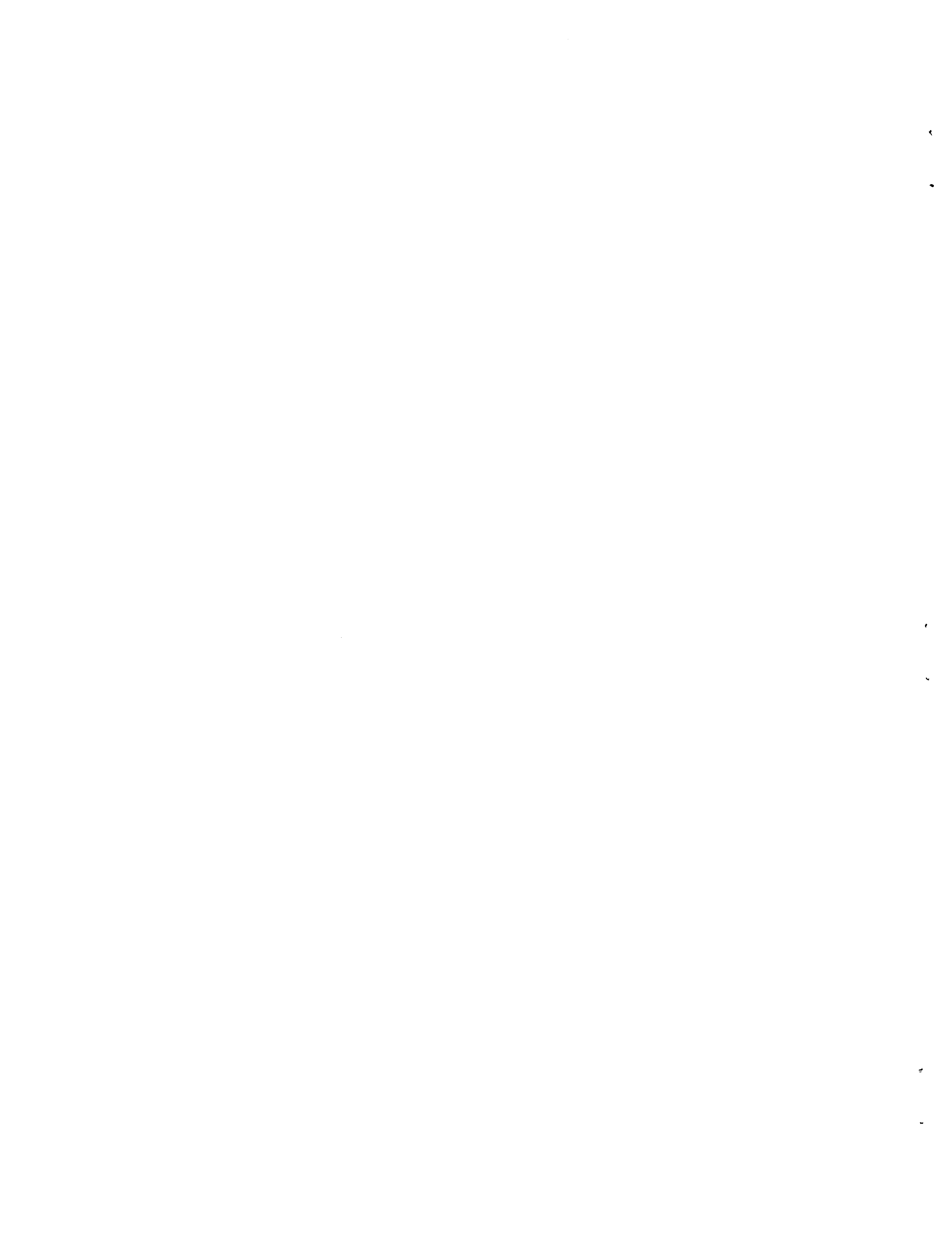
The Logical Protocol Unit: Towards an Optimal Communication Protocol

Erich Schikuta

CRPC-TR93360
November 1993

Center for Research on Parallel Computing
Rice University
P.O. Box 1892
Houston, TX 77251-1892



THE LOGICAL PROTOCOL UNIT: TOWARDS AN OPTIMAL COMMUNICATION PROTOCOL

Erich Schikuta[1]

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston , TX 77251-1892

Abstract

The performance of large software systems based on a server-slave architecture depends heavily on the design of the interprocess communication (IPC) protocol. In this paper different approaches of the logical structure of the protocol of the IPC communication are proposed and analyzed.

The term of a logical protocol unit (LPU) is defined. All basic actions of the software system are logically grouped into units and transferred jointly via the interprocess communication facilities of the underlying operating system.

