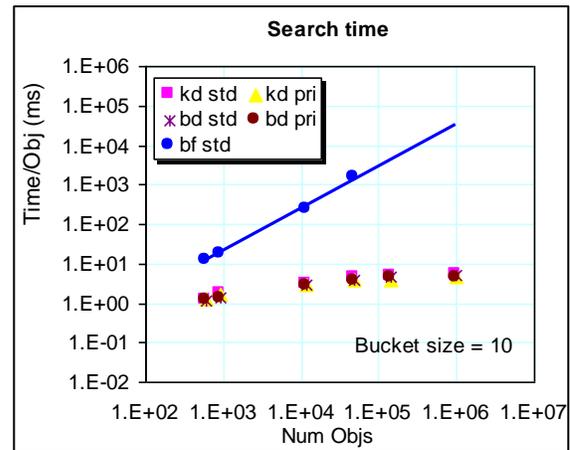


Fig 3: Comparison between the search time of the standard $O(n^2)$ NN algorithm (bf std) and the tree based ones (bd, kd). Two different search algorithms are used for the trees: std and pri (priority queue).

highly clustered distributions of points occur. Two different kind of search algorithms are then used on the two trees: “std” is a recursive search of the tree; “pri” searches the tree following a priority queue of the leaf-nodes sorted by increasing distance from the query point.

The re-clustering module

The process of organizing the data in a tree structure introduces a natural indexing on the data. The data can be reorganized on disk according to this indexing (re-clustering). On our system, this process takes about 80 μ s/object. This technique aims to minimize the amount of memory cache swaps needed by objectivity during the search for the nearest neighbors.

Fig. 4 compares the time per object required to search 5 nearest neighbors for all the objects, in the two cases of standard and re-clustered database. For a small database (500,000 objects) the time is approximately the same: the data size in this case is small (160MB) and few memory swaps are needed; for bigger databases (above 2,500,000 objects) there is a gain of 15-20% in the query execution. In general the gain can be better, considering that in this case the initial database was already partially clustered in two (ra, dec) of the 4 dimensions used (spatial organization).

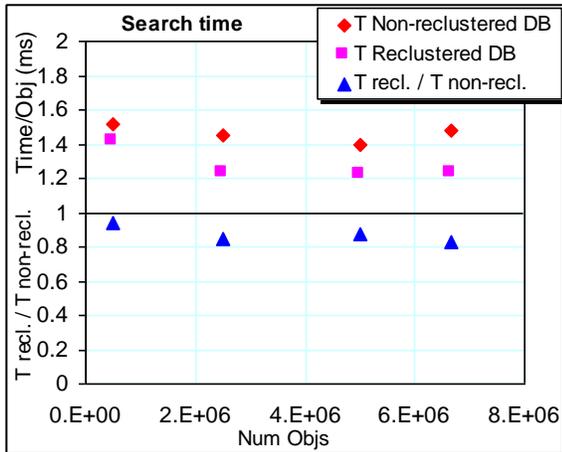


Fig 4: Comparison in the search time per object for non-reclustered and reclustered databases, for different database sizes.

The client/server architecture.

ANN/Objectivity is currently implemented as a C++ library. To allow flexibility in the choice of the programming language in the user analysis, a client/server architecture implemented via Corba interfaces has been

provided. The server has been implemented using Orbacus v4_0_5 and tested using a C client implemented with Orbit (libIDL v0_6_8). The search time per object for non-reclustered databases increases to about 1.8 ms/object. Note that this time can be optimized reducing the output log on both client and server sides. This result has to be compared with 1.4 ms/object of Fig 4: the overhead is acceptable in many, and perhaps most cases, where the increased flexibility allows rapid development and data exploration.

Future Development of the Distance Machine

The performance of ANN/objectivity can be improved by tuning of the objectivity parameters on a case-by-case basis.

Once the database reaches sizes of millions of objects and above, simply increasing the client page cache cannot increase the performance, as the cache is already small with respect to the total database size. This is especially true given that usually the NN search is performed for each object in the complete dataset. Some improvement can be obtained by a re-clustering of the database, wisely choosing the factor of growth of the containers and the number of containers per database.

There is large performance gain to be had by a parallelization of the NN search engine. The data needs to be distributed to each node partitioned by different portions of the sky. Some overlapping of the data would be implemented in order to solve the problem at the borders (the so-called shadowing techniques). TAM uses Condor⁹ as batch system, which would handle this naturally, but some problem could occur due to the already mentioned interference between GFS and Objectivity.

In order to make the definition of the space where ANN operates user-friendlier, some kind of GUI could be implemented. So far the user is required to C++ code it as a method

that operates on the schema of the persistent tag object.

References

¹ <http://cmsdoc.cern.ch/cms/grid>

² <http://www.sdss.org>

³ <http://www.sdss.jhu.edu>

⁴ <http://www.sistina.com/gfs>

⁵ <http://www.objectivity.com>

⁶ ANN Programming Manual; David M. Mount, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, www.cs.umd.edu/~mount/ANN/

⁷ Moore, A., et al., Fast Algorithms and Efficient Statistics: n-point Correlation Functions, at the MPA/MPE/ESO Conference "Mining the Sky" 2000, <http://www.cs.cmu.edu/~agray/miningthesky.pdf>

⁸ <http://sdss.fnal.gov:8000/~annis/astroCompu te.html>

⁹ <http://www.cs.wisc.edu/condor>