Using PVM to Implement a Parallel Database System[1)]

G. Bozas, R. Lehn, A. Listl, M. Pawlowski, A. Reiser
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Germany
e-mail: {bozas | lehn | listl | pawlowsk | reiser}@informatik.tu-muenchen.de

Abstract

MIDAS (**M**un**I**ch Parallel **DA**tabase **S**ystem) is a parallel relational database system. It is well suited to serve as a platform for the exploration of various strategies in order to parallelize a relational database system. This short paper depicts how the abstract architecture of MIDAS looks like. A detailed description shows the implementation of a MIDAS prototype on a network of UNIX workstations. The core of the MIDAS protoype is implemented using PVM. But pure PVM does not fulfil all demands in order to implement a parallel database system. We demonstrate how our implementation combines the PVM programming model with standard UNIX mechanisms. We use UNIX shared memory to enhance the efficiency of our implementation and we use access privileges to UNIX files to secure the database against unauthorized access.

The MIDAS Architecture

MIDAS has a client/server architecture (see figure 1). The *MIDAS server* provides a high level interface (SQL) to retrieve and manipulate efficiently the contents of a relational database. The database itself is stored in the filesystem provided by the implementation platform. *MIDAS clients* are database applications. They are sequential programs issuing transactions to the MIDAS server. Each transaction on its part consists of retrieval/update queries. A MIDAS client can run on any computer having access to the MIDAS server.

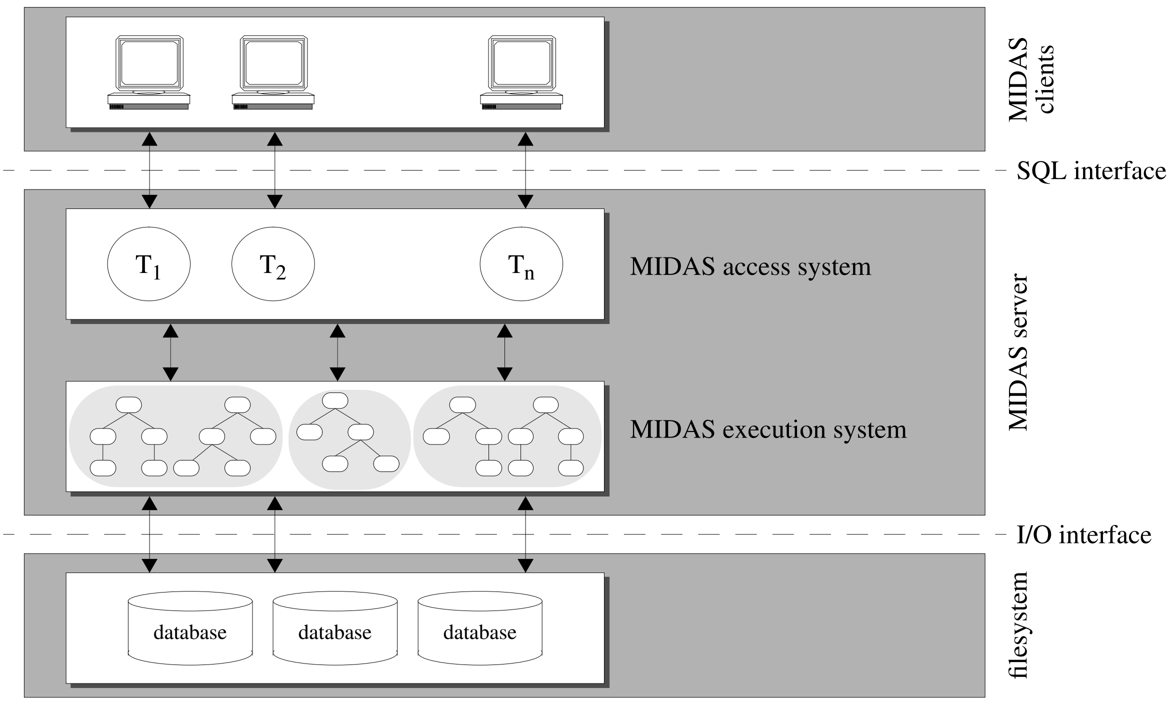

Figure 1: The MIDAS architecture

1) This research is directed by Prof. R. Bayer Ph.D. and is part of the more extensive project “Tools and Methods for Using Parallel Computer Architectures” executed at the dept. of computer science of the Technische Universität München since 1990 (sponsored by DFG contract SFB 342).

