

Inverting Middleware Framework: a Framework for Performance Analysis of Distributed OLAP Benchmark on Clusters of PCs by Filtering and Abstracting Low Level Resource Usage

Michela Taufer, Thomas Stricker, Roger Weber

Department of Computer Science
ETH Zentrum
CH-8092 Zurich, Switzerland
taufer, stricker, weber@inf.ethz.ch

Abstract

Clusters of commodity PCs are an attractive platform for parallel databases running large OLAP (On Line Analytical Processing) workloads. Using a high number of cluster nodes could result in a significant speed-up for processing a very large data set. Still engineering such systems for scalability and good performance remains a highly difficult task, since we still are lacking a deep understanding of the architectural issues of resource usage and scalability of OLAP applications on cost effective clusters of PCs. A considerable amount of work must be invested into better performance analysis tools that work well with parallel- and distributed platforms and with standard DBMSs. While standard DBMSs help the application writer to reduce programming effort, they often cause loss of control over performance issues resulting in suboptimal usage of machine resources.

To address this problem, we present a novel performance monitoring framework called inverted middleware framework (MW⁻¹). Our approach emphasizes the aspect of reverse mapping the resource abstractions introduced by the middleware layer (e.g. the DBMS) in term of resource usage cost. Inverted middleware framework assists the process of performance engineering by mapping low level performance information, monitored at the operating system layer, back to a higher layer (i.e. the application layer) filtering from performance counter samples at the operating system level and delivering good overall performance pictures at a higher level of abstraction. The framework is used side by side with the DBMS and delivers many interesting insights about the most critical resource in each of the different queries and systems configuration. As required for a larger distributed hardware/software system, our framework comprises some software instrumentation at the OS level, tools for gathering all performance relevant data and an analytical model that can be used for performance evaluation and performance prediction to newer platforms.

In this report, we demonstrate the viability of our approach with the in depth analysis of TPC-D, a standard OLAP benchmark running on clusters of commodity PCs. With the help of data provided by our performance monitoring framework, we are able to isolate and resolve a few crucial performance issues for OLAP workloads. As experts for the architecture of clusters, we intentionally limited our experimental work to two fixed distribution scheme (relations must be distributed to permit scalability to large data sets) leaving the issues of optimal data distribution and optimal use of indices to our database experts. As a result, we can give a good characterization of different OLAP workloads (i.e. the 17 queries of the TCP-D) in terms of their resource usage, quantify the optimal scalability for different queries and investigate the impact of the networking speed on the overall application performance. We can show that the disk performance and CPU speed remains the most critical resource bottleneck a majority of the queries. Queries with a lot of inter-node communication are rather limited by the communication software inefficiency within the DBMS than by the raw networking speeds. We think that our work constitutes a solid basis for future architectural decisions and system optimization in clusters of PCs that are dedicated to large parallel database systems.

Keywords: parallel databases, distributed OLAP processing, cluster architectures, cluster of PCs, performance analysis, workload characterization

1 Introduction

Clusters of commodity PCs have become a highly popular platform for distributed computing since simple PC workstations have one of the best price performance ratios in the market for computing equipment. Moving large OLAP (On Line Analytical Processing) database workloads to clusters of commodity PCs looks very tempting because of the lower cost of clusters and the significant potential for speed-up. As any other effort in parallel and distributed computing, the success depends on proper performance analysis of the entire system including hard- and software. Engineering such systems for scalability and good performance remains a highly difficult task, since we are lacking tools and instrumentation for performance analysis that work in distributed systems.

1.1 The Benefit of Middleware Layer (MW)

The functionality and the integration of application programs have greatly increased over the past 10 years. This increase in complexity of applications was enabled by numerous middleware layers solving most difficult low level programming problems on behalf of the application writer. Programming at the lowest abstraction level remains a cumbersome task and is avoided whenever possible. A *middleware layer (MW)* is a piece of installed software that allows the application writer to work on his problem at a higher level of abstraction. So in our approach, a middleware layer may be a black box layer such as ORACLE DBMS (DataBase Management Systems) or an open-source middleware packages such as MPI.

Middleware layers can effectively hide system dependent details and permits applications to be ported more easily from one machine to another. This is particularly important since the lifespan of a good application program is much larger than the lifespan of a computer platform. In parallel and distributed computing, many middleware layers address the problem of distributing large workloads, in particular, they readily solve the problems of data communication between the distributed entities, the issues of data partitioning, data allocation and load balancing.

1.2 The Problems of Middleware Layer

The fundamental problem with middleware layers is that the application writer no longer thinks at a system level suitable for performance debugging but at a level of abstraction that is much higher. The middleware layer takes care of mapping the high level commands and directives to low level system calls. But when it comes to debugging or performance tuning, most middleware layers are not properly instrumented and therefore do not map the system state or the performance indicators back to the higher API level. This is the most important hindrance to find bottlenecks and many application writers are completely lost once they have to track down bad performance in the application or even find losses of efficiency in the middleware layer code itself.

1.3 The Concept of Inverted Middleware Framework (MW^{-1})

In this report, we develop and explain the novel concept of *inverted middleware framework (MW^{-1})* for performance analysis and performance optimization. Inverted middleware framework assists the process of performance engineering by mapping the basic performance monitoring information of the operating system back to the higher level of abstraction for which the application code is written. The relationship between middleware layer and inverted middleware framework is very similar to the relationship of a parallelizing compiler for a high level programming language and the highly advanced source level debugger. We demonstrate the viability of this approach in the fairly advanced environment of “shared nothing” distributed databases running OLAP (On-Line Analytic Processing) workloads on clusters of commodity PCs.

1.4 A Typical Study Case with Suboptimal Performance

In the study described in this report, we look at the ORACLE DBMS as a middleware layer on top of the LINUX operating system and supporting the distributed TPC-D benchmark chosen as a typical example of OLAP applications. The scalability of the different TPC-D queries on PC clusters presents a picture of unpredictable performance and speed-up numbers that seem to be highly sensitive to the nature of the workload. Figure 1 shows a typical study case for the TPC-D benchmark distributed with TP-Lite (a software tool for distributing queries) on three nodes of a cluster of PCs. We expect a speed-up of almost three (shown with the upper line in the picture) however, not all the queries are sped up (i.e queries 2, 9 and 10) and some of them (i.e. queries 5 and 7) do not even end in a reasonable time due to some inefficiency in parallel processing.

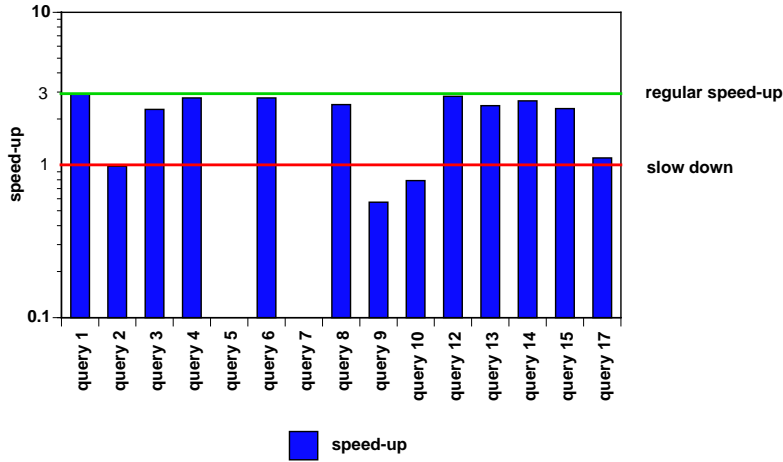


Figure 1. Scalability for the TPC-D benchmark distributed across three nodes of a cluster of commodity PCs with TP-Lite

1.5 State of the Art for Performance Tuning on DBMSs

For studying the reason of suboptimal performance values, most DBMSs incorporate elaborate instrumentation for performance monitoring and tuning [9, 6], but such instrumentation normally works on the basis of data collection within the DBMS itself and not on the basis of mapping the operating system state or the basic performance monitoring data back to an abstraction level that is more appropriate for a user. In particular, the performance monitoring instrumentation of ORACLE only accounts for the total count of operations and does neither allow to efficiently sample performance counts at certain intervals nor allow for this information to be collected efficiently from a large number processing nodes in a cluster. While some of the logical or table access counts would certainly be interesting, most performance information of the database management system does not directly relate to the usage of physical resources in a distributed system and is therefore hard to use in a framework that aims at predicting the execution time based on application and platform parameters.

