Practical Lessons in Supporting Large-Scale Computational Science

Ron Musick, Terence Critchlow
{rmusick, critchlow}@llnl.gov
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

1. Introduction

Business needs have driven the development of commercial database systems since their inception. As a result, there has been a strong focus on supporting many users, minimizing the potential corruption or loss of data, and maximizing performance metrics such as transactions-per-second and benchmark results [Gra93]. These goals have little to do with supporting business intelligence needs such as the decision support and data mining activities common in on-line analytic processing (OLAP) applications. As a result, business data are typically off-loaded to secondary systems before these activities occur. In addition, they have little to do with the needs of the scientific community, which typically revolve around a great deal of compute and I/O intensive analysis, often over large data with high dimensionality. For scientific data, in many cases the data was never collected in a DBMS in the first place, and so the analysis and visualization take place over specialized flat-file formats. This is a painful solution, because a DBMS has much to offer in the overall process of managing and exploring data.

Of late, industry and the research community have been pushing to develop DBMS-based systems that will break this mold, and provide the needed OLAP support. The recent activity in OLAP [GC97, OLA98], multi-dimensional databases [TD96], ORDBMS [SM96], and the TPC council's TPC-D [TPC98] benchmark all testify to the strength of this new direction. This is a promising change of focus. OLAP optimizations are much closer than on-line transaction processing (OLTP) to supporting the *interactive computational data analysis* (ICDA) activities that take place in scientific domains [MM97]. OLAP and ICDA do not, however, represent identical workloads. In fact, little is known about exactly how DBMS technology fails to meet ICDA needs. We explore this issue in some depth, describing an evaluation of DBMS technology for large, high-dimensional computational data (see [Mus99] for more detail). After extensive testing, we can report that the technology is much closer to being able to support ICDA than one might expect. Furthermore, there is a clear evolutionary path that should lead to full support once the technology matures.

The main function this report serves, in lieu of stable and well-known benchmarks for ICDA, is to provide a practical evaluation of the current state of DBMS technology. In Section 2 we describe the characteristics of ICDA data and workloads, while Section 3 explains the evaluation criteria. Section 4 contains the bulk of the evaluation results, focussing first on relational databases, then discussing the newer object-relational approaches that have appeared commercially in the past few years. Finally, we conclude with the future directions and research that may finally integrate ICDA and mainstream database management systems.

2. ICDA Data and Workloads

Mesh data is by far the most common abstraction for computational data in the scientific community. A mesh provides a way of breaking a surface or volume down into an interconnected grid of much smaller zones (Figure 1). Each zone stores a range of computed or collected variables. The hope is that if the zones are small enough, the micro-scale properties and interactions can be modeled with enough accuracy to provide precise predictions of macro-scale events. The main limitation on decreasing zone size (and thereby increasing the number of zones per mesh) is the additional storage and computational costs incurred by the additional zone variables. Current capabilities are at the scale of a few billion zones; a typical simulation will contain between tens of thousands and tens of millions of zones.

This work was performed under the auspices of the U.S. Department of Energy at LLNL under contract no. W-7405-Eng-48. UCRL-JC-135606 Rev. 1.

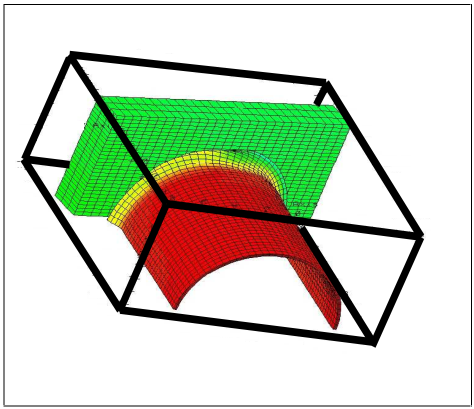

Figure 1. An Example of Mesh Data

Mesh data is typically stored in highly structured binary files, using standard low-level self-describing formats like NetCDF [RDE93] or HDF [HDF98]. The mesh is accessed through high-level APIs that provide methods to read and write individual components of the mesh directly, without reading the rest of the structure into memory. The zones in a mesh are made up of lists of points, or nodes. For example, in a 3-dimensional case, a cubic zone has 8 nodes that describe the corners of the cube. Nodes have X, Y, and Z coordinates, whose values can change if the mesh deforms over time. Variables can be assigned to zone centers, or to nodes. Variable values are recorded at each time step in a simulation, for each node and zone in the mesh.