An algorithm is given for the calculation of specific LPUs.

It is shown that the total performance of the whole system is affected dramatically by different logical protocol structures; increased by larger logical protocol units and decreased by smaller one.

This result is applicable to all types of IPC based systems, as beneath all large industrial applications. It was tested and analyzed with an existing relational data base system.

Processes and communication

Large software applications in the industry consist mainly of several processes, which run timesliced (virtually parallel) on a hardware system and communicate via the interprocess facilities [Roch85] of the underlying operating system. This design is demanded by the conventional server-slave architecture of centralized software systems. The slave processes are applications which represent the interface to users or other programs. In the following these programs are called ´user processes´.

The central server keeps up the connection with the user processes and is actually performing the work of the system (e.g. database access, system control, ...). Therefore a very time consuming part of the server task is the exchanging of information with the user processes. To partition a large program into smaller processes has the advantage of smaller program sizes (smaller main memory requirements), higher performance by workload sharing, logical structuring of tasks, etc..

It is obvious that the efficiency and the usage of the interprocess facilities is of heavy importance for the performance of the whole system. The efficiency is dependent on the operating system but

---

1 Authors permanent address: Erich Schikuta, Institute of Applied Computer Science, Dept. of Data Engineering, University of Vienna, Rathausstr. 19/4, A-1010, Vienna, Austria

the way of the usage of the IPC facilities is defined by the software system (and therefore tunable).

The logical protocol unit

For the following discussion of the problem the term of a logical protocol unit (LPU) is defined. This is a collection of information (operations and data) transferred jointly by the IPC facilities. It can be seen as an atomic transaction changing the state of the system from one synchronized situation to another. Synchronized means that the communicating processes can react to arising exception situations of the system sensibly.

Atomicity

Atomicity in our sense of protocol units is similar to the definition of an atomic action used in the area of transaction processing in database systems. It is a sequence of actions which either are performed in entirety or are not performed at all ([Ceri84]).

An action in the context of transaction processing is basically a read or write action of the database system ([Voss91]).

An action in our terminology can be any information transferred between processes.

Trigger and information actions

We distinguish between trigger and information actions. A trigger action forces the system to transfer the state of the system to a new state. An information action delivers the data necessary for the transition between states. Informally we can speak of operations and data.

An example is the insertion of an employee data record into a database system[2]. The system consists of a number of user processes, which provide the user interface, and a centralized database server. The insertion transaction consists of the information action, which is the transport of the employees information, as employee-number, name, department-number, etc., and the trigger action, which executes the insertion of the record.

The situation is different to the conventional database transaction, where normally more than one action (usually the number of actions stored in the transaction table) has to be rolled back to react to an exception. This 'reaction' is generally a recovery, which can be triggered by a user request or is automatically triggered by a special system state, like a system crash.

The length of a transaction atom is defined by the placement of a BOT (begin of transaction) and an EOT (end of transaction). This information is provided by the user program.

For an LPU this situation is not comprehensive enough, because with a protocol unit nothing is said about the actual length of the unit sequence, i.e. the number of actions. The maximal number of actions an LPU is restricted by the constraint that the system can react to any exception situation with a a single (and obviously final) recovery action. On the other hand the system has not to rollback a number of LPUs to reach a consistent state of the system again.

---

[2] Generally we use database system transactions as examples, which corresponds to the succeeding heuristic analysis.

Definition of an LPU

Hence follows the definition of an LPU (slightly different to the definition of a database transaction).

Definition:

A *Logical Protocol Unit* (LPU) is a (finite) sequence $a_1 \ldots a_n$ of basic actions which either are performed in entirety or are not performed at all and where the user program can react to an exception situation with a single recovery action.

With the preceding insertion example the whole transaction is an LPU. One possible exception can be a doubled employee-number. The server rejects the insertion and the user process can request a new number for the employee record. Another, not so obvious, exception can be the use of a wrong data type, a character input for the department-number (we suppose no user interface check). This could be checked during the evaluation of the data provided by the information action and the user could be informed. In the context of an LPU the last possible exception handling with a single recovery is the rejection of the insertion after the trigger action (the insertion request).

Computation of all LPUs

The computation of all possible LPUs can generate a rather large number of action sequences. Because of this we define an algorithm, which performs this task systematically.

