Simulating multiprocessor database system architectures[♣]

© Pavel S. Kostenetskiy
South Ural State University
kostenetskiy@yandex.ru

© Leonid B. Sokolinsky
South Ural State University
sokolinsky@acm.org

© Anthony Kulig
South Ural State University
kuliga1@yahoo.com

Abstract

In this paper we propose a new approach to simulating parallel hardware architectures based on the hierarchical database system model, and an architecture simulator is implemented based on this new approach. This simulator allows for the investigation of different hierarchical architectures under an OLTP workload. It also gives us the opportunity to investigate a wide variety of hybrid and hierarchical cluster architectures without the excessive expenses of hardware implementation.

Introduction

During the last ten years a number of problems have arisen that require the storage and processing of very large volumes of information [1]. In connection with this, it has become necessary to develop new hierarchical architectures of multiprocessor database systems, which allow us to process extremely large volumes of information in a small amount of time. In relation to these problems, hybrid hierarchical architectures are of the greatest interest [3]. Investigation of servile architectures is difficult, since the construction of multiprocessor systems demands large financial expenditures connected with acquisition and reconfiguration of expensive equipment. Therefore, this approach for many research organizations is unacceptable and thus the development of software models, representative of multiprocessor database systems, is a potentially promising solution to this problem. These models allow us to investigate different multiprocessor configurations without their hardware implementation. The investigation of multiprocessor database systems with one-level and two-level architectures has frequently been done with the help of models (for example, [4]). However, in general hybrid hierarchical architectures have not been explored.

This paper offers a new approach to simulating hierarchical architectures of parallel database systems with an arbitrary number of levels, and outlines a package of programs based on this approach, which allows for the investigation of different hierarchical configurations, including hybrid ones. This gives us the opportunity to investigate a wide variety of promising hybrid and hierarchical cluster architectures without the financial expenditures associated with hardware implementation.

1 Multiprocessor database system model

We are proposing a *Database Multiprocessor model* (*DM*-model), which can be used to investigate a wide variety of hardware-based architectures, including hybrid architectures. The *DM*-model has been developed for the parallel processing of transactions working in the OLTP regime and based on the relational data model.

Within the framework of our model, database architectures are represented in the form of *DM*-trees.

A *DM*-tree is a graph whose nodes are of one of the following three types: processor modules, disk modules and hub modules. Further we shall name the *DM*-tree's nodes as modules. The graph's edges in this context are an abstraction of bidirectional data transmission channels connecting the modules. The *DM*-tree root can only be a hub module. A processor module can be connected to a disk module only through a hub module. The disk and the processor modules are always leave-nodes of the *DM*-tree.

We will call the representation of a real multiprocessor system in the form of a *DM*-tree a virtual multiprocessor.

Within the framework of our model the processing of information is accomplished with accuracy to the quantity of information known as a packet. A packet contains one or more tuples.

Within the framework of the *DM*-model we will define a cycle as the period of time necessary for all the processor modules to have processed one packet, the disk modules to have performed one read-write operation for one packet, and the hub modules to have processed all the packets waiting for transmission at the moment the cycle begins.

The time of processing a query in the *DM*-model is calculated as the sum of cycles processed by the system between the initialization and completion of a

[♣] This work was supported by the Russian foundation for basic research (project 03-07-90031).

Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, St.-Petersburg, Russia, 2005

query. This coarse definition of the time it takes to process a query is acceptable for the OLTP-applications, since we can consider that a packet's size is incommensurately small in comparison with the database as a whole

We assign a *labor-consuming coefficient* h_{n_i} to each *DM*-tree module, which determines the time necessary for this module to process one database packet.

The time it takes the processor module to process one packet for the OLTP-applications is 10^5-10^6 times faster then it takes a disk module to perform a read-write operation for one packet, or the time it takes to transfer one packet through a hub module[5]. Therefore, we will suppose in the *DM*-model that the labor-consuming coefficient for the processor module is always equal to zero.

As the hub module n_i can transmit more than one packet simultaneously in one cycle, in addition to the labor-consuming coefficient for each hub module we introduce an interference function $f_{n_i}(m)$, in which m is the number of packets simultaneously passing through the module. Thus, the time, required for the hub module to complete one cycle is calculated by this formula:

$$t_{n_i} = h_{n_i} + f_{n_i}(m).$$

We will define N as a set of all the hub modules of the *DM*-tree, D as a set of all of the disk modules in the DM-tree, P as a set of all the processor modules in the DM-tree, and *S=(N,P,D)* as a set of all the *DM*-tree modules. By means of n_i, p_i, d_i we will denote the elements of the corresponding sets, that is the hub, processor, and disk modules.