The standard that a new DBMS-based implementation must compare favorably to is the existing legacy system in use on the scientist's desktop. For ICDA applications, this typically means comparing to native Unix (NU) fwrite and fread performance with optimal block sizes. While we would not expect DBMS-based performance on large array I/O to beat NU, a question we would like to answer is how much do we pay in performance to gain the additional query and data management capabilities? This depends of course on what typical ICDA workloads look like.

ICDA characteristics are distinct from OLAP and OLTP. ICDA applications involve *high dimensional, dense data,* so multi-dimensional database solutions are not effective. The data will be as *large* as the computational and storage platform supports, and so significant storage overhead for indexing or replication is not acceptable. The data is *write-once, read-many,* so update facilities do not need to be optimized. *Transactions* are not needed, and there are *few concurrent* users of the same data. There is a strong emphasis on *extreme data throughput rates* in order to support visualization-oriented operations on a mesh, such as iso-surface, or orthogonal slice extraction. ICDA workloads are very hard to pin down. Visualization-oriented queries lean heavily on range and multi-point queries over mesh variables. Scientists looking for anomalies, specific events or phenomena in the mesh, however, will generate workloads that rely more on point and extremal queries, thus benefiting more from query optimization. In either case, data throughput rates are the prime metric of concern.

3. Evaluation and Testing

Generating fair and repeatable comparisons between any two systems is hard due to the number of variables that impact the validity of a comparison. Unfortunately, there are no widely accepted benchmarks available for testing ICDA applications on DBMSs. This work does not define a new benchmark for ICDA. Instead, the goal has been to develop a set of tests that clarify the performance implications of supporting ICDA-style queries with a NU-based approach, compared with a DBMS-based approach. For the purposes of this study, only single node performance is compared.

Several issues arise when comparing two such open-ended approaches to data management. For example, the tests should be run on the same system, and under the same conditions with respect to system load. The test suite should also reflect a workload similar to the application that the system is meant to support. Some of the more interesting issues include:

- *Counting*: Throughput in terms of Mb/s is a key measure of interest for this evaluation. Exactly which bytes should be counted? For example, if the system reads 1Mb of useful information from a 3Mb interleaved array, but had to read the entire 3Mb to get it, was the total amount of data read 1Mb or 3Mb? For this work, the answer is based on the *apparent throughput to the application*, and therefore is 1Mb. Note that the use of apparent throughput leads to much lower rates, in terms of Mb/sec, than the maximum raw I/O throughput.
- *DBMS Implementation*: DBMS tuning plays an important part in determining performance. For each query and schema, several different embedded SQL queries using several combinations of indexes were tested. This resulted in several hundred small-scale performance tests. Tuning was performed on the system with the goal of understanding the full range of potential performance [Sha92].
- *NU Implementation:* For NU testing, the data was arranged on disk in binary files in much the same way it is for ICDA applications at LLNL. Choosing a reasonable NU implementation is a difficult task. If enough effort is expended, one can always construct a NU test that will outperform the

corresponding DBMS test. We need to compare not just performance, but the relative costs of achieving that performance. To help with this, three classes of NU tests were written for every SQL query: **poor**, **good**, and **very good**. **Poor** implementations were the simplest and fastest to build, and do no memory management. **Good** implementations make full use of system memory, while **very good** code is the most complex, it builds and makes use of indexes. Consider for example a point query on an array. **Poor** NU reads an object, checks the conditions, then reads the next object. **Good** NU reads several thousand pages of data into memory before testing conditions, while **very good** code will use the index to read only the data desired.