First we determine all trigger actions $T = t_1 .. t_n$. Then we have to categorize all information actions $I = i_1 \ldots i_m$ into groups $G = g_1 \ldots g_l$, where $k \leq m$ and

$$\text{for all } i_i, i_i \in g_j$$
$$g_j \cap g_k = \emptyset, \text{ if } j \neq k,$$
$$g_j \neq \emptyset,$$
$$\cup g_j = I$$

These groups contain all actions, which cover similar information providing functions (in our example the insertion data providing group). Then we have to map the groups G to their possible trigger actions T. A group can be mapped to more than one trigger action. Finally the possible LPUs are described by all possible sequences of elements of groups mapped to the several trigger actions.

Formally the algorithm can be expressed by the following:

Algorithm

algorithm createLPU

determine all trigger actions $T = t_1 \ldots t_n$
group all information actions in groups $G = g_1 \ldots g_l$
for each $t \in T$
find all $g \in G$ which belong to t -> G_t
an LPU is defined by the sequence [i]t with $i \in G_t$[3]

Example

Lets consider the (rather artificial) insertion example. T is defined by the insertion action IA. The record data is transferred via an attribute action AA. Therefore only one group exists. Obviously AA is mapped to IA. The LPU is therefore calculated by all possible sequences of attribute actions and a final insertion trigger. This is the same result we determined informally above.

With the following heuristic analysis a comprehensive example is presented.

The physical protocol unit (with respect to LPUs)

With the realization of the protocol and the knowledge about the LPU the size of the message unit (the unit which sent via the IPC facilities) is of importance. A conventional approach would send each action by one message unit.

The concept of an LPU can be used for developing more efficient method. The system can react to an exception within an LPU in similar way as to the exception within a normal action. So we can expand the normal message unit to hold a group of actions. We call a physically transferred message unit with respect to the LPUs a *physical protocol unit* (PPU).

Definition:

A physical protocol unit (PPU) is the unit of information, which is transferred between communicating processes via the IPC facilities of the underlying operating system.

With the design of a suitable (and hopefully optimal) communication protocol we have to determine the optimal length of a PPU.

Optimal in our context is defined by the highest practical throughput of the system. Therefore we state the following theorem.

Theorem:

The optimal PPU is defined by the PPU which can accommodate the longest LPU.

This theorem is informal supported by the common rule of thumb, which can be found in the literature and which say: "few larger units outperform many small units" [Roch85].

---

[3] We use the commonly known grammar notation. Actions contained in square brackets '[' and ']' must be applied at least once.

We will prove this theorem heuristically. We implemented protocols with different PPU length for an existing database system (the Grid File database system, GFDBS) and measured the overall throughput of the system.

Heuristic analysis

The GFDBS is a specialized relational data base system which was developed during the last few year at our research institute ([Schi90]). The internal structure of the GFDBS is the Grid File ([Niev84]), a dynamic multi-dimensional data structure, which is extremely suitable to store complex objects and to support range queries. Because of the properties of the Grid File we are now developing a parallel version of the GFDBS, the PARGRID data base system ([Schi91]).

The GFDBS consists of a data base server process and several application processes (SQL-, QBE-, user-processes) communicating by message queues. All user programs are communicating with the server process via the IPC facilities by a fixed set of data base server commands (creation, deletion of a file, input, output, update, query, etc.). All data records (input and output) are transferred via the IPC-facilities, too.

Overview of the internal GFDBS commands

The following commands are available to the user programs:

Communication with the server:

GFDBSInit: starting the communication with the server

GFDBSQuit: stopping the communication with the server

Creation and Deletion of tables

GFInitCreate: start of creation of a new data base or a new table

ADefShort/Long/Float/String: definition of an attribute

GFCreate: creation of a new table

GFDelete: deletion of a table

Manipulation of tables

GFOpen/GFClose: opening or closing of existing tables

TInit: initialization of a new tuple

APutShort/Long/Float/String: initialization of attributes with a value

TInsert/TUpdate/TDelete: insertion, update and deletion of a tuple

TQuery/TNext: query of a table

AGetShort/Long/Float/String: retrieving attribute values

Computation of the LPUs for the GFDBS

During the normal work with the GFDBS server a sequence of the commands builds a logical unit of execution for the server. Normally each command for itself is sent to the server, but now we

can determine the LPUs of the server commands and can send them via the PPUs of the protocol. In the following examples of typical LPUs (the creation of a table, query of a table) are described.

In the following we concentrate on the creation and the query of a GFDBS table only. Further database transactions can be handled in a similar manner.

According to the given algorithm 'createLPU' we first determine all trigger actions. We find

GFInitCreate	(exception: table exists)
GFCreate	(exception: attribute exists)
TInit	(exception: table unknown)
TQuery	(exception: attribute unknown)

The possible exceptions to the trigger actions are given in brackets. These exceptions would force the server process to terminate the activity and recover, and the user process to react accordingly.

