

Conquest: CONcurrent QUERies over Space and Time

Silvia Nittel Kenneth W. Ng Richard R. Muntz

{silvia,kenneth,muntz}@cs.ucla.edu
Computer Science Department
University of California
Los Angeles, CA 90095-1596

Abstract. The need and opportunity to efficiently find patterns and features in the vast and growing scientific data sets of today is apparent. In this paper, we present an extensible and distributed query processing system — Conquest (CONcurrent QUERies over Space and Time) which delivers high performance for non-traditional database query applications. Conquest is composed of components: *query management subsystem*, *query execution engine* and *user-interface subsystem*. Both operation and data parallelism are supported. Scientists can arbitrarily introduce their familiar well-known raster and vector geometry data types as well as particular algorithms for specific applications. Tools are available to help support new data types and operators so that developers can focus on programming their core logic of algorithms without paying a penalty to deal with the novel execution environment’s features.

1 Introduction

With the vastly growing availability of satellite raster data sets, either from commercial or government satellites, the need and opportunity to efficiently mine these data sets for specific phenomenon or features (e.g. weather patterns and their impact on other areas of the earth) becomes more apparent. Scientists would like to find patterns and features in large satellite raster data sets, relate their results to other data sets, and share their findings with specialists in closely related fields to collaborate on a larger scientific task. For example, at UCLA’s NASA-sponsored ESP²Net project [ESP98], scientists in Atmospheric Science and Oceanography at the Jet Propulsion Lab (JPL), the Scripps Institute in San Diego and the Department of Atmospheric Science at the University of Arizona collaborate on the task of convection detection and tracking over the western Pacific as well as its influence on the rain fall patterns over the West Coast of the U.S. Typical tasks include mining ISCCP DX and CL data sets for cloud coverage and movement, and extracting such features in the form of vector data (polygons). Results are used as input for follow-up studies which relate the cloud coverage occurrences temporally and spatially to rainfall events on the U.S. West Coast.

To perform such raster analysis and feature discovery tasks efficiently, the following requirements for a support tool exist: (1) a data model that supports

a large variety of both raster data types such as coverages and vector geometry data types (points, lines, polygons), (2) high-level query specification capabilities, and (3) a processing architecture that provides for high performance of query execution necessary for the vast amounts of data that are often processed during a query. Furthermore, such a tool needs to provide an extensible data model that allow users to define their own data types; it should also allow for extensible query processing in the sense that a scientist can define his/her own algorithms (i.e. operators) for a particular feature extraction task.

Due to the huge amount of data that is normally processed for feature discovery and extraction tasks, non-data flow oriented languages and systems, are not particularly efficient for this kind of massive data processing. In such a non-data flow environment an operator performs a computation only when all its operands are available and then outputs its result. This one-time activation is a severe restriction to massive data processing which requires high performance. In particular, for large data sets it is important to avoid writing intermediate results to secondary storage. Using, however, a data flow paradigm for such a tool makes it difficult for scientists or programmers to write operators since they are commonly more accustomed to programming in a procedural language than in a data flow or functional programming language. Also, writing operators in a procedural language makes it easier to 'wrap' available data analysis legacy code instead of re-writing it within a dataflow language.

At the UCLA Data Mining Laboratory, we have developed and introduced Conquest [St95, Sh96a]. Conquest provides a data model that supports both raster and vector geometry data types, and also allows a scientist to define their own data types and operators. Conquest provides a graphical user interface to compose and execute queries as shown in Figure 2. For the execution of queries, the approach taken in Conquest is to use a data streaming scheme; here, *streams* of data objects instead of single data items are constantly exchanged between operators, i.e. an operator repeatedly executes as long as there is more data to be consumed from its input.

Based on the data streaming paradigm, Conquest parallelizes a query execution by (a) distributing operators to different processors and (b) replicating an operator ("cloning") to several operator instances and splitting the original input stream among these identical instances. Based on available resources, a query is compiled, optimized, and executed in a workstation farm environment. However, highly efficient, parallelized materialization of a data flow diagram on a Von Neumann architecture is not trivial since it is hard to analyze the behavior of a data flow diagram before actually executing it. For example, it is often difficult to know the resource availability (memory, CPU, I/O, communication, etc.), data selectivity and other runtime properties of each operator and data flow link in advance. To alleviate the problem and provide for improved efficiency, Conquest uses a dynamic re-optimization scheme which collects runtime information, makes predictions of query and system behavior, and re-configures a query execution plan based on cost prediction on the fly during runtime [Ng98, Ng99a].

Conquest places a strong emphasis on support for writing and integrating op-

erators into the environment. Scientists often have large amounts of data analysis legacy code available that they would like to easily integrate into the environment. The legacy code might be written in different programming language which should, nevertheless, be usable in Conquest. Providing such a complex environment as Conquest, the task of writing an operator is not simple. Therefore, several choices have been made to ease this task: (1) the programming interface for *user-defined operators* in Conquest has been designed to be object-oriented and procedural in contrast to the parallel, data streaming based execution environment. A Conquest operator developer implements his/her algorithm according to its procedural description, i.e. mapping one or more input data objects to one or more output data objects, and can choose a procedural language that he/she is accustomed to, or can use the programming language that had been used for already existing data analysis code. (2) Conquest provides the operator development tool *opGen* which automates the task of integrating an operator written in a native programming language with the execution environment of Conquest. *opGen* separates the system code from the operator logic, and generates the system code automatically for an operator. The design of *opGen* was based on Design Patterns [?].

A scientific data mining tool such as Conquest also has to be able to work in a heterogeneous, distributed workstation environment, possible using a wide-area network, to make use of all available processing power in a typical scientists' environment and enable collaboration. Evolved and re-implemented over several generations, the recent version of Conquest is implemented using Iona's CORBA implementation OrbixWeb 3.1 which is used to provide location- and platform transparency for query execution; the current version of Conquest runs at UCLA, JPL (Pasadena), Hughes Research Laboratory (HRL) (Malibu), the Scripps Institute (San Diego) and at the University of Arizona (Tucson).

The remainder of this paper is organized as follows. An overview of the Conquest system is given in Section 2. Section 3 presents the approach taken in Conquest to integrate user-defined operators into the parallel execution environment. Section 4 describes capabilities of Conquest regarding user-defined data types, the integration of heterogeneous data sources and formats into the Conquest environment as well as specific issues regarding the exchange of data between platforms and machines. Related work is discussed in Section 5, while Section 6 presents our conclusions.