- *Cache*. Separating the performance impact of the cache and memory from the storage hierarchy is important. Isolating the cache allows one to predict what will happen with I/O performance as applications move from small-scale to large-scale, or toward tasks with low locality of reference.

Several relational schemata were created to represent the full complexity of the mesh data. This includes node and zone variables, lists of surface nodes and zones, mesh deformation information, material mixtures in zones, and internal file structures. The overwhelming majority of the data is found in the 2-4 dimensional tables that contain the variable data. Because ICDA workloads are focused on point and range queries over the variable data, the evaluation concentrates on this task as well.

The DBMS testing and timing was carried out in C using embedded SQL, since this is the most likely approach for building a DBMS into ICDA applications. For the evaluation, queries were constructed over databases that contained single tables with 2-4 columns of integers and floats. The raw data ranged in size from 500kb to 500Mb for point queries, and 150Mb to 500Mb for range queries. All point and range queries consisted of 1-4 conditions in the where clause. Point queries selected 1 integer or float. Range queries selected between 1% and 100% of the table. Note that none of these queries involve joins. Indexes were constructed and optimized for each query. Range queries were run both where the selected data was stored contiguously on disk due to the use of a clustered index, and where the selected data was scattered throughout the table. Thousands of individual point queries were made per test, so that the total test time took anywhere from 10 – 60 minutes. Range queries were large enough to run for the same time interval. Stored procedures were not used, since understanding *ad hoc* query performance is the goal of the testing. Slow tests were repeated 3 times, while faster tests were repeated 10 times. These tests were run using Sybase 10 and 11, and Informix IUS 9 (both as an RDBMS, and an ORDBMS) across several hardware platforms, and with hot and cold caches. Each DBMS test lead to 3 NU tests (**poor, good, very good)** on the same platform. Finally, there were no concurrent users, and no other significant loads on the machines.

We can not fully report on the results of the hundreds of small tests the testing plan lead to, nor would it serve an interesting purpose. Only a well-written (and broadly accepted) benchmark for ICDA could be used to provide a comprehensive picture of the performance capabilities of a particular DBMS. The performance numbers summarized below do, however, give a flavor of the types of results that were consistently found over the course of this evaluation.

4. Results

In this section, we break our analysis of the DBMS technology into three components. First, we describe the advantages the technology brings to ICDA, followed by the disadvantages, theoretical and practical, that arise when attempting to use it. Then, because the "traditional wisdom" about using DBMS technology for ICDA is often misleading, we attempt to dispel the more common myths associated with the technology. Finally, we conclude with a brief summary of the technology's applicability to ICDA.

4.1 RDBMS Systems

Advantages

1. *The RDBMS market is very strong and growing, and the major vendors are relatively stable. Interfaces to RDBMS data have been standardized for a long time, at several levels of abstraction, allowing customers to switch between products without dramatic modifications to legacy code.*
2. *RDBMS products are relatively mature, which has many advantages including a large pool of experienced labor, knowledgeable customer support from vendors, fairly efficient software and well understood query optimization technology.*
3. *There is a large base of third party tool vendors that offer a wide range of software for accessing and manipulating data stored in RDBMS. This tool base makes an RDBMS the foundation of a highly flexible, end-to-end data management capability, from design to storage and finally to access.*