In applications that use multiple processors and distribute a large workload for the sake of higher execution speed, proper engineering for performance and scalability becomes a crucial concern. A precise understanding of resource utilization is not just required for fine tuning a running system, but also for the prediction of scalability or the study of viability of the application on new, alternative platforms such as e.g. clusters of low cost commodity PCs.

1.6 Goals of the Performance Study in this Report

Our main contribution in this report is to explain precise resource usage and the scalability of the TPC-D benchmark running on clusters of PCs by means of our framework, the inverted middleware framework. We demonstrate that there is indeed a systematic way to filter the raw performance data provided by the operating system and turn the result into a more abstract performance picture that reflects the the resource usage of the distributed application code. As cluster architects, we can improve the configuration of future PC clusters based on this data about resource usage. Depending on the most performance critical resources, we can improve the cost/performance ratio by using cheaper or more expensive CPUs, disks, memory systems (motherboard) or interconnects between the nodes. If we can use cheaper components without penalty in performance, we can afford a larger number on nodes in a cluster.

Since cluster computing is always about the best use of commodity components, we look at parallel- and distributed systems built entirely commodity hardware and commercial software components. In particular, we combine the open source operating system LINUX with the proprietary single node DBMS of ORACLE and a lean experimental software layer for the distributing processing of queries. The high performance database system for decision support workloads can be classified as a shared-nothing architecture. Since we are interested in using cluster for very large data sets exceeding the disk capacities of a single node, our data distribution scheme relies on partitioning rather than on replication.

The rest of the paper is organized as follows: Section 2 describes our approach to distribute the OLAP workload to the nodes of a PC cluster based on the TP-Lite system and the basic characteristics TPC-D benchmark. Section 3 presents our performance monitoring framework, the *inverted middleware framework* (MW^{-1}), for distributed systems using raw performance data supplied by the operating system. Section 4 applies our methodology to three performance

issues in distributed databases. We use the framework to characterize the workload of TPC-D and classify the different queries according to their resource usage, to investigate the scalability problems of certain queries and to determine the performance impact of different interconnect speeds found in modern PC clusters. Section 5 concludes the paper with a statement about the viability of our approach to instrument a complex system of standard software properly for performance analysis and performance optimization.

2 Parallel TPC-D Benchmark on a Cluster of PCs

In this Sections, we give a brief overview on the TP-Lite approach and the implementation of the TPC-D [17] benchmark according to [1]. Furthermore, we discuss various characteristics of the platform and the middleware layer and describe the dimensions of the factor- and the parameter space of the benchmark for workload characterization.

2.1 Distributing OLAP Workloads on Clusters of PCs

Our performance analysis method was motivated strongly by OLAP (On-Line Analytic Processing) applications running in parallel on a cluster of PCs. The PowerDB project at ETH Zurich deals mainly with OLTP (On-Line Transaction Processing) workloads, but still provided many ideas and software tools for parallel queries (TP-Lite). We prefer to work with OLAP workloads (e.g. TPC-D [17]) over working with OLTP applications (e.g. TPC-C [16]) since OLAP applications always deals with large quantities of data that could potentially benefit from high speed interconnects in advanced PC clusters. Furthermore high performance workloads in OLTP do require a very large number of simultaneous queries to scale, and in general they do not result in large data transfers. Therefore, large scale OLTP jobs are much harder to generate than large scale OLAP jobs, and less interesting for computer architects.

We still use the TPC-D [17] benchmark as a representative of OLAP applications for historical reasons but could easily migrate our approach to TPC-H or TPC-R and the qualitative aspects of this article would remain the same if the those more recent benchmarks were used. Some of the newer work in the PowerDB project also deals with updates.

2.2 TP-Lite Approach

In principle, a distributed implementation of a database on a cluster of PCs works as follows: clients send their requests (i.e. SQL-transactions) to a so-called coordinator. The coordinator analyzes these requests, partitions them into several independent sub-transactions, and routes the sub-transactions to the nodes within the cluster. The goal of the partitioning and the routing is to minimize response times of queries and to maximize the throughput of SQL transactions. While the general approach is rather complex due to update operations and concurrency, we concentrate here on a query-only environment as this is sufficient for most work in OLAP. In this case, we do not have to deal with replication, concurrency, dynamic partitioning of data, and crash recovery (see [13, 14, 8] for more details). In our simplified approach based on a shared-nothing architecture, we use a static and disjoint partitioning of the data in the cluster. Queries are sent to all nodes and the union of their result tuples forms the overall answer of the query. The partitioning is such that each node has to touch about the same amount of data for query evaluation.

With TP-Lite, a query is executed in a master-slave setting using an instance of ORACLE8 on each node, its proprietary database links, and PL/SQL [12]. The coordinator runs both the master and slaves in independent transactions and we use ORACLE pipes as their communication primitive. An SQL-query is executed as follows: the master sends a message to all slaves over ORACLE pipes to initiate query execution. Each slave executes the SQL-query against the database of a dedicated node in the cluster using a database link. The result tuples are sent back to the master as messages over a shared ORACLE pipe. TP-Lite can be seen as a poor man's implementation of a parallel database.

Although Böhm et al. [1] reported that TP-Lite is easily outperformed by using a TP-monitor or a proprietary coordination layer, we have used the TP-Lite implementation in conjunction with OLAP on clusters of commodity PCs to demonstrate the ability of our inverted middleware framework to detect and identify performance bottlenecks. As seen in Section 4, our methodology is able to document and explain why the TP-Lite approach is not always scaling well when increasing the machine size to a larger amount of cluster nodes.

2.3 The TPC-D Benchmark

The TPC-D benchmark consists of a broad range of decision support applications that require complex, long running queries against large data structures. Although this benchmark came obsolete in 1999, we still use it as a representative of OLAP applications for historical reasons but we could easily migrate our approach to TPC-H or TPC-R and the qualitative aspects of this article would remain the same if these more recent benchmarks were used. Furthermore, we

are not interested in publishing new results for this benchmark, rather our aim is to demonstrate how our particular performance analysis framework may help in detecting bottlenecks and architectural problems.

The TPC-D benchmark contains 6 dimension tables (Customer, Nation, Region, Supplier, Part, PartSupp) and 2 fact tables (Order, LineItem). Out of the 17 queries of the TPC-D benchmark, we mainly used query 1, 3, 4, 8 and 12 for the experiments since they read large parts of the fact tables. As mentioned above, our implementation of a parallel database uses static partitioning of data. While the dimension tables are fully replicated on all nodes, the data of the fact tables is disjointly distributed over the nodes of the cluster. Since the queries under consideration run against large parts of the fact tables, we achieve a speed up with the cluster simply due to the reduced volume of data touched by each node. For the experiments, we used two different partitioning schemes. The first scheme (1 – *part*) partitions only the LineItem fact table, the second one (2 – *part*) partitions both fact tables. Further, note that our partitioning scheme and the selection of the queries guarantee that the union of the result tuples generated by the nodes in the cluster is equal to the result set of the query against the entire database. However, it is beyond the scope of this paper to discuss this issue and related ones in more detail.

2.4 Platform Characterization

Clusters of commodity PCs are becoming an increasingly popular new platform for large processing tasks. Therefore, we look at the execution of parallel queries on top of multiple instances of ORACLE running on each node of a cluster of PCs under the LINUX operating system. We refer to the database software needed to do this task as middleware layer. An application-run in our experiment depends on several parameters controlling the workload and the execution environment. The parameters under investigation are called factors. Most factors are external and under the control of the application writer. However, our approach also deals with intrinsic parameters that affect performance but are not directly visible to the application writer such as the set of factors related to the characteristics of the platform (i.e. the architecture of the PC cluster) that we define the *platform factors*. Among them we consider the factors at the following levels:

the clock rate of the CPU (t_{clock_cycle})	400 MHz Pentium II or 1000 MHz Pentium III
the average disk read performance ¹ ($disk_rate$)	22.0 MB/s (slow disk) or 30.5 MB/s (fast disk)
the average disk access time (t_{rand_access})	7.3 ms (slow disk) or 6.8 ms (fast disk)
the speed of the network interconnect ($network_speed$)	100 Mbit/s (Fast Ethernet) or 1000 Mbit/s (Gigabit Ethernet)
the number of slaves in the cluster (ns)	1, 3 or 6 processing nodes ²

Besides the factors (parameters under investigation), some parameters are held at constant level for this investigation. For the sake of better explanation, they are written as variable in the analytical performance model of the inverted middleware framework. Among them are the clock cycle per instruction ($CPI = 1$) linking CPU clock-frequencies to integer performance, the size of a memory block ($blk = 512$), which is an important constant in the I/O buffering system of LINUX, the physical capacity of the disks ($disk_size = 18$ GB), the total size of the tables in the TPC-D database benchmark (10GB) and the memory size (256MB).