2 The Conquest System Architecture

The overall system architecture of Conquest is shown in Figure 2. The system consists of three parts: the *query management subsystem*, the *query execution engine*, and the *query user interface subsystem*. The query execution system is responsible for the distributed, parallel execution of a scientific query. The query user interface supports composition of scientific queries graphically, and execution of these queries without programming effort. The query management subsystem performs the compilation and optimization of a query, and includes the

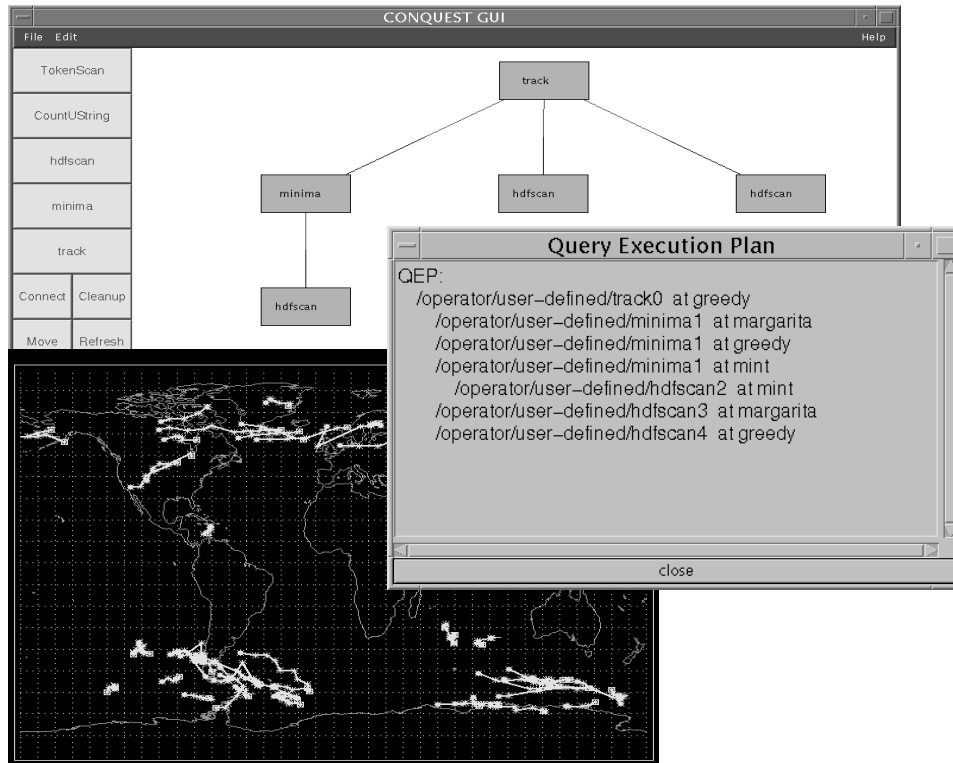


Fig. 1. A typical Conquest query execution

system catalog (or registry), and the system measurement module for this task. The latter is used for dynamic re-optimization of long-running queries which is also performed by this subsystem. We will discuss each of these three subsystems in the following sections. The current implementation of Conquest is based on the Common Object Request Broker Architecture (CORBA) [Ob98]¹, and we assume in our discussion that readers have basic knowledge of this architecture. The use of CORBA as an underlying infrastructure allows for platform- and location-independent implementation, parallelization and execution of queries.

2.1 Query management subsystem

The *query manager* is a well-known CORBA object within Conquest and for Conquest clients, and interacts with the front-end graphical or textual user interface to accept query requests. Upon receiving a query request, which is internally expressed as an algebraic expression, the query manager creates a *query*

¹ The system we choose is IONA's OrbixWeb 3.0, which is a Java-based CORBA implementation [Io97]. Conquest is primarily written in Java.

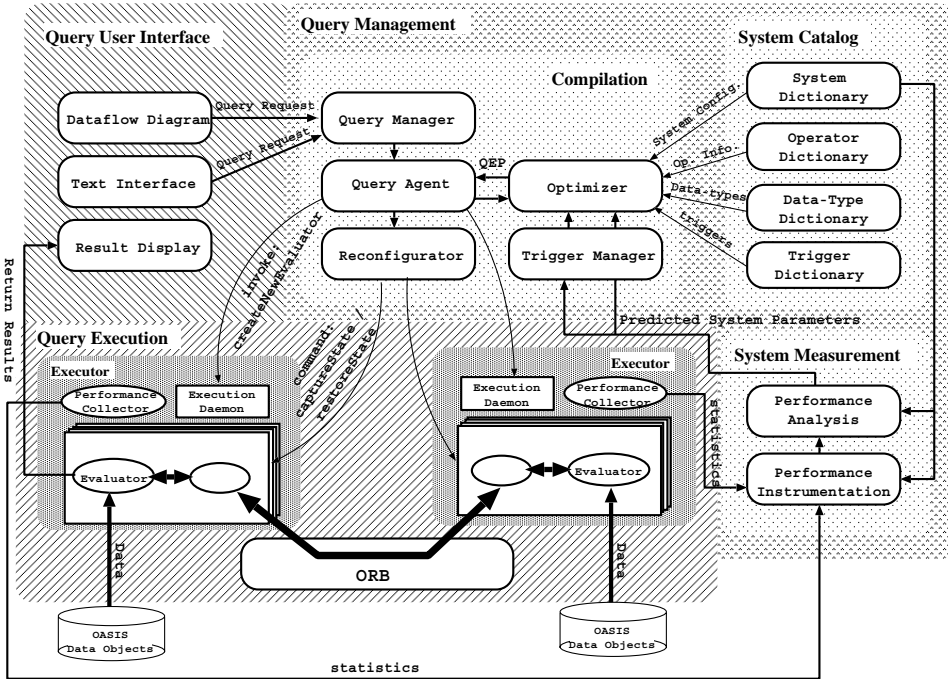


Fig. 2. The Conquest system architecture.

agent which is responsible for executing and answering this particular query. The query agent sends the algebraic expression to the *query optimizer* which generates a query execution plan (QEP) [Sh96b, Ng99b]. To materialize and execute the QEP, the query agent contacts several execution demons each of which runs on either a local or remote host machine. Using CORBA, Conquest easily supports a distributed execution that allows to include remote sites for query execution. Remote sites are normally used if data used for a query is available on this site, thus, avoiding to stage the data to a local machine. An execution demon is constantly running on all host machine that are registered within Conquest; each demon listens to requests of a query agent. The execution demon dynamically starts local *execution servers* on request which are also CORBA objects². An execution server starts several *evaluators*, and sets up the communication between evaluators according to the QEP. An evaluator is responsible for executing one Conquest operator according to a data streaming paradigm. For performance reasons, evaluators are implemented as pure Java processes (not as a more heavy-weight CORBA objects).

The query agent plays a central role over the life cycle of a query evaluation. This component is responsible for coordinating the re-optimization and

² An execution daemon actually maintains a pool of shared and unshared execution servers to reduce the response time due to launching an execution server.

re-configuration of a query; it monitors the evaluators so that in case of an evaluator crashes, the query is either reconfigured or aborted. Furthermore, it responds to the requests forwarded from the GUI or client to pause, restore, and abort the query under execution (scientific queries are typically long-running), and it kills the related evaluators when the query is finished or aborted.

