Abstract

Dynamic load balancing shall optimize the resource utilization in parallel systems and computer networks in order to reduce the application response times. While simple schemes are mathematically tractable and can increase resource utilization for certain applications, while implemented remote execution facilities can avoid cpu overload peaks in computing clusters, load balancing schemes for realistic systems and complex load profiles, especially for database management systems, need to consider data and communication. This paper explains the problem and potential benefits of dynamic load balancing, emphasizing on the exploitation of data affinities and consideration of communication cost. The sophisticated techniques developed in the HiCon system are presented in detail and validated by several real measurements from parallel and distributed database processing.

Table of Contents

Dynamic Load Balancing for Parallel Database Processing

Wolfgang Becker

Faculty Report No. 1997 / 08

Institute of Parallel and Distributed High-Performance Systems
(IPVR), University of Stuttgart
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
Phone: +49 711 7816 411
Email wbecker@informatik.uni-stuttgart.de

1 Dynamic Load Balancing for Real Systems and Applications

The basic motivation to investigate and realize load balancing facilities comes from the very simple experience that several applications in a parallel system or computer network usually do not exploit the system very good. Instead, many compute nodes are idle while others are overloaded. The most well-known environments are workstation networks. Without buying new resources, load balancing as a system service can offer much more resources and better response times to all users, and can effectively avoid disasters that sometimes can arise from bad load distribution.

Many approaches try to equalize the number of processes on the compute nodes or equalize the task queue lengths at the nodes in the network - actually most of them just avoid that nodes run empty while others are busy. Load balancing becomes more complicated as the system consists of heterogeneous nodes, i.e. faster and slower processors, different amount of main memory and different number of cpus per node (SMP). Similarly, most real applications and tasks in the system are heterogeneous. They consume different amounts of resources, i.e. run short or longer times and use different percentage of the cpu time. Further, reality usually brings a mix of several, maybe parallelized, applications. Additionally, the arrival of tasks is not coordinated or planned, so load balancing has to cope with changing, unpredictable load profiles. Another complication is that parallel applications are not just sets of independent tasks, but the tasks are structured by precedence relationships. Load balancing should favor tasks on critical paths and tasks that entail large parallelism to reduce application response time and increase resource utilization. Other effects like context switch overhead or paging delays due to active memory usage severely influence the system performance, so load balancing should limit the overall system load appropriately. Load balancing should further be adaptive, i.e. identify the currently important performance factors, create estimations by profiling the system behavior and find the trade-off between load balancing overhead and improvement.

Although these requirements already make real world load balancing quite difficult, the following section explains that applications in various styles work on shared data and exchange data, which also has significant influence on the system resource utilization and application performance. Hence, load balancing should consider data, communication and networks appropriately.

2 Data Access and Communication Influence Application Performance

Data and communication are not of primary interest to the load balancing service. Load balancing is focussed on application response time and system resource utilization, measured by task elapsed times, application elapsed times, cpu utilizations, paging rates and perhaps disk and network utilization.

But applications often access remote data in form of network file sharing or global distributed databases. Access to data which do not reside on the local node, forces the application to request the data and wait for their arrival, and induces network load. The delay time could be used by other processes on the node, the network utilization should be coordinated with other network users to avoid congestion. If remote data access is performed by issuing remote procedure calls and make the data owner node execute the operations, then additional cpu load at this node will arise and other tasks on this node may be slowed down.

Application tasks usually pass intermediate results to their successor tasks, whether they send messages, or pass results to the client which passes them with the next server calls, or they store them into temporary files. This data flow also causes communication delays and network load. Tasks in parallel applications exchange synchronization messages to control the overall execution. These messages implement precedence relationships between tasks and can serve for result collection and aggregation. They also cause delays due to message latencies and network load.

Cooperation in parallel applications can, depending on the programming paradigm, take place by sending data to the task that needs them, or by accessing a virtual shared memory which locates the requested data and sends them to the requester. This data communication infers delays, network load, cpu load at remote node and search overhead to locate the data, which also yields computations and messages.

Finally, load balancing can cause data communication for process or application migration. Migrating a running process entails sending its data segment and maybe its code segment. This can be done in one strike or by demand paging, and increases the migration cost by additional delays and network load. Similarly, when migrating whole running (parallel) applications, whether moving all processes or starting the next

processing phase on other nodes, the communication of the application's active data set yields additional communication delays and network load.

Depending on network throughput and latency, and on the impact of the communication or data access speed on the application performance, load balancing can obtain large improvements by considering this.

3 How Can Load Balancing Consider Data and Communication

The relevant effects from data access and communication for a load balancing service are the additional delays, the reduced cpu utilization and the additional network utilization or network congestion. Now, load balancing needs a simple model and few critical factors to measure and to decide upon. The critical factors can be obtained by static data about the system resources and application behavior, dynamically by measuring the system and application behavior and by getting recent profile estimations from the applications. On system level, network delays (process to process latency per short message), network throughputs (bytes of continuous data streams per second) and the network topology (which nodes are directly connected, routing, hardware multicasting support) are of interest. One level above, the application data distribution within the system, average remote data access response times, the network utilization by communications or remote data access operations, and the cpu utilization and consumption for remote data operations are relevant. On application level, the message frequencies, the communication topology and the message sizes can be profiled or pre-estimated, and data reference patterns (record or data range names, read/write probability, repetition degree of access patterns) can be obtained by observation or from direct application hints. Finally, general estimations about applications' sensitivity to communication power (throughput, latency, temporary deviations) are interesting as guidelines for adaptive load balancing, to decide which factors are currently relevant.

Instead of developing a general approach to evaluate these informations for load balancing decisions, a brief review of existing techniques will be followed by a closer look at one of them, the HiCon approach:

- In general, there are some load balancing schemes which avoid communication, limit communication to the network capacity, or find the trade-off between increased resource usage by task distribution and parallel execution and the loss from communication delay and overhead. They do it by placing communicating tasks on the same node or cluster, by placing the tasks to where their data reside or by migrating / replicating the data to where the tasks are or going to be.
- A common static load balancing technique [10][11][31] is to match a task graph with a network graph. The task graph consists of the application tasks as nodes and communication relationships as edges, weighted by the communication intensity or frequency or amount. The network graph's nodes represent processors (compute nodes) and the edges direct communication lines, weighted by the throughput of the lines. Tasks can be assigned to processors by recursively partitioning the task and network graphs and always assign one task graph part to one network graph part. Both graphs are partitioned so that heavy weight linked nodes remain together in one partition. Matching obeys boundary conditions like maximum node load, maximum node load skew, maximum network line utilization etc. Alternatively, the graph matching can be done immediately by solving the corresponding optimization problem [29][35][12][36][25]. Recent research on supercomputers with extreme low latency networks [33], however, show that it can be profitable to distribute communicating tasks, because message passing between nodes is faster than the context switch from sender to receiver process within one node.
- Static load balancing methods that assign groups of tasks with precedence relationships to a set of processors in order to minimize the finish time of the last task, can be extended to consider also the data flow between the tasks [2][13][15][22][26]. Without looking at data, these algorithms place the largest tasks or the tasks on the critical path onto the fastest processor or onto the next available processor. Data flow is included in the model by adding a delay to the start of each task (the amount of time depends on the network speed and the data volume), if the task gets data from a predecessor that executed on another node. Hence, successor tasks are preferably placed on the same node as their predecessors.
- Transaction routing schemes place transactions to the database server that has the largest portion of the required data [3][38]. Therefore, load balancing knows the location of the database partitions and the average data reference patterns of the most important transaction types. Besides distributing the computational load by equalizing the number of database requests per unit of time between the proces-

sors, the transactions may not be distributed by chance but - as far as possible without compute load skews - to the node where most of the addressed database partitions reside. More sophisticated methods have an explicit load model including the cost of remote database requests (subtransactions), and send each query to the node where the sum of compute times, message delays and remote compute times is minimal. Further, the cost factors of communication, execution and general additional overhead per remote execution can be adjusted by tracing the system behavior and periodic regression analysis.

- Similarly, dynamic load balancing can receive data reference pattern estimations per task from the client [8], can generate / correct these estimations by observing the real access profiles [7], can dynamically observe the system data and copy distribution, and can observe the real data movement costs for remote accesses. Using this information, the expected time for data access on each available server can be estimated and compared.
- Dynamic load balancing schemes can estimate the data communication portion of tasks or task types by tracing the cpu utilization divided by number of running processes or observing the recent cpu usage of tasks [33]. Tasks that do not fully need their cpu share, either perform I/O or communicate a certain amount of time, and hence their nodes can take more concurrent tasks.
- While the techniques above served for task placement decisions, load balancing can also consider data communication cost for task migration decisions [32]. So, selecting a suitable process for migration from an overload node can take the one with the smallest data segment or the smallest number of accessed local files. Thus, the data communication entailed by the migration is kept small.

4 Load Balancing Runtime Support for Parallel Database Systems

Today's general load balancing systems are usually queueing and remote execution facilities or process migration facilities. They ignore the existence of data or do not consider them in a sophistication that makes them applicable for parallel database systems. So it is interesting to describe the architecture of one of the major general systems that can cope with parallel database operation, the HiCon load balancing model [8]. HiCon load balancing is application independent, but was designed especially for data intensive and cooperating complex parallel applications. The main concepts emphasize the key issues for dynamic load balancing systems:

- *Centralized and Distributed Structure*. Centralized load balancing (often called 'global') provides one central agent for collecting load information and for decision making [14][16][18][28]. It has several strong advantages: The measurement informations and predictions consistently reside in one place and may not be distributed or replicated among the system which would cause message traffic and could lead to outdated, inconsistent informations and contradictory load balancing decisions. Centralized load balancing can exploit the global knowledge about system and application behavior and can utilize sophisticated strategies. Centralized load balancing usually comes along with central task queueing. This allows for load control, because tasks can be queued until enough resources are available. Otherwise, heavy load situations saturate the system and reduce the throughput due to process switching, memory paging and network congestion. Central queueing also enables late decisions, because tasks may not be assigned when they are born but when they are removed from the queue for immediate execution. Central load balancing can avoid load imbalances before they arise, whereas distributed load balancing always has tasks waiting / running at the node where they originated and try to defeat the skews.

  Distributed load balancing (often called 'decentralized') becomes necessary for large parallel and distributed systems. These concepts try to keep load balancing efforts (resource consumption as well as delays) constant regardless to the system size, prevent single points of failure by local decision autonomy and reduce information and task exchange between nodes by simple, restrictive load balancing policies [4][23][27][30].

