Using PRIMA-DBMS as a Testbed for Parallel Complex-Object Processing

C. Hübel, B. Mitschang, M. Gesmann, A. Grasnickel, W. Käfer, H. Schöning, T. Härder
Sonderforschungsbereich 124, University of Kaiserslautern, Germany
{huebel,mitsch,gesmann,grasnick,kaefer,schoenin,haerder}@informatik.uni-kl.de

Abstract

The PRIMA-DBMS approach is explained by introducing PRIMA's architecture and query processing framework. The PRIMA-DBMS constitutes a testbed that is flexible enough to support evaluation and validation of quite a variation of different strategies for complex-object processing taking into account different parallelization levels and different hardware environments. Thus, PRIMA marks an important step towards our main research goal concerning measures for efficient complex-object processing: the measures that are in competition with each other are query optimization, query evaluation strategies, and massive storage, that all benefit from parallelism.

The programming environment that supports the parallel DBMS processing is introduced with special emphasis on its ability for parametrization and configuration. A case study of the PRIMA testbed illustrates our first investigations and demonstrates a methodology for evaluation and tuning of PRIMA configurations.

1. Motivation

Complex applications, such as design applications, AI applications, and even enhanced business applications provoked a lot of research in and evaluation of enhanced data models and query languages. The reason for this is that those applications can benefit significantly from DBMS that efficiently support complex objects.

Basically, there are three orthogonal measures in order to meet the necessary performance requirements. Firstly, query processing can be tremendously influenced by query optimization. Secondly, efficient evaluation of the optimized query plans depends on a high performance query evaluation system, and thirdly, exploitation of massive storage (e.g. cached or materialized complex objects, access structures) complements the previous two. Nowadays and triggered by evolution in hardware (especially the advent of shared-memory multiprocessor machines and high bandwidth networks) parallelism in query processing becomes very attractive as a means towards more efficient realization of these performance measures to query processing. Especially for complex-object processing intra-query parallelism becomes interesting, in addition to the well-known inter-query parallelism.

In order to investigate these performance-crucial measures, we have developed a DBMS testbed that is flexible enough to support evaluation and validation of strategies for complex-object processing considering different parallelization levels and different hardware environments. An overview of the basic concepts underlying our DBMS-testbed approach and a brief outlook on some case studies are given in the following chapters.

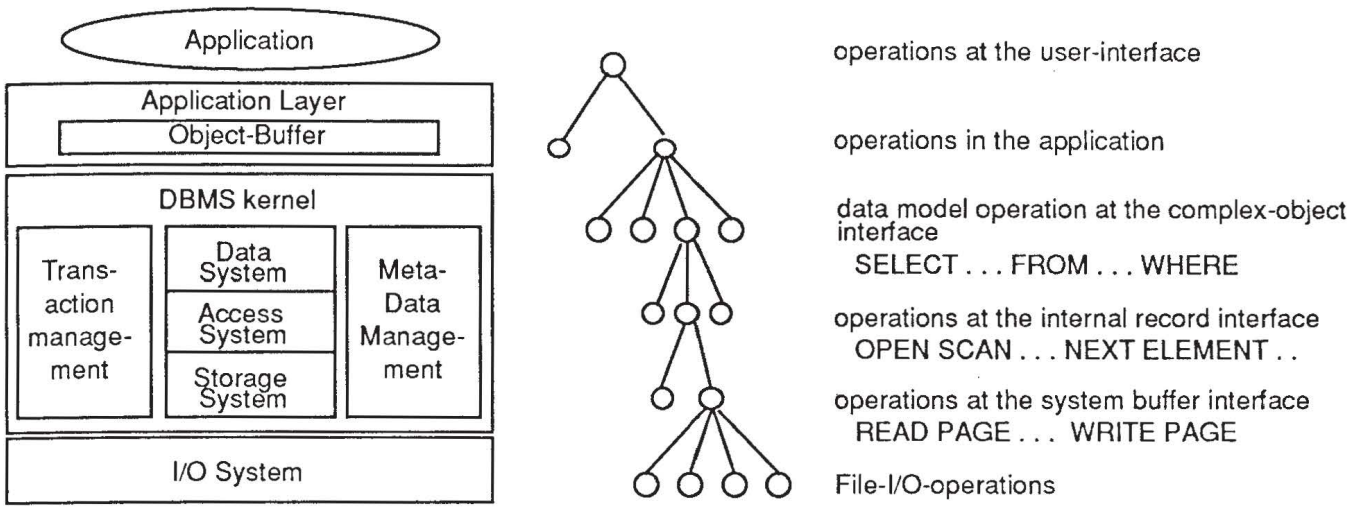
2. Client/server-based architecture for database processing

DBMS modularization as well as its embedding into an adequate runtime environment are necessary prerequisites for flexible and parallel database processing. In this chapter we focus on our DBMS architecture and query processing concepts, whereas the subsequent chapter deals with the environment issues.