The *optimizer*, *reconfigurator* and *trigger manager* cooperate to implement the triggered run-time re-optimization [Ng99a]. The trigger manager evaluates trigger rules registered in the *trigger dictionary* to determine whether a run-time optimization should perform or not. Each trigger rule consists of three parts: *event*, *condition* and *action*. An event can be a dramatic change of a system parameter, alarm clock time-out and so on. The condition part specifies the criteria a dynamic optimization should be carried out while the action part suggests possible plan reconfiguration options to the optimizer. The optimizer is responsible for generating a new optimal QEP taking the actual state of the query in account; the reconfigurator carries out the reconfiguration from the old QEP to the new QEP. For details about re-optimization and reconfiguration, we refer readers to [Ng98] and [Ng99a].

The system catalog maintains both system configuration information and various dictionaries. The system catalog is a tree-structured database which interfaces with other parts of the Conquest system via a CORBA interface. It contains the system configuration database, the operator dictionary, the user-defined type (UDT) dictionary, as well as the trigger dictionary.

The system measurement module provides *instrumentation* and *analysis* services. The instrumentation service collects various system and query performance information, and the analysis service predicts the system information using mean-based forecasting methods, which compute averages of collected parameter values over a certain period as the forecasted value [Wo97], and provides the analysis information to the optimizer and trigger manager. More sophisticated algorithms for performance prediction are under development.

2.2 Query execution engine

The query execution engine is the core of the Conquest system. It loads user-defined operators and other user-defined entities³, organizes the "pipelines" between these operators to realize the user-specified query according to the query data flow diagram (input from the optimizer), executes the query in parallel, and supports dynamic query re-optimization, i.e. changing the query execution plan during the execution of a query.

For a Conquest execution environment, we assume a local area network and a cluster of workstations and/or PCs each of which can be a different platform; therefore, all computer resource in an environment can be used for query execution. Each machine is registered in the Conquest system catalog, and is

³ When we say "user-defined," we really mean "user-extensible," because there are system built-in operators and other built-in entities. The Conquest system treats built-in operators/entities just like other user-defined operators/entities.

considered as a potential *executor* of a Conquest query. Physical instances of user-defined operators are executed inside an *evaluator* as shown in Fig. 3. A typical evaluator contains:

1. a user-defined operator instance that performs the computation;
2. input buffers that temporarily store input data objects from producer evaluators;
3. output buffers that temporarily store output data objects to be distributed to consumer evaluators;
4. a distributor (either built-in or user-defined) that sends produced data objects to consumer evaluators.

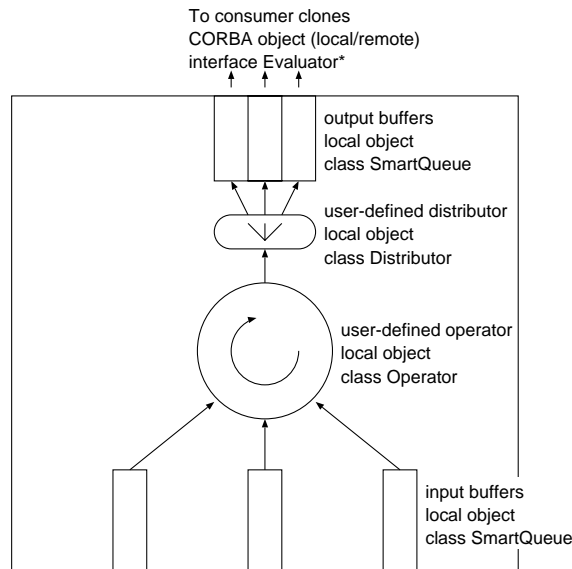


Fig. 3. The Conquest evaluator object.

A logical operator used in a query might be split up into several evaluator instances during execution, each of which processes a portion of the input data (on one machine or on several machines). Evaluators can be distributed over any number of hosts, and evaluators on different hosts are connected via the Object Request Brokers (ORB) on each platform (using the IIOP protocol). Evaluators take care of synchronization issues such that input/output buffers do not get overflow or underflow; they also batch small data items which are output data of an operator to larger data blocks since larger data blocks are more efficient to transmit over the network (as input data to a remote evaluator). Evaluators provide functionality to assist dynamic re-configuration of a query execution plan (QEP), e.g., check-pointing operators that are suspended, support of saving

and restoring buffers (and other elements of execution state) before and after reconfiguration, etc.

As mentioned, an evaluator is created as a thread in a *execution server*. Evaluators on the same machine can share one execution server process with other evaluators from the same query, or share it with evaluators from different queries, or be specified to execute in a separate process. Executing evaluators from different queries in the same execution server is not safe because a misbehaved evaluator can cause the execution server process to crash and abort other queries sharing the same execution server. Executing evaluators of the same query in separate execution servers, however, is “over-safe” in many cases, but has proven to be necessary for running operators wrapped from legacy C/FORTRAN programs which use global static variables. All of the three styles are supported in Conquest.

When a query is executing, the *performance collector* collects various performance statistical information about operators and the environment. The statistical information is used as the basis for run-time query optimization.

2.3 Query user interface subsystem

The query user interface subsystem provides several ways to access the Conquest query execution subsystem. A query can be defined by using (a) the graphical user interface (GUI), (b) the text interface, or (c) the query management CORBA interface.

The GUI is provided to allow scientists an easy-to-use interface to define Conquest queries in a graphical manner. The GUI extracts operator definitions from the operator dictionary in the system catalog module, and displays them as buttons in the GUI. The GUI allows the user to compose a data flow diagram by selecting operators from the operator dictionary display, dragging and dropping them to a composition area, and connecting them with arrows representing data flow between operators. Furthermore, clicking on an operator starts a display for an operator’s parameters, and allows to enter values (e.g. file system path for a file). The text interface does all this using a textual language (e.g., SQL).

The result display is user-defined and displays the query result (e.g., cyclones) either in text format, or (more commonly) in visualized form. The result can also be stored to a persistent store such as a file or a spatial-temporal database.

Another user interface not shown in is for manipulating the system catalog. A graphical user interface is provided for users or administrators to add, remove or modify system configuration information, user-defined operators, user-defined operators, and user-defined triggers.

3 Integrating User-Defined Operators in a Parallel Execution Environment

Our goal is to fully support parallelism [Ng99b] and run-time query re-optimization [Ng99a] in an extensible distributed environment, and still accommodate user-defined operators in the Conquest stream processing environment. Implementing

an operator in such an environment, however, requires a fair amount of support code to fit into the execution and optimization environment in addition to the core data processing logic within the operator. As a result, programming an operator in such an execution environment can be quite cumbersome.