  When comparing centralized and distributed approaches, it must be noted that a stable heavy load by long running tasks is no problem for centralized approaches, because there are not many decisions. Task arrival bursts are critical because they cause much work for the centralized component. But, without centralized load balancing they generate severe load skews because they usually arise from one or few sources, e.g. parallel applications. So, centralized load balancing itself is overload but prevents load skews while distributed techniques will encounter a long period of slowly stepwise equalizing the skews.

Ideally, load balancing should have a mixed structure, i.e. a centralized agent per closely coupled cluster and distributed cooperation between neighbored clusters [6][19][20][39] or even a hierarchical structure [1][37]. The HiCon load balancing system provides one central component per cluster while neighbored clusters cooperate decentralized. Central task queueing takes place within a cluster; between clusters whole executing applications (task groups) are migrated if a significant load skew is detected. Between cluster, the HiCon system does not migrate executing tasks, but just routes all following tasks of this application to the neighbor cluster.

- *Client / Server Execution Model*. The HiCon approach provides a load balancing service for client - server processing. Tasks are issued by clients and are queued and assigned by the load balancing component to appropriate server processes of the respective server class. The concept supports mixed parallel applications and multiuser operation.

  It is important that the execution model for the load balancing decisions matches the execution profiles of the target application area. It should restrict itself to few relevant characteristics and measurement entities. The client - server execution model has been established as the basic processing concept for database management systems and for almost all large, distributed applications. Especially in database processing, fixed SPMD-style problem decomposition onto processors is not flexible enough, because it does not work well for large and small problems, for parallel queries and multiuser operation or for complex structured queries with unpredictable intermediate result sizes. Server class calls, however, allow for a flexible dynamic task distribution within parallel and distributed systems, because each call may be assigned to an arbitrary server instance on any node.

  For load balancing purposes, client - server has the additional advantage, that server classes provide a natural grouping of tasks that show similar profiles. So, load balancing can observe - and later on predict - the execution profiles and data access patterns per server class. The decoupling of applications into discrete tasks (server class calls) is also a simplification for the load model, because the behavior of whole, complex structured and communicating processes is hard to model appropriately. The concept of statically allocated server instances (processes) allows for rather fine grained task distribution and high parallelization degree, because a remote execution does not necessarily incur process start and connection setup overhead.

- *Flexible Data Management System*. For parallel database processing, load balancing additionally needs to have an idea of data, i.e. a model of how tasks operate on data and how data can be accessed remotely or can be moved or replicated and what resource consumption and delays come along with it. This was explained above.

  The approach of a virtual shared memory / data store fits very well into database processing: Within the HiCon system, tasks communicate by accessing global data items, either shared within an application or globally between all applications. The global data can be arbitrarily structured, volatile or permanent. No remote data access operations are performed, but the data items are migrated or replicated to the requesting task on demand. Therefore, a distributed runtime system supports transparent data partitioning / replication / migration, realizes a global database or virtual shared memory with changing data owner and probable owner search concepts, and provides consistent data access by locks and copy invalidation. The HiCon load balancing concept does not require this data management component. It just gets informations from it and knows about its facilities. Arbitrary database management or virtual shared memory components could be employed. It must be stated that none of today's commercial database systems provides dynamic data migration and replication in a comparable degree of flexibility. So, load balancing must know about the facilities of the data management system which the respective application uses. The important issues for load balancing are the efforts for data communication, the location of the data and the expected data access patterns of the application tasks.

  For illustration, the scheme of the runtime data management system is described which is currently used within the HiCon system. It is a general, flexible approach to management of logically common, physically distributed data. Data granularity is defined by the application. It can be few bytes, records in main memory or in a file or even whole files. The owner of a data record is the server who performed the latest update operation on it. This server is asked for copies and may grant ownership to other servers for updates. Copies are requested for read-only access and for updates the distributed copies are invalidated. This policy optimizes the data access cost, provided that the servers show a certain data reference locality: Repeated read access on local data copies is cheap and the number and distribution

of copies automatically adapts to the read / write proportion of the accesses. Load balancing tries to increase this access locality by assigning tasks to where the required data or copies of them reside. To ensure consistency, servers precede their data access by acquiring appropriate read / write locks.

The concept of cooperation via common data enables a quite flexible distribution of the tasks among the system, even for server classes that are not context free. For load balancing, the common data are the objects where it can include data affinity in its cost model. And this model comprises in a simple way communication between tasks, passing of intermediate results (pipelining) and access to common, global, maybe persistent data.

- *System Load Measurement and Load Pre-estimations*. It is important that load balancing acts dynamically by measuring the current system load. Especially in multiuser environments like OLTP there is no central a-priori planning but tasks arrive uncoordinated. For balancing large complex queries, dynamic load balancing is also important, because the actual amounts of data and computing efforts cannot be pre-estimated accurately. Further, load balancing should be adaptive for three reasons:

  1. Adaptive generation and correction of pre-estimations for task profiles and system load behavior. Dynamic load balancing decisions base on informations about application requirements and system utilization. These informations are often inaccurate or unavailable. Even load balancing concepts that make no explicit assumptions about the future task or system behavior but react on current measurements only, implicitly employ an extrapolation of zeroth degree, i.e. assume constant behavior. More sophisticated concepts can use load profile estimations from their own observations or from applications [17][21][24][32][34]. Comparing the previous estimations with the actual system behavior, load balancing can assess the accuracy and the value of the estimations and can correct them appropriately. For example, in the area of transaction routing [38] periodically perform regression analysis to determine the correlation between assumed and actually observed response times, and therefrom reduce the derivations for further routing decisions by correction factors.

  2. Regulation of difficult decision factors by feedback. Another weakness of dynamic load balancing is the vague execution model on which the decision algorithm is based. In real, parallel and distributed systems and large real applications it is extremely difficult to capture all effects and dependencies within a compact model. Because decisions have to be committed at runtime they must restrict to simple execution models. Hence, the correlation between the factors which load balancing considers (e.g. processor run queue lengths) and the overall throughput which is to be optimized, is not strong enough in all situations. The costs for remote data access, for example, depend on the size and complexity of the data, on the network utilization, on the lock wait times and on the utilization of the remote data owner - factors hard to merge into a compact formula. So, load balancing has to compare the actual effects of such decision factors in certain situations and globally adjust further decisions by appropriate correction factors, not knowing detailed reasons.

  3. Adaptive optimization of load balancing's cost - efficiency proportion. Load balancing profit is curtailed by its overhead, because it causes compute load, communication load and delays for information collection and decision making. Without adaption, load balancing assumes that its overhead is always appropriate and worthwhile which is not true for all situations and load profiles. Hence, load balancing must minimize its efforts, or even better, observe and regulate its cost - efficiency proportion. Decentralized load balancing, for example, should exchange less load informations in times of overall heavy system load, because there should not be many migrations anyway. Similarly, it should switch from the more expensive sender initiation to receiver initiation, i.e. not the many overloaded nodes should start searching for an underload node but the few idle nodes should pick up load from some overload colleague.

  The HiCon load balancing model employes a couple of adaption techniques, like adaptive correction of task size and data access profile estimations of the clients, adaptive determination of current remote access cost for certain data types in the current situation by feedback or adaptive observation of the load balancing component's compute load and decision delays with according simplification of the load balancing policy.

- *Critical Paths and Task Priorities*. To optimize the throughput of uncorrelated concurrent tasks, it is sufficient to consider each task isolated together with the load of the compute nodes. However, especially in large parallel applications there are dependencies between the tasks that should be considered by load balancing. Complex execution graphs in relational database queries are a good example. While several

static scheduling algorithms have been developed, the integration of such dependency considerations into dynamic load balancing decisions rises different problems. First, the precedence relationships are not given at system start-up but arrive every now and then during runtime and have to be integrated into the current situation. Second, there is uncoordinated multi user bustle, i.e. precedence relationships maybe known within task groups, but the groups as well as other unrelated tasks arise and run at arbitrary times. The third is the inaccuracy of the precedence relationships announced by the clients. Missing tasks, violated precedence relationships and wrong task size estimations should not severely destroy the profits of the load balancing decisions.

Despite these problems, dynamic load balancing should be able to profit from knowledge about inter task dependencies. E.g. the HiCon approach [5] decouples the priority calculation from the assignment decision and from the consideration of data affinities. Clients can dynamically announce arbitrary task dependency graphs containing precedence relationships and task size estimations. Load balancing therefrom computes and stores task priorities according to the critical path algorithms including a special prioritizing of tasks that entail large parallelism. The critical path is this path along a task's successor tasks which adds up the largest task sizes; it determines the response time of the whole task group. Later on, arriving tasks that are recognized as previously announced are sorted into the central task queue according to their priority. While residing in the central queue, the priority of tasks grows linearly. Tasks not belonging to a group just get a priority proportional to their estimated task size.

- *Consideration of Data Affinities*. In the HiCon model, tasks are queued centrally per workstation cluster, and load balancing assigns tasks in advance to servers for a certain amount of time. Selecting the best suitable server S for a task is guided by the minimum response time and the maximum system throughput, weighted depending on the overall system load.

  It is instructive to take a closer look into the decision model, as far as it is concerned with data affinity consideration. Following formula shows the general cost model which is used to determine the best suited server instance for a task execution: $S = MIN_s(t_{compute}(s,a) + t_{dataComm}(s,a) \times d_{overrate} + t_{remWork}(s))$

  $t_{compute}(s,a)$ is the expected elapsed time to process task a at server s. It is computed by the estimated task size (instructions), the expected available processing power at server's node at the estimated task start time, and an adaptive correction factor. The available processing power is derived from the processor speed, weighted by task's floating point and integer demands, the number of SMP processors on the node and the number of concurrently executing tasks on the node, weighted by an adaptive factor estimating the cpu utilization of these task types. This factor considers the communication portion of the tasks' execution, and is updated by exponential smoothing from system level cpu utilization measurements.

  $t_{remainingWork}(s)$ is the expected time until server s will have finished all the tasks currently in its local queue. It is the sum of the expected calculation and communication times of the tasks queued at this server minus the time the server has spend on its current task up to now.

  While the main formula determines the server with minimum task response time, it can be shifted towards the server which gives optimum resource utilization by simply overrating the data communication cost. Factor doverrate overrates the costs depending on the overall system utilization by application tasks (which can be measured by the number of tasks queued centrally in the cluster), and on the business of the cluster load balancing component (which can be measured by the number of events waiting to be dispatched). Data communication overweighting achieves that less parallelism is exploited and data affinity becomes more important. In consequence, it reduces communication cost and ensures full cpu utilization, which are important issues in high load situations when global system throughput is the main optimization goal.