The main motivation behind the migration of OLAP database work to clusters of PCs is to run huge databases of terabyte-scale on PC clusters that are equipped with a large number of inexpensive disks. For faster experimentation with different cluster configurations, we scale the problem size down to 10GB and artificially limit memory size on each node to keep the original balance in the storage hierarchy.

3 Performance Evaluation in the Context of Distributed Systems

3.1 Inverting the Middleware Layer Functionality (MW^{-1})

To address the problem of filtering and abstracting the raw data, we propose to create a framework that we will call “inverted middleware”. This framework is working in the environment of distributed computing and comprises software instrumentation at the OS level, performance data gathering tools and an analytical model that is tailored to a particular application.

¹The disk characteristics in detail are: slow disks, Seagate SCSI ST318203LW: with a min/avg/max throughput of 14.5/22.0/26.9 MB/s and access time 7.3 msec, the fast disks are Seagate SCSI disks ST318404LW with a throughput of 22.8/30.5/36.3 MB/s and 6.8 msec access time.

²The power database project aims at the scalability to a larger number of nodes (16, 32, or 64 processors), however measuring such task requires very large data sets. With our current experiments, we use 10 GB of data which is adequate for 1, 3, 6 processors. We plan to extend our work to the new 128 node “Xibalba” database cluster as soon as we manage the engineering challenge of generating and distributing data sets of 100 GB to 10 TB size with LINUX.

Unlike previous work that focuses purely on instrumentation, our approach emphasizes the aspect of mapping and reverse mapping of abstractions in the middleware layer and inverted middleware framework respectively. We propose that the inverted middleware framework is developed separately and that its functionality is not to be integrated with the middleware layer itself. This approach might impose some limitations in what performance data can be gathered and what problems can be analyzed, but our experience with manual instrumentation of specific middleware layers [15] has shown that developers rarely care to instrument their middleware layer for backward mapping system state into user level abstractions. Most middleware layers are proprietary or far too complex to modify and can therefore only be used as a black box in large software systems.

Figure 2 summarizes the relationships between middleware layer and inverted middleware framework as two parts of a complex software system permitting the development and the performance analysis of applications hand-in-hand. The graph shows some obvious structural symmetries (note the interesting combination of horizontal and rotational symmetries).

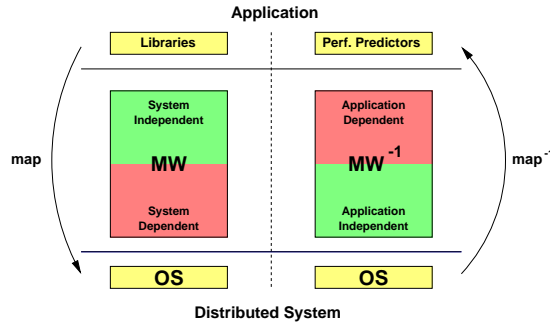


Figure 2. Symmetric structure of middleware layer and inverted middleware framework executing with an application code on a particular system platform.

In complex systems, the mapping of high level abstractions to operating system specific functions resembles a chain of mathematical functions leading from the high level application calls to the API of the middleware layer and to the low level system calls of the underlying operating systems. Therefore, it is most obvious to use a similarly layered implementation of inverse mapping functions to project the system state and the performance monitoring data into the abstract world of the application layer. This approach of decomposing complex distributed systems into layers should be viable as long as the distributed system in question shows the property of nearly-complete decomposability as stated by Courtois [4, 5]. According to the criteria given by Courtois, we found this property in the distributed system used for our distributed OLAP application. Therefore, we can break our system into subsystems and propose their aggregation in single blocks or layers. This technique permits to study the interactions among layers (weak interactions) without knowing and considering the interactions within the layers themselves (strong interactions) and therefore we can treat each layers of the system separately as a black box and try to find an inverse to its functionality. Such is required since we do not have source code or access to the internals of the commercial ORACLE database management system running on each cluster node. Figure 3 shows the decomposition and the three layers considered

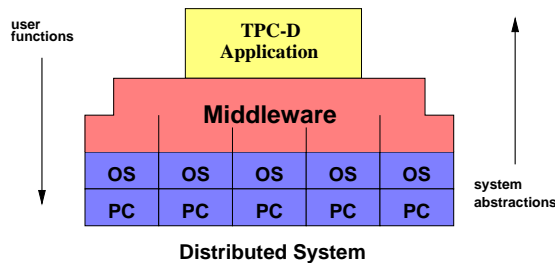


Figure 3. Middleware layer functionalities for distributed computing, i.e. task parallelism, distributed task execution, automated data distribution.

for our system: the application executing the SQL query, the conventional DBMS layer running on each node and the distribution system that parts and distributes the work among multiple cluster nodes.

Most platforms for distributed computing (hardware and operating system) are already well instrumented with detailed performance monitoring facilities like high precision timers, performance counters or system call tracing facilities

inside the operating system, but those are running at each node of the distributed system separately. So for most systems the information for performance engineering is already there, but currently it can not be used at the application level due to a middleware environment. It is the functionality of the inverted middleware framework to make this data available for performance studies and performance optimization.

3.2 The Structure of Inverted Middleware Framework

The software system of an inverted middleware framework is structured into three different layers as well: an application-specific layer, a distribution-specific layer and a system-specific layer. Figure 4 shows the three layers and how they relate to the other components of the distributed system.

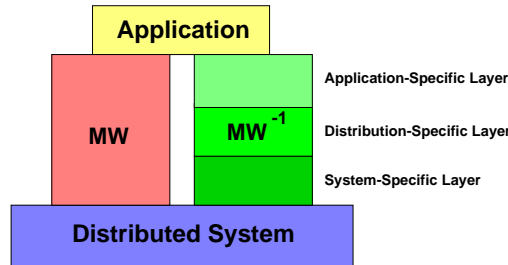


Figure 4. Structure of inverted middleware framework in the distributed environment.

The *system-specific layer* of inverted middleware framework monitors and collects system-specific performance data. System-specific performance data includes information on resource usage and bottlenecks gathered over the lifespan of the application-run. In our current prototype, we monitor the following resources: the local CPUs usage, the local disks usage and the local network usage as sampled on each platform node.

The *distribution-specific layer* gathers the performance related data from several nodes and patches it into a single coherent view of the whole system to be handed over to the application-specific layer. Inverted middleware framework inherits the master-slave setting from its middleware layer counterpart. The information gathered by the master is already properly filtered by the slaves and ready for processing at the application-specific layer.

The *application-specific layer* uses the global performance data of the entire system for application-level optimizations and performance predictions. The top-most layer uses a performance model of the application itself to map response variables collected from the global view of the system onto suitable suggestion for performance tuning, e.g. changes of parameters that are performance relevant factors of the computation.

3.3 Problems Addressed in Inverted Middleware Framework

We consider a distributed system running an application code over an extended amount of time. Therefore, we have to cope with a large amount of performance related information, which we call response variable according to [10]. An exhaustive representation of all relevant performance data is rather expensive, in particular if the distributed system has a large number of nodes to be monitored. A huge amount of information would have to be exchanged within the inverted middleware layers leading to a high overhead. The amount of system information must be filtered. Instead of recording every event possible, some parameters must be captured by sampling. If samples are not taken frequently enough, it is impossible to make an accurate statement about the system, but if they are taken too frequently, the system is perturbed by floods of monitoring traffic. The **granularity of samples** must be a trade-off between accuracy of the system view and the cost for the system monitoring. Compression into a representation of minimal size and appropriate granularity is necessary. In our experiments, we refer to a local sample every one second.

The performance data (response variables) we collect from the nodes comprises:

- the number of instructions (i.e. user instructions $IC_{user}(j)$ and system instructions $IC_{sys}(j)$) on the coordinator and nodes CPUs. j ranges from 1 to $ns + 1$ where ns is the factor expressing the number of nodes.
- the number of sequential disk accesses ($seq(j)$) and non sequential disk accesses ($no_seq(j)$) to the disk of the coordinator and each node (j ranges from 1 to $ns + 1$).
- the average stride $avg_stride(j)$ for the disk access in each node and the coordinator. By average stride, we relate to the average movement of the disk head between two random accesses to a disk. In the LINUX operating system, sequential disk accesses show up as reads with a stride of two blocks (blk) while non sequential accesses show up as larger strides.