3.1 Integrating User-Defined Operators

To (partially) alleviate the overhead, an earlier version of the Conquest query service proposed a declarative API called “CODL” (*Conquest Operator Definition Language*) [Da95] that assists scientists in developing user-defined operators. The overall goal of “CODL” is to simplify and minimize the amount of code a programmer has to write. A CODL definition is composed of four sections: *input*, *output*, *parameter* and *state*. In the *input* and *output* section, the input and output data schema of an operator are specified. The *parameter* section contains the definitions of read-only execution parameters which are well-known Conquest data types and whose values are specified at run-time by the optimizer. Variables defined in the *state* section are used to accumulate and record execution information (e.g., aggregate read records) internally at run-time. A code generator scans the CODL definition and generates an operator implementation skeleton in the C language. A user fills in the skeleton code the core processing logic in order to define an operator.

Although CODL is successful in alleviating some of the tediousness of operator implementation, there are some areas for improvement. First, the source code generated by CODL is not complete yet. It is still cumbersome to change and complete an operator definition. An inexperienced operator implementor might mistakenly delete or alter some support code, since the support code (required for Conquest execution control) generated by CODL is integrated with the user-defined operation code in an operator implementation. Also, due to this fact, any changes to the extension of execution services in Conquest (e.g., the new dynamic QEP re-configuration) can require operator implementors to modify and recompile each operator definition manually. Thus, each change to the Conquest environment might result in scientists having to rewrite existing operators. Finally, CODL does not provide an approach for wrapping legacy code for operators which is important requirement since scientists have invested in building analysis tools for over 20 years with proven algorithms and analytical process that are widely used today, and would like to wrap and integrate this ‘legacy’ code as operators rather than re-implement the analysis processes. The tradeoffs and resulting design principles and implementation issues for the Conquest operator programming interface are discussed next.

3.2 Operator development tool *opGen*

We developed the *operator development tool* to alleviate the problems mention above that are not solved by CODL. The operator development tool accepts C++ code as input and produces C++ and Java code as output. The choice of C++ was motivated by its common use as a programming language in computing

system. Java, on the other hand, is the primary implementation language of the Conquest system. However, the principles presented can also be applied to other object oriented programming languages⁴.

In designing the Conquest operator development tool, we had to consider how to integrate a user-defined operator programmed in a native programming language (e.g., C++) with the Conquest execution environment which is programmed in OrbixWeb (a Java-based implementation of the CORBA architecture) [Io97]. We separated the code that is needed to execute programs in a native language other than Java from the operator implementation itself. As a result, developers can focus on implementing operators with their familiar languages without considering the language used for the execution environment. Our approach is illustrated in Figure 4.

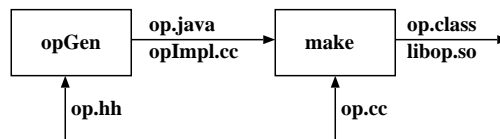


Fig. 4. Implementing Operators

Given a C++ header file in which operator attributes and function interfaces, and a properly specified input and output data schema are defined, the operator development tool **opGen** is used to generate Java and JNI (Java Native Interface) code for an operator under development. An operator implementor works as if he/she is programming in a pure C++ environment (or other programming language) and does not need to be concerned with idiosyncrasies of the *real* execution environment. To program a user-defined operator, the implementor completes two required functions: **init()** and **next()**. The **init()** function initializes operator object variables with parameters and/or default values. The **next()** function performs an iteration of the computation. The implementor can choose either processing one input data object per iteration (hence there may be no output after an iteration) or out-putting one data object per iteration (hence one or more input data objects are processed). Both approaches are supported. Conquest supports run-time query re-optimization as an optional feature [Ng99a].

To participate in a run-time re-optimization, an operator has to provide three more functions: **isSuspendable()**, **getContext()** and **setContext()**. The **is-**

⁴ Fortran, a popular programming language used in scientific computing, is not object oriented. Our current approach to handle the legacy Fortran code is to identify them as library functions called by C++ operator objects. A similar attempt can be found in [Br96]. Since we have de-coupled the execution support code (in Java) from user-defined operation implementation, **the update of an operator implementation, e.g., a pure C++ implementation replaces the existing Fortran code - unclear to me**, does not cause any change in use of this operator.

Suspendable() function indicates whether the computation can be suspended at the current point (between two iterations). The getContext() function captures the execution state of this operator for rebuilding the execution state later on via the setContext() function. The details about run-time re-optimization can be found in [Ng98, Ng99a]. After having finished the code for the user-defined operator in C++, an operator implementor uses a makefile generated by **opGen** to produce a Java executable class and a C++ shared library. At run-time, the Conquest environment executes Java classes which calls the actual user-defined functions in C++ shared libraries.

3.3 Example

In the following, we use the *minima* operator as an example to illustrate the concepts used in implementing an operator. Given a snapshot of extracted sea-level pressure data which contains 44 (latitudes) \times 72 (longitudes) sea level pressure values, the user-defined operator *minima* is responsible for extracting the location of local minima in a sea level pressure field with certain constraints. An algorithm chosen here is often based on a user's (typically a scientist) unique definition of the phenomenon. In our example, a local minimum is detected at point (x, y) if the sea-level pressure recorded at that locale is a certain defined amount less than the average value computed using a 5×5 neighborhood of grid points centered at (x, y) . A threshold was set to 5.5 millibars to permit the detection of large, shallow low pressure areas. If the spatial resolution of the data set is coarse, the *minima* operator may also smooth the grid data with a surface pressure function to better estimate locations of extracted minima.

The C++ header file of the minima operator is as shown in Figure 5(a). Given the input data type (i.e., NDimArray) and the output data type (i.e., SetOfPoints), the operator development tool produces, as mentioned above, **minima.java**, *minimaImpl.cc* and a makefile. The **minima.java** is as shown in Figure 5(b). The Java operator object and the C++ operator object are "bridged" with the *mPeer* specified as an attribute of the Java object class. That means that access to a method of the Java operator which is used in the internal Conquest environment is transferred for execution to the respective method of the user-written C++ operator which performs the desired computation. After the scientist has implemented the core logic of the minima extraction algorithm, the makefile is used to generate executable code that is used in the Conquest execution environment for running a query. Note, that if additional library functions which can be developed in the same native language (i.e., C++) or another language (e.g., FORTRAN) are used, the makefile can be modified to include them in the library path; this allows to implement an operator in heterogeneous programming languages.

