Scalable Distributed Data Structures for High-Performance Databases

Yakham Ndiaye, Aly Wane Diène, Witold Litwin[1] & Tore Risch[2]

Abstract

Present databases, whether on centralized or parallel DBMSs, do not deal well with scalability. We present an architecture for Wintel multicomputers termed AMOS-SDDS, coupling a high-performance main-memory DBMS AMOS-II and a manager of Scalable Distributed Data Structures SDDS-2000. SDDS-2000 provides the scalable data partitioning in distributed RAM, supporting parallel scans with function shipping. AMOS-SDDS potentially allows for scalable data volumes arbitrarily exceeding the storage capacities of a single AMOS-II. We validate our system by experiments with join queries. We show a 30% improvement of response time even for volumes of data that AMOS-II still could manage alone at its site. The results encourage the use of multicomputers for high-performance databases.

Key words: Multicomputer, scalability, distributed data structures, parallel processing

1 Introduction

Popular PCs and WSs connected through typical 10/100 Mb/s networks became a standard for organizations. The concept of *(network) multicomputers* emerges to designate such configurations. collection of PCs and workstations Research on multicomputers becomes popular, as they offer potentially storage and processing capabilities rivaling supercomputers at a fraction of the cost. One outcome are Scalable Distributed Data Structures (SDDSs), a new class of data structures specifically for multicomputers aimed at their new storage and processing capabilities, [LNS93]. The data of an SDDS are partitioned for storage over some server nodes of the multicomputer. One prefers the distributed main memory (DRAM), as a DRAM file may easily reach Gbytes while being accessible in a fraction of that on a disk. The SDDS file scales transparently for the application to potentially any number of sites, hence any size. It supports the parallel scans. The data access computations do not involve any centralized directory. Several SDDSs are know[3]. In particular, the LH* schemes provide the scalable distributed linear hash partitioning. Likewise, the RP* schemes provide the scalable distributed range partitioning, [LNS94], [K98].

At CERIA we design the experimental system for SDDSs on Wintel multicomputers that we termed SDDS-2000. It support several variants of both LH* and RP* schemes. Some are the high-availability schemes, able to tolerate server sites failures [LMRS00].

High-performance database management uses basically two technologies. One is the RAM database, e.g., of the well known Object-Relational DBMS AMOS-II [RJK99]. A RAM database offers best access performance, but is of very limited size and scalability, bound by a single node RAM storage capacity, usually a GB at most at present. AMOS II therefore is configurable as a distributed multi-base system where each node also may wrap external data sources [RJ99]. In the experiments reported here we use only the single-site AMOS-II RAM database configuration.

Another technology is the *parallel* database, typically on a share-nothing multicomputer or supercomputer (also often called now *switched* multicomputer). This technology allows for large sizes, but is at present disk based, hence the database access is typically much slower than a RAM database. Its scalability is also limited since every known distributed DBMS supports at present only a static partitioning and on a few dozens of sites at most. To add a

[1] CERIA Université Paris IX Dauphine, Place du Mal. de Lattre, 75775 Paris Cedex 16, France. {Yakham.Ndiaye, Ali-Wane.Diene, Witold.Litwin}@dauphine.fr

[2] Uppsala Universitet, Dept. of Information Science, Box 513, Uppsala, SE-751 20, Sweden Tore.Risch@dis.uu.se

storage site is a cumbersome administrative operation and is only possible for a database on network multicomputer. It typically requires to stop the use of the database, and to manually redistribute the data. See the manuals of DB2 for such DBA operations for the hashing partitioning and, e.g., of Non-Stop SQL DBMS for the range partitioning.

Scalability and high-performance are now major goals for a database. [GW99], [CW98], [CACM97], [G96], [GW96], [MC99]. To experiment with the merge of both technologies, we have coupled sigle-site AMOS-II and SDDS-2000 into the scalable distributed system we have termed AMOS-SDDS. SDDS-2000 serves as the scalable RAM data storage and access manager, allowing for database sizes impossible for a single site AMOS-II. AMOS-II manages at the applications node the database queries through its object-relational declarative language AMOSQL. Data distribution and scalability are transparent to the user. AMOS-SDDS is the 1st system of this type to the best of our knowledge.