  $t_{dataComm}(s,a)$ is the expected elapsed time for data communication during task a's execution on servers. It basically sums up the access cost to non-local data records:

  $$t_{dataComm}(s,a) = dataTimeAdapt_{tt} \times \sum_{i=1}^{N} \sum_{j=1}^{N_i} t_{dataAccess}(s,a,i,j)$$

Data access patterns are provided by clients as set of N probable accesses to ranges Ni of data. Per announced data reference dataRefi,j(a) with given probability for write access, following costs are expected: $t_{dataAccess}(s,a,i,j)$ =

0	if *ownerServer(dataRefi,j(a))=s*,
dataRangeWriteProbi(a) x $dataCommCost_d$	if $s \in$ *copyHolders(dataRefi,j(a))*,
$dataCommCost_d$	if *ownerServer(dataRefi,j(a))* $\in$ *local cluster*,
$dataCommCostRemote_d$	otherwise.

Data access cost are differentiated between intra and inter cluster access. For read access a local copy is sufficient, for write access the original is required. These things depend on the underlying data management system. *dataCommCost* (and similarly *dataCommCostRemote*) is adjusted by observing real elapsed times per remote access to this data type, with exponential smoothing.

ownerServer and *copyHolders* are information tables telling the probable current data distribution over the clusters and among the servers within a cluster. Each time a task is assigned, the probable data distribution is updated according to the task's reference estimation, and every now and then the data management system sends partial updates of the real current data distribution.

While the formula for $t_{dataAccess}$ estimates the data access costs, the comparison between the servers uses a modification, where copy access costs are reduced by the factor dataReadWritecs to encourage the creation of copies for repeated data accesses. This factor is regulated by observing the average number of read accesses to a data type *d* between two write accesses.

$dataTimeAdapt_{tt}$ is another adaptive correction factor per task type, which is updated by observation and exponential smoothing. It represents the relationship between the total data access cost per task execution and the cost estimated by the formula above, based on the client's estimation and the estimated data distribution and the estimated network power (cost per remote data access).

For migration decisions between clusters, HiCon load balancing also considers the cost for moving the data to the remote cluster: Applications are migrated only (among other restrictions), if their expected remaining processing time is larger than the migration cost for the recently accessed data by the application. The costs are estimated similarly to the formulas shown above. This is important to avoid unworthy migration efforts.

5 Experience

Concluding, the experiences from a number of real measurements that have been performed with the load balancing system, show the correctness and necessity of the techniques. Overall, the HiCon model was shown to be successful for different application domains like numerical simulation, image processing and database operations in various demanding configurations and load profiles.

Looking at parallel database applications executing dedicated on a workstations cluster, sequential execution, random task distribution and complex load balancing, e.g. according to the HiCon concept can be compared. In workstation clusters, communication and remote data access is rather expensive compared to parallel machines. Hence, the communication to processing power relationships will be valid for the near future and become valid for parallel systems too, because with growing processor power the data locality and caching issues are becoming increasingly important.

Random distribution across about five nodes can - compared to sequential execution - give significant acceleration for different applications: response time reduction to 40% for breath first recursive graph searches, reduction to 37% for complex database query execution. A load mix of database operations on R-Tree access structures - a sequence of parallel polygon inserts followed by a number of parallel spatial join operations can be accelerated to about 47% of sequential response time. Sophisticated load balancing, however can, by considering the different host speeds, the actual host loads the different task sizes and the expected data communication, reduce the response times to 27% (search), 21% (complex query) and 42% (R-Tree operations) compared to sequential processing. The additional 5% gain at the R-Tree operations, for example, results mainly from reducing the exploited parallelism in phases of high insert activities.

In multiuser operation, several of the respective applications were thrown concurrently into the system. Unbalanced execution means that applications executed sequentially on one of the nodes each. The response time then depends on the slowest node in the cluster, because all nodes got roughly the same

application load. Simple round robin load distribution even sometimes deteriorates the throughput, according to the common experience that in heavy load situations the system should not be disturbed unnecessarily by load balancing. Sophisticated load balancing however, can give enormous improvements (response time reduction) to about 57% (searches) and 60% (complex queries) compared to unbalanced execution. In the search applications, the data affinities are less important because the runtime system automatically distributes copies of the required data to the nodes, independent from the load balancing policy, which is a good idea for the large sets of read-only data. Hence, in the search application it was more important to avoid idle times by distributing tasks equally both looking at estimated task sizes and actual node loads. A closer look at the complex query executions shows that the concept of prioritizing tasks on critical paths is the reason for 10% throughput gain while the other earnings result from data affinity consideration.

In a very heterogeneous load mix, consisting of two parallel image recognitions, two parallel database query scenarios and two parallel finite element analysis calculations on a workstation cluster, load balancing has to face all different kinds of complexities as described in the first section. While “first free” load balancing (assign each task to the first server that becomes idle) yields 93% of the execution time compared to random task distribution, HiCon load balancing could reduce the overall response time to 42% of the random distribution. Detailed analysis told that the achievements mainly resulted from proper utilization of the resources in the overloaded system, and significant network delay reduction while all cpus could be kept busy.

If several clusters are coupled, communication is usually even more expensive and should be considered more carefully. As an extreme case, parallel applications were distributed across workstation clusters located in Stuttgart, Bonn, Toulouse and Belfast [7][9]. The wide area network imposed large latencies in the order of 100 ms per message, compared to 0.5 ms within a cluster. Seven parallel image recognition applications were started within one cluster with a distance of several seconds between each. When choosing applications randomly for migration between clusters, the overall elapsed time even increases to 105% compared to no inter-cluster load balancing. When selecting the best suited application, including the consideration of the data movements, migration can reduce elapsed time to 60%, and if migrations are limited to those where it could be worthwhile (with respect to the data movement cost and the expected remaining application run time), load balancing can reduce to 57% of the execution time.

Metacomputing views coupled systems as one large computing resource. Measurements with distant workstations in Stuttgart and Toulouse configured as one cluster (note that load balancing still can distinct data communication cost between nodes in and between these clusters), gave interesting results [9]. While ignoring data affinity was a complete disaster, several complex parallel database queries could be fruitfully distributed across Europe. Load balancing stuck mainly to one cluster unless routing was switched to a faster ATM network. Sometimes it could be observed that load balancing first kept the data intensive tasks within one cluster. After the applications unfolded their parallelism and the overall load increased, load balancing first spread few tasks across Europe. With the time, however, these tasks pulled the execution towards the remote cluster, because successor tasks operate on the intermediate result data of their predecessor tasks, encouraging load balancing to perform further executions also at this site.

The benefits from task priority consideration where evaluated by executing complex parallel database query graphs on a workstation cluster [5]. Additional problems compared to most static scheduling setups are different node capacities and the fact that there are much more tasks than available processors. In summary, the trials showed that ignoring priorities or using priorities simply according to each task’s size is insufficient. Taking critical paths and entailed parallelism into account results in 5 to 10% increase of throughput. In a multiuser access environment, i.e. several query graphs executing concurrently, the dependency considerations yield even more improvement for following reason: In single user mode, simple load balancing can achieve a nearly ‘highest level first scheduling’ like distribution because the executable tasks arrive in this order. In multiuser concurrence however, load balancing has to prefer - and reserve capacities for - tasks from other graphs which are more urgent, although they may have arrived a bit later.

6 Conclusion

Dynamic load balancing for parallel and distributed database management systems is a key to efficient execution of large and complex queries that run dedicated or in multiuser concurrence. This paper introduced the data communication and remote data access issues for dynamic load balancing. It compiled the basic approaches and techniques to face these challenges. After this, the data communication related concepts of the HiCon system were presented in detail, and some measurement scenarios illustrated the need, applicability and the benefits of the approach. An important conclusion is that the consideration of data and communication is actually important for load balancing, however mostly ignored up to now. Load balancing facilities can benefit significantly by considering data affinities and communicating task groups appropriately. The HiCon concept was presented in detail as major representative for sophisticated and successful policies that are applicable for parallel database processing.

References

1. I. Ahmad, A. Ghafoor, G. Fox, Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers, Journal of Parallel and Distributed Computing 20, 1994
2. F. Anger, J. Hwang, Y. Chow, Scheduling with Sufficient Loosely Coupled Processors, Journal of Parallel and Distributed Computing 9, 1990
3. A. Barak, A. Shiloh, A Distributed Load-balancing Policy for a Multicomputer, Software-Practice and Experience Vol. 15 No. 9, 1985
4. K.Baumgartner, B.Wah, A Global Load Balancing Strategy for a Distributed Computer System, Workshop on the Future Trends of Distributed Computing Systems in the 1990's, 1988
5. W. Becker, G. Waldmann, Exploiting Inter Task Dependencies for Dynamic Load Balancing, Proc. IEEE 3rd Int. Symp. on High-Performance Distributed Computing (HPDC), 1994
6. W. Becker, J. Zedelmayr, Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures, Mitteilungen GI, Workshop Parallele Algorithmen und Rechnerstrukturen, 1994
7. W. Becker, G. Waldmann, Adaption in Dynamic Load Balancing: Potential and Techniques, Tagungsband 3. Fachtagung Arbeitsplatz-Rechensysteme (APS), 1995
8. W. Becker, Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen, Ph.D. Thesis, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
9. W. Becker, Fine Grained Workload Distribution Across Workstation Clusters of European Computing Centers Coupled by Broadband Networks, Faculty report 1995 / 9, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
10. S. Bokhari, A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System, IEEE Transactions on Software Engineering Vol. 7 No. 6, 1981
11. N. Bowen, C. Nikolaou, A. Ghafoor, Hierarchical Workload Allocation for Distributed Systems, Proceedings Parallel Processing, 1988
12. N. Bowen, C. Nikolaou, A. Ghafoor, On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems, IEEE Transactions on Computers Vol. 41 No. 3, 1992
13. T. Chou, J. Abraham, Load Balancing in Distributed Systems, IEEE Transactions on Software Engineering, Vol. 8 No. 4, 1982
14. Y. Chow, W. Kohler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, IEEE Transactions on Computers Vol. 28 No. 5, 1979
15. J. Colin, P. Chretienne, C.P.M. Scheduling with Small Communication Delays and Task Duplication, Operations Research Vol. 39 No. 4, 1991
16. G. Copeland, W. Alexander, E. Boughter, T. Keller, Data Placement in Bubba, SIGMOD, 1988

17.M. Devarakonda, R. Iyer, Predictability of Process Resource Usage: A Measurement-Based Study on UNIX, IEEE Transactions on Software Engineering, Vol. 15 No. 12, 1989

18.K. Efe, B. Groselj, Minimizing Control Overheads in Adaptive Load Sharing, Proc. 9th Int. Conf. Distributed Computing Systems, 1989

19.D. Evans, W. Butt, Load balancing with network partitioning using host groups, Parallel Computing 20, 1994

