Infrastructure for Building Parallel Database Systems for Multi-dimensional Data [*]

Chialin Chang[†], Renato Ferreira[†], Alan Sussman[†], Joel Saltz[†+]

[†] Dept. of Computer Science
University of Maryland
College Park, MD 20742

[+] Dept. of Pathology
Johns Hopkins Medical Inst.
Baltimore, MD 21287

{chialin,renato,als,saltz}@cs.umd.edu

Abstract

Our study of a large set of scientific applications over the past three years indicates that the processing for multi-dimensional datasets is often highly stylized. The basic processing step usually consists of mapping the individual input items to the output grid and computing output items by aggregating, in some way, all the input items mapped to the corresponding grid point. In this paper, we discuss the design and performance of T2, an infrastructure for building parallel database systems that integrates storage, retrieval and processing of multi-dimensional datasets. It achieves its primary advantage from the ability to integrate data retrieval and processing for a wide variety of applications and from the ability to maintain and jointly process multiple datasets with different underlying grids. We present preliminary performance results comparing the implementation of two applications using the T2 services with custom-built integrated implementations.

1 Introduction

As computational power and storage capacity increase, processing and analyzing large volumes of data play an increasingly important part in many domains of scientific research. Typical examples of very large scientific datasets include long running simulations of time-dependent phenomena that periodically generate snapshots of their state (e.g. hydrodynamics and chemical transport simulation for estimating pollution impact on water bodies, magnetohydrodynamics simulation of planetary magnetospheres, simulation of a flame sweeping through a volume, airplane wake simulations), archives of raw and processed remote sensing data (e.g. AVHRR , Thematic Mapper, MODIS), and archives of medical images (e.g. high resolution confocal light microscopy, CT imaging, MRI, sonography).

These datasets are usually *multi-dimensional*. That is, each data item in a dataset is associated with a point in a multi-dimensional *attribute space* defined by the attributes of the data item. The data dimensions can be spatial coordinates, time, or varying experimental conditions such as temperature, velocity or magnetic field. Access to data items of such a dataset is often described by a *range query*, which retrieves all the data items whose associated points fall within a given multi-dimensional box in the dataset's underlying attribute space. The increasing importance of such datasets has been recognized by the database community and several research and commercial systems have been developed for managing and/or visualizing them [4, 8, 13].

These systems, however, focus on lineage management, retrieval and visualization of multi-dimensional datasets. They provide little or no support for analyzing or processing these datasets – the assumption is that these operations are too application-specific to warrant common support. As a result, applications that process these datasets are usually decoupled from data storage and management, resulting in inefficiency due to copying and loss of locality. Furthermore, every application developer has to implement support for managing and scheduling the processing.

Over the past three years, we have been working with several scientific research groups to understand the processing requirements for such applications [1, 2, 6, 16]. Our study of a large set of applications indicates that the processing for such datasets is often highly stylized and shares several important characteristics. Usually, both the input dataset as well as the result being computed are multi-dimensional. The basic processing step usually consists of retrieving the input data items selected by a range query, mapping the retrieved data items to output items and computing output

[*] This research was supported by the National Science Foundation under Grants #BIR9318183 and #ACI-9619020 (UC Subcontract # 10152408), ARPA under Grant #DABT 63-94-C-0049 (Caltech Subcontract #9503), and the Office of Naval Research under Grant #N6600197C8534. The Maryland IBM SP2 used for the experiments was provided by NSF CISE Institutional Infrastructure Award #CDA9401151 and a grant from IBM.

items by aggregating, in some way, all the retrieved input items mapped to the same output data items. Mapping between input and output items is done by projecting the input data points to points in the output attribute space and finding the output data items that correspond to the projected points. Furthermore, the correctness of the output usually does not depend on the order the input data items are aggregated. These aggregation functions correspond to the *distributive* and *algebraic* aggregation functions defined by Gray et. al [10]. For example, remote-sensing earth images are usually generated by performing atmospheric correction on 10 days worth of raw telemetry data, projecting all the data to a latitude-longitude grid and selecting those measurements that provide the clearest view. As another example, chemical contamination studies simulate circulation patterns in water bodies with an unstructured grid over fine-grain time steps and chemical transport on a different grid over coarse-grain time steps. This is achieved by projecting the fluid velocity information from the circulation grid, possibly averaged over multiple fine-grain time steps, to the chemical transport grid and computing smoothed fluid velocities for the points in the chemical transport grid.