Below, we present the architecture of AMOS-SDDS. We describe the processing of distributed queries and the coupling technology using the foreign (external) functions of AMOS-II. We show experimental performance measures of a benchmark join query under various conditions. The results prove the validity of our design. AMOS-SDDS is not only globally efficient, but can be even faster than AMOS-II for volumes of data that AMOS-II may still store if no join index is specified there.

Section 2 recalls principles of an SDDS. Section 3 presents AMOS-II. Section 4 introduces the AMOS-SDDS architecture. Section 5 presents the performance study. Section 6 concludes the paper and presents the future research directions.

2 Scalable Distributed Data Structures

An SDDS is a collection of records constituted each basically from a key and from some non-key data. The storage space at each server called *bucket* can contain many records. Buckets are numbered 0,1… The SDDS is created as bucket 0. Inserts that overflow the file trigger bucket splits. Each splits appends a new bucket that receives about half of records from the split bucket.

An applications searches, inserts and updates SDDS records through the SDDS *client* site. The client sends out the query using unicast or a multicast messages. To address the servers, each client has the *image* of the actual file. The image maps keys to buckets constituting the SDSS, typically using a hash function or an index. It may also contain the multicast address for the SDDS, shared by all its buckets. This address is especially used for queries involving non-key data. Such queries translate typically to parallel scans.

Initially, the image contains bucket 0 only. When splits occur an image may become inaccurate as splits are not posted synchronously to the clients. A client may send a key search or an insert etc. to an incorrect server. To avoid an addressing error, each server checks through its *checking* algorithm whether it is the correct one for the received key. If not, it forwards the query to another server determined though the *forwarding* algorithm. The correct server that finally receives the query, possibly after no more than a few hops. sends back to the client the *Image Adjustment Message* (IAM). The IAM allows the client to adjust its image so that at least the addressing error that triggered the IAM cannot occur again. The IAMs make the every image following the actual state of the file less or more adequately.

The client that sends out a parallel scan, through multicast or a series of unicast messages, ends it up through some *termination* protocol. A *probabilistic* termination occurs when no reply message comes after some reasonable time-out. It is tuned to the SDDS size, servers and network speed etc. to reasonably guarantee that all servers that should reply

[3] See (http://ceria.dauphine.fr/ceria.html) for the whole references

do it. A *deterministic* termination allows in contrast to test for sure whether all the servers that should reply did it. The deterministic termination algorithms are SDDS specific.

For AMOS-SDDS, we applied the RP* scheme for the scalable distributed range partitioning. Like in a B-tree, records in an RP* file are lexicographically ordered according to their keys. RP* supports efficiently the range queries. Each bucket has its *range* defined in its header by two values λ and Λ called the *minimal key* and the *maximum key*. A bucket may contain key c iff $\lambda < c \leq \Lambda$. Bucket 0 has the initial range of $(-\infty, +\infty)$. It splits into two buckets when the number of records to store exceeds the bucket *capacity* of $b >> 1$ records. Bucket 1 is then appended and receives the higher half of the bucket 0. This process iterates for every bucket that overflows when the file scales. At any split, if $\boldsymbol{c}$ denotes the corresponding median key, after the split the range $(\lambda,\Lambda]$ of split bucket decreases to $(\lambda, \boldsymbol{c}]$, while new bucket gets range $(\boldsymbol{c}, \Lambda]$. This process creates and maintains the range partitioning.

3 AMOS-II DBMS

Amos II is a distributed RAM multi-database system where one can choose between setting up wrappers for external data sources or storing data locally in the RAM databases. For this experiment the purpose was to use the RAM storage manager and OO query processor of Amos II to provide a fast SDDS-based RAM OODBMS. Thus here only the Amos II configuration of single site OO RAM databases was used [R00], [RJK99]. AMOS-II offers a declarative query language AMOSQL that can be embedded into C and Lisp. An external program can interface AMOS-II in two ways:

- AMOS-II can be through the *call-level interface* functions (*callin* interface); especially through the *a_execute* function dynamically executing an AMOSQL query.
- AMOS-II call external routines (functions) via the *callout* interface.