- the amount of traffic transferred over the network interconnect: (i.e. Fast Ethernet, Gigabit Ethernet). We distinguish between the amount of *bytes received* ($size_rec(j)$) by each node and the coordinator on network device, and the amount of *bytes sent* ($size_trans(j)$) from each node as well as from the coordinator on network device. Again, j ranges from 1 to $ns + 1$.

This information is sampled as a changing rate over the time of an execution.

One important point in the design and development of inverted middleware framework is how deeply the system-specific layer should be integrated with the operating system. Our best solution is to implement the system-specific layer of inverted middleware outside the kernel as a daemon. This renders monitoring slightly slower and less accurate than in a kernel implementation, but remains much easier to maintain with new version of the OS kernel (which happen rather frequently). The prototype is based on LINUX */proc file* mechanism to write performance data. However, the information available in the */proc file* are significantly extended. The inverted middleware framework uses the *hardware performance counters* of the Pentium processor that are made accessible through a library we have implemented ourselves. All information for performance analysis is gathered at every node of our distributed system in parallel. Our inverted middleware framework is responsible for putting together the information in a global view. The monitors in the nodes send the performance information immediately to the monitoring master. The **network overhead** for these frequent data transmissions is kept low using the UDP/IP protocols, which are known to work quite well in clusters of PCs where links are short, full crossbar switches are common and transmission errors are infrequent. Many programmers are still building distributed monitoring tools with TCP/IP involving retransmission and flow control. We learned that for the best sampling accuracy and for the least system disturbance it is better to use UDP/IP and deal with the loss of information. Such loss of messages can be treated like sampling errors. Moreover, the use of the UDP/IP protocol optimally fits our specific **notion of time** within the distributed system. Communicating monitoring data through TCP/IP would result in unwanted synchronization through acknowledge messages, which adds perturbation to our applications.

Maintaining a consistent notion of time in a distributed system is an important aspect of our inverted middleware framework. The monitoring master has to be able to patch together the performance data of the several nodes in a consistent manner with a global wall clock time scale. To address this issue, a consistent, single notion of time among the several nodes of the distributed system has to be maintained. To overcome this problem, the framework has to introduce some mechanisms of synchronization during the performance sampling without introducing additional monitoring intrusions. To cope with the problem of maintaining a global notion of time and at the same time reducing monitoring intrusions, we propose a notion of time based on accurate built-in cycle counters. We avoid unnecessary synchronizations through a highly precise synchronization at the start of the monitoring session and, as the execution progresses, relaxing to a loose synchronization [7] in which the monitoring tool synchronizes sampling by looking at the highly accurate cycle counters in the CPUs. The built-in cycle counters mechanism acts as *virtual barriers*. The cycle counters have to be delivered as timestamps in the performance packets containing the sample information sent to the monitoring master.

3.4 Performance Model

The application-specific layer of inverted middleware framework includes a performance model of the application that can translate the elementary response variables into high level answers to performance questions and performance predictions that are suitable for optimizing the user controlled factors of an application.

Our current model is a simple set of formulas which allow the estimation of resource usage in case of a transparent middleware characterized by efficient resource allocation and control.

Table 1 shows the set of equations making up the analytical model used for the estimates of total execution time based on data about the detailed resource usage. The model aggregates low level response variables as an effect of the factors chosen in Section 2.4. Since we are dealing with resource constraints, the entire table talks about max values of the time components given by the several nodes and the coordinator. Note that the time component of CPU usage is directly related to the amount of instructions on the nodes, the time for the disks usage is the sum of the time for sequential and non sequential accesses to the disks while the communication time depends on the communication rate and size both gathered through the inverted middleware framework in the master-slave setting.

3.5 Performance Optimization on the Application Layer

ORACLE provides many means to estimate and measure the costs for executing SQL operations. For instance, a number of internal system performance tables record how many resources have been consumed by the current session, e.g. the number of disk accesses, CPU consumption, or the number of requested locks. Our monitoring tool completes

resource	estimated time	variable factors	response variables
CPU	$\max_{j=1..ns+1} (CPI * (IC_{user}(j) + IC_{sys}(j)) * t_{clock_cycle}(j))$	$t_{clock_cycle}(j)$	$IC_{user}(j), IC_{sys}(j)$
disks	$\max_{j=1..ns+1} (t_{seq}(j) + t_{no_seq}(j))$ $t_{seq}(j) = \frac{2*blk*seq(j)}{disk_rate(j)}$ $t_{no_seq}(j) = (no_seq(j) * t_{avg_stride}(j))$ $t_{avg_stride}(j) = \frac{(blk*avg_stride(j))}{disk_size(j)} * t_{rand_access}$	$disk_rate(j)$ $disk_size(j)$ $t_{rand_access}(j)$	$seq(j)$ $no_seq(j)$ $t_{avg_stride}(j)$
network	$\max_{j=1..ns+1} (\frac{size_rec(j)+size_trans(j)}{rate_rec(j)+rate_trans(j)})$		$size_rec(j), size_trans(j)$ $rate_rec(j), rate_trans(j)$

Table 1. Set of equations making up the analytical model used for the estimates of total execution time based on data about the detailed resource usage

this information with aggregated performance measures of the nodes in the cluster (see Table 1). In contrast to the performance tables in ORACLE, we not only know who consumed how many resources during the execution of a query, but also when and where these resource consumptions occurred. An interesting question at this point is: how can we exploit this knowledge in order to optimize the application.

According to Figure 4, the application-specific layer of the inverted middleware should suggest optimizations or provide feedback for the application based on the performance data gathered by the distribution-specific layer and the system-specific layer. In the optimal case, the inverted middleware would be able to tune the application automatically such that changes of hardware or query characteristics would immediately lead to a better (hopefully optimal) configuration of the application. However, having a database as the middleware, automatic tuning of the application is not entirely feasible due to the complexity of the SQL interface. Commodity databases like ORACLE can only adjust some specific performance settings: e.g. ORACLE uses a cost-based query planer to find an optimal execution strategy taking statistical information about the data distribution and performance characteristics of the operating system into account. However, important aspects like physical design (what indexes, how much normalization, how much replication, how to cluster tables) or partitioning of data are far beyond of what commodity database systems are able to optimize. Therefore, we assume that the application-specific layer further incorporates some human experts which are able to tune the application based on the footprints generated by the monitoring tool.

Example: The query execution plan describes the algorithm that the database uses in order to evaluate the given SQL query. In most cases, one could apply a number of different plans to a single query. Choosing the optimal one, however, is not an easy task. Given the performance data of the lower levels of the inverted middleware framework, an expert could identify bottlenecks in resource allocation and could relate them to the operations performed by the database according to the execution plan. In order to remove these bottlenecks, the expert could suggest to create additional indexes or to partition data in a different way (among other possibilities). In Section 4, we give an example for such a scenario with the two partitioning schemes introduced in Section 2.

c

4 Performance Issues Studied using the Inverted Middleware Framework

In this chapter, we will look at three important performance issues including the workload characterization of the TPC-D queries, their scalability and the dependency on the network interconnect in a cluster of commodity PCs. With a rigorous performance analysis based on the use of the inverted middleware framework, we can present some solid explanations and remedies for the issues and problems found.

4.1 Workload Characterization of Parallel TPC-D

The TPC-D benchmark uses 17 different queries that fall into a few different categories. As database specialists, we would typically classify them into categories based on the number and the extend of join operations required to work through the relation tables affected by each query. Our current approach as cluster system architects and performance evaluation specialists is quite different; we propose an alternative classification of the queries based on the most critical machine resources used during their execution.

The several machine resources in our distributed system (local or remote) differ a lot in speed and availability. In our performance model that is part of the inverted middleware framework, we identify three different machine resources that can be a critical limit to faster execution of a query: the CPU usage, the disk usage and the network usage for inter-node communication in the case of parallel processing. We consider CPU, disk and network usages because of their

direct significance to the database applications. Moreover, we add a fourth category for queries whose performance can not be modeled accurately based on resource usage because revealing an obvious inefficiency in one of the DBMS layers. This approach and the related set of resources is certainly not limited to database applications. We successfully use a similar method for the analysis and modeling of parallel scientific codes [15].

A typical example of a query limited by CPU usage is query 1, a typical example of a query limited by disk usage is query 4. Query 3 exhibits an interesting dependency on the communication subsystem when executed on multiple nodes and is therefore a good example for a query limited by network usage. Finally, query 8 shows the limitation of our performance evaluation techniques since its resource usage appears to be totally inconclusive. Unlike in all other cases where the processed output of our monitoring tools and our analytic model explain the execution time within an average error of less than 10%,¹ in this case most of the execution time remains without direct allocation to machine resources in the system.