In this paper, we present *T2*, an infrastructure for building parallel database systems that enables integration of storage, retrieval and processing of multi-dimensional datasets on a parallel machine. T2 allows for customization for various applications with the stylized processing structure described above, while providing support for index generation, data retrieval, memory management, scheduling of processing across a parallel machine and user interaction. It achieves its primary advantage from the ability to integrate data retrieval and processing for a wide variety of applications and from the ability to maintain and jointly process multiple datasets with different underlying attribute spaces. Several runtime support libraries and file systems have been developed to support efficient I/O in a parallel environment [7, 11, 14, 18]. These systems are analogous to T2 in that: (1) they plan data movements in advance to minimize disk access and communication overheads, and (2) in some cases, they attempt to optimize I/O performance by masking I/O latency with computation and with interprocessor communication. Also, T2 schedules its operations based on the completion of disk I/O requests, which is similar to the strategy used by disk-directed I/O [11] and server-directed I/O [14]. However, T2 differs from those systems in a number of ways. First, T2 is able to carry out range queries directed at irregular spatially indexed datasets, such as satellite data consisting of two-dimensional strips embedded in a three-dimensional attribute space, digitized microscopy data stored as heterogeneous collections of spatially registered meshes, and water contamination simulation data represented by unstructured meshes over simulated regions of bays and estuaries. Second, computation is an integral part of the T2 framework. Users provide T2 with procedures to carry out data preprocessing and analysis, and the required computations are performed in parallel with I/O and interprocessor communication. With the collective I/O interfaces provided by many parallel I/O systems, data processing usually cannot begin until the entire collective I/O operation completes. Third, data placement algorithms optimized for range queries are integrated as part of the T2 framework. Analytic and simulation studies [12] have shown that these algorithms allow T2 to exploit the disk bandwidth of the entire system, and evenly partition the workload across all the processors and disks.

T2 has been developed as a set of modular services implemented in C++. To build a version of T2 customized for a particular application, a *domain engineer*, who is authorized to customize the system, has to provide functions to pre-process the input dataset, access the individual input data items, project points between the input and output attribute spaces, and aggregate multiple input data items that map to the same output item. Several extensible database systems that can be tailored to support particular applications have also been proposed [3, 5, 17]. In addition to the functionality provided by a general-purpose relational database system, these systems also provide support for adding new storage methods, new data types, new access methods, and new operations. The incorporation of user-defined access methods and operations into a computation model as general as the relational model allows these systems to support a large number of applications. However, it also makes query optimization very difficult. A number of researchers have begun to address this problem [15]. T2 on the other hand, implements a more restrictive processing structure that directly mirrors the processing structure of a set of applications that process multi-dimensional datasets. Good performance in T2 is achieved through careful scheduling of the operations and good utilization of the system resources, not by rearranging the algebraic operators in a relational query tree, as is done in relational database systems.

In Section 2 we present an overview of T2 and discuss how customization is done. Section 3 provides preliminary performance results for two applications implemented with T2. We conclude in Section 4 with a description of the current status of both the T2 design and implementation, and discuss future work.