20.P. Gopinath, R. Gupta, A Hybrid Approach to Load Balancing in Distributed Systems, Symposium on Experiences with Distributed and Multiprocessor Systems USENIX, 1991

21.K. Goswami, M. Devarakonda, R. Iyer, Prediction-Based Dynamic Load-Sharing Heuristics, IEEE Transactions on Parallel and Distributed Systems Vol. 4 No. 6, 1993

22.B. Indurkhya, H. Stone, L. Cheng, Optimal Partitioning of Randomly Generated Distributed Programs, IEEE Transactions on Software Engineering Vol. 12 No. 3, 1986

23.L. Kale, Comparing the Performance of two Dynamic Load Distribution Methods, Proceedings Parallel Processing, 1988

24.T. Kunz, The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, IEEE Transactions on Software Engineering, Vol. 17 No. 7, 1991

25.C. Lee, D. Lee, M. Kim, Optimal Task Assignment in Linear Array Networks, IEEE Transactions on Computers, Vol. 41 No. 7, 1992

26.T. Lewis, H. El-Rewini, Parallax: A Tool for Parallel Program Scheduling, IEEE Parallel and Distributed Technology, 1993

27.F. Lin, R. Keller, The Gradient Model Load Balancing Method, IEEE Transactions on Software Engineering Vol. 13 No. 1, 1987

28.H. Lin, C. Raghavendra, A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC), IEEE Transactions on Software Engineering Vol. 18 No. 2, 1992

29.V. Lo, Heuristic Algorithms for Task Assignment in Distributed Computing Systems, IEEE Transactions on Computers Vol. 37 No. 11, 1988

30.R. Lüling, B. Monien, F. Ramme, Load Balancing in Large Networks: A Comparative Study, IEEE Symposium Parallel and Distributed Processing, 1991

31.P. Ma, E. Lee, M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, IEEE Transactions on Computers Vol. 31 No. 1, 1982

32.W. Osser, Automatic Process Selection for Load Balancing, Master Thesis, University of California, Santa Cruz, 1992

33.R. Pollak, A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer, Proc. Int. Symp. Parallel and Distributed Supercomputing, 1995

34.E. Rahm, R. Marek, Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems, Proc. 19th VLDB Conference, 1993

35.C. Shen, W. Tsai, A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion, IEEE Transactions on Computers, Vol. 34 No. 3, 1985

36.B. Stramm, F. Berman, Communication-Sensitive Heuristics and Algorithms for Mapping Compilers, ACM SIGPLAN Vol. 23, No. 9, 1988

37.A. van Tilborg, L. Wittie, Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers, IEEE Transaction on Computers Vol. 33 No. 9, 1984

38.P. Yu, A. Leff, Y. Lee, On Robust Transaction Routing and Load Sharing, ACM Transactions on Database Systems Vol. 16 No. 3, 1991

39.S. Zhou, X. Zheng, J. Wang, P. Delisle, Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Canada, 1992

Abstract

Dynamic load balancing shall optimize the resource utilization in parallel systems and computer networks in order to reduce the application response times. While simple schemes are mathematically tractable and can increase resource utilization for certain applications, while implemented remote execution facilities can avoid cpu overload peaks in computing clusters, load balancing schemes for realistic systems and complex load profiles, especially for database management systems, need to consider data and communication. This paper explains the problem and potential benefits of dynamic load balancing, emphasizing on the exploitation of data affinities and consideration of communication cost. The sophisticated techniques developed in the HiCon system are presented in detail and validated by several real measurements from parallel and distributed database processing.

Table of Contents

Dynamic Load Balancing for Parallel Database Processing

Wolfgang Becker

Faculty Report No. 1997 / 08

Institute of Parallel and Distributed High-Performance Systems
(IPVR), University of Stuttgart
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
Phone: +49 711 7816 411
Email wbecker@informatik.uni-stuttgart.de

1 Dynamic Load Balancing for Real Systems and Applications

The basic motivation to investigate and realize load balancing facilities comes from the very simple experience that several applications in a parallel system or computer network usually do not exploit the system very good. Instead, many compute nodes are idle while others are overloaded. The most well-known environments are workstation networks. Without buying new resources, load balancing as a system service can offer much more resources and better response times to all users, and can effectively avoid disasters that sometimes can arise from bad load distribution.

Many approaches try to equalize the number of processes on the compute nodes or equalize the task queue lengths at the nodes in the network - actually most of them just avoid that nodes run empty while others are busy. Load balancing becomes more complicated as the system consists of heterogeneous nodes, i.e. faster and slower processors, different amount of main memory and different number of cpus per node (SMP). Similarly, most real applications and tasks in the system are heterogeneous. They consume different amounts of resources, i.e. run short or longer times and use different percentage of the cpu time. Further, reality usually brings a mix of several, maybe parallelized, applications. Additionally, the arrival of tasks is not coordinated or planned, so load balancing has to cope with changing, unpredictable load profiles. Another complication is that parallel applications are not just sets of independent tasks, but the tasks are structured by precedence relationships. Load balancing should favor tasks on critical paths and tasks that entail large parallelism to reduce application response time and increase resource utilization. Other effects like context switch overhead or paging delays due to active memory usage severely influence the system performance, so load balancing should limit the overall system load appropriately. Load balancing should further be adaptive, i.e. identify the currently important performance factors, create estimations by profiling the system behavior and find the trade-off between load balancing overhead and improvement.

Although these requirements already make real world load balancing quite difficult, the following section explains that applications in various styles work on shared data and exchange data, which also has significant influence on the system resource utilization and application performance. Hence, load balancing should consider data, communication and networks appropriately.

2 Data Access and Communication Influence Application Performance

Data and communication are not of primary interest to the load balancing service. Load balancing is focussed on application response time and system resource utilization, measured by task elapsed times, application elapsed times, cpu utilizations, paging rates and perhaps disk and network utilization.

But applications often access remote data in form of network file sharing or global distributed databases. Access to data which do not reside on the local node, forces the application to request the data and wait for their arrival, and induces network load. The delay time could be used by other processes on the node, the network utilization should be coordinated with other network users to avoid congestion. If remote data access is performed by issuing remote procedure calls and make the data owner node execute the operations, then additional cpu load at this node will arise and other tasks on this node may be slowed down.

Application tasks usually pass intermediate results to their successor tasks, whether they send messages, or pass results to the client which passes them with the next server calls, or they store them into temporary files. This data flow also causes communication delays and network load. Tasks in parallel applications exchange synchronization messages to control the overall execution. These messages implement precedence relationships between tasks and can serve for result collection and aggregation. They also cause delays due to message latencies and network load.

Cooperation in parallel applications can, depending on the programming paradigm, take place by sending data to the task that needs them, or by accessing a virtual shared memory which locates the requested data and sends them to the requester. This data communication infers delays, network load, cpu load at remote node and search overhead to locate the data, which also yields computations and messages.

Finally, load balancing can cause data communication for process or application migration. Migrating a running process entails sending its data segment and maybe its code segment. This can be done in one strike or by demand paging, and increases the migration cost by additional delays and network load. Similarly, when migrating whole running (parallel) applications, whether moving all processes or starting the next

processing phase on other nodes, the communication of the application's active data set yields additional communication delays and network load.

Depending on network throughput and latency, and on the impact of the communication or data access speed on the application performance, load balancing can obtain large improvements by considering this.

3 How Can Load Balancing Consider Data and Communication

The relevant effects from data access and communication for a load balancing service are the additional delays, the reduced cpu utilization and the additional network utilization or network congestion. Now, load balancing needs a simple model and few critical factors to measure and to decide upon. The critical factors can be obtained by static data about the system resources and application behavior, dynamically by measuring the system and application behavior and by getting recent profile estimations from the applications. On system level, network delays (process to process latency per short message), network throughputs (bytes of continuous data streams per second) and the network topology (which nodes are directly connected, routing, hardware multicasting support) are of interest. One level above, the application data distribution within the system, average remote data access response times, the network utilization by communications or remote data access operations, and the cpu utilization and consumption for remote data operations are relevant. On application level, the message frequencies, the communication topology and the message sizes can be profiled or pre-estimated, and data reference patterns (record or data range names, read/write probability, repetition degree of access patterns) can be obtained by observation or from direct application hints. Finally, general estimations about applications' sensitivity to communication power (throughput, latency, temporary deviations) are interesting as guidelines for adaptive load balancing, to decide which factors are currently relevant.

Instead of developing a general approach to evaluate these informations for load balancing decisions, a brief review of existing techniques will be followed by a closer look at one of them, the HiCon approach:

- In general, there are some load balancing schemes which avoid communication, limit communication to the network capacity, or find the trade-off between increased resource usage by task distribution and parallel execution and the loss from communication delay and overhead. They do it by placing communicating tasks on the same node or cluster, by placing the tasks to where their data reside or by migrating / replicating the data to where the tasks are or going to be.
- A common static load balancing technique [10][11][31] is to match a task graph with a network graph. The task graph consists of the application tasks as nodes and communication relationships as edges, weighted by the communication intensity or frequency or amount. The network graph's nodes represent processors (compute nodes) and the edges direct communication lines, weighted by the throughput of the lines. Tasks can be assigned to processors by recursively partitioning the task and network graphs and always assign one task graph part to one network graph part. Both graphs are partitioned so that heavy weight linked nodes remain together in one partition. Matching obeys boundary conditions like maximum node load, maximum node load skew, maximum network line utilization etc. Alternatively, the graph matching can be done immediately by solving the corresponding optimization problem [29][35][12][36][25]. Recent research on supercomputers with extreme low latency networks [33], however, show that it can be profitable to distribute communicating tasks, because message passing between nodes is faster than the context switch from sender to receiver process within one node.
- Static load balancing methods that assign groups of tasks with precedence relationships to a set of processors in order to minimize the finish time of the last task, can be extended to consider also the data flow between the tasks [2][13][15][22][26]. Without looking at data, these algorithms place the largest tasks or the tasks on the critical path onto the fastest processor or onto the next available processor. Data flow is included in the model by adding a delay to the start of each task (the amount of time depends on the network speed and the data volume), if the task gets data from a predecessor that executed on another node. Hence, successor tasks are preferably placed on the same node as their predecessors.
- Transaction routing schemes place transactions to the database server that has the largest portion of the required data [3][38]. Therefore, load balancing knows the location of the database partitions and the average data reference patterns of the most important transaction types. Besides distributing the computational load by equalizing the number of database requests per unit of time between the proces-

sors, the transactions may not be distributed by chance but - as far as possible without compute load skews - to the node where most of the addressed database partitions reside. More sophisticated methods have an explicit load model including the cost of remote database requests (subtransactions), and send each query to the node where the sum of compute times, message delays and remote compute times is minimal. Further, the cost factors of communication, execution and general additional overhead per remote execution can be adjusted by tracing the system behavior and periodic regression analysis.