2.1 PRIMA architecture

The architectural framework of the PRIMA system [HMMS87], shown in Fig. 1, is composed of a DBMS kernel with *neutral*, i.e., application independent, data management functions and an application layer providing application-specific support. The data retrieved from the kernel is embedded into the application layer by means of an object-buffer. The object-buffer manager supports fast pointer-like access to the objects loaded. The kernel is refined by a modular architecture being a prerequisite for flexibility and extensibility of the components and the kernel itself. The kernel provides at its interface a complex-object data model and language [Mi89]. The implementation model behind distinguishes three different layers (i.e. data, access, and storage system) for mapping the complex objects (here called mol-

0-8186-2660-7/92 $3.00



a) Modularized and layered architecture

b) Decomposition of operations

Fig. 1: PRIMA modularization and operation-decomposition

ecules) onto elementary objects (called atoms), which in turn are mapped onto blocks stored on external devices. For more detailed information on the PRIMA kernel the interested reader is referred to [HMMS87]. As illustrated in Fig.1, operations at one layer are decomposed into operations of the next underlying layer.

Based on the system modularization and on the operation decomposition we have developed a client/server model that defines the framework for database processing in PRIMA: in order to do its task, a client can issue requests for services to be performed by respective servers. Fig. 2 illustrates a simplified view to the server components and their client/server relationships. The data system consists of several servers (not all are shown in Fig. 2). The *Compiler* server is responsible for query compilation and generation of query evaluation plans that are executable by the *DML-execution* server. Both issue requests to the *Meta-Data* server and to themselves (e.g. in case of nested query expressions). In order to perform query evaluation plans, the *DML-execution* server uses *Access System* services, which in turn rely on *Storage System* services. All servers use *Transaction Management* services. The realization of the PRIMA client/server model is based on the Remote Cooperation System (short: RCS). It provides the fundamental functionality that is needed for the implementation of the client/server model in such a way that both the underlying operating system as well as the actual hardware are transparent at its interface. More detailed information on RCS and on the realization of the client/server model are given in the next chapter.

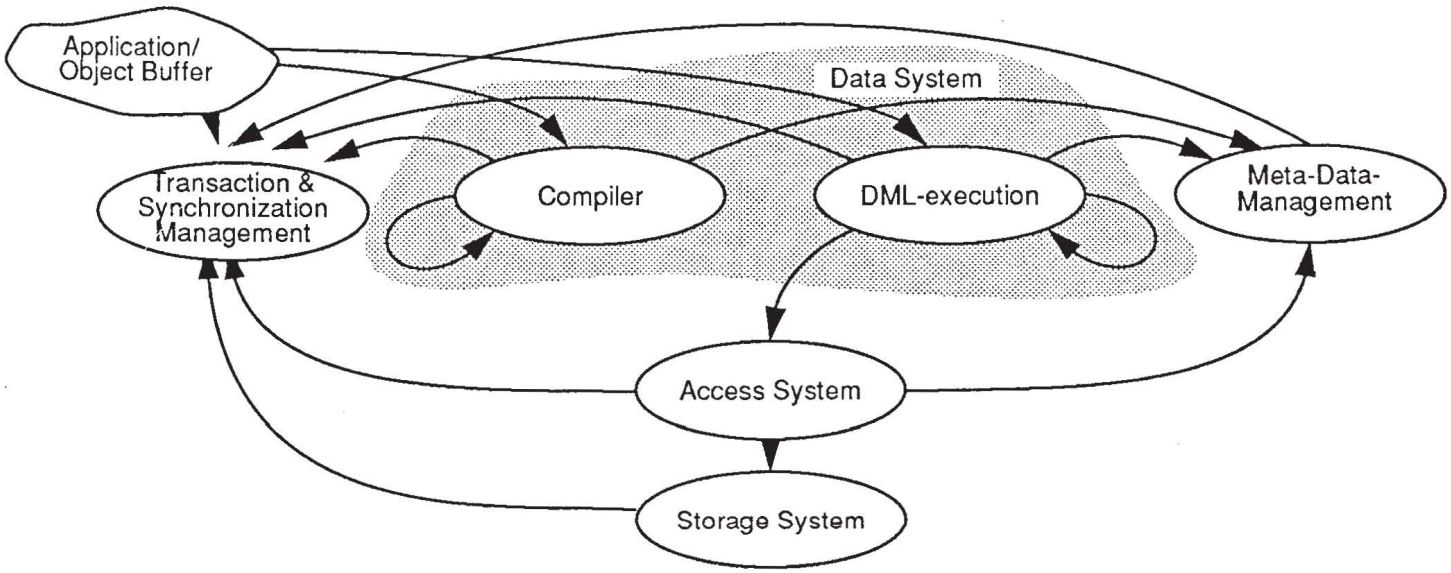


Fig. 2: Decomposition into server components

2.2 Query Processing in PRIMA

The complex-object data model provided at the kernel interface is called Molecule Atom Data model (MAD) and its language is called Molecule Query Language (MQL). Here, complex objects are assembled from primitive atoms: molecule construction basically follows references from one atom to other atoms in order to get the constituent atoms of the complex object. From a more general point of view this is quite similar to object assembly in object-oriented databases, where references between objects are the prevailing representation scheme for object relationships and therefore used for assembly [ZM90]. Molecule construction and of course also molecule qualification are the main tasks of *DML-execution* that requests atoms (internal records) from the *Access System* server, which in turn uses services of the *Storage System* to retrieve the pages that contain the atoms needed.