Figure 5.a (left) breaks down the usage time allocated to the specific machine resources by the inverted middleware

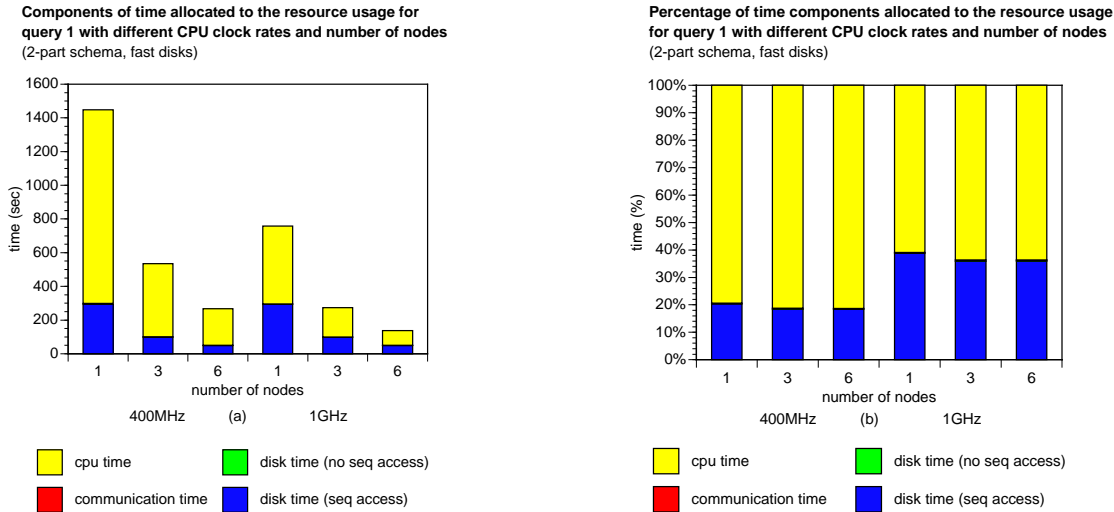


Figure 5. Components of the time allocated to the resource usage (i.e. CPU, disk and network usage) for query 1 (a) and related percentage of the components (b) with different CPU clock rates and number of nodes.

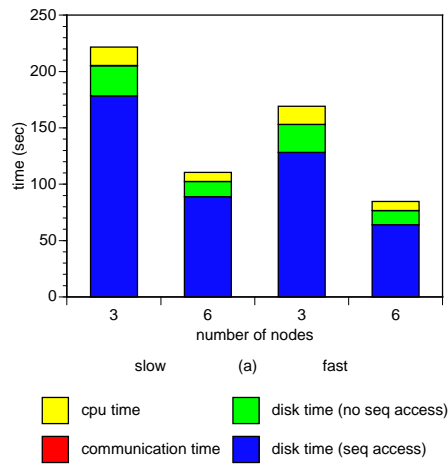
framework for query 1 with two different generations of PC clusters that differ in the CPU clock rates. The time components are four: CPU usage, sequential access and non sequential access to the disk subsystem and inter-node communication. Figure 5.b (right) shows the percentage of these components. The inverted middleware framework shows that the CPU usage accounts for 80% to 60% of the execution time, depending on the clock rates of the CPUs used. The component of the execution time attributed to CPU usage diminishes by a factor of 2.5 when we upgrade the CPUs from 400MHz to 1 GHz proving that query 1 is fully CPU limited. Looking at the scalability with a growing number of nodes, we state that the fractions of execution times stay the same and that the limiting factor remains the CPU even for larger clusters. The precise extent of scalability will be discussed in the next subsection.

Figure 6.a (left) breaks down the usage time allocated to the specific machine resources by the inverted middleware framework for query 4 with two different disk types used in the nodes of our clusters. Again, Figure 6.b (right) shows the percentage of these components. The data provided by the inverted middleware framework indicates that the disk usage accounts for more than 90% of the execution time in both cases of disk subsystems. The inverted middleware framework is further capable to distinguish sequential from non sequential (random) read disk accesses. As expected for OLAP workload, the emphasis is on sequential read access with a ratio of 90%/10% for the slower disks and 85%/15% for the faster disks. Going to faster disks does significantly decrease the execution time as can be seen in Figure 5.a. The fraction of execution time allocated to disk operations by the inverted middleware framework decreases with faster disks as expected. As for scalability with 1, 3 or 6 nodes, the fractions of CPU-, disk- and network usage stay the same for all distributed configurations.

We identified query 3 as a representative of a network limited query. Figure 7.a (left) shows the usage time allocated to the specific machine resources by the inverted middleware framework for query 3 (i.e. CPU, disk and network usage) with two different network interconnects commonly found in clusters of PCs (i.e. Fast Ethernet and Gigabit Ethernet).

¹We did calibrate the analytical model with numerous measurements and we do have the data to show its accuracy, but we had to omit the figures due to space constraints.

Components of time allocated to the resource usage for query 4 with different disk speeds and number of nodes (2-part schema, 1GHz)



Percentage of time components allocated to the resource usage for query 4 with different disk speeds and number of nodes (2-part schema, 1GHz)

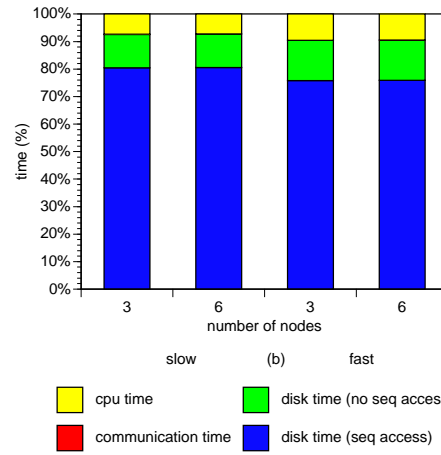
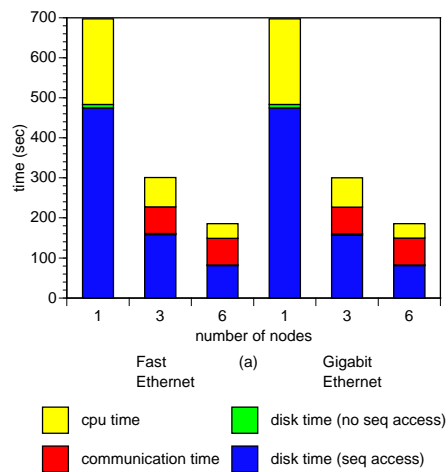


Figure 6. Components of the time allocated to the resource usage (i.e. CPU, disk and network usage) for query 4 (a) and related percentage of the components (b) with different disk speeds and number of nodes.

Components of time allocated to the resource usage for query 3 with different networks and number of nodes (2-part schema, fast disks)



Percentage of time components allocated to the resource usage for query 1 with different networks and number of nodes (2-part schema, fast disks)

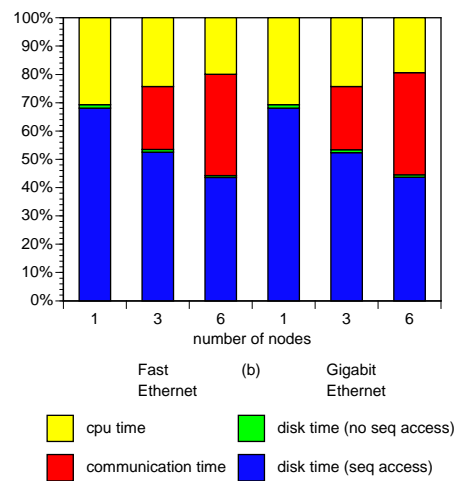


Figure 7. Components of the time allocated to the resource usage (i.e. CPU, disk and network usage) for query 3 (a) and related percentage of the components (b) with different networks and number of nodes.

As expected, there is no inter-node communication in the uniprocessor case (1 processor) and the communication becomes only visible as we distribute the workload to multiple nodes (3 and 6 nodes) in the PC cluster. The distribution of the resources without the network is about 30% CPU and 70% disk and stays that way for larger clusters pointing at almost linear scalability for the CPU and disk components and at the good explanation of the non scalability by the communication work. Surprisingly, there is no improvement with the addition of Gigabit Ethernet that is 10 times faster than Fast Ethernet. The scalability and the impact of the network will be discussed below - for now we just state the evidence that query 3 is a network limited query in parallel OLAP on clusters of PCs.