Creation of a table

The creation of a table consists of 3 actions: start, attribute definition and creation.

GFInitCreate	(start of the creation process)
ADef*	(a sequence of attribute definitions)
GFCreate	(creation of the table)

The 'ADef*' actions are grouped together to an attribute information group. The mapping of the groups (in this case only one) reveals 2 LPU sequences.

- GFInitCreate is sent as an LPU to the server. This is in accordance with the definition of an LPU, because the user process has to have the possibility to react to an exception, like the situation that the table already exists.
- The ADef* commands represent the information actions. The GFCreate command represent the trigger action. The user program can react to an exception of the ADef*, like the erroneous specification of equal attribute names.

The creation of a table is therefore split into 2 LPUs.

Query of a table

The query of a table consists of 3 actions either: start, definition of the query restrictions, execution.

TInit	(start of the query definition)
APut*	(definition of the restrictions)
TQuery	(start of the query)

This case is handled in a similar way as the table creation.

- TInit is one LPU. A possible exception could be that the Grid File is unknown.

- The APut* commands represent the information actions again. TQuery is the trigger actions, which concludes the LPU. In the case of an exception (attribute not available, etc.) the user program is informed and can react with only one action.

The query transaction is split into 2 LPUs.

The same calculations in similar manner have to be done for the insertion, update, deletion and retrieving transactions of tuples.

Performance Evaluation of different PPU lengths

For the evaluation of different PPU lengths a standardized performance test suite was created (Watz90]). A mix of frequently used data base transactions was built and the transaction times recorded. The following data base transactions were checked:

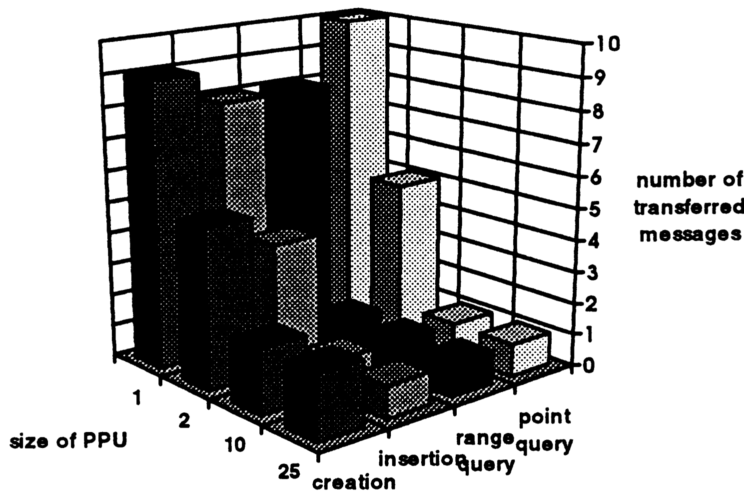
- creation of a table
- insertion of tuples
- range-queries
- point-queries

The size of the PPU sequence (number of blocked commands) was varied. The following charts show the performance of the system for different PPU sizes (block sizes). The performance behavior is shown by two characteristics:

- the number of sent message blocks
- the consumed time

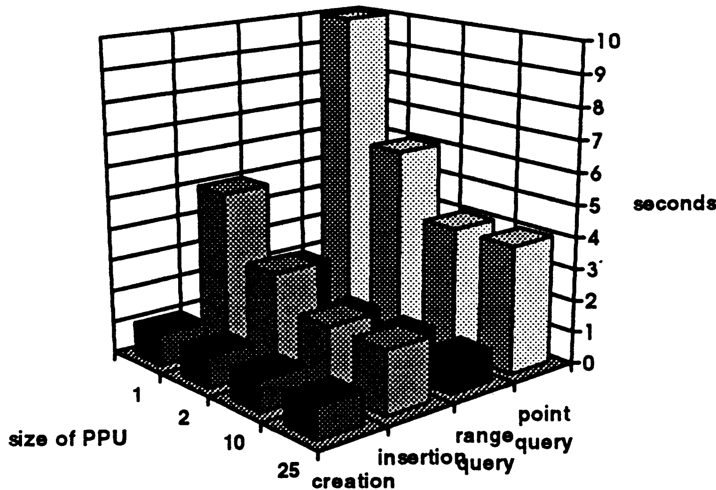
Basis of measure was the size of the largest basic action. This is represented in the figures by the PPU size of 1. The idea is that in the conventional case at least 1 basic action can be transferred with one message unit. So all values which are smaller than the value for size 1 (the conventional situation) represent an improvement achieved by the LPU concept.