Molecule construction and qualification relies on an operator approach. That is, *DML-execution* consists of operators for complex-object assembly, join, recursion, sort, etc. The query evaluation plan, that is fed into DML-execution looks like a graph where the nodes are the operators and the edges represent the dataflow in the graph. At this abstraction level it is quite obvious that inter- as well as intra-operator parallelism (two forms of intra-query parallelism) can be exploited in addition to the well-known inter-query parallelism. For example, operators can be processed in a pipelined fashion or executed in parallel if independent from each other. Furtheron, there are a lot of parallel join or sort techniques as well as several approaches to parallel assembly of complex objects, e.g. by issuing requests for atoms in parallel. There are also ways to exploit parallelism in the *Access System* server, mainly for maintaining primary and derived data such as access paths and replicated data (due to clustering approaches). More detailed information on molecule processing in PRIMA can be found in [HMS91], and [HSS88] contains an overview of conceivable parallelization approaches in complex-object processing.

2.3 Difference to other approaches

The purpose for developing PRIMA was to get a flexible testbed that allows for detailed investigations in complex-object processing with special emphasis on parallelism. Therefore, our research interests deffers from most others that concentrate e.g. on parallel join or on sort operations on declustered data relying on relational operators and using lots of different hardware platforms [DG90,BA90, LD89/. For complex objects data partitioning does not seem as useful because of the highly meshed structures due to sub-object sharing. Since complex-object processing has a different flavor than processing flat relational data, the Volcano project (cf. [KGM91] and [Gr90]) recently introduced an assembly operator for complex objects and does also provide parallelization concepts by means of the exchange operator (we can view RCS useful for a realization of Volcano's exchange operator). There, research emphasis is concentrated on operator issues and extensibility in query evaluation, whereas our approach comprises parallelism in all system layers since the whole architecture reflects the structures required by the client/server model.

3. Programming environment supporting parallel/cooperative database processing

As suggested in the previous chapter, the interaction/cooperation among the components of our PRIMA DBMS follows the client/server model, i.e., each component offers a couple of services that can be activated from other components to perform their individual functionality. This hierachical service activation leads to a tree of *service activation units* as illustrated in Fig.1b. Following the major goal of our investigations, this tree has to be executed in parallel whenever the service semantics permits it. Hence, it has to be performed in a distributed processing environment given by tightly or closely coupled multiprocessors or by loosely coupled multi-computers. In order to separate and isolate the mechanisms that support parallel and cooperative work in a distributed environment from the application, we have implemented the above mentioned **Remote Cooperation System** (RCS). It serves as runtime environment for the PRIMA DBMS [HKS91]. The general requirements that RCS must satisfy can be summarized by the following three issues related to independent views to the system:

- Application view: RCS has to support **cooperation and parallel execution** of service activations in a distributed client/server system. The encapsulation issue enhances the stability of the application programs w.r.t. the actual realization of RCS functionality.
- Administration view: It should provide **a high degree of hardware as well as software (operating system) independency** in order to allow for **portability and adaptability** of RCS and its application system. This is realized by means of adequate **system configuration facilities**.
- Application designer's view: RCS should also support **monitoring and analyzing facilities** that allow for the **valuation and tuning** of the actual RCS con-

figuration of its application system (i.e., the PRIMA DBMS).

These topics are discussed in more detail through sections 3.1 to 3.3.

3.1 RCS Characteristics and Functionality

The most important features of RCS seen from an application programmers point of view are pointed out in the following:

- A system component may play **server role as well as client role** for other system components at the same time. E.g., the access component (cf. Fig 2) uses as client the functionality of the storage component and provides its own functionality in a server role to the DML-execution component.
- Each service request is bracketed between **service initiation** and **result reception**. The activation of a particular service is performed asynchronously allowing for parallelism among a client activation unit and all of its initiated services.
- Each server component can accept **multiple service initiations** and can perform the associated operations independently. This feature is used extensively within the Compiler as well as the DML-execution component (cf. Fig. 2) for recursive service activations.

Fig. 3 gives a **very simple** example of client/server cooperation based on RCS functions. After local initialization (Rc_Init) the client-site connects itself to the servers needed. Then, the processing starts: client-site initiates as needed services (Remote_Service_Initiation) at several server-sites. Optionally, it can activate multiple services of a single server. On server-site, service activations are independently accepted (Accept) and processed. For client usages, RCS provides functions to look in a non-blocking manner or to wait (i.e., blocked) for service results. Fig. 3 shows the wait case (Wait_For_Service_Termination) where the client-site resumes when the server-site returns results. At the end of processing, the client disconnects itself from the connected servers and terminates thereafter.