We classify the queries whose total usage time allocated to the specific machine resources by the inverted middleware framework for is much shorter than the total run-time (less than 15% in average) as DBMS inefficient or DBMS-dependent. The inverted middleware framework reports that for such queries, none of the monitored resources are intensively used. The lack of a critical machine resource indicates an internal problem in the DBMS and the system seems to be capable to execute such a query much faster. Certainly, there might be an additional non-monitored

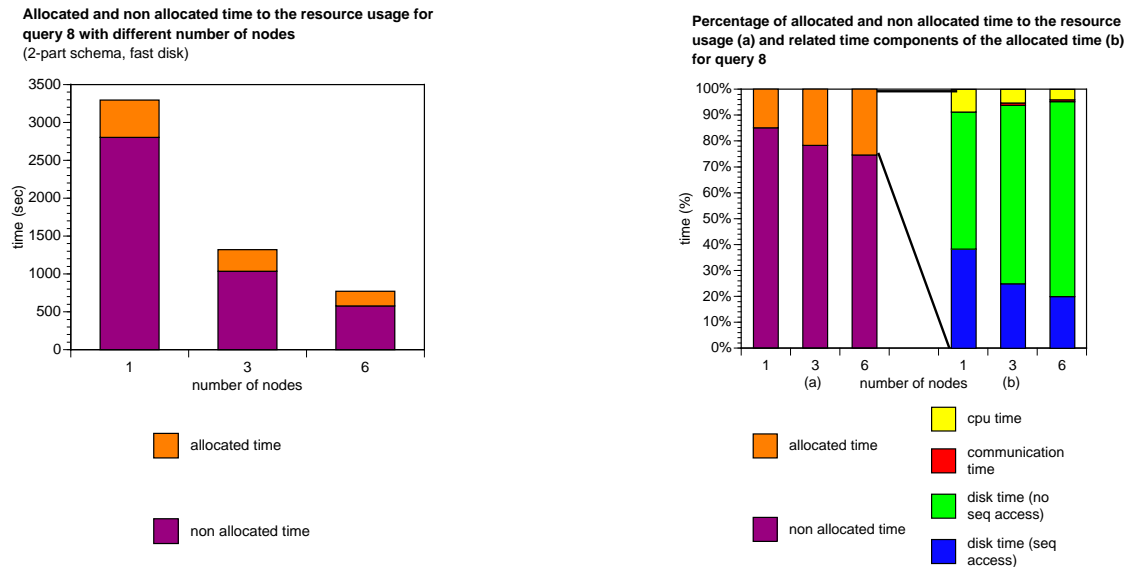


Figure 8. Execution time component allocated to the use of specific machine resources and time that can not be allocated by the inverted middleware framework in query 8. The percentages are further broken down into components of the allocatable part by the inverted middleware framework.

resource or a combination of resources that could be the bottleneck, but we did not find any and it remains unlikely that the entire set of monitoring tools of a modern operating system would not record any indication. In parallel scientific codes, we found such behaviors due to load imbalance and due to ill fated synchronization algorithms [15]. Query 8 is characterized by such a behavior. Figure 8.a (left), indicates the time allocated to the resource usage by the inverted middleware framework with the non allocated time for this query. We can see that the phenomenon is present to the same extent in the centralized case (1 processor) as well as in the parallel cases (3 or 6 nodes). The time spent based on the profiled workload of the CPU, disks and network covers less than 20% for the query running on a single node and increases only slightly to over the 25% for six nodes as indicated in Figure 8.b (right figure first three bars). A further breakdown of the time explained by the inverted middleware framework in query 8 is a bit inconclusive. The part of query 8 we can explain seems to be disk limited with a large emphasis of random disk accesses. It also seems that the sequential accesses and the CPU usage does scale with a larger number of nodes while the non sequential disk accesses do not scale (right figure last three bars). Such a situation indicates a high potential of optimization through tuning parameters of the application code or the DBMS. In fact, we have access to a few intrinsic parameters of the system that affect the performance. The upper DBMS layer that does the distribution of the relation tables of TPC-D accepts hints about a more or less aggressive strategy for partitioning vs. replication of the relation tables in TPC-D. We therefore used our framework to investigate the response to such parameter variation. The inverted middleware framework with the collected performance monitoring information combined with an analytical model of resource usage permits to characterize the workload of TPC-D OLAP application more closely through a classification of the queries according to their individual resource usage. With our framework, we can classify the queries as CPU limited, disk limited, communication limited or inefficient due to DBMS limitations.

4.2 Scalability of Parallel TPC-D in a Cluster of PCs

Figure 5.a and Figure 5.b show that CPU usage is the most important component in the processing of query 1 and that this component scales almost perfectly with a growing number of nodes involved in a parallel processing of this query. The second most important resource limitation is disk usage. Since we use a shared-nothing architecture to distribute the workload, the number of disks increases with the number of nodes involved. The inter-node communication stays negligible in this query since very small amount of data is transferred. Figure 6.a and Figure 6.b confirm a similar picture for query 4. Processing this query results in heavy on disk usage. Again this is invariant to changes in the number of nodes because the total disk performance scales with an increasing number of nodes. The second most used resource is CPU usage which scales perfectly to larger systems.

For query 3, the network usage does no longer scale nicely with the increase of the number of slaves, while the other resources (i.e. CPU usage and sequential access to disk) have good scalability (see Figure 7.a and Figure 7.b). The

reason for the limited scalability is a growing percentage of execution time due to inter-node communication with 3 and 6 nodes involved. Furthermore, the time components scale exactly in the same way for 1000 BaseT and 100 BaseT, and therefore, the loss of scalability seems to be independent of the strength of the interconnection network.

The amount of communication required to process a parallel query is not necessary a reason for bad scalability. In our clusters of PCs, the nodes are interconnected with a full crossbar switch and therefore, the network resources available to the parallel processing of OLAP workload could actually grow linearly with the number of processing nodes involved. Furthermore, the total amount of communication data reported by the monitors and collected by the inverted middleware framework is fairly small and does not point directly at a bottleneck. Even a slow, shared medium Fast Ethernet (hub) would be able to provide the necessary raw bandwidth to move the data. Therefore, a proper communication performance study requires more than the total amount of data transferred.

The global sampling of the communication data by means of the inverted middleware framework is accurately synchronized by a virtual wall clock time and permits to identify bursty and unbalanced network traffic in the parallel application.

Communication speed of query 3 on coordinator and each of three/six processing nodes connected by Gigabit Ethernet

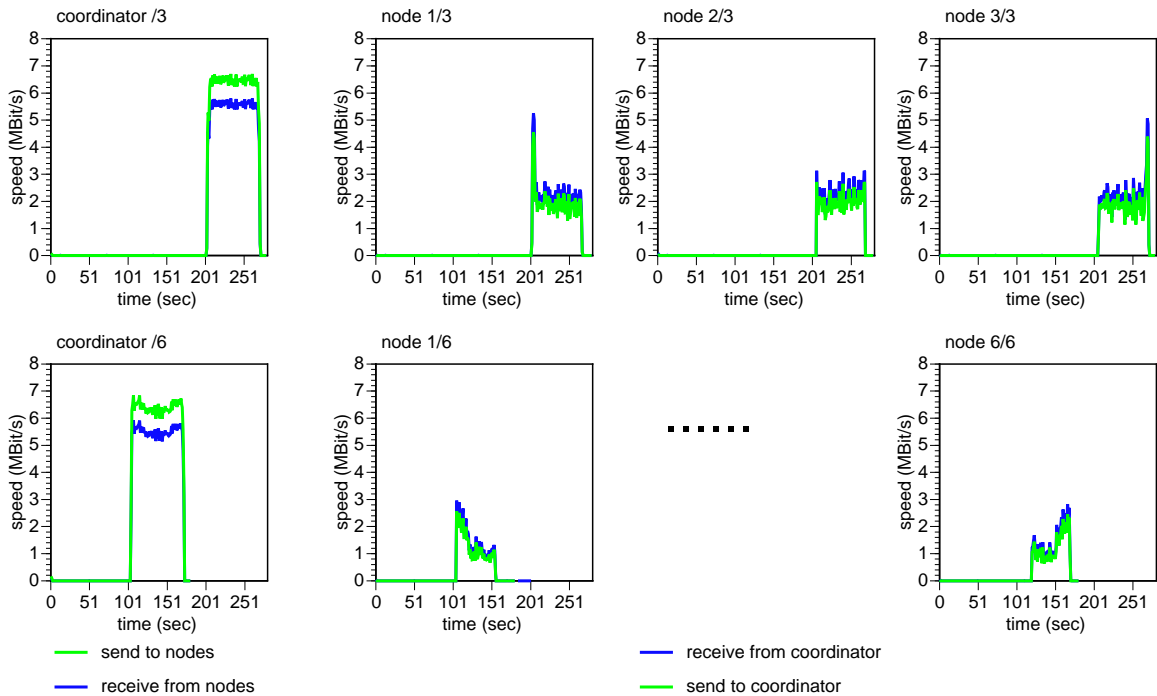


Figure 9. Communication activity during the the processing of query 3 for three/six nodes on the coordinator node (left) and on each of the three/six processor nodes (right).