2 System Architecture

A complete customized T2 application consists of one or more *clients*, a *front-end process*, and a parallel *back-end*, as shown in Figure 1. A client program, implemented for a specific domain, generates requests to the front-end. The front-end translates the requests from a format defined by the application into T2 *queries* in a format understood by

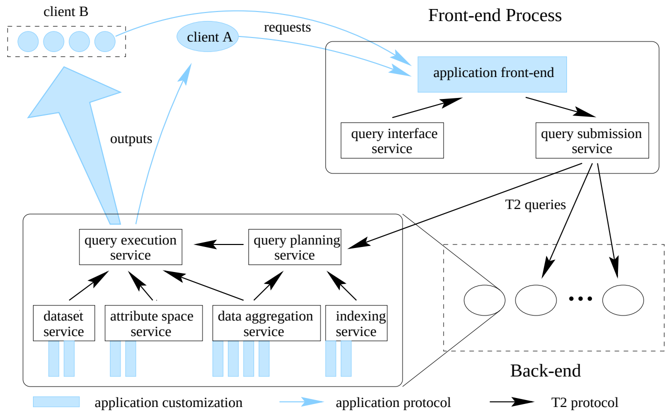

Figure 1. A customized T2 application.

the back-end, and forwards the queries to the back-end. The back-end, which is responsible for storing the datasets, performs the required data retrieval and processing, as specified by the queries. The results of the queries are then transferred back to the clients, or other destinations specified by the queries. The destinations could be sequential or parallel programs, as shown by clients A and B in Figure 1, respectively. T2 provides a set of modular services for the implementation of the front-end process and the back-end. Some of the services can be extended with user-provided functions, which are represented by the light bars in Figure 1. These functions are invoked by T2 at runtime to process the queries. T2 does not provide support for implementing clients, since their behavior can be entirely domain-specific.

The front-end process usually runs on a separate machine from the back-end. The interaction between the front-end and the clients uses a protocol defined by the application, while the interaction with the back-end is facilitated through the following T2 services:

- *the query interface service*, which supports browsing of datasets and user-provided functions registered with the back-end, and assists the front-end in translating domain-specific requests into T2 queries, and

- *the query submission service*, which buffers the queries that are ready to be processed and sends one or more queries to the back-end whenever the back-end is ready.

A T2 query specifies the dataset(s) of interest, the user-provided aggregation function to use for data processing, and where to send the output to. Currently, T2 can send the output through either a socket interface or a Meta-Chaos [9] interface. The socket interface is used for sequential clients, while the Meta-Chaos interface is mainly used for parallel clients.

The back-end, which runs on a parallel machine, stores and manages datasets, and performs the actual data retrieval and processing required by the queries. T2 assumes a shared-nothing architecture with a disk farm attached to the processors. Datasets are partitioned into data blocks, which are assigned to disks using either a user-provided placement algorithm or the default algorithm provided by T2. Each back-end process consists of the following T2 services:

- *the attribute space service*, which manages the registration and use of multi-dimensional attribute spaces and the user-defined projection functions, which project points between attribute spaces,

- *the dataset service*, which manages the datasets stored on the back-end and provides utility functions for loading datasets into the T2 back-end,

- *the indexing service*, which manages dataset indices, which are used to efficiently identify the portions of datasets that must be retrieved and processed,

- *the data aggregation service*, which manages the user-defined functions employed in data aggregation operations,

- *the query planning service*, which determines a query plan that can be used to efficiently process a set of queries based on resource availability, and

- *the query execution service*, which manages all the resources in the system and carries out the query plans generated by the query planning service.

Of these services, the attribute space service, the dataset service, the indexing service and the aggregation service allows new datasets and functions to be registered with the back-end. This is achieved in T2 through C++ class inheritance. That is, for each of these services, T2 provides a set of C++ base classes with virtual functions that are expected to be implemented by their derived classes. Adding a domain-specific entry into a service requires the definition of a user-defined class derived from the appropriate T2 base class and the implementation of the virtual functions for that base class. For example, T2 considers a dataset as a set of data blocks, each of which consists of a set of data items. In T2, data blocks are the unit of data retrieval. Adding a new dataset to the dataset service requires naming the source of the data blocks (i.e. file names), and defining a derived dataset class that includes the implementation of a virtual function that can be used to iterate through the data items in a given data block.