During service evaluation, dependencies can occur due to semantic relations among several service activations (e.g., between fix-page and unfix-page service activations of the PRIMA transaction/synchronization server). In order to cope with this kind of dependencies (not shown in Fig. 3) RCS provides an event mechanism based on wait/signal event functions. Additionally, RCS carries out some kind of *global storage* which is shared from a logical point of view. That is, data allocated in global storage can be transmitted from one component to another with a *call by reference* semantics. Particularly, it is possible to transmit referenced complex data structures like lists, trees, etc. simply by allocating their nodes within global storage. Again, RCS hides the actual realization from the application system.

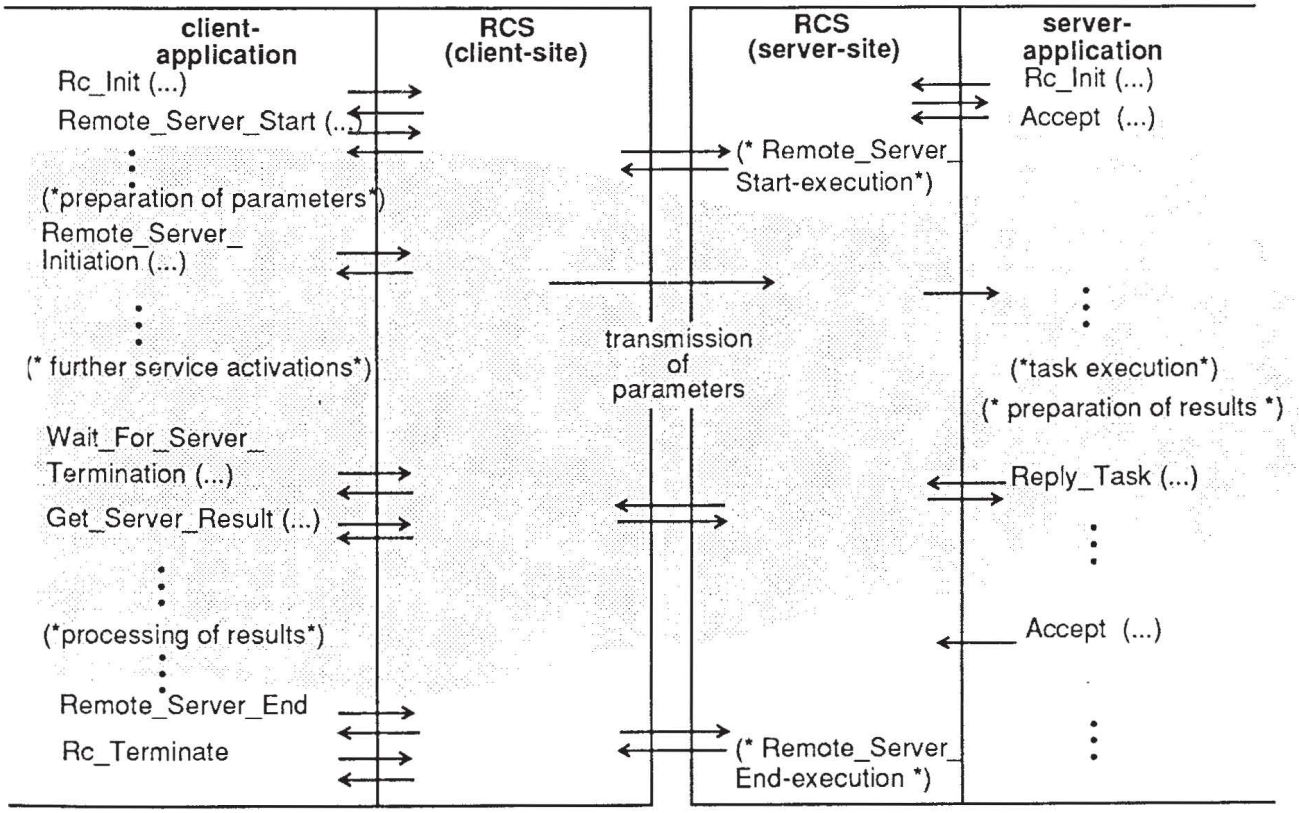


Fig. 3: A sample schedule of client server cooperation

3.2 Parameterization and Configuration

The configuration parameters supported by RCS comprise the underlying hardware and related Operating System (OS), as well as the aspects of embedding system components (and thus their services) into the OS processing units.

Until now, RCS allows for use of the following hardware/OS platforms: Sun workstations under SunOS, Apollo workstations under AEGIS, Sequent shared-memory multiprocessor under Dynix, and Siemens mainframe under BS2000. That is, all the RCS cooperation primitives show a high degree of hardware/OS independence, and isolate the application programs (at least those parts that deal with cooperation) from particular hardware/OS features (concerning memory and communication). For performance reasons, RCS tries to exploit as much as possible existing features (e.g. shared memory for communication purposes or for realization of global storage) of the underlying platform without making this fact visible to the application programs. Application components (that have no other hardware/software dependency, e.g. no need of special graphic hardware or software packages) can be attached to different platforms by easily editing a configuration file when system starts up.