The MIDAS server is composed of two layers: The *MIDAS access system* and the *MIDAS execution system*.

- The MIDAS access system supports parallelism between different MIDAS clients. This kind of parallelism is usually called *inter-transaction parallelism*. For that purpose, the access system provides a mechanism so that an arbitrary and varying number of clients can issue their queries to the MIDAS server in parallel.
  The second task of the access system is to compile, optimize and parallelize the queries issued by the clients. The queries are purely descriptive (SQL). They do not contain any parallel constructs. The access system generates query execution plans (QuEP) which can be performed by the execution system in parallel in order to answer the queries of the clients.
- The *MIDAS execution system* is responsible for the parallel execution of queries in accordance with their query execution plans. This kind of parallelism is usually called *intra-query parallelism*. It is achieved by the capability of the execution server to work on different parts of one QuEP simultaneously. Interim results can be exchanged between the workers involved in one QuEP.
  Furthermore, the MIDAS execution system has to provide an efficient access to the physical database stored on non volatile memory like magnetic disks.

The MIDAS Prototype

Flexibility and scalability are the two main goals which must be accomplished by the implementation of the MIDAS server. PVM seems to be well suited because it provides much flexibility. The possibility to start dynamically new PVM tasks helps to implement a scalable system. Whenever a new MIDAS client arrives, the MIDAS server can increase its degree of parallelism by starting new PVM tasks within the MIDAS server. These new PVM tasks are dedicated to the additional work caused by new MIDAS clients.

All components of the MIDAS server are implemented as a set of PVM tasks. All MIDAS clients are conventional UNIX processes. It is not possible to implement the clients as PVM tasks, because we must protect the database against unauthorized access. The physical database is stored in files of the UNIX file system. Protection bits grant access rights based on user-IDs. The PVM tasks making up the MIDAS server must be allowed to access the database files directly, whereas the MIDAS clients must be prevented from direct access because of security reasons. Therefore the clients and the MIDAS server must have different user-IDs. This is not possible within one PVM program [1].

The detailed description of the MIDAS prototype follows figure 2. A set of UNIX workstations builds the set of hosts composing the virtual machine on top of which the MIDAS server is running. The MIDAS server consists of a *static* and a *dynamic* part. Its static part is made of a fixed number of PVM tasks which does not change during the lifetime of the MIDAS server. Its dynamic part grows and shrinks during the lifetime of the MIDAS server.

MIDAS access system: The implementation of the MIDAS access system consists of one *connection server* and a varying number of *application servers*. The numbers within the following description refer to figure 2.

The *connection server* is a PVM task belonging to the static part of MIDAS. It builds the entry point to the MIDAS server for the MIDAS clients. It has an UNIX socket globally known (①) to all clients. Whenever a new client wants to connect to the MIDAS server, it applies for an application server. This is done by sending a connection request to the globally known UNIX socket (②).
The job of the connection server is to listen to this globally known UNIX socket. If a connection request of a new MIDAS client comes in, the connection server starts a new application server

for this client (③). This application server is a PVM task belonging to the dynamic part of MIDAS. The connection server takes the actual load of the virtual machine into account in order to choose the host where the application server will be started. By that means, the connection server is able to distribute the load among the hosts of the virtual machine evenly. Once that the application server is started, it is exclusively at the MIDAS client's disposal. The client and its application server are fellows and belong together until the client disappears. Furthermore, a communication link (④) will be installed between the client and its application server. This communication link is a pair of UNIX sockets. In this manner, we achieve a scalable system. The number of application servers grows with the number of clients actually connected to the MIDAS server. No single component of MIDAS becomes a bottleneck.