The callout capability realizes the AMOSQL object-relational capability usually called *external* or *foreign* functions. Those make it possible to extend the manipulation capabilities of AMOS-II in order to meet the specific needs of the users. In particular, data structures and their manipulation's routines can be developed in C or Lisp and fully integrated into AMOS-II.

4 Coupling SDDS 2000 and AMOS-II

We based the coupling on the callin and the callout interfaces, as in Figure 1, with more details in Figure 2. The calls-in uses the *a_execute* function that provides a query-based interface to the application at each client site. SDDS-2000 manages the RP* file as an external storage. A record basically corresponds to one tuple whose OID is the RP* record key, and other attributes (AMOSQL functions values) are non-key fields in some specific internal format. AMOS-II at the client calls SDDS-2000 through its callout interface using specifically designed so-called *ship* external functions. The RP* client ships the query to those RP* servers it should currently reach according to its image. At each server, AMOS-II is used as local query processor of the query shipped to the server. It is called-in by the RP* server, and later, internally, calls-out the server again for the records delivery, through AMOS-SDDS server specific foreign function calls. The records are filtered by the server AMOS-SDDS and those that satisfy the query predicate are returned to SDSS-2000. The latter sends them back to the client. Such a capability is often called *function shipping.* It is crucial for the efficiency of the scalable distributed/parallel processing, as it filters locally the records. Otherwise, the whole bucket be sent to the client, possibly overloading both the client and the network, especially in this scalable environment.

At the client site, the SDDS client assembles the records received from the servers and returns them to AMOS-II which performs *postprocessing* and finally returns the selected data to the application.

Summing up, in AMOS-SDDS, each SDDS-2000 server plays two roles:

1. It stores local data on the SDDS servers and offers access primitives locally to its server AMOS-II system through the callout interface.
2. It is an SDDS specific scalable distributed communication platform between Amos-SDDS clients and servers.

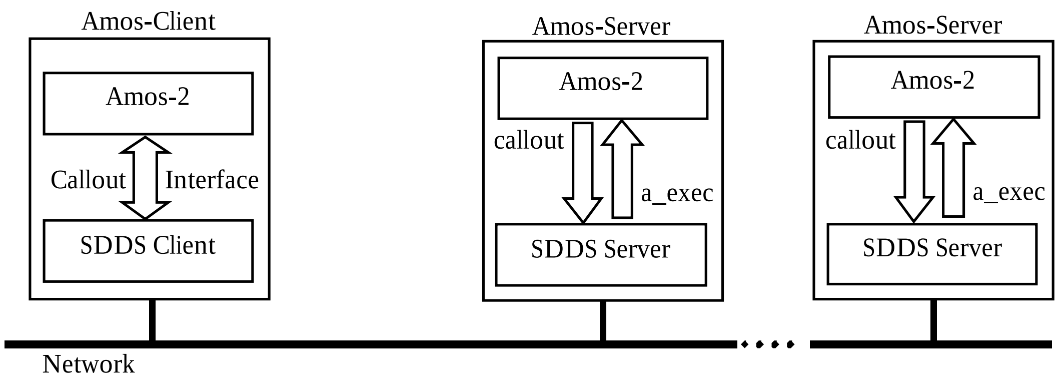

Figure 1 : AMOS-SDDS Architecture

Role (2) requires that the Amos-SDDS client decomposes the application query into sub-queries suitable for the parallel processing. While there are typically many ways to decompose a query in this way, the Amos-SDDS client handles any sub-queries basically as follows, Figure 2:

1. It calls-out the ship function.
2. The function passes the query to the SDDS client that multicasts it, using the UDP protocol, to SDDS servers.
3. Each SDDS server passes the embedded query to the local AMOS-II system through the call-in.
4. Each AMOS-II server system processes the query using local external functions to reach the data stored in the local SDDS bucket.
5. The result is put in an AMOS-II *scan* variable, which can contain several tuples.
6. Each SDDS server buffers the results contained in scan variable and sends the buffers to the SDDS client. For small buffers a unicast UDP message or a TCP connection is used. The latter is mandatory for buffers over 64K.
7. The SDDS client returns the result to the AMOS-II ship function. It performs some termination protocol to detect the end of the distributed processing.
8. The AMOS-II client receives the results from all the buckets. It eventually returns the final result to the application.