Besides this kind of configuration, one can change the embedding of application components into OS processing units (i.e. processes). In the standard case, each application component is mapped to a separate process, which is the unit of processor attachment. That is, all service activations of one server are executed by a single process; hence, by a single processor. However, RCS achieves the required processing independence of multiple server activations by a multi-task mechanism, i.e. by an intra-process task organization. This means that parallelism is normally exploited in an inter-server fashion only. In order to exploit different degrees of parallelism, RCS allows for changing this mapping strategy. One can specify *multi-server processes* as well as *multi-process servers*. Selecting the *multi-server process* mapping that is, multiple servers attached to a single process, allows for reduced cooperation overhead between the inner servers, thus compensating the loss of parallelism. In the other case, i.e., when mapping one server to multiple processes, the possible parallelism can be exploited effectively.

The latter case brings up some difficulties since it affects load balancing that marks a sole problem in the whole DBMS field. Where selecting the multi-process server mapping, one has to extend in today's implementation of the RCS the application programs by appropriate load balancing strategies. To improve this situation, we are working on RCS enhancements that should allow us to separate load balancing from the remainder application and to plug server-specific load balancing strategies into RCS. This feature will significantly improve the easiness of system configuration since RCS will be able to hide the kind of server embedding totally from the application.

3.3 Measuring and Performance-Analysis

Valuation and tuning of distributed and parallel client/server application systems require tailored measuring and analyzing support. So far, we have worked on a suitable performance analysis methodology and implemented adequate measuring and analyzing facilities within the RCS framework [GHKSS91]. These facilities allow for the online/offline monitoring of system dynamics and load distribution, as well as monitoring of load dependencies between server, process, and processor. Offline monitoring and a further analyzing facility are based on log-file informations that are generated during runtime. The main goal of these tools is analyzing relationships between components in a (distributed) client/server application. A visualization feature helps in recognizing hints for distribution strategies and optimized system configurations, and in identifying bottlenecks within the processing behavior of client/server application systems.

4. A DBMS Testbed Scenario

It is not clear from the beginning how a good parameterization and configuration for parallel complex-object processing in PRIMA-DBMS look like, when load and size of the database, as well as the underlying hardware (and OS) are subject to change. One of the research issues is to extract some rules that help as guidelines for parameterization, configuration, and tuning. In approaching this, we have implemented the measurement and analysis tools mentioned. A general testing phase of these tools is already passed and we are now using them to evaluate the effects of variations of parameterization and configuration. Right now, we are working on a first evaluation of different PRIMA configurations.

There are three extreme positions of system configurations to be distinguished:

- The **single processor configuration**, where all PRIMA servers are attached to the same processor.
- The **LAN configuration**, where each of the PRIMA components is executed by a separate processor.
- The **shared-memory and multiprocessor (SMMP) configuration**, where PRIMA is totally distributed onto the processors of our Sequent machine.

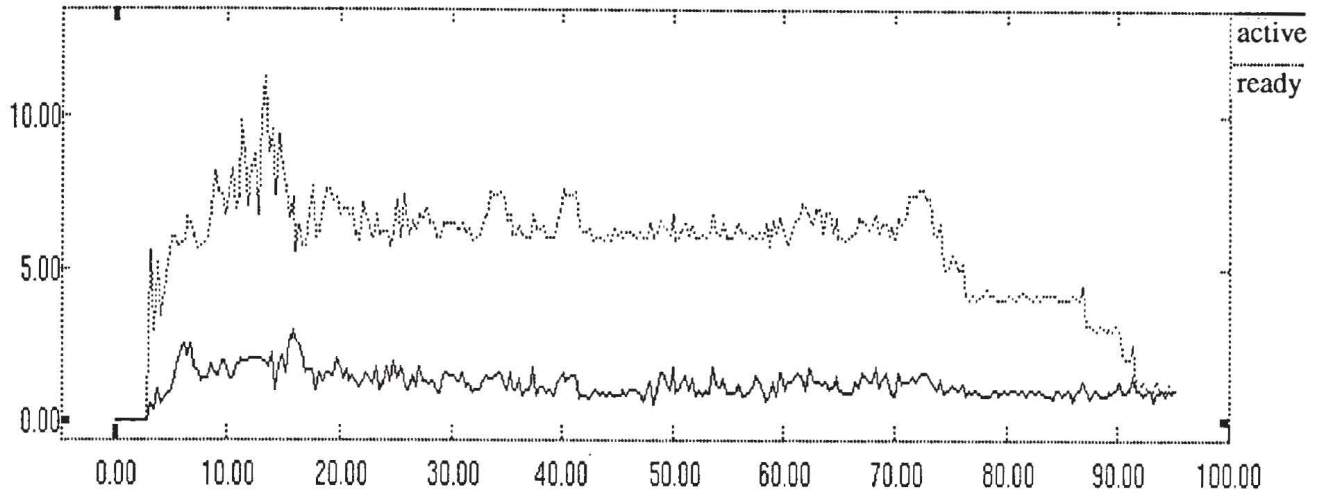


Fig. 4: Total number of (active/ready) tasks drawn over the observation time

Our first investigations aim at how to find out a LAN configuration being optimal/good for a given workload. The following example demonstrates the methodology that we have applied to tune a system configuration and to identify some weaknesses of our PRIMA implementation. Knowing that a "good" configuration lies somewhere between a single processor and a totally distributed LAN configuration, we start our evaluations with one of these extreme positions: initially, **each of the PRIMA components is carried out by a single process sharing a single processor** (later refered to as WS1).