Customizing the T2 back-end therefore requires the domain engineer to register new projection functions, new datasets, new dataset indices and new aggregation functions with the appropriate services. Dataset indices are used by T2 to quickly identify the set of data blocks that need to be processed for a given query. The user-defined projection functions map the input data items to output items by projecting the coordinates of the input data items to the attribute space of the output. The user-defined aggregation functions implement the domain-specific aggregation operations.

The query planning service and the query execution service implement the core functionality necessary for efficient query processing. The query plan generated by the query planning service specifies the set of input data that needs to be retrieved from disks for the queries and the order in which the input data is retrieved for processing. Currently, T2 processes a query by having each back-end processor compute independent intermediate results using only the input data on the disk(s) assigned to that processor, with a final merge step to aggregate intermediate results across all processors. Our experience shows that, with the T2 default data placement algorithm, this strategy works well for our current set of driving applications. However, as for computing a parallel reduction operation in a parallel loop, there are multiple ways to process a T2 query. In general, choosing among these strategies is based on several factors, including the mapping between the input and output items, the placement of the input data on the disks, and the relative costs of computation, interprocessor communication and disk accesses, and is a topic that is currently under investigation.

The query execution service is responsible for efficiently carrying out the query plan generated by the query planning service. The primary feature of the query execution service is its ability to integrate data retrieval and processing for a wide variety of applications. This is achieved by pushing the processing operations into the storage manager and allowing processing operations to access the buffer used to hold data arriving from disk. As a result, it avoids one or more levels of copying that would be needed in a layered architecture where the storage manager and the processing belonged to different layers. To further reduce query execution time, the query execution service overlaps the disk operations, network operations and the actual processing as much as possible. It does this by maintaining explicit queues for each kind of operation (data retrieval, message sends and receives, data processing) and switches between them as required. Pending asynchronous I/O and communication operations left in the operation queues are polled and, upon their completion, new asynchronous functions are initiated when more work is expected and buffer space is available. Data is therefore retrieved and processed in a pipelined fashion.

3 Preliminary Experimental Results

To show the effectiveness of T2, we have customized the various T2 services for two applications: (1) Titan, a remote-sensing image database, and (2) the Virtual Microscope, a system for serving digitized high-power light microscope data. In this section, we present preliminary performance results comparing the two original systems with their counterparts implemented in T2, all running on an IBM SP-2 at the University of Maryland. The Maryland SP-2 consists of 16 RS6000/390 processing nodes, running AIX 4.2, with six disks (IBM Starfire 7200) attached to each processing node. All systems are compiled with the IBM C/C++ compiler, and the datasets for both applications are distributed across a subset of the disks, using the same default data placement algorithm provided by T2. In these experiments, we focus on the performance of the query execution service, and therefore the results presented in this section do not include the time to send the output back to the requesting clients.

3.1 Titan

Titan [6, 16] is a custom-built image database for storing and processing remotely sensed data. It currently contains about 24 GB of data from the AVHRR sensor on the National Oceanic and Atmospheric Administration NOAA-7 satellite. Each data item in the AVHRR dataset is referred to as an *instantaneous field of view* (IFOV), and consists of three key attributes that specify the spatio-temporal coordinates and five data attributes that contain observations in different parts of the electromagnetic spectrum. Titan dedicates one of the sixteen SP-2 processing nodes as a front-end node, which interacts with Java GUI client programs, and uses the other fifteen nodes as the back-end processing and data retrieval nodes. Four disks on each of the back-end nodes are used to store the AVHRR dataset. In prior work [6] we have shown that Titan delivers good performance for both small and large queries. Customization of T2 is done by implementing wrapper functions that directly call the kernel of Titan to both navigate and process the AVHRR data blocks.

Several sample queries were used to evaluate the performance of the T2 implementation against that of Titan. These queries select IFOVs that correspond to various areas of the world over a 10-day period and each generates a composite image by selecting the "best" IFOV among all IFOVs that map to the same output pixel, based on the clarity of the IFOV and the angular position of the satellite when the observation was made. Table 1 shows the total amount of data read from disk to resolve the sample queries, and the query processing time for both Titan and the T2 implementation. Note that one of the SP-2 nodes, and its four disks, was not available at the time the experiments were conducted, so all results reported in this section were obtained using fourteen back-end nodes.