- Similarly, dynamic load balancing can receive data reference pattern estimations per task from the client [8], can generate / correct these estimations by observing the real access profiles [7], can dynamically observe the system data and copy distribution, and can observe the real data movement costs for remote accesses. Using this information, the expected time for data access on each available server can be estimated and compared.
- Dynamic load balancing schemes can estimate the data communication portion of tasks or task types by tracing the cpu utilization divided by number of running processes or observing the recent cpu usage of tasks [33]. Tasks that do not fully need their cpu share, either perform I/O or communicate a certain amount of time, and hence their nodes can take more concurrent tasks.
- While the techniques above served for task placement decisions, load balancing can also consider data communication cost for task migration decisions [32]. So, selecting a suitable process for migration from an overload node can take the one with the smallest data segment or the smallest number of accessed local files. Thus, the data communication entailed by the migration is kept small.

4 Load Balancing Runtime Support for Parallel Database Systems

Today's general load balancing systems are usually queueing and remote execution facilities or process migration facilities. They ignore the existence of data or do not consider them in a sophistication that makes them applicable for parallel database systems. So it is interesting to describe the architecture of one of the major general systems that can cope with parallel database operation, the HiCon load balancing model [8]. HiCon load balancing is application independent, but was designed especially for data intensive and cooperating complex parallel applications. The main concepts emphasize the key issues for dynamic load balancing systems:

- *Centralized and Distributed Structure*. Centralized load balancing (often called 'global') provides one central agent for collecting load information and for decision making [14][16][18][28]. It has several strong advantages: The measurement informations and predictions consistently reside in one place and may not be distributed or replicated among the system which would cause message traffic and could lead to outdated, inconsistent informations and contradictory load balancing decisions. Centralized load balancing can exploit the global knowledge about system and application behavior and can utilize sophisticated strategies. Centralized load balancing usually comes along with central task queueing. This allows for load control, because tasks can be queued until enough resources are available. Otherwise, heavy load situations saturate the system and reduce the throughput due to process switching, memory paging and network congestion. Central queueing also enables late decisions, because tasks may not be assigned when they are born but when they are removed from the queue for immediate execution. Central load balancing can avoid load imbalances before they arise, whereas distributed load balancing always has tasks waiting / running at the node where they originated and try to defeat the skews.

  Distributed load balancing (often called 'decentralized') becomes necessary for large parallel and distributed systems. These concepts try to keep load balancing efforts (resource consumption as well as delays) constant regardless to the system size, prevent single points of failure by local decision autonomy and reduce information and task exchange between nodes by simple, restrictive load balancing policies [4][23][27][30].

  When comparing centralized and distributed approaches, it must be noted that a stable heavy load by long running tasks is no problem for centralized approaches, because there are not many decisions. Task arrival bursts are critical because they cause much work for the centralized component. But, without centralized load balancing they generate severe load skews because they usually arise from one or few sources, e.g. parallel applications. So, centralized load balancing itself is overload but prevents load skews while distributed techniques will encounter a long period of slowly stepwise equalizing the skews.

Ideally, load balancing should have a mixed structure, i.e. a centralized agent per closely coupled cluster and distributed cooperation between neighbored clusters [6][19][20][39] or even a hierarchical structure [1][37]. The HiCon load balancing system provides one central component per cluster while neighbored clusters cooperate decentralized. Central task queueing takes place within a cluster; between clusters whole executing applications (task groups) are migrated if a significant load skew is detected. Between cluster, the HiCon system does not migrate executing tasks, but just routes all following tasks of this application to the neighbor cluster.

- *Client / Server Execution Model*. The HiCon approach provides a load balancing service for client - server processing. Tasks are issued by clients and are queued and assigned by the load balancing component to appropriate server processes of the respective server class. The concept supports mixed parallel applications and multiuser operation.

  It is important that the execution model for the load balancing decisions matches the execution profiles of the target application area. It should restrict itself to few relevant characteristics and measurement entities. The client - server execution model has been established as the basic processing concept for database management systems and for almost all large, distributed applications. Especially in database processing, fixed SPMD-style problem decomposition onto processors is not flexible enough, because it does not work well for large and small problems, for parallel queries and multiuser operation or for complex structured queries with unpredictable intermediate result sizes. Server class calls, however, allow for a flexible dynamic task distribution within parallel and distributed systems, because each call may be assigned to an arbitrary server instance on any node.

  For load balancing purposes, client - server has the additional advantage, that server classes provide a natural grouping of tasks that show similar profiles. So, load balancing can observe - and later on predict - the execution profiles and data access patterns per server class. The decoupling of applications into discrete tasks (server class calls) is also a simplification for the load model, because the behavior of whole, complex structured and communicating processes is hard to model appropriately. The concept of statically allocated server instances (processes) allows for rather fine grained task distribution and high parallelization degree, because a remote execution does not necessarily incur process start and connection setup overhead.

- *Flexible Data Management System*. For parallel database processing, load balancing additionally needs to have an idea of data, i.e. a model of how tasks operate on data and how data can be accessed remotely or can be moved or replicated and what resource consumption and delays come along with it. This was explained above.

  The approach of a virtual shared memory / data store fits very well into database processing: Within the HiCon system, tasks communicate by accessing global data items, either shared within an application or globally between all applications. The global data can be arbitrarily structured, volatile or permanent. No remote data access operations are performed, but the data items are migrated or replicated to the requesting task on demand. Therefore, a distributed runtime system supports transparent data partitioning / replication / migration, realizes a global database or virtual shared memory with changing data owner and probable owner search concepts, and provides consistent data access by locks and copy invalidation. The HiCon load balancing concept does not require this data management component. It just gets informations from it and knows about its facilities. Arbitrary database management or virtual shared memory components could be employed. It must be stated that none of today's commercial database systems provides dynamic data migration and replication in a comparable degree of flexibility. So, load balancing must know about the facilities of the data management system which the respective application uses. The important issues for load balancing are the efforts for data communication, the location of the data and the expected data access patterns of the application tasks.

  For illustration, the scheme of the runtime data management system is described which is currently used within the HiCon system. It is a general, flexible approach to management of logically common, physically distributed data. Data granularity is defined by the application. It can be few bytes, records in main memory or in a file or even whole files. The owner of a data record is the server who performed the latest update operation on it. This server is asked for copies and may grant ownership to other servers for updates. Copies are requested for read-only access and for updates the distributed copies are invalidated. This policy optimizes the data access cost, provided that the servers show a certain data reference locality: Repeated read access on local data copies is cheap and the number and distribution

of copies automatically adapts to the read / write proportion of the accesses. Load balancing tries to increase this access locality by assigning tasks to where the required data or copies of them reside. To ensure consistency, servers precede their data access by acquiring appropriate read / write locks.

The concept of cooperation via common data enables a quite flexible distribution of the tasks among the system, even for server classes that are not context free. For load balancing, the common data are the objects where it can include data affinity in its cost model. And this model comprises in a simple way communication between tasks, passing of intermediate results (pipelining) and access to common, global, maybe persistent data.

- *System Load Measurement and Load Pre-estimations*. It is important that load balancing acts dynamically by measuring the current system load. Especially in multiuser environments like OLTP there is no central a-priori planning but tasks arrive uncoordinated. For balancing large complex queries, dynamic load balancing is also important, because the actual amounts of data and computing efforts cannot be pre-estimated accurately. Further, load balancing should be adaptive for three reasons:

  1. Adaptive generation and correction of pre-estimations for task profiles and system load behavior. Dynamic load balancing decisions base on informations about application requirements and system utilization. These informations are often inaccurate or unavailable. Even load balancing concepts that make no explicit assumptions about the future task or system behavior but react on current measurements only, implicitly employ an extrapolation of zeroth degree, i.e. assume constant behavior. More sophisticated concepts can use load profile estimations from their own observations or from applications [17][21][24][32][34]. Comparing the previous estimations with the actual system behavior, load balancing can assess the accuracy and the value of the estimations and can correct them appropriately. For example, in the area of transaction routing [38] periodically perform regression analysis to determine the correlation between assumed and actually observed response times, and therefrom reduce the derivations for further routing decisions by correction factors.

  2. Regulation of difficult decision factors by feedback. Another weakness of dynamic load balancing is the vague execution model on which the decision algorithm is based. In real, parallel and distributed systems and large real applications it is extremely difficult to capture all effects and dependencies within a compact model. Because decisions have to be committed at runtime they must restrict to simple execution models. Hence, the correlation between the factors which load balancing considers (e.g. processor run queue lengths) and the overall throughput which is to be optimized, is not strong enough in all situations. The costs for remote data access, for example, depend on the size and complexity of the data, on the network utilization, on the lock wait times and on the utilization of the remote data owner - factors hard to merge into a compact formula. So, load balancing has to compare the actual effects of such decision factors in certain situations and globally adjust further decisions by appropriate correction factors, not knowing detailed reasons.

  3. Adaptive optimization of load balancing's cost - efficiency proportion. Load balancing profit is curtailed by its overhead, because it causes compute load, communication load and delays for information collection and decision making. Without adaption, load balancing assumes that its overhead is always appropriate and worthwhile which is not true for all situations and load profiles. Hence, load balancing must minimize its efforts, or even better, observe and regulate its cost - efficiency proportion. Decentralized load balancing, for example, should exchange less load informations in times of overall heavy system load, because there should not be many migrations anyway. Similarly, it should switch from the more expensive sender initiation to receiver initiation, i.e. not the many overloaded nodes should start searching for an underload node but the few idle nodes should pick up load from some overload colleague.

  The HiCon load balancing model employes a couple of adaption techniques, like adaptive correction of task size and data access profile estimations of the clients, adaptive determination of current remote access cost for certain data types in the current situation by feedback or adaptive observation of the load balancing component's compute load and decision delays with according simplification of the load balancing policy.

- *Critical Paths and Task Priorities*. To optimize the throughput of uncorrelated concurrent tasks, it is sufficient to consider each task isolated together with the load of the compute nodes. However, especially in large parallel applications there are dependencies between the tasks that should be considered by load balancing. Complex execution graphs in relational database queries are a good example. While several

static scheduling algorithms have been developed, the integration of such dependency considerations into dynamic load balancing decisions rises different problems. First, the precedence relationships are not given at system start-up but arrive every now and then during runtime and have to be integrated into the current situation. Second, there is uncoordinated multi user bustle, i.e. precedence relationships maybe known within task groups, but the groups as well as other unrelated tasks arise and run at arbitrary times. The third is the inaccuracy of the precedence relationships announced by the clients. Missing tasks, violated precedence relationships and wrong task size estimations should not severely destroy the profits of the load balancing decisions.