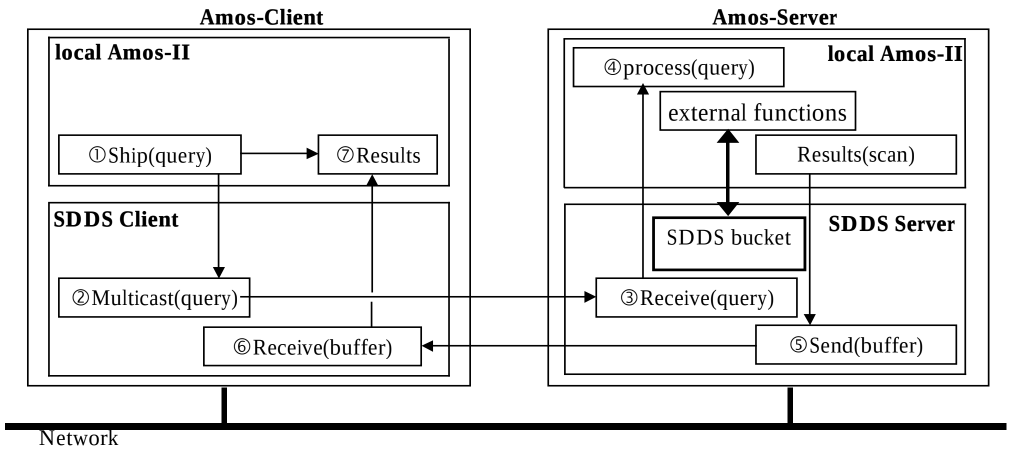

Figure 2 : AMOS-SDDS scalable parallel query processing

5 Experimental analysis

The implementation of the AMOS-SDDS required several design choices. One should decide for instance on the buffer sizes for the different communications between the client and the servers. If the server nodes have different CPU speeds, their bucket sizes may differ accordingly for the load balancing. Next, it may be advantageous for a scan query, to contain at once several (how many ?) values, e.g., of the distributed parallel join attribute. Furthermore, it may, or may not, be effective to send out these values sorted by the client. One trades here the halving of the server scan time against the additional sort time at the client. Also, one may need to decide whether the deterministic or probabilistic termination should be used. For the latter, one should optimize the timeout.

It did not appear that these and other design choices can be made on the theoretical basis only. Furthermore, it did not seem possible to determine in that way the overall efficiency of the system. Both AMOS-II and SDDS-2000 are highly complex software systems and their coupling obviously adds-on on that. In particular, there does not seem to be any easy theoretical way to measure the efficiency of the call-in and call-out coupling. Likewise, while the parallel processing speeds-up the file scans, the communication between client and the servers and the distributed processing itself introduces delays on their own. In brief, the experimental analysis of such systems is a must.

5.1 The hardware and the experiment data

The multicomputer at our disposal was constituted from seven Windows NT workstations on the 100Mbit/s Ethernet network. There were four HP-LN Pentium I 90 MHz workstations with 48 MB of RAM, two Pentium II 266MHz with 64 MB of RAM, and one Pentium II 350MHz with 128 MB of RAM workstation. Some of these sites were used as clients other as servers, depending on the experiment.

The experiments assumed the use of a large table **Person** (ssn, name, city). Tuples were inserted as records in the RP* file using **ssn** as the key. We have generated between 1,000 to 6000 tuples of average 25 bytes that were distributed successively on up to six servers.