Disadvantages

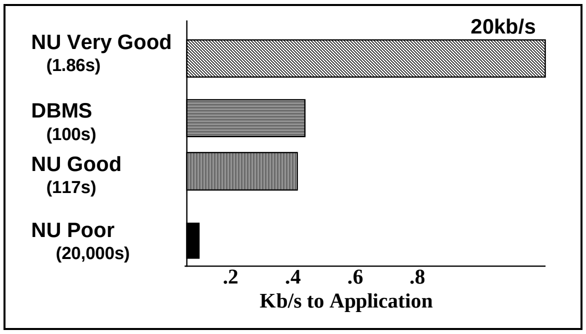

Figure 2. Point Queries

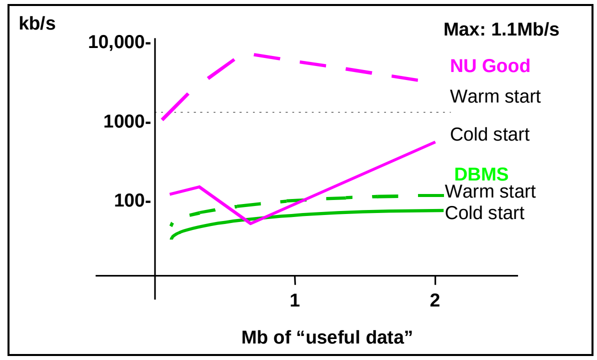

Figure 3. Range Queries

4. *A significant drawback of relational systems is the clumsiness of the data modeling paradigm. The model is capable of representing arbitrarily complex objects, but the result is often far from intuitive, or efficient.*
   The table-based type system in an RDBMS is very simple and straightforward. However, the extent of conceptual mismatch between this data model and ones found in ICDA application code means that the user will be required to maintain *two* data models – that of the application, and that of the DBMS. The translation code between the two can be non-trivial, easily reaching thousands of lines of code. For example, consider that, in the relational model, even the simple definition of a vector becomes tedious. Since each table is officially a *set* of tuples (rather than an ordered list), the only way to insure that the jth element of array X is returned is to have a table that stores the array index along with a second feature that is actually the value of X. This increases both the required storage space, and the complexity of accessing elements of X in a query. This becomes much worse in multi-dimensional cases. Another uncomfortable aspect is the strong typing that must occur in the schema. In the table above, the "value" must have a fixed type. A mesh, though, may have zone and node variables of mixed types. Efficiently representing this flexibility in a relational schema is challenging. On the positive side, once in a table, we can ask very interesting, *ad hoc* queries over the values of X.
5. *Performance of these systems is very poor for ICDA, independent of the vendor. In our performance studies, slowdowns of 5 to 40x were seen for the simple range and multi-point queries critical for ICDA.*
   The tests performed over the course of the evaluation clearly demonstrated that RDBMSs do not provide the throughput performance needed for ICDA applications.
   The data in Figure 2 and Figure 3 are from point and range queries, respectively. Point queries are the types of queries that one would expect a DBMS to consistently perform well on, mainly due to query optimization, and the relative ease of constructing and using indexes.
   As shown in Figure 2, for point queries over large data with a warm cache, the DBMS performed basically on-par with the **good** NU implementations. None of the NU implementations achieved more than 7% of the character read I/O capacity for the disk. The bulk of the tests showed results that utilized a much smaller fraction of the expected I/O bandwidth. This is not surprising, given that only *apparent* throughput to the application is being counted. The **poor** and **good** NU versions were I/O bound due to reading in data not useful to the application (and thus not counted). While the **very good** NU code shows extremely good relative performance, there is some overhead costs associated with reading the index. DBMS performance does not approach **very good** NU performance due to server communication, and other overhead costs mentioned below (#8).
   The range and multi-point queries (Figure 3) were much less consistent between vendors than the point queries. When compared to **good** NU implementations, total throughput rates varied from 5 to 40x slower, even with substantial tuning. These tests were carried out with data sizes that ranged up to 150Mb of "useful" data out of 500Mb total, and for the test shown in Figure 3, maximum apparent throughput to the application of 1.1Mb/s. The NU implementations reached maximum potential throughput within the first several Mb of data being read. The DBMS implementations flattened out much sooner, and at much lower throughputs.
   An interesting note is that for small data, while the NU implementations ran anywhere from 10-50 times faster with a warm cache, the DBMS implementations consistently gained only about 10%. For larger datasets, the cache had much smaller beneficial effects. This most likely reflects

the fact that the RDBMS is a compute-bound system.