4 User-Defined Raster and Vector Geometry Data Types

Large scale non-traditional databases systems in the geospatial and GIS area often involve the processing and handling of a large variety of data objects such

<pre> #ifndef MINIMA_HH #define MINIMA_HH // #include "Operator.hh" #include "NDimArray.hh" #include "SetOfPoints.hh" // class minima : public Operator { public: minima(); // default constructor ~minima(); // destructor // required member functions void init(PParam *paramSeq); PUDT *next(JNIEnv *env, jobject obj); // other member functions private: // parameter info long mSmooth; // fit minima to a spline? 1-Yes; 0-No // state information int mCount; // frame count int mNumMin; // External variables used in minutil.cc // other methods ... }; // handle to minima typedef minima* Pminima; // #endif </pre>	<pre> // User-defined operator -- // standard to all operators except the operator name import edu.ucla.ConquestV2.user.*; import java.io.*; public class minima implements Operator { // load shared library of operator in C++ static { System.loadLibrary("minima"); } // pointer to C++ class protected int mPeer = 0; protected UDTInput mInBufSeq[] = null; // constructor public minima() { // create a C++ operator object mPeer = (int)nativeCreate(); } // native methods linked to C++ public static native int nativeCreate(); public native void nativeInit(Param[] params); public native UDT[] nativeNext(); public native void nativeClose(); public native boolean nativeIsSuspendable(); public native byte[] nativeGetContext(); public native void nativeSetContext(byte[]); // other methods ... } </pre>
(a) minima.hh	(b) minima.java

Fig. 5. Definition of User-Defined Operator Minima

as raster, vector geometry, etc. which can not be represented conveniently within a relational data model. Therefore, abstract data types have been introduced to model more complex, user-defined data objects for these applications. A large variety of today's database systems such as the *PREDATOR* database system [Se97b] view the world as an integrated collection of data types, each of which supports a declarative, optimizable query language, and optimizes queries in an ORDBMS with enhanced abstract data types [Se97a].

4.1 User-Defined Spatial Data Types

Conquest provides a set of well-known spatial data types based on the OpenGIS object model. The Conquest data model supports vector data types such as points, polygons, line string, etc. as well as raster data types. Both categories are available in different temporal spatial reference systems such that e.g. vector data can be represented in 2D and 3D Cartesian Coordinate systems as well as in 2D and 3D Geographic Coordinate systems. Furthermore, Conquest also

allows users to define their own spatial data types, and use them in combination or without the predefined set of spatial types. Types are normally chosen based on the operators used.

Data that is used for queries is available via heterogeneous data sources such as databases, GIS, or archive file formats such as netCDF, HDF or even data set-specific formats. Traditional query systems require that data is made available in a uniform format, and normally is ingested into a centralized or distributed DBMS before queries can be executed. On the other hand, heterogeneous DBMS and their query components [Ni97, Pap95, To97, Ha97, Ro97] provide a middleware that makes a uniform (spatial) data model and query language available to the scientist/users, and provides automatic mapping of data types and queries to the data model and query language (if available) of the underlying data stores. Thereby, the middleware takes advantages of the query capabilities of the data stores, and processes parts of a query within the data stores. Such a model, however, does not work well with a data streaming query execution model as used in Conquest.

Therefore, Conquest 'wraps' data sources via so-called *data source scan operators*, and 'streams' the data into the Conquest environment for execution. A data source scan operator has, as all operators in Conquest, a predefined input data type, and output data type. E.g. an input data type for a DBMSScan operator is the table name, the return attribute names, and the where-clause to subset the table content. The DBMS returns the data via a cursor or the complete data might be buffered in the DBMSScan operator. Next, the operator uses the data to instantiate Conquest data objects of the equivalent Conquest data type, and make the data available in a data streaming fashion to the first consumer operator. Similarly, an HDFScan operator that accesses multidimensional arrays in HDF files exports the Conquest data type *n-dimensional array*; the input to the operator are the number of dimension of the array, the file path, and other data. Parts of these parameters are provided by the user when defining the query (e.g. file path), others are determined by Conquest. All input parameters for data source scan operators are bound at compile time.

A similar scheme is applied to store the results of a query. A query result can either be streamed to a visualization operator implemented using the IDL package, or to an operator that stores the data either to a DBMS, GIS, file, or any defined data store and format.

4.2 Conquest Schema Translator

Using Conquest in the geoscientific area, a strong emphasis on user-defined operators exists. Unlike in traditional database systems, the emphasis on operators is more significant than the emphasis on data types. This is due to the fact that scientists have been investing in building analysis tools for over 20 years with proven algorithms and analytical processes that are widely used today. Thus, scientists are more interested in porting analysis code as operators to Conquest, rather than defining a uniform data model and adapting analysis algorithms to this model.

These kinds of data types include various geometric object types of different dimensionality such as points, lines, polygons and volumes. These data objects may further be used to construct more complex data types in particular applications, e.g., set of points, temporal sequences of polygons, etc. To support efficient geoscientific computing, we have proposed the *field model*, which supports a variety of OpenGIS compliant data types by capturing recurring characteristics of spatial-temporal data, in the previous version of Conquest [Sh96a].

Though the *field* model has been proven to be simple but powerful to describe geoscientific data, scientists rather expect the data types commonly used in their current analysis tools to be recognized in the Conquest system so that experiments or parts of experiments can be ported directly to Conquest. This leads to the problem that Conquest might support a variety of operators that originate from different sources each one expecting slightly different input parameters⁵. Thus, structural conflicts may occur as a result of schema conformation. Therefore, data records from a 'producer operator' (of data items) have to be translated to conform to the target schema of the 'consumer operator' according to the data streaming paradigm. This leads to different issues.

First, it is obvious that the schema translation will introduce additional overhead. For example, assume that a producer operator and a consumer operator use different data representations. If a structural conflict occurs, data records need to be re-structured to conform to the input schema of the consumer operator before they can be consumed by the operator. This cost has to be reasonably small; otherwise, it is not worthwhile to integrate these two data models.

Second, the schema translation has to be available in both directions. For example, a schema represented in well-known data types should be translatable to an equivalent schema represented with equivalent, but slightly different data types, and vice versa. With both translations available, the choice of operators during query optimization will not be limited due to the data model issue. However, if there are n data types and all of them can be translated directly, the number of required translation functions will be $n(n - 1)!$. On the other hand, such an approach is not flexible since new translation functions must be added to all existing types when a new type is introduced. Therefore, the best option could be to choose one neutral data model as the media. All other data types are equipped with translation functions to and from the chosen model. Whenever a new data type is introduced, if it is with translation functions from/to the chosen model, it can then be translated from/to other types via the media model. Since the Conquest field model has been proven to be simple but powerful in representing geoscientific spatial-temporal data objects, we chose it as the media.

Third, the schema translation should "fit" in the *stream processing* paradigm that is used in the Conquest execution environment. In other words, operators in a QEP can continuously consume portions of input streams and produces portions of output stream using the schema translation in between as another operator. Fourth, the translation structure should be extensible. A well-known

⁵ Note, that our intend is to simplify the task of wrapping 'legacy' analysis code.

data type can be added into the Conquest query execution environment whenever the corresponding translation rules are defined. With this feature, scientists can start with a small subset of the well-known data types and then define more data types that are suitable for the geo-science studies later.