Despite these problems, dynamic load balancing should be able to profit from knowledge about inter task dependencies. E.g. the HiCon approach [5] decouples the priority calculation from the assignment decision and from the consideration of data affinities. Clients can dynamically announce arbitrary task dependency graphs containing precedence relationships and task size estimations. Load balancing therefrom computes and stores task priorities according to the critical path algorithms including a special prioritizing of tasks that entail large parallelism. The critical path is this path along a task's successor tasks which adds up the largest task sizes; it determines the response time of the whole task group. Later on, arriving tasks that are recognized as previously announced are sorted into the central task queue according to their priority. While residing in the central queue, the priority of tasks grows linearly. Tasks not belonging to a group just get a priority proportional to their estimated task size.

- *Consideration of Data Affinities*. In the HiCon model, tasks are queued centrally per workstation cluster, and load balancing assigns tasks in advance to servers for a certain amount of time. Selecting the best suitable server S for a task is guided by the minimum response time and the maximum system throughput, weighted depending on the overall system load.

  It is instructive to take a closer look into the decision model, as far as it is concerned with data affinity consideration. Following formula shows the general cost model which is used to determine the best suited server instance for a task execution: $S = MIN_s(t_{compute}(s,a) + t_{dataComm}(s,a) \times d_{overrate} + t_{remWork}(s))$

  $t_{compute}(s,a)$ is the expected elapsed time to process task a at server s. It is computed by the estimated task size (instructions), the expected available processing power at server's node at the estimated task start time, and an adaptive correction factor. The available processing power is derived from the processor speed, weighted by task's floating point and integer demands, the number of SMP processors on the node and the number of concurrently executing tasks on the node, weighted by an adaptive factor estimating the cpu utilization of these task types. This factor considers the communication portion of the tasks' execution, and is updated by exponential smoothing from system level cpu utilization measurements.

  $t_{remainingWork}(s)$ is the expected time until server s will have finished all the tasks currently in its local queue. It is the sum of the expected calculation and communication times of the tasks queued at this server minus the time the server has spend on its current task up to now.

  While the main formula determines the server with minimum task response time, it can be shifted towards the server which gives optimum resource utilization by simply overrating the data communication cost. Factor doverrate overrates the costs depending on the overall system utilization by application tasks (which can be measured by the number of tasks queued centrally in the cluster), and on the business of the cluster load balancing component (which can be measured by the number of events waiting to be dispatched). Data communication overweighting achieves that less parallelism is exploited and data affinity becomes more important. In consequence, it reduces communication cost and ensures full cpu utilization, which are important issues in high load situations when global system throughput is the main optimization goal.

  $t_{dataComm}(s,a)$ is the expected elapsed time for data communication during task a's execution on servers. It basically sums up the access cost to non-local data records:

  $$t_{dataComm}(s,a) = dataTimeAdapt_{tt} \times \sum_{i=1}^{N} \sum_{j=1}^{N_i} t_{dataAccess}(s,a,i,j)$$

Data access patterns are provided by clients as set of N probable accesses to ranges Ni of data. Per announced data reference dataRefi,j(a) with given probability for write access, following costs are expected: $t_{dataAccess}(s,a,i,j)$ =

0	if *ownerServer(dataRefi,j(a))=s*,
dataRangeWriteProbi(a) x $dataCommCost_d$	if $s \in$ *copyHolders(dataRefi,j(a))*,
$dataCommCost_d$	if *ownerServer(dataRefi,j(a))* $\in$ *local cluster*,
$dataCommCostRemote_d$	otherwise.

Data access cost are differentiated between intra and inter cluster access. For read access a local copy is sufficient, for write access the original is required. These things depend on the underlying data management system. *dataCommCost* (and similarly *dataCommCostRemote*) is adjusted by observing real elapsed times per remote access to this data type, with exponential smoothing.

ownerServer and *copyHolders* are information tables telling the probable current data distribution over the clusters and among the servers within a cluster. Each time a task is assigned, the probable data distribution is updated according to the task's reference estimation, and every now and then the data management system sends partial updates of the real current data distribution.

While the formula for $t_{dataAccess}$ estimates the data access costs, the comparison between the servers uses a modification, where copy access costs are reduced by the factor dataReadWritecs to encourage the creation of copies for repeated data accesses. This factor is regulated by observing the average number of read accesses to a data type *d* between two write accesses.

$dataTimeAdapt_{tt}$ is another adaptive correction factor per task type, which is updated by observation and exponential smoothing. It represents the relationship between the total data access cost per task execution and the cost estimated by the formula above, based on the client's estimation and the estimated data distribution and the estimated network power (cost per remote data access).

For migration decisions between clusters, HiCon load balancing also considers the cost for moving the data to the remote cluster: Applications are migrated only (among other restrictions), if their expected remaining processing time is larger than the migration cost for the recently accessed data by the application. The costs are estimated similarly to the formulas shown above. This is important to avoid unworthy migration efforts.

5 Experience

Concluding, the experiences from a number of real measurements that have been performed with the load balancing system, show the correctness and necessity of the techniques. Overall, the HiCon model was shown to be successful for different application domains like numerical simulation, image processing and database operations in various demanding configurations and load profiles.

Looking at parallel database applications executing dedicated on a workstations cluster, sequential execution, random task distribution and complex load balancing, e.g. according to the HiCon concept can be compared. In workstation clusters, communication and remote data access is rather expensive compared to parallel machines. Hence, the communication to processing power relationships will be valid for the near future and become valid for parallel systems too, because with growing processor power the data locality and caching issues are becoming increasingly important.

Random distribution across about five nodes can - compared to sequential execution - give significant acceleration for different applications: response time reduction to 40% for breath first recursive graph searches, reduction to 37% for complex database query execution. A load mix of database operations on R-Tree access structures - a sequence of parallel polygon inserts followed by a number of parallel spatial join operations can be accelerated to about 47% of sequential response time. Sophisticated load balancing, however can, by considering the different host speeds, the actual host loads the different task sizes and the expected data communication, reduce the response times to 27% (search), 21% (complex query) and 42% (R-Tree operations) compared to sequential processing. The additional 5% gain at the R-Tree operations, for example, results mainly from reducing the exploited parallelism in phases of high insert activities.

In multiuser operation, several of the respective applications were thrown concurrently into the system. Unbalanced execution means that applications executed sequentially on one of the nodes each. The response time then depends on the slowest node in the cluster, because all nodes got roughly the same

application load. Simple round robin load distribution even sometimes deteriorates the throughput, according to the common experience that in heavy load situations the system should not be disturbed unnecessarily by load balancing. Sophisticated load balancing however, can give enormous improvements (response time reduction) to about 57% (searches) and 60% (complex queries) compared to unbalanced execution. In the search applications, the data affinities are less important because the runtime system automatically distributes copies of the required data to the nodes, independent from the load balancing policy, which is a good idea for the large sets of read-only data. Hence, in the search application it was more important to avoid idle times by distributing tasks equally both looking at estimated task sizes and actual node loads. A closer look at the complex query executions shows that the concept of prioritizing tasks on critical paths is the reason for 10% throughput gain while the other earnings result from data affinity consideration.

In a very heterogeneous load mix, consisting of two parallel image recognitions, two parallel database query scenarios and two parallel finite element analysis calculations on a workstation cluster, load balancing has to face all different kinds of complexities as described in the first section. While “first free” load balancing (assign each task to the first server that becomes idle) yields 93% of the execution time compared to random task distribution, HiCon load balancing could reduce the overall response time to 42% of the random distribution. Detailed analysis told that the achievements mainly resulted from proper utilization of the resources in the overloaded system, and significant network delay reduction while all cpus could be kept busy.

If several clusters are coupled, communication is usually even more expensive and should be considered more carefully. As an extreme case, parallel applications were distributed across workstation clusters located in Stuttgart, Bonn, Toulouse and Belfast [7][9]. The wide area network imposed large latencies in the order of 100 ms per message, compared to 0.5 ms within a cluster. Seven parallel image recognition applications were started within one cluster with a distance of several seconds between each. When choosing applications randomly for migration between clusters, the overall elapsed time even increases to 105% compared to no inter-cluster load balancing. When selecting the best suited application, including the consideration of the data movements, migration can reduce elapsed time to 60%, and if migrations are limited to those where it could be worthwhile (with respect to the data movement cost and the expected remaining application run time), load balancing can reduce to 57% of the execution time.

Metacomputing views coupled systems as one large computing resource. Measurements with distant workstations in Stuttgart and Toulouse configured as one cluster (note that load balancing still can distinct data communication cost between nodes in and between these clusters), gave interesting results [9]. While ignoring data affinity was a complete disaster, several complex parallel database queries could be fruitfully distributed across Europe. Load balancing stuck mainly to one cluster unless routing was switched to a faster ATM network. Sometimes it could be observed that load balancing first kept the data intensive tasks within one cluster. After the applications unfolded their parallelism and the overall load increased, load balancing first spread few tasks across Europe. With the time, however, these tasks pulled the execution towards the remote cluster, because successor tasks operate on the intermediate result data of their predecessor tasks, encouraging load balancing to perform further executions also at this site.

The benefits from task priority consideration where evaluated by executing complex parallel database query graphs on a workstation cluster [5]. Additional problems compared to most static scheduling setups are different node capacities and the fact that there are much more tasks than available processors. In summary, the trials showed that ignoring priorities or using priorities simply according to each task’s size is insufficient. Taking critical paths and entailed parallelism into account results in 5 to 10% increase of throughput. In a multiuser access environment, i.e. several query graphs executing concurrently, the dependency considerations yield even more improvement for following reason: In single user mode, simple load balancing can achieve a nearly ‘highest level first scheduling’ like distribution because the executable tasks arrive in this order. In multiuser concurrence however, load balancing has to prefer - and reserve capacities for - tasks from other graphs which are more urgent, although they may have arrived a bit later.

6 Conclusion

Dynamic load balancing for parallel and distributed database management systems is a key to efficient execution of large and complex queries that run dedicated or in multiuser concurrence. This paper introduced the data communication and remote data access issues for dynamic load balancing. It compiled the basic approaches and techniques to face these challenges. After this, the data communication related concepts of the HiCon system were presented in detail, and some measurement scenarios illustrated the need, applicability and the benefits of the approach. An important conclusion is that the consideration of data and communication is actually important for load balancing, however mostly ignored up to now. Load balancing facilities can benefit significantly by considering data affinities and communicating task groups appropriately. The HiCon concept was presented in detail as major representative for sophisticated and successful policies that are applicable for parallel database processing.