Figure 9 on the left shows the communication activity on the coordinator node for query 3 while coordinating the processing of the query with three/six nodes (coordinator/3 and coordinator/6). At the same time the figure displays on the right the communication activity for the same query on the three/six individual processing nodes (node n/3 and node n/6). The data in the pictures are gathered at the distribution-specific layer by the inverted middleware framework.

In the top-right charts we consider the communication on each of the three nodes for the processing of the query with three nodes, while in the bottom-right charts we look at the communication on the first and last nodes for the processing of the query with six nodes. The charts related to the processing on three nodes (top) depict the transferred number of bytes per second as a function of time where 0s denotes the start of query execution and 280s the end of query execution for three nodes. The same query finishes earlier i.e. roughly at time 170s when the work is distributed among six nodes (see charts on the bottom). In the left charts we look at the communication work on the coordinator and distinguish between “sends to the nodes” and “receives from the nodes”. In the right charts we distinguish between “sends to the coordinator” and “receives from the coordinator”. The graphs show a network traffic that is quite bursty

and imbalance, most of the traffic is concentrated on the coordinator, while the processing nodes clearly communicate at roughly one third or one sixth of the speed of the coordinator.

The graphs refuted our initial assumption that communication is highly asymmetrical due to the algorithm in TP-Lite that processes partial queries on each node and finally gathers the results in the coordinator. In reality, the communication traffic is quite symmetrical and during the data transfer the coordinator sends almost as many bytes of request as the nodes send for its answers. Furthermore, the peak communication rates in the coordinator is determined to be just 6.4 MBit/s in a network that can sustain one Gigabit/s (i.e. 160 times more) under good conditions.

The inverted middleware framework provides precise allocation of total execution time to each resource usage and offers the possibility to record resource usage for arbitrary sampling intervals. This helps to find and to isolate the cause for the loss of scalability in the processing of non scalable queries. An accurate sampling of all communication activity over the entire execution time is required to discover and deal with performance limitation due to bursts and hot spots in the communication patterns.

4.3 Performance Impacts of the Network Interconnect Speed

During the evaluation, procurement and installation of a large cluster for database research, we studied the question of the least interconnect technology that is sufficient for the planned work on parallel databases. For a Beowulf type cluster, a commodity Fast Ethernet switch would be sufficient. For enhanced clusters, we would opt for a Fast Ethernet switch that provides full bisection bandwidth (or non blocking ports in networking terminology). For an advanced cluster, we consider high performance interconnects like Gigabit Ethernet or Myrinet. Depending on the performance networking, a cluster node can cost between 100 and 1000 per node installed. Our study should answer the question of whether clusters with higher interconnect speeds are worthwhile for parallel OLAP processing or not. For our measurements, we equipped our cluster of PCs with Gigabit Ethernet and a fully non-blocking 16 port switch in addition to non-blocking Fast Ethernet.

The study of the benefit of a higher speed interconnect is limited to query 3, an example query that obviously becomes communication bound in the parallel configurations of 3 or 6 nodes. As seen in the Figure 9, the communication activity is bursty and takes place towards the end of the query. Furthermore, the communication takes place between the coordinator and all the nodes at the same time (two peaks, one for the 3 node experiment and another for the 6 node experiment), but most importantly the speed of communication is just 6-7 MBit/s which is two order of magnitudes below the 100 MBit/s that is possible with Fast Ethernet and three order of magnitudes slower than Gigabit Ethernet. It is not visible from the chart if the inefficiency is due to further burstyness or packet collisions within the sampling interval or if there is software reason for this bad performance and a separate experiment must be used to verify if there is any benefit at all of using a faster networking technology (which would also handle bursty traffic faster).

Figure 10.a and Figure 10.b depict the trace of communication activity on a node (transfers from and to a coordinator)

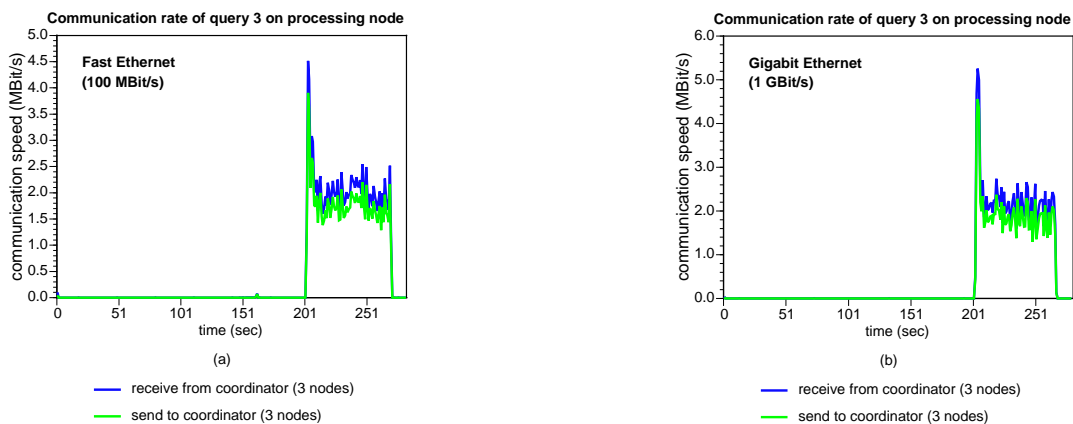


Figure 10. Slave communication rates of query 3 for Gigabit Ethernet (a) and for Fast Ethernet (b) on a cluster with three slaves and CPUs with 1 GHz clock rate.

for query 3 in a system of three nodes of cluster connected by Fast Ethernet and Gigabit Ethernet respectively. The two similar traces with completely equal peak and average performance prove that communication behavior is exactly the same for a 100 MBit and a 1 GBit network and that there is no benefit at all of purchasing a network with a higher throughput. Although Fast Ethernet and Gigabit Ethernet differ significantly in the throughput available, the latency and overhead for small packets are not that different.

In our first approach for a parallel processing of OLAP database workloads, we used ORACLE database links to ship data from the nodes to the coordinator in the cluster. These database links are based on the ORACLE NET8 transport protocol and use 2PC (two phase commit) to ensure transactional guarantees. Our experiments clearly show that ORACLE database links are not efficient enough to work on large communication loads since they require a lot of slow synchronization. So finally, the latency of the network or the processing overhead in the DBMS might become the cause of the slow-down of the communication while the bandwidth and the throughput has no longer any influence on the performance. Moving the application on low latency networks (like Myrinet) could still miss to solve the communication problems because of the high overheads in the DBMS. Even if it helps a bit, it is not expected to be cost effective since the bandwidth of Myrinet yet remains unused most of the time.

Again, the inverted middleware framework with its accurate sampling and its global representation of resource usage in the distributed system permits an in-depth analysis of communication problems and properly explains the inefficient use of the network while processing queries like query 3 in TPC-D. With the inverted middleware framework, we are finally able to explain why the TP-Lite approach behaved so much worse than the other approaches in [1].

5 Architectural Implications of the Resource Usage Observed

Following the classification of the queries reported in the previous section, we can calculate the CPI value from the measured performance data for the different classes of TPC-D queries and find numbers that are quite interesting. CPU limited queries are indicative for the basic CPI value of OLAP workloads, but since we are on a parallel and distributed system we are experiencing several modifications to those basic values. If the nodes processing a query spend a lot of time waiting for “peripherals” such as disks (SCSI cards) or the network (Ethernet cards), the cycles are not attributed to processing instructions but to idle loops in the operating system kernel. Therefore, the CPI is inflated as soon as disk or network do most of the work and are frequently waited for.

For strictly CPU limited queries (e.g. query 1), we measure a CPI of about 1.5 on the older and the newer types of nodes, regardless of clock rate of either 400MHz or 1GHz. Researchers of UCB and UIUC looked at CPI figures for TPC-C and TPC-D on large symmetric shared- memory multiprocessors (SMPs). We are not aware of any studies that deal with massively parallel supercomputer nodes or entire clusters of PCs up to this date.