Finally, the schema translation should be transparent to geo-scientific operator implementors, thus programmers can focus on the core data processing logic. Once the well-known data types are registered in the system catalog, scientists should not get involved to resolve the difference of underlying data models when programming an operator.

As a consequence of these decisions, we introduce an operator that does not correspond to any operator in the logical algebra. The purpose of this operator is not to perform any logical data manipulation but to complete the data and schema translation. This operator is called the *translator* operator which is comparable to the “glue” operators in Starburst [Lo88] and “enforcer” operators in Volcano [Gr94] if we consider the schema as a “property” of data records. However, while the “glue” and “enforcer” operators change properties of the data stream such as the order of elements the *translator* operator in Conquest changes the properties (schema) of the data items, i.e. the elements of the data stream.

The *translator* operator is inserted into a QEP by the optimizer if schema translation is necessary which is transparent to operator implementor. Also due to the data streaming paradigm used in Conquest, the operator is only inserted in-between two actual data processing operators. The additional cost introduced by schema translation is available to the optimizer via the cost function of the *translator* operator. As a result, the optimizer can decide on schema translation in a QEP optimally.

In designing the *translator* operator, we de-couple the schema translation functions of well-known data types from execution support code required in the Conquest execution environment by adopting the *Bridge pattern design* [Ga95], as shown in Figure 6. This separation allows a new well-known data type can be introduced any time since the appropriate schema translation function is dynamically linked at QEP execution. Two major schema translation functions should be defined for each well-known data type: *fromField()*, which translates the schema from the Conquest field model to the well-known data types; and *toField()*, which translates the schema from the well-known data types to the Conquest field model. We note that *fromField()* and *toField()* methods are optional to application developers. If these two object functions are not defined, it allows less optimization options but does not affect the validity for the query processing.

4.3 Marshalling and Un-marshalling

User-defined types typically refer to other (user-defined) types. E.g. if a *setOfPoints* is modeled as a set of point locations and associated values, a *trajectory* of *setOfPoints* is complex since it is represented as an ordered list of references to *setOfPoints* objects. In a workstation farm environment, which is typically used for executing a parallelized query in Conquest, complex data objects have

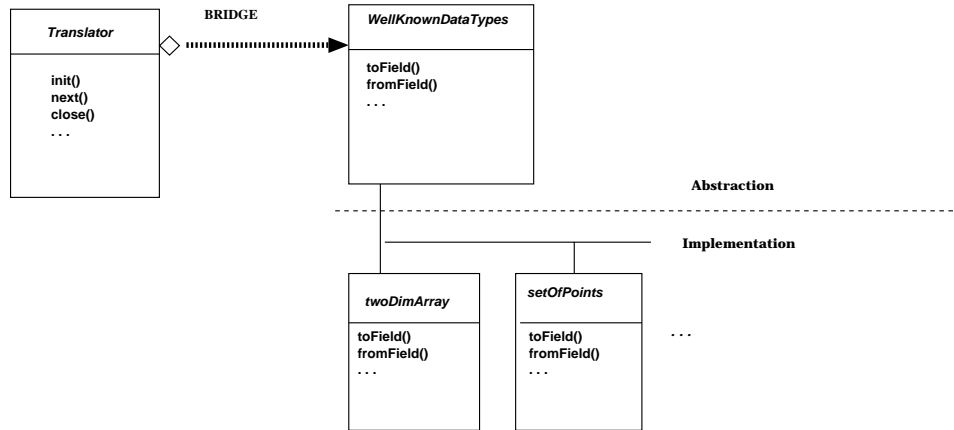


Fig. 6. Schema Translator Operator

to be passed among processes and machines; the machines might additionally be different platforms. Therefore, operations to “flatten” data objects, i.e. to represent a complex data object via a simple structure, and operations to rebuild those complex data objects are required. Also, efficient exchange of data streams is achieved by grouping and packing several data items into a larger package. However, without knowing the semantics of a data type, data objects that are grouped may be misinterpreted during the communication process. For example, data objects each of which contains a list of points may not be equal in size. To properly identify the single items, the system has to know how many points each data object contains. Therefore, a semantic-oriented means to support data marshalling is necessary. Furthermore, the process of data passing should be transparent to geo-scientific operator implementors, and scientists/programmers should not need to get involved in resolving any data communication problems when programming an operator. Finally, an uniform interface should be provided such that a user-defined type developed in programming language A can be automatically ‘imported’ into an equivalent data type implemented in a different language. Also, as we already described, schema translator operators might be inserted by the Conquest system during compilation and optimization which also need to rely on a uniform interface to read and translate data item. Normally, an environment such as CORBA would take care of packaging data in a platform-independent way, and unpacking it correctly on a different platform, but since operators are executed via regular C++ or Java processes, the Conquest execution environment has to provide for Conquest-specific means to flatten data and exchange it platform-independently.

To address these issues, we propose a novel approach to specify user-defined data types, which requires developers to introduce a new data type by specifying two functions: **toByteArray()** and **fromByteArray()**. **toByteArray()** is an object function which creates and returns a byte array of *this* object instance. In

other words, this data type object instance is represented in the form of a byte array. A typical `toByteArray()` routine follows the steps:

1. compute the required length of the byte array to store attribute values;
2. allocate memory for the designated byte array;
3. copy attribute values to the byte array;
4. return the byte array as the result.

In case there exist attributes that are references (pointers to other object instances), those referenced object instances must be duplicated into the byte array so that they are able to be re-created in another memory space.

In contrast to `toByteArray()` which is an object *instance* function (object method), `fromByteArray()` is a *static* function of a user-defined type class (class method) which creates and initializes an object instance with a given byte array. A typical `toByteArray()` routine may follow the steps:

1. create a new object instance of the user-defined data type class;
2. allocate memory for (referencing) attributes if necessary;
3. set attribute values with data in the byte array;
4. return the reference of the newly created user-defined data type object as the result.

Additionally, to allow data types being defined in different languages, Conquest provides a uniform interface for each language so that a UDT can be defined in one programming language, but still automatically can be represented in other languages by Conquest. For example, to define a type **SetOfPoints** (a set of points), a developer can specify the definition in C++ as shown in Figure 7(a). The two required functions together with other functions associated with type `SetOfPoints` are defined. We note that the `UDT.hh` file defines the C++ interfaces for the two mandatory functions, and hence they are inherited by each C++ user-defined type. To apply such a user-defined type in Conquest, an associated Java "wrapper" class has to be specified a Java class is due to the fact that Conquest is implemented with OrbixWeb and Java), which is shown in Figure 7(b). In the Java "wrapper" class, an attribute "mPeer" references the "actual" user-defined data type object (i.e., a C++ object) to *bridge* the data objects implemented in different programming languages. Access to a Conquest Java data type object will be transferred to the actual C++ object instance, hence performing the desired manipulation.