5.2 Benchmark queries

To benchmark our system, we have chosen as the first measure, as usual, a computationally complex join query. The query searched for couples of persons living in the same city. As the distribution of the records according to the city is random, persons in the same city were likely to be stored at different buckets. The join selectivity, i.e., the ratio of the selected tuple from the Cartesian product, was about 1.6 %. On the one hand, we have formulated and run our query, termed for this purpose Query 1, towards AMOS-II alone with all the data within. We knew that the join is then evaluated using the nested loop strategy. We did not build any index in Amos II on the joined column (city). For AMOS-SDDS, we have experimented with various ship functions and design choices for the SDDS manager and the AMOSes, providing different execution strategies. All those obeyed nevertheless to the same well-known *distributed/parallel nested loop* strategy. In our case, termed Query 2, it worked specifically as follows:

- The outer loop:
    - The client AMOS requests all the tuples.
    - The join attribute (city) values are extracted from the tuples successively received, packed into buffers, and eventually sorted.
    - The client multicasts each packet prepared in this way to the servers.
- The inner loop:
    - For each packet received, each server performs the scan of its bucket, selects the matching tuples(persons living in the same city) and sends them back, in packets, to the client..
    - For each tuple received, the client post processes it with its matching tuples, to produce the couples.

The execution strategies of Query 2 experimented with different termination protocols, various packet, and buffer sizes, fixed versus variable bucket sizes for load balancing, sorted versus unsorted packets of the outer loop etc.

5.3 Performance measurements

Table 1 presents execution times of Query 1 according to the file size that varies from 1000 to 6000 records if no join index is use. AMOS-II (alone) runs on the HP-LN workstation.

Number of records	1000	2000	3000	4000	5000	6000
Execution time	**8s**	**33s**	**79s**	**145s**	**230s**	**337s**

Table 1. Execution times of Query 1

Table 2 shows the best execution times respectively for the same tuples, but for Query 2. The Amos-SDDS client runs on the same HP-LN workstation. The tuples are stored in RP* files distributed on six servers. Other details and more measurements are in [NY2000].

Number of records	1000	2000	3000	4000	5000	6000
Execution time	**8s**	**23s**	**50s**	**91s**	**150s**	**228s**

Table 2. Execution times of Query 2

These results correspond to load balancing through larger buckets on faster servers. The deterministic termination also proved better than the probabilistic one. This is a somehow surprising result that holds whatever was the reasonable time-out. The optimal outer loop packet size turned out to be 50. It appeared also advantageous to use sorted buckets. The replies were received using TCP/IP that proved more effective here than UDP.

5.4 Discussion

The results clearly prove the validity of AMOS-SDDS. For the smallest file of 1000 records, it performs as well as AMOS alone without indexing. This is quite remarkable by itself, as the distributed processing is potentially slower than the local RAM use for small files. For larger files, AMOS-SDDS scales clearly much better. For the 6000 tuples, the gain reaches 33%. It would clearly increase for even larger files, assuming they still enter the AMOS-II RAM. We recall nevertheless that the prime goal of AMOS-SDDS is not to improve over AMOS-II, but to allow for the database sizes prohibitive for the latter.

Only the relativity of times in Tables 1 and 2 matters really. The Pentium I 90 Mhz HP-LN machines in our possession were the top of the line HP workstations a few years ago, but are several times slower that a typical PC today. First results on 700 MHz Pentium III workstations we just got tend to show 8 times faster figures, predictably corresponding to the CPU speed ratio.

The results match the basic intuition. The execution time of Query 1, a nested loop we recall, is indeed O(n^2) for n records. (With a nested loop over a hash index on the join column the complexity would have been O(n)). Time for Query 2 for M servers and unsorted buffers of t keys sent out by the client is roughly $O(n * (n/M)) + O(n/t) + O(r)$), where r is the number of result tuples. This evaluation holds in our case since processing time of network calls and of scans is slow with respect to the network transfer speed. To sort the buffer halves on the average the scan time, but adds sort times. We sort the buffer in the elementary way, simply by the sequential insertion. This leads to the time of $O(n/2 * (n/M)) + O(n/t) + O(t) + O(n) + O(r)$). The 1st component is for both Query 2 execution schemes the dominant one. Decreasing it M or $2M$ times logically makes Query 2 faster and more scalable than Query 1.

Notice nevertheless the rather heavy cost of the coupling mechanics and of the network use. For $M = 6$ one could at 1st glance expect the execution time of Query 2 almost six times smaller than that of Query 1. Despite all the optimizing we are still quite far of it. In particular, the $O(r)$ component is not negligible in our experiences. Our 1.6 % selectivity is intentionally very small, but still leads to up to 500 K tuples selected from the servers to the client, for the 6000-record file.