Myths

6. *"Stable code."*
   Any complex piece of software, such as a commercial database system or a legacy application, optimized to perform well on a specific platform will never be stable as long as the underlying platforms are regularly upgraded.
7. *"Scientific data is too complex for RDBMSs."*
   Complexity of the underlying data is certainly an important issue, but alone is not sufficient to disregard this approach for data management. It is certainly possible, and in many cases profitable, to use an RDBMS for managing scientific data. The Human Genome Project has a wide range of data requirements, from acquisition, to high-level search and browse, to detailed analysis. The data is highly structured and interrelated. While no one, standard set of tools is used in this community, RDBMSs are seen more often than any other choice RDBMSs [CGM98].
8. *"RDBMSs are fast due to excellent query optimization benefits."*
   Depending on the performance metric chosen, this is just simply not true – the performance study demonstrates that DBMSs are overwhelmingly compute-bound. Query optimization speeds things up but does not get close to the throughput capabilities of the I/O subsystem. Stored procedures can ensure the query optimization overhead occurs only once. However, a large part of the code path *per tuple returned* involves locking, alignment, transaction management, tuple management, page management, and update, delete and insert facilities. Some relational systems require that each tuple returned to the client calls the server to invoke much of the code path listed above. Other systems allow one server call to fill large buffers and pass them back to the client. Independent of which RDBMS is used, however, the transaction semantics, consistency and correctness guarantees result in only a small fraction of the potential I/O bandwidth being used to respond to a query, even for large range and multi-point queries.

RDBMS Summary

The main difficulty with RDBMSs, of course, is their performance. RDBMS solutions require too much space, and are too slow, to support legacy ICDA applications. This problem is independent of which vendor is used; it falls out as a result of the relational model, and the requirements of OLTP customers. These approaches are not practical for ICDA currently, and they are not likely to be in the foreseeable future.

4.2 ORDBMS Technology

There are several small vendors of ORDBMSs, and three giants: IBM DB2, Oracle 8, and Informix Universal Server (IUS). DB2 is using its DataJoiner technology. Oracle is providing OR capabilities through cartridges, which are essentially component technology for DBMSs, and Informix is using a similar approach called data blades. For evaluation purposes, we worked with the IUS 9.12 because it was available, and the architecture has a few special features that make it extremely attractive for ICDA applications. The conclusions drawn below, however, apply to ORDBMS products in general.

Advantages

1. *ORDBMSs have been in the planning stages for the major DBMS vendors for several years now, and the advantages of merging object and relational approaches have been proclaimed by the research community for much longer. Nearly all of the potential for DBMS-oriented growth lies in this area, at levels that bode well for market share, vendor support, and third-party tools [SM96]*.
2. *ORDBMS combines, in theory, the modeling capabilities of ODBMSs with all the RDBMS advantages mentioned above.*
   The IUS allows the user to build user-defined types (UDTs), and provide methods for accessing them. The system provides hooks for presenting an interface to those new objects in terms of relational tables, and provides a mechanism to supply statistical information on the underlying data to the query optimizer. The data blade approach allows collections of UDTs, and their corresponding interfaces, to be packaged and provided as an add-on to the DBMS. One of the special features of the IUS is its underlying technology called the *Virtual Table Interface* (**VTI)**. The VTI is the mechanism used to define the interface of the UDTs to the database. The interesting aspect of the VTI is that it allows the data being described in the UDTs to sit outside the DBMS storage manager. This means that the legacy data does not need to be copied or modified. The VTI essentially wraps it in such a way that the ORDBMS can provide query access to it. Thus the query capabilities of an RDBMS are added to an ICDA application without impacting the storage costs or legacy access speeds.