The time spent by the system to complete one cycle is calculated by the formula:

$$t_i = \max(\max(t_n), \max(t_d)), \forall n \in N, \forall d \in D.$$

In according with this, the overall amount of time it takes for the system to complete a series of transactions composed of k cycles is calculated by the formula:

$$T = \sum_{i=1}^{k} t_i$$

Within the framework of the proposed model a cycle is divided into 2 stages:

1) Each processor module accesses a disk module;
2) All hub modules transmit their packets to an adjacent level.

All the processor modules can access the disk modules each cycle. There are two types of disk access: reading from a disk and writing to a disk. The difference between these types of access is as follows: when reading from a disk, the packet of information begins its movement along the architecture tree from the side of the disk module towards the processor which is accessing the disk module. In the case of writing to a disk, movement occurs in the opposite direction. The path of the packet's delivery from its current location to its destination is calculated dynamically by a deterministic algorithm.

In this implementation, the main characteristic of the simulated architecture is the time, over which the processing of a given number of cycles is completed.

2 Methods of simulation

This section is devoted to outlining the methods used to simulate the hardware implementation of a DMS (Database Multiprocessor Simulator).

The C programming language has been chosen for the software implementation of the DMS.

The *Database Multiprocessor Simulator* is constructed from the following virtual components: processor modules, disk modules, and hub modules. The methods of simulating objects of the above mentioned types are considered in subsections 2.1–2.3.

2.1 Processor modules

A list of processes is assigned to each processor module. Processes in this context are meant to simulate the processor module's processing of database queries.

Each process is assigned its own probability of being the active process at any given time on its processor. For the simulation of an idle processor, included in the list of processes is an 'idle process', which also is assigned an activity probability. During each cycle a single process from each processor modules process list (excluding processes that have already finished execution) is activated. The process may perform a single access operation to one and only one of the system's disks.

2.2 Disk modules

The disk module performs the following simple algorithm each cycle:

1) If there are any packets in an input buffer, one of them is deleted, as well as its corresponding packet in the expected packet list. This simulates the processing of an incoming packet.
2) If there are unsent packets on the disk, all of them are sent into the input buffer of a higher-level hub.

2.3 Hub modules

The hub modules send packets of information, created during processor access requests, from the senders to the addressers. Within the context of our model the hubs can send off packets only to modules directly connected to them. A given packet's delivery path from the sender to the addresser is unknown before transmission. Therefore, each cycle the hub modules calculate the number of the port, to which it must direct each the packets it has received. In one cycle, all packets should be processed one time. In other words,

every packet should be transferred one and only one level up or down along the *DM*-tree each cycle. After the hub module finishes processing a packet, that packet is marked as having been processed on the given cycle.

Thus, the path of a packets' delivery from sender to addresser is calculated dynamically, based on the packet's flow along the tree. Calculation of the path of a packet is based on a deterministic algorithm. We can describe this algorithm in the following abstract manner:

1) Traverse up to the root of a minimum subtree,
2) Traverse down to the addresser.

If some packets have to pass through the hub simultaneously during one cycle, then an interference function is added to the time required for the hub module to complete one cycle:

$$f(m) = e^{\frac{m}{\delta}}$$

In this formula m is the number of packets passing through the hub simultaneously, and δ is a parameter.

Conclusion

In this paper a new approach to simulating hierarchical hardware-based architectures of parallel database systems was described. A package of programs was designed based on the approach described in this paper, and was named the DMS (Database Multiprocessor Simulator). The DMS allows us to model many different architectures of parallel database systems. It gives us the opportunity to investigate a wide variety of promising architectures, including hybrid hierarchical cluster architectures, without large expenditures on their hardware implementation. With the help of the DMS program package we can perform modeling of multiprocessor database systems on a computer with any number of processors, including a single processor computer. This approach has been applied for modeling hybrid hierarchical architectures of the CDN class [1]. Preliminary experiments have confirmed the adequateness and effectiveness of the proposed model.

References

[1] Sokolinsky L.B. Survey of Architectures of Parallel Database Systems // Programming and Computer Software. 2004. Vol. 30. No. 6. P. 337-346.

[2] Sokolinsky L.B. Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // Programming and Computer Software. 2001. Vol. 27. No. 6. P. 297-308.

[3] Sokolinsky L.B. Design and Evaluation of Database Multiprocessor Architecture with High Data Availability // 12th International DEXA Workshop, Munich, Germany, 3-7 September, 2001, Proceedings. IEEE Computer Society. 2001. P. 115-120.

[4] Bhide A. An Analysis of Three Transaction Processing Architectures // Fourteenth International Conference on Very Large Data Bases (VLDB'88), August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings. -Morgan Kaufmann. -1988. -P. 339-350.

[5] Gray J., Graefe G. The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb // SIGMOD Record. -1997. -Vol. 26, No. 4. -P. 63-68.