6 Conclusion

The aim of our work was the coupling an SDDS with a high-performance DBMS, to improve the current technologies for high-performance database. AMOS-SDDS is to our knowledge 1st attempt towards the system for a scalable RAM database. On the one hand, its scalability abolishes the tight and cumbersome in practice storage limits of a single site RAM DBMS technology. On the other hands, its RAM based query processing and scalable data partitioning, transparent to the application, improves over the current technology of a parallel DBMS. The scalable and high performance systems, able to explore large volume of data, including external ones, become a need for the modern applications [GW99], [CW98], [CACM97], [G96], [GW96], [MC99].

The on-going work includes experiments with the AMOS-SDDS on a newer multicomputer constituted from several 700 Mhz Pentium III workstations. We are also studying implementation of other join strategies, especially those for LH* files studied in [ASS94], possibly faster than the distributed nested loop. A future ambitious objective is the scalable distributed query optimizer for the AMOS-SDDS client.

Our coupling technique is not limited to AMOS. Other object-relational DBMSs supporting foreign functions can be potentially coupled to SDDS-2000 as well. We are currently studying the DB2 user defined table functions for this purpose. The success may offer an attractive alternative to its static partitioning scheme.

References:

[ASS94] Amin, M., Schneider, D., and Singh, V., An Adaptive, Load Balancing Parallel Join Algorithm. 6th International Conference on Management of Data, Bangalore, India, December, 1994.

[CACM97] Comm. Of ACM. Special Issue on high-performance Computing.(Oct. 1997).

[CW98] Clement T. Yu, Weiyi Meng. Principles of Database Query Processing for Advanced Applications. .Morgan Kaufmann, 1998.

[G96] Gray, J. Super-Servers: Commodity Computer Clusters Pose a Software Challenge. Microsoft, 1996. http://www.research.microsoft.com/

[GW96] Grimshaw, A, Wulf, W. The Legion Vision of a Worldwide Virtual Computer. Comm. of ACM, Jan. 1997.

[GW99] Groff James R. & Weinberg Paul N.. SQL The complete Reference. Osborne/McGraw-Hill 1999

[K98] Knuth D. The Art of Computer Programming. 3rd Ed. Addison Wesley, 1998.

[LMRS00] Litwin, W. Menon, J., Risch, T., Schwarz Th. Design Issues For Scalable Availability LH* Schemes with Record Grouping. Distributed Data and Structures. Carleton Scientific, (publ.) 2000.

[LNS93] Litwin, W. Neimat, M-A., Schneider, D. LH* : Linear Hashing for Distributed Files. ACM-SIGMOD Intl. Conf. On Management of Data, 1993.

[LNS94] Litwin, W., Neimat, M-A., Schneider, D. RP* : A Family of Order-Preserving Scalable Distributed Data Structures. 20th Intl. Conf on Very Large Data Bases (VLDB), 1994.

[MC99] Practical Lessons in supporting Large-Scale Computational Science, Ron Musick, Terence Critchlow ACM Sigmod Record, December 1999.

[R00] T. Risch, Amos2 External Interfaces, (http://ww.dis.uu.se/~udbl/amos/doc/external.pdf), Dept.. of Information Science, Uppsala University, Feb. 2000

[RJ99] T. Risch, V. Josifovski,: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations, Presented at 25th Intl. Conf. On Very Large Databases, Edinburgh, Scotland, September 1999

[RJK99] T. Risch, V. Josifovski, T. Katchaounov: AMOS II Concepts, Dept.. of information Science, (http://ww.dis.uu.se/~udbl/amos/doc/amos_concepts.html)Uppsala University, Nov. 1999.

[SND98] M.T. Seck, S. Ndiaye, A.W. Diene. Implémentation d'une bibliothèque de primitives d'accès aux fichiers distribués et scalables RP*. CARI 1998

[NY2000] Couplage de Structures de Données Distribuées et Scalables avec un SGBD relationnel-objet, Rapport de recherche CERIA. 2000-4-15, Ndiaye Y, (Avril 2000).