To ease developers programming burdens, the tool **UDTGen** provided with the Conquest system automatically generates all Java and JNI (Java Native Interface) code with the header file of C++. As a consequence, Conquest developers are relieved of as much of the Conquest coding as possible.

5 Related Work

In the last two decades, scientists have developed a large variety of specialized science tools that allow them to perform their science tasks either on workstations or on large supercomputers. These tools range from general-purpose

```

#ifndef SETOFPOINTS_HH
#define SETOFPOINTS_HH
#include "UDT.hh"

// user-defined type definitions -- SetOfPoints
class SetOfPoints : public UDT {
public:
// constructor
SetOfPoints( long year, long timeframe, long numpoints );

// two required member functions
static SetOfPoints *fromByteArray( char * );
char *toByteArray();

// other member functions
long getYear() const; // return the year
long getTimeFrame() const; // return the time frame
long getNumPoints() const; // return the number of points
...

private:
long mNumPoints; // number of data items
long mYear; // year
long mTimeFrame; // time frame
float *mX; // x-axis
float *mY; // y-axis
float *mValue; // value
};

// handle for SetOfPoints
typedef SetOfPoints* PSetOfPoints;
#endif

```

(a) SetOfPoints.hh

```

// User-defined type
import edu.ucla.ConquestV2.user.*;
public class SetOfPoints extends UDT
{
static { System.loadLibrary( "SetOfPoints" ); }
// pointer to C++ class
protected int mPeer = 0;
// constructor - with C++ object reference specification
public SetOfPoints( int peer ) {
mPeer = peer;
}
// A required method for C++ interface
public int getCPeer() {
return( mPeer );
}
// required native methods
public static native int nativeFromByteArray( byte[] data );
public native byte[] nativeToByteArray();
// two methods required by the execution environment
public static UDT fromByteArray( byte[] data ) {
return( new SetOfPoints( nativeFromByteArray( data ) ) );
}
public byte[] toByteArray() {
return( nativeToByteArray() );
}

// methods from C++ definition
public static native int createNew( int year, int timeframe, int numpoints );
public native long getYear(); // return the year
public native long getTimeFrame(); // return the time frame
public native long getNumPoints(); // return the number of points
...
}

```

(b) SetOfPoints.java

Fig. 7. Definition of UDT SetOfPoints

image visualization and animation tools that import common image formats, to scientific visualization tools with an extensible library of data processing and analysis functions that can import diverse archive file formats commonly uses to represent scientific data sets. On the other end of the scale, scientists use programs that have specifically been implemented to accomplish a single analysis or visualization. Due to performance reasons, scientists are often restricted to perform an analysis on a small subset of the original data set only. Conquest, on the other hand, provides the capabilities of tools in the later two categories, allows to combine different analysis tasks to a larger experiment, and supports the efficient execution on complete data sets.

Since long-running queries in database systems are common for non-traditional application domains such as geoscientific computing, a high degree of parallelism within a query execution (intra-query parallelism) is desired. Due to the nature of the application domain, user-defined data types have to be made available by the database system; therefore, an extensible query optimizer has to be supported by the query system to allow the inclusion of user-defined data types and operators in the optimization process. An early system to support an extensible optimizer was the Volcano system [Gr93, Gr94]. Conquest extends the

open-next-close stream processing model proposed in Volcano with more features such as dynamic re-optimization.

Related work has been done in the area of mapping a data model with geoscientific data types to differing geoscientific data types made available by heterogeneous data sources in the *geoPOM* (Geoscientific Persistent Object Manager) system [Ni96] as part of the *OASIS* project. *geoPOM* provides geoscientific data types and internally supports the optimized mapping to stores such as archive file format libraries and geoscientific DBMS, and generates code so that user-defined *geoPOM* types are mapped to spatial types of the local spatial data repositories. Often, semantic and syntactic discrepancies between geospatial data types exist, and have to be reconciled during the mapping process. In *geoPOM*, an approach is presented to ensure that all mappings of one geospatial data type of the *geoPOM* data model to the equivalent, but differing geospatial are almost identical, so that queries are answered with a similar accuracy. In *Conquest*, however, we take a unrestricted approach, and allow a user to define rules for mapping the output data type of one operator to the input data type of another operator.

Distributed applications are usually too complicated to be implemented without system support. Therefore, systems have been developed that provide an auxiliary interface definition language (IDL) to assist code development for clients and servers in distributed environments [Su95]. This language, e.g. the Interface Definition Language of OMG [Ob98], allows to define a server interface in a uniform language that can be mapped automatically to different programming languages for the subsequent implementation of the server code. This approach requires programmers be familiar with both the interface language and the host programming language. This requirement may be too severe for scientists. To remedy this disadvantage, [Par95] proposed the approach to generate system support stub code by processing already existing C++ header files, thus, allowing users to work with a single programming language, and generating the system support code automatically. Developing *Conquest*'s tools for user-defined data types and operators, we adopted a similar idea which is, however, more specific to generating Java object code and native interface code.

There are several approaches to build wrapper architectures for legacy data resources (e.g., [Pap95, To97, Ro97, Ni97]). Usually, a middleware is built to mediate between different data resources and query manager in these systems. However, to have such a middleware not only complicates the execution environment but also potentially violates the stream processing paradigm. Therefore, we “wrap” an existing data source in the form of an operator object function (data source scan operator), stream the data into the *Conquest* environment, and execute the query within its environment. During query optimization, functions to subset data that is retrieved from a data store could be ‘pushed’ to the data store for execution; this work however, is not supported in the actual *Conquest* environment.

6 Conclusions

High performance query processing in non-traditional database application domains such as geoscientific computing is desired due to vastly growing data sets and complex analysis algorithms. In this paper, we have presented the Conquest system — an extensible and distributed query processing system that supports both data and operator support for query evaluation. The contributions can be summarized as follows.

- We have designed and implemented a distributed query processing system that delivers high computation performance with advanced distributed system architecture, i.e. CORBA and IIOP protocol. Query operations and data access can be executed across heterogeneous platforms over the internet.
- Both pipelined and bushy parallelism is supported in Conquest. In addition, operators may be cloned for multiple instances each of which processes a portion of input data streams hence implementing data parallelism.
- Scientists can arbitrarily introduce their particular algorithms for specific application queries. With the provided operator development tool, operator implementors can focus on programming the core logic of complex algorithms instead of paying attention to interfacing to the execution environment.
- Well-known raster and vector geometry data types can be specified in the Conquest system so that scientists can use their familiar data structures to present individual phenomena and experiments. In addition, our schema translation approach allows heterogeneous data types working together for collaborative scientific experiments.
- The analysis computation against massive data sets with complex algorithms often result in long running queries. During the execution, system configuration and characteristics may change hence resulting in sub-optimal query evaluation. Conquest provides triggered run-time re-optimization as an option so that query plan may dynamically reconfigured to adapt to the changing environment and data characteristics.