The results show that the T2 performance is very close to that of Titan. Table 2 shows the time both implementations spent in processing the data blocks (the *comp* column), communicating and merging the intermediate results (the *comm* column), and the time spent for submitting and polling the

sample query	Australia	Africa	South America	North America	global
input data size	65 MB	191 MB	155 MB	375 MB	1.6 GB
Titan	3.8	11.0	8.5	22.3	107.5
T2	3.9	10.7	8.4	23.5	113.1

Table 1. The total number of bytes read to resolve the Titan sample queries, and the total query processing times (in sec.) for Titan and T2.

system	total time	comp	comm	others
Titan	107.5	104.3	1.3	1.9
T2	113.1	107.9	3.9	1.3

Table 2. Query processing breakdowns (in sec.) for the global query.

system	total processing time	computation	others
VM	1.74	0.36	1.38
T2	3.22	1.39	1.86

Table 3. Query processing times (in sec.) for VM and T2.

disk reads plus other software overhead (the *others* column) for the global query. The table shows that both implementations spent most of their time in processing the data blocks. T2, however, incurs more overhead in the computation and the communication than Titan. The computation overhead is caused by the wrapper functions, which in turn invoke the Titan computation functions. The communication overhead is due to the fact that each Titan back-end process only handles one query at a time, and therefore can estimate the sizes of all incoming messages to be received, whereas a T2 back-end process may receive messages from multiple simultaneous queries so cannot accurately predict message sizes. As a result, each Titan back-end node processor can directly post a non-blocking receive, while the current implementation of the T2 back-end probes for the size of an incoming message before posting a corresponding receive. We are redesigning the communication subsystem of T2 to reduce the communication overhead.

The query processing time reported does not include the time for generating the query plan. In both implementations, generating a query plan after the index is read from disk takes less than a second. The index for the AVHRR dataset is about 11 MB, and takes about 3.5 seconds to load into memory for both implementations. However, since Titan only uses one index for all Titan queries, it keeps the index in the memory of the front-end, which generates and distributes query plans to all the back-end nodes. T2, on the other hand, has the back-end read the index into memory for each query and discard the index after the query plan is computed. This is because there might be multiple indices in the T2 system, and caching multiple indices may significantly reduce the amount of memory available on the back-end nodes for query processing.

3.2 The Virtual Microscope

The Virtual Microscope (VM) system [2] provides the ability to access high power, high resolution digital images of entire pathology slides. A VM query specifies a high-power digitized microscopy image, the region of interest and the desired magnification for display, and an output image is computed by subsampling the high-power input image. No interprocessor communication is required in the current implementation of the Virtual Microscope.

A sample query that retrieves 357 data blocks (72 MB) is used to evaluate the performance of the two implementations. Table 3 shows the query processing time for running the sample query on eight back-end processing nodes. The results show that the T2 implementation is about 85% slower than that of the original VM implementation. Since the Virtual Microscope currently uses a simple subsampling algorithm to generate low magnification images from the stored high magnification data, very little computation is performed on the data blocks. As a result, high overhead due to excess virtual function invocation is incurred by the T2 implementation.

We are currently looking into ways to eliminate excessive function calls through sophisticated compiler optimization techniques (e.g., inter-procedural dataflow analysis). However, the functionality of VM is not limited to simply subsampling microscopy images at various magnification. Future projects that use VM will involve image compression/decompression three dimensional image reconstruction, and other complex image processing operations, all of which require significant computation to be performed on the data. This would offset the seemingly high cost observed by the current VM implementation with T2.