References

1. I. Ahmad, A. Ghafoor, G. Fox, Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers, Journal of Parallel and Distributed Computing 20, 1994
2. F. Anger, J. Hwang, Y. Chow, Scheduling with Sufficient Loosely Coupled Processors, Journal of Parallel and Distributed Computing 9, 1990
3. A. Barak, A. Shiloh, A Distributed Load-balancing Policy for a Multicomputer, Software-Practice and Experience Vol. 15 No. 9, 1985
4. K.Baumgartner, B.Wah, A Global Load Balancing Strategy for a Distributed Computer System, Workshop on the Future Trends of Distributed Computing Systems in the 1990's, 1988
5. W. Becker, G. Waldmann, Exploiting Inter Task Dependencies for Dynamic Load Balancing, Proc. IEEE 3rd Int. Symp. on High-Performance Distributed Computing (HPDC), 1994
6. W. Becker, J. Zedelmayr, Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures, Mitteilungen GI, Workshop Parallele Algorithmen und Rechnerstrukturen, 1994
7. W. Becker, G. Waldmann, Adaption in Dynamic Load Balancing: Potential and Techniques, Tagungsband 3. Fachtagung Arbeitsplatz-Rechensysteme (APS), 1995
8. W. Becker, Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen, Ph.D. Thesis, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
9. W. Becker, Fine Grained Workload Distribution Across Workstation Clusters of European Computing Centers Coupled by Broadband Networks, Faculty report 1995 / 9, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
10. S. Bokhari, A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System, IEEE Transactions on Software Engineering Vol. 7 No. 6, 1981
11. N. Bowen, C. Nikolaou, A. Ghafoor, Hierarchical Workload Allocation for Distributed Systems, Proceedings Parallel Processing, 1988
12. N. Bowen, C. Nikolaou, A. Ghafoor, On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems, IEEE Transactions on Computers Vol. 41 No. 3, 1992
13. T. Chou, J. Abraham, Load Balancing in Distributed Systems, IEEE Transactions on Software Engineering, Vol. 8 No. 4, 1982
14. Y. Chow, W. Kohler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, IEEE Transactions on Computers Vol. 28 No. 5, 1979
15. J. Colin, P. Chretienne, C.P.M. Scheduling with Small Communication Delays and Task Duplication, Operations Research Vol. 39 No. 4, 1991
16. G. Copeland, W. Alexander, E. Boughter, T. Keller, Data Placement in Bubba, SIGMOD, 1988

17.M. Devarakonda, R. Iyer, Predictability of Process Resource Usage: A Measurement-Based Study on UNIX, IEEE Transactions on Software Engineering, Vol. 15 No. 12, 1989

18.K. Efe, B. Groselj, Minimizing Control Overheads in Adaptive Load Sharing, Proc. 9th Int. Conf. Distributed Computing Systems, 1989

19.D. Evans, W. Butt, Load balancing with network partitioning using host groups, Parallel Computing 20, 1994

20.P. Gopinath, R. Gupta, A Hybrid Approach to Load Balancing in Distributed Systems, Symposium on Experiences with Distributed and Multiprocessor Systems USENIX, 1991

21.K. Goswami, M. Devarakonda, R. Iyer, Prediction-Based Dynamic Load-Sharing Heuristics, IEEE Transactions on Parallel and Distributed Systems Vol. 4 No. 6, 1993

22.B. Indurkhya, H. Stone, L. Cheng, Optimal Partitioning of Randomly Generated Distributed Programs, IEEE Transactions on Software Engineering Vol. 12 No. 3, 1986

23.L. Kale, Comparing the Performance of two Dynamic Load Distribution Methods, Proceedings Parallel Processing, 1988

24.T. Kunz, The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, IEEE Transactions on Software Engineering, Vol. 17 No. 7, 1991

25.C. Lee, D. Lee, M. Kim, Optimal Task Assignment in Linear Array Networks, IEEE Transactions on Computers, Vol. 41 No. 7, 1992

26.T. Lewis, H. El-Rewini, Parallax: A Tool for Parallel Program Scheduling, IEEE Parallel and Distributed Technology, 1993

27.F. Lin, R. Keller, The Gradient Model Load Balancing Method, IEEE Transactions on Software Engineering Vol. 13 No. 1, 1987

28.H. Lin, C. Raghavendra, A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC), IEEE Transactions on Software Engineering Vol. 18 No. 2, 1992

29.V. Lo, Heuristic Algorithms for Task Assignment in Distributed Computing Systems, IEEE Transactions on Computers Vol. 37 No. 11, 1988

30.R. Lüling, B. Monien, F. Ramme, Load Balancing in Large Networks: A Comparative Study, IEEE Symposium Parallel and Distributed Processing, 1991

31.P. Ma, E. Lee, M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, IEEE Transactions on Computers Vol. 31 No. 1, 1982

32.W. Osser, Automatic Process Selection for Load Balancing, Master Thesis, University of California, Santa Cruz, 1992

33.R. Pollak, A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer, Proc. Int. Symp. Parallel and Distributed Supercomputing, 1995

34.E. Rahm, R. Marek, Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems, Proc. 19th VLDB Conference, 1993

35.C. Shen, W. Tsai, A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion, IEEE Transactions on Computers, Vol. 34 No. 3, 1985

36.B. Stramm, F. Berman, Communication-Sensitive Heuristics and Algorithms for Mapping Compilers, ACM SIGPLAN Vol. 23, No. 9, 1988

37.A. van Tilborg, L. Wittie, Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers, IEEE Transaction on Computers Vol. 33 No. 9, 1984

38.P. Yu, A. Leff, Y. Lee, On Robust Transaction Routing and Load Sharing, ACM Transactions on Database Systems Vol. 16 No. 3, 1991

39.S. Zhou, X. Zheng, J. Wang, P. Delisle, Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Canada, 1992

Abstract

Dynamic load balancing shall optimize the resource utilization in parallel systems and computer networks in order to reduce the application response times. While simple schemes are mathematically tractable and can increase resource utilization for certain applications, while implemented remote execution facilities can avoid cpu overload peaks in computing clusters, load balancing schemes for realistic systems and complex load profiles, especially for database management systems, need to consider data and communication. This paper explains the problem and potential benefits of dynamic load balancing, emphasizing on the exploitation of data affinities and consideration of communication cost. The sophisticated techniques developed in the HiCon system are presented in detail and validated by several real measurements from parallel and distributed database processing.

Table of Contents

Dynamic Load Balancing for Parallel Database Processing

Wolfgang Becker

Faculty Report No. 1997 / 08

Institute of Parallel and Distributed High-Performance Systems
(IPVR), University of Stuttgart
Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany
Phone: +49 711 7816 411
Email wbecker@informatik.uni-stuttgart.de

1 Dynamic Load Balancing for Real Systems and Applications

The basic motivation to investigate and realize load balancing facilities comes from the very simple experience that several applications in a parallel system or computer network usually do not exploit the system very good. Instead, many compute nodes are idle while others are overloaded. The most well-known environments are workstation networks. Without buying new resources, load balancing as a system service can offer much more resources and better response times to all users, and can effectively avoid disasters that sometimes can arise from bad load distribution.

Many approaches try to equalize the number of processes on the compute nodes or equalize the task queue lengths at the nodes in the network - actually most of them just avoid that nodes run empty while others are busy. Load balancing becomes more complicated as the system consists of heterogeneous nodes, i.e. faster and slower processors, different amount of main memory and different number of cpus per node (SMP). Similarly, most real applications and tasks in the system are heterogeneous. They consume different amounts of resources, i.e. run short or longer times and use different percentage of the cpu time. Further, reality usually brings a mix of several, maybe parallelized, applications. Additionally, the arrival of tasks is not coordinated or planned, so load balancing has to cope with changing, unpredictable load profiles. Another complication is that parallel applications are not just sets of independent tasks, but the tasks are structured by precedence relationships. Load balancing should favor tasks on critical paths and tasks that entail large parallelism to reduce application response time and increase resource utilization. Other effects like context switch overhead or paging delays due to active memory usage severely influence the system performance, so load balancing should limit the overall system load appropriately. Load balancing should further be adaptive, i.e. identify the currently important performance factors, create estimations by profiling the system behavior and find the trade-off between load balancing overhead and improvement.

Although these requirements already make real world load balancing quite difficult, the following section explains that applications in various styles work on shared data and exchange data, which also has significant influence on the system resource utilization and application performance. Hence, load balancing should consider data, communication and networks appropriately.

2 Data Access and Communication Influence Application Performance

Data and communication are not of primary interest to the load balancing service. Load balancing is focussed on application response time and system resource utilization, measured by task elapsed times, application elapsed times, cpu utilizations, paging rates and perhaps disk and network utilization.

But applications often access remote data in form of network file sharing or global distributed databases. Access to data which do not reside on the local node, forces the application to request the data and wait for their arrival, and induces network load. The delay time could be used by other processes on the node, the network utilization should be coordinated with other network users to avoid congestion. If remote data access is performed by issuing remote procedure calls and make the data owner node execute the operations, then additional cpu load at this node will arise and other tasks on this node may be slowed down.

Application tasks usually pass intermediate results to their successor tasks, whether they send messages, or pass results to the client which passes them with the next server calls, or they store them into temporary files. This data flow also causes communication delays and network load. Tasks in parallel applications exchange synchronization messages to control the overall execution. These messages implement precedence relationships between tasks and can serve for result collection and aggregation. They also cause delays due to message latencies and network load.

Cooperation in parallel applications can, depending on the programming paradigm, take place by sending data to the task that needs them, or by accessing a virtual shared memory which locates the requested data and sends them to the requester. This data communication infers delays, network load, cpu load at remote node and search overhead to locate the data, which also yields computations and messages.

Finally, load balancing can cause data communication for process or application migration. Migrating a running process entails sending its data segment and maybe its code segment. This can be done in one strike or by demand paging, and increases the migration cost by additional delays and network load. Similarly, when migrating whole running (parallel) applications, whether moving all processes or starting the next

processing phase on other nodes, the communication of the application's active data set yields additional communication delays and network load.

Depending on network throughput and latency, and on the impact of the communication or data access speed on the application performance, load balancing can obtain large improvements by considering this.

3 How Can Load Balancing Consider Data and Communication

The relevant effects from data access and communication for a load balancing service are the additional delays, the reduced cpu utilization and the additional network utilization or network congestion. Now, load balancing needs a simple model and few critical factors to measure and to decide upon. The critical factors can be obtained by static data about the system resources and application behavior, dynamically by measuring the system and application behavior and by getting recent profile estimations from the applications. On system level, network delays (process to process latency per short message), network throughputs (bytes of continuous data streams per second) and the network topology (which nodes are directly connected, routing, hardware multicasting support) are of interest. One level above, the application data distribution within the system, average remote data access response times, the network utilization by communications or remote data access operations, and the cpu utilization and consumption for remote data operations are relevant. On application level, the message frequencies, the communication topology and the message sizes can be profiled or pre-estimated, and data reference patterns (record or data range names, read/write probability, repetition degree of access patterns) can be obtained by observation or from direct application hints. Finally, general estimations about applications' sensitivity to communication power (throughput, latency, temporary deviations) are interesting as guidelines for adaptive load balancing, to decide which factors are currently relevant.