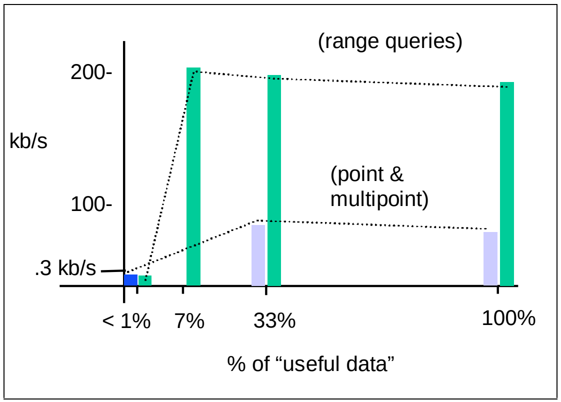

Figure 4. ORDBMS Example Query Speeds

3. *ORDBMSs have great potential for speed, and should be able to provide near-native Unix fread and fwrite performance levels. Currently, however, it is only within 3-15x of NU.*
   The IUS VTI requires building the access methods to the external data that provide a new storage manager for that data. This provides a great deal of flexibility, but can also be a lot of work. The benefit is that much of the RDBMS code path involving locking, page management, update delete and insert facilities, etc. will be avoided. Furthermore, if handled properly, the communication between the server and the client can be cut tremendously. For large requests, the throughput capability of this new approach should be capable of reaching near-NU performance. Performance testing was done comparing the IUS to relational systems, and to NU. This testing was performed on a simple binary dataset (a set of vector variables and a corresponding schema and interface). The IUS showed a 3-5x performance enhancement on the range and multi-point queries as compared to the RDBMSs. Unfortunately, it is still 3-15x slower than NU, depending on the query and the NU implementation being used. One limiting aspect of IUS performance is that each tuple returned still needs to be individually processed by the server. An example of the test results is shown in Figure 4.

Disadvantages

4. *There are no best-of-breed approaches yet for how best to integrate external, or "OO" data into the relational query interface. Since there is also no vendor agreement on this issue, any application that relies on one style of interface will be largely tied to a single vendor.*
   At the abstract level, the large ORDBMS vendors look very similar. At the level of how the OO–relational interface is actually specified, created, and used, there is little vendor agreement. The current design has several weak points. First, query optimization is difficult. The developer has the non-trivial task of collecting and providing the information the query optimizer needs to build the new OO data into the relational "fold". Second, it is no longer easy to do several things that are natural in an RDBMS environment. For example, there is no simple SQL command to create an index for the OO data. Third, wrapping the OO data and accessing it through the relational system stunts much of the potential flexibility of the system because the data must be presented in terms of relational tables and external functions. A better approach would be to construct functions that provide access to the object by taking arguments and returning a value. The downside of this is that any query that attempts to access the data in a way not preconceived and supported through the functions may require that new code be written. This is the same major drawback faced with the NU approach! In other words, because the data is not explicitly represented (as with relational tables), supporting true *ad hoc* queries over the data with simple and intuitive SQL statements may be very difficult to achieve.
5. *There is no delivery yet on the potential advantages mentioned above. This is a new approach, and the instantiations of it being offered by the major vendors are not ready for general consumption.*
   The ORDBMS has great potential, but at this point, much of it is unrealized: The OO modeling capabilities are only partially accounted for. The RDBMS advantages of market, external vendors, standards and maturity are currently not shared by the ORDBMS products. The ORDBMS can be made faster than the corresponding RDBMS for ICDA applications, and should eventually reach near-native Unix speeds. Right now however, this goal is still a distant one, for the reasons discussed above. While ORDBSs may be the wave of the future, and may enable a unification of ICDA, OLAP, and OLTP data management systems, this will not come to pass within the next few years. Furthermore, the current products are complicated and are introducing a completely new way to interact with, and manage data. The products are not mature, and the new OR mode of interacting with data is neither well nor widely understood.
6. *Tying external data into the relational system takes a great deal of effort on the part of the DBMS developer. To internalize an external object, several chunks of code must be written*
   To build an interface to data, one must:
   A. Develop the user-level interface, using the interface defined in B and C. For example,

modifying a visualization tool to support DBMS queries.

B. Describe relational tables that define the ORDBMS interface to the data. The difficulty of the design depends on the specific project, the implementation is fairly simple.

C. Define the new functions that complete the interface to the underlying data. Both the design and the implementation difficulty are driven by the functions being created.

D. Define the access methods used to replace DBMS storage functionality. For example, methods are needed to open and close tables, begin and end scans, get the next element, and etc. The definition includes function names, locations of the code, etc. Implementation is in extended SQL.