The basic heuristics used to find out an optimal system configuration are based on two general insights:

- First, multiple processors are needed only if a single processor is saturated, i.e. it constitutes a bottleneck in DBMS processing.
- Second, the total response time of a distributed client/server system is positively influenced by a uniform and homogenous load of each system component.

The workload of the example considered is determined by a small database and a statically defined set of 24 MQL statements with in the mean 6 statements in parallel. The system configuration comprises *Transaction Management* server, *DML-execution* server, *Access System* and *Storage System* server, as well as *Meta-Data Management* server.

As one might have expected, the applied processor was overloaded, i.e. the cpu works permanently during the total observation time (no idle time). Fig. 4 shows the number of tasks of all PRIMA servers drawn over the observation time. The mean values for the active as well as the ready tasks indicates the degree of overloading, since these tasks could be performed concurrently if additional processors and/or processes would be available. In other words, it makes sense to distribute some PRIMA servers to a further processor and/or to change the mapping strategy for particular servers from a single-process to a multi-process mapping. In order to determine which server(s) should be attached to an additional processor, we must have a closer look at the server specific workload characteristics. Doing so, the DML-execution server is identified, since it significantly influences the system behavior as is illustrated in Fig. 5. The curves indicating the workload of the *DML-execution* server (given by the number of active/ready