The *application server* receives queries from its client. The queries are transformed into QuEPs by compilation, optimization and parallelization. The application server initiates and controls the parallel execution of these QuEPs by scheduling. For that purpose, it starts and removes dynamically so-called interpreters (⑤). They belong to the MIDAS execution system (see below). Finally, the application server sends back the results produced by the execution of the QuEPs.
The application server disappears as soon as its client disconnects from the MIDAS server.

MIDAS execution system: The MIDAS execution system is designed as a set of *cache/lock servers* and a set of *interpreters*. The cache/lock servers belong to the static part of MIDAS. Their number is constant. Exactly one cache/lock server is running on each host of the virtual machine. The interpreters belong to the dynamic part of MIDAS. Their actual number depends

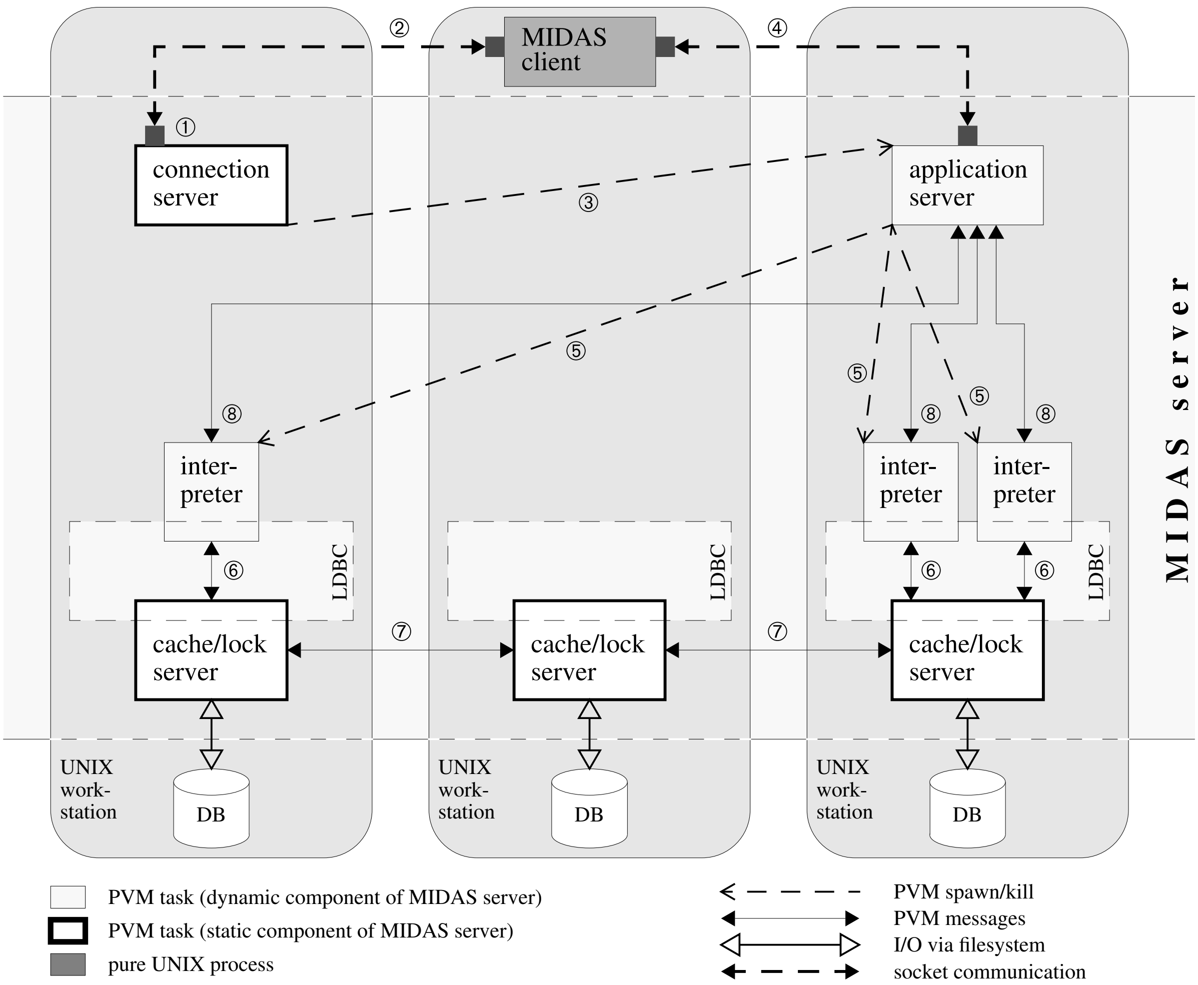

Figure 2: The MIDAS Prototype

on the number of QuEPs actually performed by the MIDAS execution system and the chosen degree of parallelism for each QuEP.

The *cache/lock server* is a PVM task. It provides an efficient and transparent mechanism so that all interpreters can access all database pages. For that purpose, caching techniques are used [2]. Each cache/lock server installs a UNIX shared memory on its host. This UNIX shared memory makes up the local database cache (LDBC). It is divided into frames. Each frame can hold one database page. If an interpreter requires an access to a database page during the execution of a QuEP, it sends a request (⑥) to the cache/lock server running on its host. The cache/lock server puts the required page into a frame of the local database cache. The requesting interpreter is informed (⑥) about the address of the frame holding the required page. Efficiency is the reason why we use UNIX shared memory in order to implement the LDBC. Pages can be accessed by several interpreters sequentially or even simultaneously without copying them from one address space to another. Our MIDAS prototype uses UNIX shared memory, because PVM does not provide a mechanism to share parts of the address space of different PVM tasks running on the same host. But at least, PVM is flexible enough to combine the UNIX shared memory mechanism with the message passing library of PVM.