E. Write the code for the access methods. These methods can be arbitrarily complex, depending on how intelligent the memory management and disk management is, how complete the qualification system is, and how complex the external data is. The implementation is done using a set of libraries that replace the Unix-based file and memory management routines with Informix versions, since the IUS runs in a CPU extended virtual process.

F. Compile the access methods, and place them in the DBMS extension code.

G. Reconfigure the DBMS to create and handle the extent virtual processes properly.

H. Create tables in the ORDBMS according to the definitions above.

7. *The interface to the new capabilities is buggy, fragile, and limited. There were quite a few bugs and VTI design problems in this version of the IUS, some more serious than others.*

   A. The VTI use of a CPU extent virtual process causes several *serious* difficulties. First, standard RDBMS mirroring facilities do not work here. Second, while POSIX operations are supported, Unix file system operations are frowned upon, and *alloc is forbidden. When writing the access methods, all interactions with external data must be written using the Informix libraries. Thus a large base of legacy code is not available for use.

   B. Bugs in the code for the access methods can corrupt the database. During the course of the case study, the test database (and once, unfortunately, the group's research database) had to be rebuilt from scratch literally hundreds of times. The potential consequences of buggy user code are high for ICDA applications, and should give serious pause to users.

   C. Several annoying aspects mar the interface. For example, in the access methods, every value returned to the server needs to be strongly typed, then loaded into a "pdatum" structure and sent to the server. This is slow, due to the server communication requirements, and messy since we do not know the type of a particular mesh variable ahead of time.

   D. There were a host of distracting bugs, including a broken qualification evaluation routine, incorrect specifications for the various machine architectures (that lead to database-corrupting crashes), and faulty type evaluation code.

8. *The extent of new options that are opened to the users is bewildering.*
   For the IUS, data can be stored with relational tables consisting of a variety of standard and extended types. Each of these options has a unique set of capabilities and limitations attached to them, such as query flexibility, bulk loading capability, different access methods and indexing options. They have varying degrees of self-description, ease of use, and potential performance.

Myths

9. *"Saves time"*
   At this point, utilizing the OR aspect of the system to provide access to ICDA data is so difficult, that the time might be better spent amortized over smaller projects focused at extending legacy ICDA systems to allow *ad hoc* queries, indexes, etc.

10. *"OO-level modeling"*
    Full object-oriented models are not yet supported. Furthermore, in building the external data into the relational interface, much of the object-oriented aspect of that data is lost. SQL, the relational DBMS interface, is still the primary interface to the data, with the result being a somewhat less satisfying middle-ground than one could hope for.

11. *"More flexible"*
    In building the OO data into the relational system, several restrictions are placed on the object – views if you like. The views limit the flexibility that might otherwise exist in the object. Furthermore, the access methods created to support the external object will only be effective under certain rigid usage patterns.
    This myth has a more uncertain status than others. We have argued that the number of options open to the developer is so large, that it is nearly overwhelming. But each of these options carries its own set of limitations, and the end result is that the current OR integration approach introduces limits on the overall data model. Given a good design, the ORDBMS interface should provide

very flexible access to the underlying ICDA data. Unfortunately, the question of how to create a good *object-relational* design is largely unexplored territory. This is one of the most significant, and yet subtle dangers for the ORDBMS community. Powerful, but overly complex and hard-to-use technology is a niche commodity at best.

ORDBMS Summary

The potential for this technology is striking in terms of performance, flexibility, and ties to the business marketplace. This technology is only a few years old, and exhibits several issues, stemming primarily from the novelty of the approach, which remain to be worked out. The main danger for ORDBMSs is failure of the approach due to the complexity of using it.

5. Future Directions and Conclusions

One point became very clear during this evaluation: three prerequisites must be met simultaneously in ICDA applications. Unfortunately, every approach to data management seems able to support only two of the requirements concurrently. The prerequisites are: (1) small storage costs; (2) standards-based *ad hoc* query support; and (3) reasonable throughput (Mb/s) performance. Most combinations of two actually seem to be supported. For example, blobs do not consume much overhead storage, and have reasonable throughput performance, but do not provide good SQL access into the file. Alternatively, by keeping a copy of the data (thereby doubling or tripling storage requirements) in the original flat-file format outside the DBMS, legacy applications such as visualization will not see a performance degradation, and will get the *ad hoc* query capability on the data stored in the DBMS. Current legacy systems support low storage costs with great performance, but do not support *ad hoc* queries against the data.