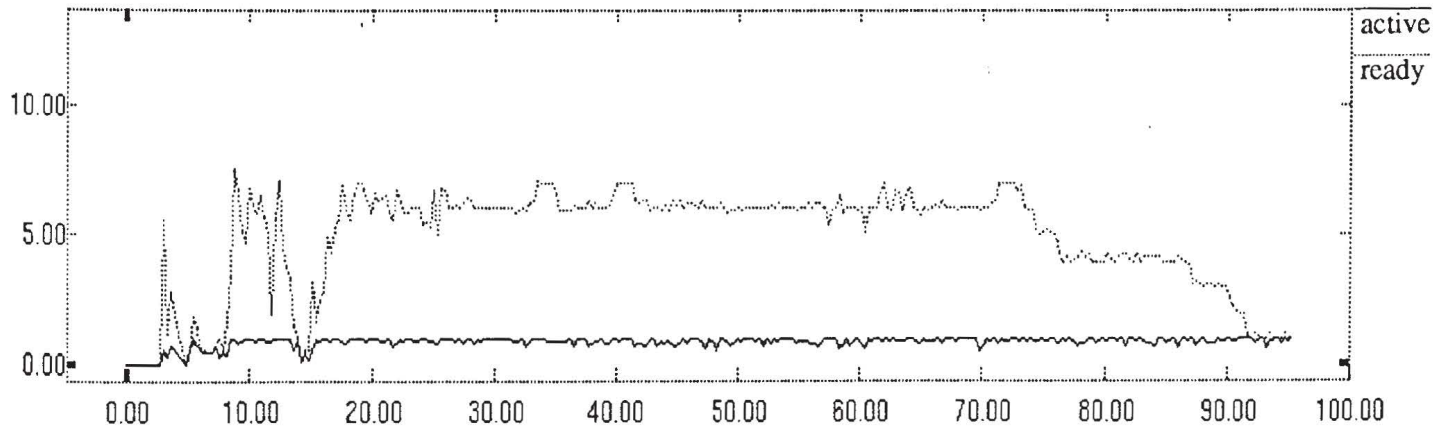


Fig. 5: Number of (active/ready) tasks performed by the DML-execution server

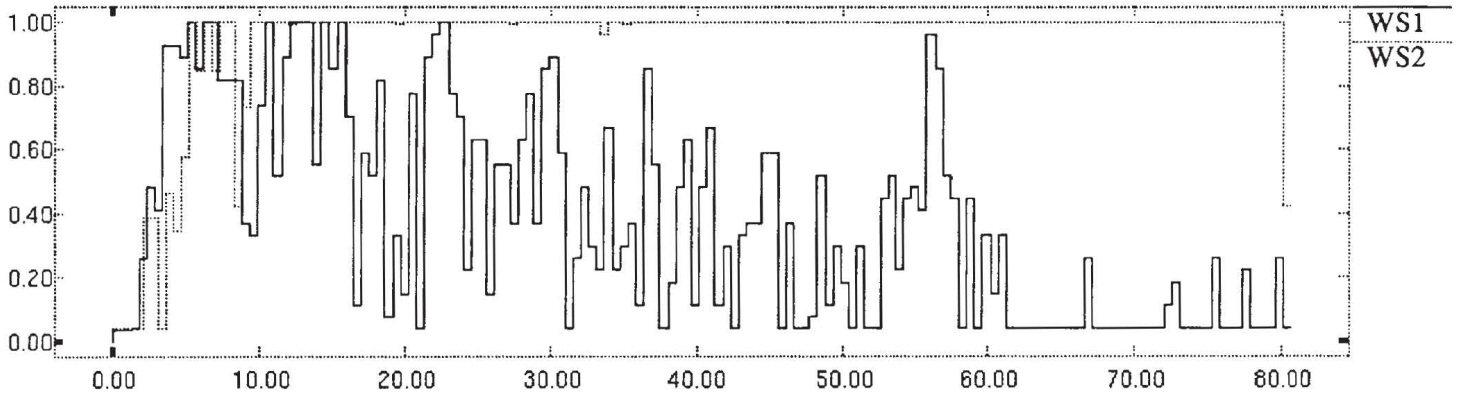


Fig. 6: Saturation degree of WS1 and WS2 (1.00 indicates total processor saturation

tasks) look quite similar to the overall task curves shown in Fig. 4.

To improve the system behavior, i.e., to reduce PRIMA's processing time, we **reconfigured and attached the DML-execution server to a second workstation (WS2) in our LAN.** Besides the 20% reduction of the overall processing time this decreased the processor load to 64% in the mean.

Having a closer look at the processor specific workload one can identify the new/old bottleneck: the *DML-execution* server. The processor that performs this server (WS2) is permanently busy (cf. Fig. 6)

There are two measures of improvement:

- **Change of mapping strategy**, i.e., duplicate the process that performs the DML-execution server and attach them to a different processors.
- **Analyzation of the inner processing strategy of the DML-execution server**, in order to identify the actual reason for this bottleneck.

The first measure addressed will ameliorate the bottleneck situation in any case, the second one may make additional sense. Fig. 7 illustrates the states of all such tasks triggered by executing a particular task of the *DML-execution* server. Obviously, the *DML-execution* task (task identification number one) initiates most of its subtasks in a synchronous manner. Reviewing the server implementation one might recognize whether this fact is problem inherent or it results from an inappropriate implementation. In the latter case one can improve the server's programming by exploiting asynchronity. A further look at Fig. 7 shows a misrelation between the periods in which the task state is ready and in which it is active. This may mean that the corresponding servers do not perform their tasks in the right schedule, since they work for others (clients) while blocking the *DML-execution* task. Further detailed observations may validate this supposition and a modification of the scheduling

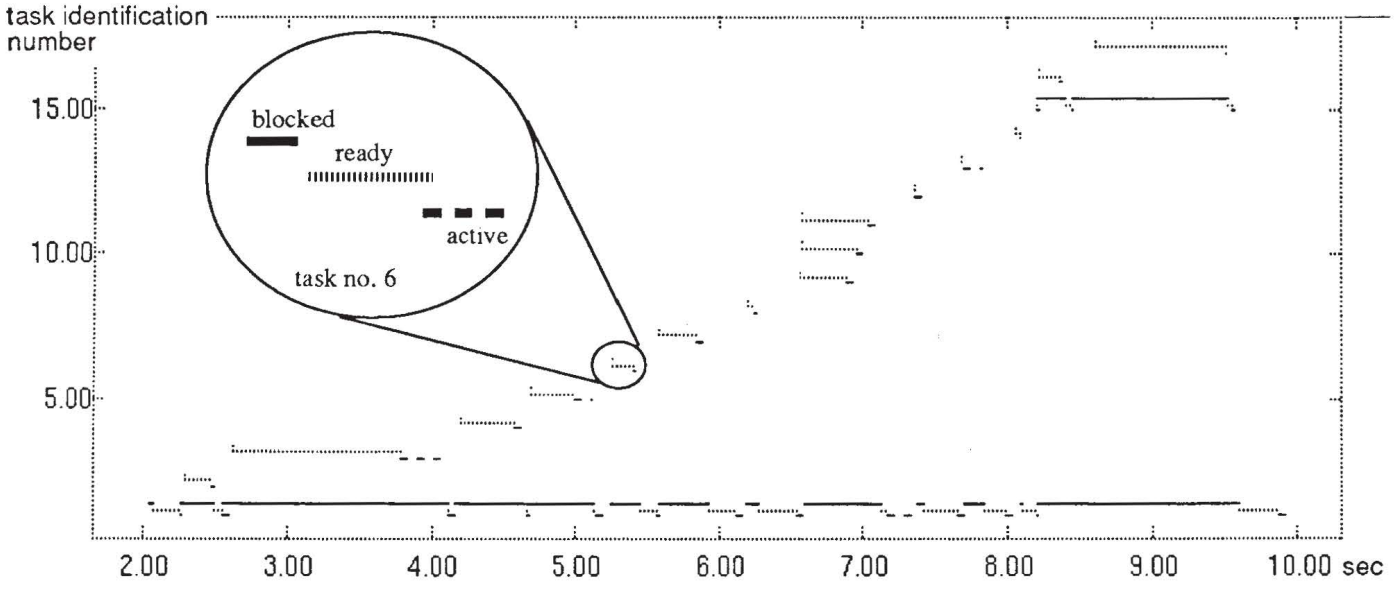


Fig. 7: State/time diagram of all tasks triggered by executing a particular task of the DML-execution server

strategy will become necessary, what in turn, is easily achieved by the underlying RCS functionality.