The usage of an own LDBC on each host of the virtual machine causes a cache coherency problem. It is solved by using a variation of the invalidation approach combined with dynamic page ownership to guarantee weak cache coherency [2, 3]. Two phase locking is used in order to synchronize different transactions. The communication between the cache/lock servers which is necessary to realize cache coherency and concurrency control is done by PVM messages (⑦). In this case, a thread mechanism would be very helpful to avoid the implementation of a complicated scheduling strategy in order to prevent deadlock situations.

The *interpreters* are PVM tasks as well. They are started and removed by the application servers (⑤). Their job is to execute one QuEP in parallel. Each interpreter works on one part of a QuEP which is sent to it by the application server (⑧). The lifetime of an interpreter coincides with the time necessary to execute its part of the QuEP assigned to it. The access to database pages is performed locally by the mechanism described above. For that purpose, each interpreter must be attached to the UNIX shared memory building the LDBC.

Conclusion

PVM is suited to implement a parallel database system. We have chosen PVM because it is convenient to use and flexible to scale the database system dynamically. Furthermore, PVM provides a high degree of fault tolerance which is important to database systems, also.
One drawback of PVM is the absence of different access rights to the database stored in the UNIX filesystem within one PVM program. Another drawback is the lack of shared memory to implement an efficient database cache. Threads within PVM are desired to make the MIDAS implementation easier.

Bibliography

[1] A. Geist et al.: *PVM 3 User's Guide and Reference Manual*. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, May 1994

[2] A. Listl, M. Pawlowski: *Parallel Cache Management of a RDBMS*. Technical Report SFB 342/18/92, Technische Universität München, August 1992

[3] A. Listl: *Using Subpages for Coherency Control in Parallel Database Systems*. In Proceedings of *PARLE'94*, Athens, Greece, July 1994