There are several issues with respect to high performance query processing that require further investigation. First, we currently use mean-based methods to forecast the trend of system parameter changes and data characteristics during execution. More sophisticated algorithms are under study and development for more accurate estimation hence performing query evaluation optimally. In addition, we have designed and implemented a triggered run-time re-optimization technique for guaranteeing high performance by taking advantages of up-to-date system parameters and data characteristics. Nonetheless, the mechanisms are not enough to handle the situation when an unpredictable accident happens, e.g., network partition due to a broken physical cable. In such cases, query evaluation must be re-started from scratch. The cost to re-start evaluation could be expensive for long running queries, which are common in non-traditional database application domains. Therefore, how to apply the techniques we developed for query plan reconfiguration for providing a fault tolerance query processing environment is meaningful for those long running applications.

Acknowledgements The authors would like to give thanks to Dr. Eddie C. Shek who has started the Conquest project and given us insightful comments. Thanks are due to Zhenghao Wang, Siddhartha Mathur, Suhas Joshi, murali mani and all other members of Data Mining Laboratory in Computer Science Department at UCLA. Randy Blohm and Howard Lee at JPL helped to build experimental applications and gave us many useful feed backs.

The source code of Conquest and some examples can be downloaded from <http://dml.cs.ucla.edu/projects/oasis/Conquest/conquest.html>.

References

- [Br96] R.E. Brucoleri. WRAPGEN - A Tool for the Use of Fortran and C Together in Portable Programs. *ACM SIGPLAN*, vol.31, number 4, April 1996.
- [Co89] P.T. Cox, F.R. Giles, T. Pietrzykowski. Prograph, a step towards liberating programming from textual conditioning, *IEEE Workshop on Visual Languages*, pp. 150-6, Rome, October 1989.
- [Da95] Data Mining Laboratory. The Conquest Execution Server 1.0 Programming Manual. Department of Computer Science, UCLA, June 1995.
- [ESP98] ESP2Net Project. http://dml.cs.ucla.edu/projects/dml_esp/index.html. Data Mining Lab, Department of Computer Science, UCLA, April 1998.
- [Ga95] E. Gamma, R. Helm, R. Johnson and J. Vlissides. BRIDGE. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [Gr93] G. Graefe and W.J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search, *Proceedings of the Ninth International Conference on Data Engineering*, April 1993.
- [Gr94] G. Graefe, R.L. Cole, D.L. Davison, W.J. McKenna and R.H. Wolniewicz. Extensible Query Optimization and Parallel Execution in Volcano, *Query Processing for Advanced Database Systems*, 1994.
- [To97] Tork Roth, M. and Schwarz, P. =”Don’t Scrap It, Wrap it! A Wrapper Architecture for Legacy Data Sources, *Proc. of the 23rd Conference on Very Large Databases*, Athens, Greece, 1997.
- [Ha97] L.M. Haas, and D. Kossmann, and E.L. Wimmers, and J. Yang. Optimizing Queries across Diverse Data Sources, *Proc. of the 23rd Conference on Very Large Databases*, Athens, Greece, 1997.
- [Io97] IONA Technologies PLC. OrbixWeb Programmer’s Reference, November 1997.
- [Lo88] G.M. Lohman. Grammar-Like Functional Rules for Representing Query Optimization Alternatives, *Proc. of ACM SIGMOD*, June 1988.
- [Me96] E. Mesrobian, R.R. Muntz, E.C. Shek, S. Nittel, M. Kriguer, M. La Rouche and F. Fabbrocino. OASIS: An EOSDIS Science Computing Facility, *International Symposium on Optical Science, Engineering, and Instrumentation, Conference on Earth Observing System*, August 1996.
- [Ni96] S. Nittel, J. Yang and R.R. Muntz. Mapping a Common Geoscientific Object Model to Heterogeneous Spatial Data Repositories, *Proc. of the 4th ACM International Workshop on Advances in Geographic Information Systems*, Rockville, Maryland, November 1996.
- [Ni97] S. Nittel, R.R. Muntz and E. Mesrobian. geoPOM: A Heterogeneous Geoscientific Persistent Object System, *Proc. of the 9th International Conference on Scientific and Statistical Database Management*, Olympia, Washington, August 1997.

- [Ng98] K. Ng, Z. Wang, R. R. Muntz and E. Shek. On reconfiguring query execution plans in distributed object-relational DBMS, in Proceedings of *the International Conference on Parallel and Distributed Systems*, Tainan, Taiwan, December 1998.
- [Ng99a] K. Ng, Z. Wang, R. R. Muntz and S. Nittel. Dynamic Query Re-Optimization. *The 11th International Conference on Scientific and Statistical Database Management*, Cleveland, Ohio, July 1999.
- [Ng99b] K. Ng and R. R. Muntz. Parallelizing User-Defined Functions in Distributed Object-Relational DBMS. *International Database Engineering and Applications Symposium*, Montreal, Canada, August 1999.
- [Ob98] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, <http://www.omg.org/library/c2indx.html>, February 1998.
- [Pap95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina and J. Ullman. A Query Translation Schema for Rapid Implementation of Wrappers, *Proc. of the 4th International Conference on Deductive and Object-Oriented Databases, DOOD'95, Singapore*, December 1995.
- [Par95] G.D. Parrington. A Stub Generation System For C++, *Department of Computing Science, The University of Newcastle upon Tyne*, Technical Report number 510, 1995.
- [Pi95] E. Pitoura, O. Bukhres and A. Elmagarmid. Object Orientation in Multi-database Systems, *ACM Computing Survey*, vol.27, number 2, June 1995.
- [Ro97] M.T. Roth and P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. *Proc. of the 23rd VLDB Conference*, August 1997.
- [St95] P. Stolorz et al. Fast Spatio-Temporal Data Mining of Large Geophysical Datasets. *The First International Conference on Knowledge Discovery and Data Mining*, Montreal, Quebec, Canada, August 1995.
- [Sh96a] E.C. Shek, E. Mesrobian and R.R. Muntz. On Heterogeneous Distributed Geoscientific Query Processing. *The Sixth International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems*, New Orleans, Louisiana, February 1996.
- [Sh96b] E.C. Shek, R.R. Muntz, E. Mesrobian and K. Ng. Scalable Exploratory Data Mining of Distributed Geoscientific Data. *The Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, August 1996.
- [Se97a] P. Seshadri, M.Livny and R. Ramakrishnan. The Case for Enhanced Abstract Data Types. *Proceedings of VLDB conference*, 1997.
- [Se97b] P. Seshadri and M. Paskin. Predator: An OR-DBMS with Enhanced Data Types. *Proceedings of ACM SIGMOD*, 1997.
- [Su95] SunSoft. NEO Programming Interfaces Reference. *Sun Microsystems, Inc.*, 1995.
- [Wo97] R. Wolski. Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. *Proceedings of the 6th High-Performance Distributed Computing Conference*, August, 1997.