Keeton *et al.* [11] look at the CPU usage of the TPC-C Benchmark using Informix DBMS running on an Quad Pentium Pro, shared-memory, shared-disk configuration. The basic CPI including memory usage is 2.90, which is much higher than the CPI values found in computational benchmarks like SPEC95. Those values are for OLTP and so the difference to our values of 1.4 to 1.5 for OLAP is not surprising. Cao *et al.* [3, 2] look at the characterization of the TPC-D benchmark with Microsoft’s SQL Server and Windows NT on top of SMP server with Quad Pentium Pro, shared-memory, shared-disk configuration. They report a basic CPI of 1.27. This slightly lower figure is for a high end SMP with moderate parallelism, while our work deals with cluster nodes in a massively parallel setting.

We carefully study the dependency the CPI on the different queries, i.e. the workload. For disk limited queries (e.g. query 4), the CPU that is frequently waiting for the disk accesses inflates the overall CPI considerably. The CPI measured on clusters with the faster disks is about 10 while the CPI for slower disks is about 15. Communication limited queries are unique for parallel and distributed computing platforms that rely on networking technology for inter-node communication. The resulting CPI values of 3 to 4 reflect the idle loops encountered for waiting for data from the network. Finally, the DBMS software limited queries show extremely high CPI values that can only be interpreted by a failure of the monitoring environment to track down the resource usage properly. Query 8 is characterized by a CPI of about 90.

The good values of the CPI for CPU and disk limited queries indicates that a cluster architect should focus on purchasing fast disks and purchasing fast CPU. This is consistent with the believe of many processor architects that see a major driver for ever better higher MIPS ratings in microprocessor with higher clock rates and more ILP in those database workloads.

The performance picture of our study turned out to be devastating to high speed communication in clusters of PCs, which was a primary target of the clusters built in our CoPs (Clusters of PCs) project.

Distributing OLAP queries to a large number of nodes with partitioned data can result in large amounts of inter-node communication for certain queries. Unfortunately, the communication is completely dominated by software overheads within the DBMS and adding a faster network does not help. More precisely neither better latencies nor better bandwidth results in much improvement. The precise analysis of the network usage over time indicates that there is a bottleneck due communication, which takes place between the coordinator and all the nodes at the same time. The bottleneck can be removed by improving the scheduling of the partial queries and by balancing of the load communication more carefully. As expected for a database application, the performance of the disks is highly critical to overall performance. Using a distributed file-system with RAID capability based on remote disk accesses may

improve the OLAP performance beyond the limitations of simple parallelization.

6 Conclusion

Performance analysis in parallel databases running on clusters of commodity PCs remains a highly difficult task, since we are still lacking many of the tools and instrumentation that could give us the performance data we need to understand parallel- and distributed systems executing OLAP workloads in standard DBMSs.

As a contribution to address this important problem, we present a framework, the inverted middleware framework, for collecting and filtering the raw performance data given by the operating system in a distributed high performance database system built from common commodity PCs and commercial DBMSs. Our framework combines the usual monitoring instruments at the operating system level with an effective strategy for collection and interpretation of this monitoring information at the overall systems level. Together with some simple analytical model of overall resource usage, we can assess the high level performance indicators at a level of abstraction that is appropriate for an application writer. In particular, our system is able to give some new insights about the use of specific machine resources in the system. In the existing monitoring system, we capture CPU usage (CPI), two kinds of disk usage (sequential and non-sequential), communication system usage and the work for memory system usage is currently progress. Unlike the built-in performance monitors of most database management systems our operating system based monitoring solution is fully operational in a cluster setting with distributed processing and accounts for accumulated resource usage as well as for a time-variant resource usage by sampling and collecting performance data at arbitrary intervals. Information about peak resource usage and temporary bottlenecks can be derived from the time-variant resource usage traces.

To demonstrate the viability of our approach and to draw proper architectural conclusion about the construction of future PC clusters for high performance database systems, we use our framework to investigate parallel high performance databases executing OLAP workloads on clusters of commodity PCs. We successfully analyze a highly complex hardware-software system that relies primarily on standard hardware and commercial software components that were provided to us by a database research group. The configuration evaluated includes an ORACLE DBMS for SQL processing at each node, LINUX operating system to manage the node's resources as well as different Ethernet switches to take care of inter-node communication. The experimental software shell to distribute the queries to different nodes (TP-Lite) is taken from a database research project.

In our measurement effort, we execute a 10GB TPC-D benchmark and to characterize the workload precisely according to the resource usage encountered in the cluster. We gain significant insight into the question of the impact of high performance networking to this sort of application. As a most interesting result, we can classify the queries of TPC-D into CPU limited, disk limited or communication limited queries including a fourth class that remains inefficient due to DBMS software limitations. Knowing the critical resource for each query, we can properly explain scalability (or lack thereof) for each query on cluster with three and six nodes. Based on a simple analytic model that factors the total execution time into parts attributed to each resource, we can estimate the scalability for large number of nodes. To our biggest surprise, the inter-node communication is not the key to better scalability unless the communication facilities of standard DBMS software are drastically improved. In particular, we have shown that the Gigabit high speed network in our high end cluster of PCs does not improve scalability of the application due to inefficiencies in the commercial DBMS software that is used in conjunction with TP-Lite.

References

- [1] K. Böhm, T. Grabs, U. Röhm, and H.-J. Schek. Evaluating the Coordination Overhead of Synchronous Replica Maintenance in a Cluster of Databases. In A. Bode, W. Karl T. Ludwig, and R. Wismüller, editors, *Proceedings of the 6th International Euro-Par Conference*, Lecture Notes in Computer Science, pages 435–444, Munich, Germany, August 2000. Springer.
- [2] Q. Cao, J. Torrellas, and H. Jagadish. Unified Fine-Granularity Buffering of Index and Data: Approach and Implementation. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'00)*, pages 175–186, Washington - Brussels - Tokyo, September 2000. IEEE.
- [3] Q. Cao, P. Trancoso, J.-L. Larriba-Pey, J. Torrellas, R. Knighten, and Y. Won. Detailed Characterization of a Quad Pentium Pro Server Running TPC-D. In *International Conference on Computer Design (ICCD '99)*, pages 108–117, Washington - Brussels - Tokyo, October 1999. IEEE.
- [4] P. J. Courtois. Decomposability—queueing and computer system applications, 1977.

- [5] P. J. Courtois. On time and space decomposition of complex structures. *Communications of the ACM*, 28(6), June 1985.
- [6] C. Dye. *Oracle Distirbuted Systems*. 1st Edition. O'Reilly, 1999.
- [7] G.C. Fox. What have we learnt from using real parallel machines to solve real problems? In *Proceedings of the 3rd conference on Hypercube concurrent computers and applications*, pages Vol.2 pp 897–955, Pasadena, CA, USA, Jan 1988.
- [8] T. Grabs, K. Böhm, and H.-J. Schek. High-level Parallelisation in a Database Cluster: a Feasibility Study Using Document Services. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001. IEEE Computer Society.
- [9] M. Gurry. *Oracle SQL Tuning Pocket Reference*. O'Reilly, 2001.
- [10] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley Professional Computing, 1996.
- [11] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker. Performance characterization of a Quad Pentium Pro SMP using OLTP Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-98)*, volume 26,3 of *ACM Computer Architecture News*, pages 15–26, New York, June 27–July 1 1998. ACM Press.
- [12] Oracle Corporation. Oracle8. <http://www.oracle.com/>, 1997.
- [13] U. Röhm, K. Böhm, and H.-J. Schek. OLAP Query Routing and Physical Design in a Database Cluster. In *Proceedings of the International Conference on Extending Database Technology*, Konstanz, Germany, March 2000.
- [14] U. Röhm, K. Böhm, and H.-J. Schek. Cache-Aware Query Routing in a Cluster of Databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2001.
- [15] M. Tauffer and T. Stricker. Accurate Performance Evaluation, Modelling and Prediction of a Message Passing Simulation Code based on Middleware. In *Proceeding of the SC98 Conference*, Orlando, FL, USA, Nov 7-13 1998.
- [16] The Transaction Processing Performance Council. TPC Benchmark C Standard Specification Revision 3.4. <http://www.tpc.org>, 1998.
- [17] The Transaction Processing Performance Council. TPC Benchmark D Standard Specification Revision 1.3.1. <http://www.tpc.org>, 1998.