To support ICDA applications, the best case scenario would be to have a large, stable vendor who provides an operational query engine on top of a data server with flat-file like performance. This (hypothetical) system would have features such as:

1. *Large, multi-dimensional array data stored* outside *the DBMS storage manager. This way, external applications could use the data without taking a performance hit, and without the large storage overheads.*
2. *Mechanisms to plug this data into the query engine, in a way that queries could combine data stored in the DBMS with data external to it. If the standard SQL interface were to be used, this might require being able to create relational views of the external data, and then populating the views on the fly when a query hits the external data. Population would allow passing large collections of data from the external access methods.*
3. *Avoidance of as much of the DBMS code path as possible. To protect the performance gains, have the hooks in #2 be inserted so as to avoid as many of the following as possible: locking, alignment, transaction management, tuple management, page management, insert and delete facilities. These features are much less interesting when there are few users on WORM data that is stored* outside *of the DBMS.*

ORDBMS vendors are beginning to close in on these targets, and may be viable options for supporting ICDA applications within 5-7 years. Technologies and trends worth watching include:

1. ORDBMS technology, as currently this is closest to fulfilling ICDA requirements.
2. The response of the ODBMS vendors to the new ORDBMS competition. They might be pushed into a new design cycle that leapfrogs current ORDBMS status.
3. The datacube and multi-dimensional database approaches to OLAP, data mining and other business intelligence applications. Datacube and MDDB approaches work on large multi-dimensional data that can be numeric or text. They may eventually provide much of the functionality needed for ICDA.
4. The progress of underlying scientific data models and formats to a common standard, such as the vector bundle model. A common underlying data model alone might be enough to be able to easily make use of commercial scientific visualization packages like EnSight, and others that share that model. While this won’t realize all the potential benefits of being able to use commercial business DBMS technology, it would be a good start.

Acknowledgments

This work has benefited tremendously through discussions with Mark Miller.

Bibliography

[CGM98] T. Critchlow, M. Ganesh, R. Musick, K. Fidelis, and T. Slezak. DataFoundry: Information Management for Scientific Data. Submitted to *IEEE Transaction on Information Technology in Biomedicine*, 1999.

[GC97] S. Goil, and A. Choudhary. High Performance OLAP and Data Mining on

Parallel Computers. *Data Mining and Knowledge Discovery Journal* , 1:4, 1997.

[Gra93] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, San Francisco, 1993.

[HDF98] HDF home page can be found at: http://hdf.ncsa.uiuc.edu/

[MM97] D. Malone and E. May. Critical Database Technologies for High Energy Physics. In *Proceedings of the 23rd International Conference on VLDB*, Athens, Greece, 1997.

[Mus99] R. Musick. Supporting Large-Scale Computational Science. Lawrence Livermore National Laboratory. Tech Report UCRL-ID-129903

[OLA98] OLAP Council Benchmarks found at http://www.olapcouncil.org/ research/bmarkly.htm

[RDE93] R. Rew, G. Davis and S. Emmerson. *NetCDF User's Guide, An Interface for Data Access, Version 2.3*. Unidata Program Center, Boulder, CO 1993.

[Sha92] D. Shasha. "Database Tuning, a Principled Approach". Prentice Hall, NJ, 1992.

[SM96] M. Stonebraker with D. Moore. "Object-Relational DBMSs: the Next Great Wave", Morgan Kaufmann, San Francisco, 1996.

[TD96] R. Tanler and K. Drost. Multidimensional Analysis of Warehoused Data. "Data Warehousing: Strategies, technologies and techniques", Rob Mattison. McGraw-Hill, New York, 1996.

[TPC98] Transaction Processing Council benchmarks found at http://www.tpc.org.