All test runs were performed for 50 Grid Files with 2 to 10 attributes. In the following only the mean values for 7 attribute Grid Files are shown. This represents the average file size in our application profile. With increasing number of attributes the results show that the size of the LPUs obtain obviously stronger influence on the performance of the system.

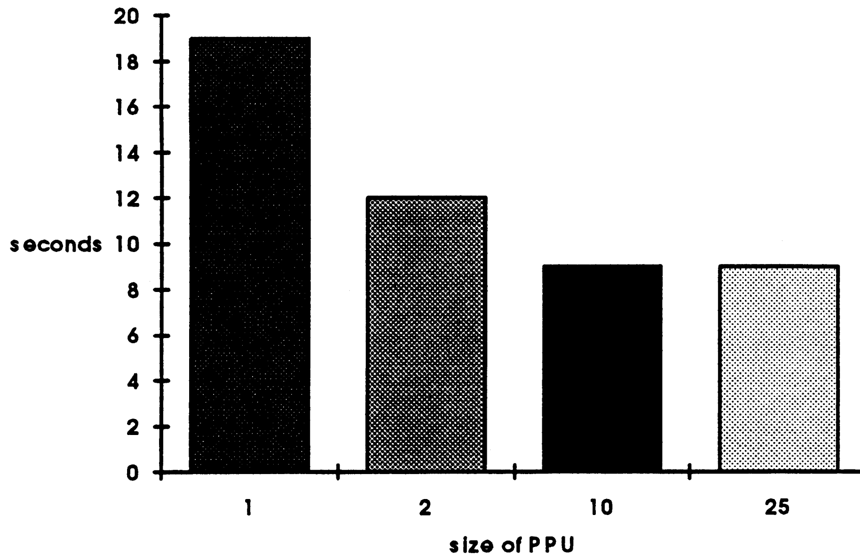


When the size of the PPU is big enough to hold an LPU completely, no more performance gain can be achieved by an increase of the PPU size (see range and point queries for PPU size 10 and 25). Therefore we can conclude that we have always to find the largest possible LPU (the LPU with the longest action sequence) in our system and implement PPUs, which can hold this LPU.

The next figure gives us the amount of time consumed for the same test suit.



The last figure shows the overall sum (creation + insertion + range query + point query) of the consumed time, which indicates that the throughput of the whole system was doubled by the introduction of LPUs.



This figure shows apparently the improvement of the communication throughput of the LPU concept (time for PPU size 10) to the conventional method of basic action sending (time for PPU size 1).

It confirms the above stated property too. Beyond a given PPU size no further performance increase can be achieved (time for PPU size 25). In contrary, it can lead to a decrease because of the systems overhead for very large message blocks.

Results

The results of the tests can be summarized shortly:

- LPUs increase the performance of the system dramatically (with the data base system the performance was doubled).
- PPU should be large enough to hold an LPU completely.
- To large PPU give no further performance improvement, in contrary they can decrease the systems throughput.

With realization of LPUs for the IPC protocol of message passing systems, longest possible sequences for LPUs have to be established. The optimal size of an PPU can only be decided by an exhaustive heuristic analysis. The LPU concept provides the appropriate tool for the aim to increase the performance of an IPC based system.

Acknowledgment

Many researchers and students made contributions to the GFDBS-project. On this occasion I want to thank Harald Watzer for the implementation and evaluation of the new LPU-message passing protocol. My special thanks goes to the CRPC for providing a stimulating and supportive atmosphere that contributed to this work.

This research was in part supported by the grant J0742-PHY of the Austrian FWF.

References

[Ceri84]
Ceri S., Pelagatti G., *Distributed databases, Principles & Systems*, McGraw-Hill 1984

[Niev84]
J. Nievergelt, H. Hinterberger, K.C. Sevcik, *The Grid File: a data structure for relational data base systems*, ACM Trans. Database Systems, 9, 1, 38-71, (1984)

[Roch85]
Rochkind M., *Advanced UNIX Programming*, Prentice-Hall 1985

[Schi90]
Schikuta E., *The Grid File Data Base System*, Proc. 10th SCCC International Conference on Comp. Science, Santiago de Chile, Chile, 1990

[Schi91]
Schikuta E., *A Grid File Based Highly Parallel Relational Data Base System*, Proc. 4th ISMM Int. Conference Parallel and Distributed Computing and System, Washington, D.C., USA, 1991

[Vossen91]
Vossen G., *DataModels, Database Languages and Database Management Systems*, Addison Wesley, 1991

[Watz90]
Watzer H., *Ein Hochleistungs-Message Passing System für das GFDBS*, master thesis, Univ. of Vienna, 1990, in German