4 Current Status and Future Work

We have presented T2, a customizable parallel database that integrates storage, retrieval and processing of multi-dimensional datasets. We have described the various services provided by T2, and provided preliminary performance results for two applications implemented as customized T2 instances. The performance of the T2 implementation of the AVHRR database is very close to that of the Titan implementation, with both performing large amounts of computation on their input datasets. On the other hand, the T2 implementation of the Virtual Microscope system is about 85% slower than the original implementation, which performs very little computation on its input datasets. We are currently in the process of optimizing the query execution service to reduce this overhead. We are also evaluating the tradeoff between different query processing strategies and developing cost models for the query planning service. In addition, we are starting to investigate techniques for extending T2 to tertiary storage, to efficiently store and process datasets that are too large to fit into secondary storage. Finally, we are beginning to investigate how T2 can be integrated with services provided by more general purpose databases (e.g., relational or object-relational commercial databases). That would allow, for instance, the ability to efficiently do a join between data stored in a relational database and multi-dimensional data stored in T2.

Acknowledgements

We would like to thank Tahsin Kurc for his help in understanding the University of Texas bay and estuary simulation codes, and Henrique Andrade for help in the customization of the Virtual Microscope application.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. Tuning the performance of I/O-intensive parallel applications. In *Proceedings of the Fourth ACM Workshop on I/O in Parallel and Distributed Systems*, May 1996.

[2] A. Afework, M. D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital dynamic telepathology - the Virtual Microscope. In *Proceedings of the 1998 AMIA Annual Fall Symposium*. American Medical Informatics Association, Nov. 1998.

[3] D. Batory, J. Barnett, J. Garza, K. Smith, K. Tsukuda, B. Twichell, and T. Wise. GENESIS: An extensible database management system. *IEEE Transactions on Software Engineering*, 14(11):1711–1730, Nov. 1988.

[4] P. Baumann, P. Furtado, R. Ritsch, and N. Widmann. Geo/environmental and medical data management in the RasDaMan system. In *Proceedings of the 23th VLDB Conference*, pages 548–552, Aug. 1997.

[5] M. J. Carey, D. J. DeWitt, G. Graefe, D. M. Haight, J. R. Richardson, D. T. Schuh, E. J. Shekita, and S. L. Vandenberg. The EXODUS extensible DBMS project: An overview. In D. Zdonik, editor, *Readings on Object-Oriented Database Systems*, pages 474–499. Morgan Kaufman, San Mateo, CA, 1990.

[6] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high performance remote-sensing database. In *Proceedings of the 1997 International Conference on Data Engineering*, pages 375–384. IEEE Computer Society Press, Apr. 1997.

[7] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, Aug. 1996.

[8] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu. Client–server Paradise. In *Proceedings of the 20th VLDB Conference*, pages 558–569. Morgan Kaufmann Publishers, Inc., 1994.

[9] G. Edjlali, A. Sussman, and J. Saltz. Interoperability of data parallel runtime libraries. In *Proceedings of the Eleventh International Parallel Processing Symposium*. IEEE Computer Society Press, Apr. 1997.

[10] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the 1996 International Conference on Data Engineering*, pages 152–159, Feb. 1996.

[11] D. Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74. ACM Press, Nov. 1994.

[12] B. Moon and J. H. Saltz. Scalability analysis of declustering methods for multidimensional range queries. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):310–327, March/April 1998.

[13] The Oracle 8 spatial data cartridge, 1997. *http://www.oracle.com/st/cartridges/spatial/*.

[14] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective I/O in Panda. In *Proceedings of Supercomputing '95*, San Diego, CA, Dec. 1995. IEEE Computer Society Press.

[15] P. Seshadri, M. Livny, and R. Ramakrishnan. The case for enhanced abstract data types. In *Proceedings of the 23th VLDB Conference*, Athens, Greece, Aug. 1997.

[16] C. T. Shock, C. Chang, B. Moon, A. Acharya, L. Davis, J. Saltz, and A. Sussman. The design and evaluation of a high-performance earth science database. *Parallel Computing*, 24(1):65–90, Jan. 1998.

[17] M. Stonebraker, L. Rowe, and M. Hirohama. The implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, Mar. 1990.

[18] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. Passion: Optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.