5. Conclusions

So far, this example has sketch some steps of a methodology for optimizing the PRIMA configuration (as well as its implementation) under the presumption of a fixed load. By changing the workload we can evaluate the dependencies between the configuration decisions and the load characteristics. In order to meet our goal of finding some general rules for system configuration, we have to consider additionally the SMMP configuration case. A preliminary conclusion drawn from our investigations so far, comprises the following, rather abstract guidelines for optimal setting of system parameters, that are now validated by further testbed evaluations:

- **Data allocation**: Especially support for shared data is crucial. For the LAN configuration it is quite clear that those components that share data (e.g. Access server and Storage server share the DB-buffer allocated in RCS global storage) do better share processors in order not to share data over the network; for the SMMP scenario this is not so critical since the DB-buffer (also allocated in RCS global storage) is supposed to be part of the shared memory so that all components have fast and cheap access.
- **Kind of parallelization**: Kind of parallelization determines dataflow characteristics. For example, pipelined processing opts for a 'continuous' stream of data between server and client, whereas independent parallel processing just needs result transmission between both.
- **Workload characteristics**: As illustrated in the example, bottleneck identification and elimination is very important. If the saturation of processors is reached, then it is necessary to attach servers to other processors or if task-saturation for server processes occurs, then it is worthwhile to replicate this server, eventually on another processor.

In addition to the investigations for optimal system parameter settings w.r.t. parallelism, further research topics under consideration concentrate on the evaluation of the sole as well as the cumulated effects of the other measures mentioned, thus working on different query processing strategies, optimization strategies, and different clustering and indexing techniques.

In this paper we concentrate on complex-object processing on the DBMS kernel only. Considering that in engineering applications the way of processing (in the application and application layer) is typically determined by mechanisms like check-out/check-in, local databases, long-living transactions, etc., for evaluating and tuning the overall system behavior it becomes important to consider both application processing and DBMS processing. This identifies additional guidlines for further investigations in our PRIMA framework.

6. Literature

BA90 Boral, H., Alexander, W., Clay, L., Copeland, G., Danfurth, S., Franklin, M., Hart, B., Smith, M., Valduriez, P.: Prototyping Bubba, A Highly Parallel Database System, in: Knoweldge and Data Engineering, Vol. 2, No. 1, March 1990.

DG90 DeWitt, D.J., Ghandeharizadeh, S., Schneider, D.A., Bricker, A., Hsiao, H.-I., Rasmussen, R.: The Gamma Database Machine Project, in: Knowledge and Data Engineering, Vol. 2, No. 1, March 1990.

Gr90 Graefe, G.: Volcano, an Extensible and Parallel Query Evaluation System, Research Report University of Colorado at Boulder, CU-CS-481-90, 1990.

GHKSS91 Gesmann, M., Hübel, C., Käfer, W., Schöning, H., Sutter, B.: Messen und Bewerten paralleler Client/Server-Architekturen - am Beispiel des kooperierenden Non-Standard-Datenbanksystems PRIMA, in: Proc. der GI/NTG-Fachtagung "Messen, Modellieren und Bewerten", MMB91, München 1991.

HKS91 Hübel, Ch., Käfer, W., Sutter, B.: Ein Client/Server-System als Basiskomponente für ein kooperierendes Datenbanksystem, in: Proc. der GI/ITG-Fachtagung "Kommunikation in Verteilten Systemen", KIVS-91, Mannheim, Feb. 1991.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. of the 13th VLDB, Brighton, 1987.

HMS91 Härder, T., Mitschang, B., Schöning, H.: Query Processing for Complex Objects, appears in: Data and Knoweldge Engineering, 1991.

HSS88 Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, in: Proc. of the International Symposium on Databases in Parallel and Distributed Systems, Austin, Texas, 1988.

HSS89 Härder, T., Schöning, H., Sikeler, A.: Evaluation of Hardware Architectures for Parallel Execution of Complex Database Operations, in: Proc. 3rd Annual Parallel Processing Symposium Fullerton, CA, USA 1989, pp 564-578.

KGM91 Keller, T., Graefe, G., Maier, D.: Efficient Assembly of Complex Objects, Research Report, University of Coloardo at Boulder, CU-CS-502-90, submitted to SIGMOD '91.

LD89 Lorie, R., Daudenarde, J., Hallmark, G., Stamos, J. Young, H.: Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience, Data Engineering, Vol. 12, No. 1, March 1989.

Mi89 Mitschang, B.: Extending the Relational Algebra to Capture Complex Objects, in: Proc. 15th Int. VLDB Conf., Amsterdam, 1989.

ZM90 Zdonik, S., Maier, D.: Readings in Object-Oriented Database Systems, Morgan Kaufmann Publ., 1990.