Instead of developing a general approach to evaluate these informations for load balancing decisions, a brief review of existing techniques will be followed by a closer look at one of them, the HiCon approach:

- In general, there are some load balancing schemes which avoid communication, limit communication to the network capacity, or find the trade-off between increased resource usage by task distribution and parallel execution and the loss from communication delay and overhead. They do it by placing communicating tasks on the same node or cluster, by placing the tasks to where their data reside or by migrating / replicating the data to where the tasks are or going to be.
- A common static load balancing technique [10][11][31] is to match a task graph with a network graph. The task graph consists of the application tasks as nodes and communication relationships as edges, weighted by the communication intensity or frequency or amount. The network graph's nodes represent processors (compute nodes) and the edges direct communication lines, weighted by the throughput of the lines. Tasks can be assigned to processors by recursively partitioning the task and network graphs and always assign one task graph part to one network graph part. Both graphs are partitioned so that heavy weight linked nodes remain together in one partition. Matching obeys boundary conditions like maximum node load, maximum node load skew, maximum network line utilization etc. Alternatively, the graph matching can be done immediately by solving the corresponding optimization problem [29][35][12][36][25]. Recent research on supercomputers with extreme low latency networks [33], however, show that it can be profitable to distribute communicating tasks, because message passing between nodes is faster than the context switch from sender to receiver process within one node.
- Static load balancing methods that assign groups of tasks with precedence relationships to a set of processors in order to minimize the finish time of the last task, can be extended to consider also the data flow between the tasks [2][13][15][22][26]. Without looking at data, these algorithms place the largest tasks or the tasks on the critical path onto the fastest processor or onto the next available processor. Data flow is included in the model by adding a delay to the start of each task (the amount of time depends on the network speed and the data volume), if the task gets data from a predecessor that executed on another node. Hence, successor tasks are preferably placed on the same node as their predecessors.
- Transaction routing schemes place transactions to the database server that has the largest portion of the required data [3][38]. Therefore, load balancing knows the location of the database partitions and the average data reference patterns of the most important transaction types. Besides distributing the computational load by equalizing the number of database requests per unit of time between the proces-

sors, the transactions may not be distributed by chance but - as far as possible without compute load skews - to the node where most of the addressed database partitions reside. More sophisticated methods have an explicit load model including the cost of remote database requests (subtransactions), and send each query to the node where the sum of compute times, message delays and remote compute times is minimal. Further, the cost factors of communication, execution and general additional overhead per remote execution can be adjusted by tracing the system behavior and periodic regression analysis.

- Similarly, dynamic load balancing can receive data reference pattern estimations per task from the client [8], can generate / correct these estimations by observing the real access profiles [7], can dynamically observe the system data and copy distribution, and can observe the real data movement costs for remote accesses. Using this information, the expected time for data access on each available server can be estimated and compared.
- Dynamic load balancing schemes can estimate the data communication portion of tasks or task types by tracing the cpu utilization divided by number of running processes or observing the recent cpu usage of tasks [33]. Tasks that do not fully need their cpu share, either perform I/O or communicate a certain amount of time, and hence their nodes can take more concurrent tasks.
- While the techniques above served for task placement decisions, load balancing can also consider data communication cost for task migration decisions [32]. So, selecting a suitable process for migration from an overload node can take the one with the smallest data segment or the smallest number of accessed local files. Thus, the data communication entailed by the migration is kept small.

4 Load Balancing Runtime Support for Parallel Database Systems

Today's general load balancing systems are usually queueing and remote execution facilities or process migration facilities. They ignore the existence of data or do not consider them in a sophistication that makes them applicable for parallel database systems. So it is interesting to describe the architecture of one of the major general systems that can cope with parallel database operation, the HiCon load balancing model [8]. HiCon load balancing is application independent, but was designed especially for data intensive and cooperating complex parallel applications. The main concepts emphasize the key issues for dynamic load balancing systems:

- *Centralized and Distributed Structure*. Centralized load balancing (often called 'global') provides one central agent for collecting load information and for decision making [14][16][18][28]. It has several strong advantages: The measurement informations and predictions consistently reside in one place and may not be distributed or replicated among the system which would cause message traffic and could lead to outdated, inconsistent informations and contradictory load balancing decisions. Centralized load balancing can exploit the global knowledge about system and application behavior and can utilize sophisticated strategies. Centralized load balancing usually comes along with central task queueing. This allows for load control, because tasks can be queued until enough resources are available. Otherwise, heavy load situations saturate the system and reduce the throughput due to process switching, memory paging and network congestion. Central queueing also enables late decisions, because tasks may not be assigned when they are born but when they are removed from the queue for immediate execution. Central load balancing can avoid load imbalances before they arise, whereas distributed load balancing always has tasks waiting / running at the node where they originated and try to defeat the skews.

  Distributed load balancing (often called 'decentralized') becomes necessary for large parallel and distributed systems. These concepts try to keep load balancing efforts (resource consumption as well as delays) constant regardless to the system size, prevent single points of failure by local decision autonomy and reduce information and task exchange between nodes by simple, restrictive load balancing policies [4][23][27][30].

  When comparing centralized and distributed approaches, it must be noted that a stable heavy load by long running tasks is no problem for centralized approaches, because there are not many decisions. Task arrival bursts are critical because they cause much work for the centralized component. But, without centralized load balancing they generate severe load skews because they usually arise from one or few sources, e.g. parallel applications. So, centralized load balancing itself is overload but prevents load skews while distributed techniques will encounter a long period of slowly stepwise equalizing the skews.

Ideally, load balancing should have a mixed structure, i.e. a centralized agent per closely coupled cluster and distributed cooperation between neighbored clusters [6][19][20][39] or even a hierarchical structure [1][37]. The HiCon load balancing system provides one central component per cluster while neighbored clusters cooperate decentralized. Central task queueing takes place within a cluster; between clusters whole executing applications (task groups) are migrated if a significant load skew is detected. Between cluster, the HiCon system does not migrate executing tasks, but just routes all following tasks of this application to the neighbor cluster.

- *Client / Server Execution Model*. The HiCon approach provides a load balancing service for client - server processing. Tasks are issued by clients and are queued and assigned by the load balancing component to appropriate server processes of the respective server class. The concept supports mixed parallel applications and multiuser operation.

  It is important that the execution model for the load balancing decisions matches the execution profiles of the target application area. It should restrict itself to few relevant characteristics and measurement entities. The client - server execution model has been established as the basic processing concept for database management systems and for almost all large, distributed applications. Especially in database processing, fixed SPMD-style problem decomposition onto processors is not flexible enough, because it does not work well for large and small problems, for parallel queries and multiuser operation or for complex structured queries with unpredictable intermediate result sizes. Server class calls, however, allow for a flexible dynamic task distribution within parallel and distributed systems, because each call may be assigned to an arbitrary server instance on any node.

  For load balancing purposes, client - server has the additional advantage, that server classes provide a natural grouping of tasks that show similar profiles. So, load balancing can observe - and later on predict - the execution profiles and data access patterns per server class. The decoupling of applications into discrete tasks (server class calls) is also a simplification for the load model, because the behavior of whole, complex structured and communicating processes is hard to model appropriately. The concept of statically allocated server instances (processes) allows for rather fine grained task distribution and high parallelization degree, because a remote execution does not necessarily incur process start and connection setup overhead.

- *Flexible Data Management System*. For parallel database processing, load balancing additionally needs to have an idea of data, i.e. a model of how tasks operate on data and how data can be accessed remotely or can be moved or replicated and what resource consumption and delays come along with it. This was explained above.

  The approach of a virtual shared memory / data store fits very well into database processing: Within the HiCon system, tasks communicate by accessing global data items, either shared within an application or globally between all applications. The global data can be arbitrarily structured, volatile or permanent. No remote data access operations are performed, but the data items are migrated or replicated to the requesting task on demand. Therefore, a distributed runtime system supports transparent data partitioning / replication / migration, realizes a global database or virtual shared memory with changing data owner and probable owner search concepts, and provides consistent data access by locks and copy invalidation. The HiCon load balancing concept does not require this data management component. It just gets informations from it and knows about its facilities. Arbitrary database management or virtual shared memory components could be employed. It must be stated that none of today's commercial database systems provides dynamic data migration and replication in a comparable degree of flexibility. So, load balancing must know about the facilities of the data management system which the respective application uses. The important issues for load balancing are the efforts for data communication, the location of the data and the expected data access patterns of the application tasks.

  For illustration, the scheme of the runtime data management system is described which is currently used within the HiCon system. It is a general, flexible approach to management of logically common, physically distributed data. Data granularity is defined by the application. It can be few bytes, records in main memory or in a file or even whole files. The owner of a data record is the server who performed the latest update operation on it. This server is asked for copies and may grant ownership to other servers for updates. Copies are requested for read-only access and for updates the distributed copies are invalidated. This policy optimizes the data access cost, provided that the servers show a certain data reference locality: Repeated read access on local data copies is cheap and the number and distribution

of copies automatically adapts to the read / write proportion of the accesses. Load balancing tries to increase this access locality by assigning tasks to where the required data or copies of them reside. To ensure consistency, servers precede their data access by acquiring appropriate read / write locks.

The concept of cooperation via common data enables a quite flexible distribution of the tasks among the system, even for server classes that are not context free. For load balancing, the common data are the objects where it can include data affinity in its cost model. And this model comprises in a simple way communication between tasks, passing of intermediate results (pipelining) and access to common, global, maybe persistent data.

- *System Load Measurement and Load Pre-estimations*. It is important that load balancing acts dynamically by measuring the current system load. Especially in multiuser environments like OLTP there is no central a-priori planning but tasks arrive uncoordinated. For balancing large complex queries, dynamic load balancing is also important, because the actual amounts of data and computing efforts cannot be pre-estimated accurately. Further, load balancing should be adaptive for three reasons:

  1. Adaptive generation and correction of pre-estimations for task profiles and system load behavior. Dynamic load balancing decisions base on informations about application requirements and system utilization. These informations are often inaccurate or unavailable. Even load balancing concepts that make no explicit assumptions about the future task or system behavior but react on current measurements only, implicitly employ an extrapolation of zeroth degree, i.e. assume constant behavior. More sophisticated concepts can use load profile estimations from their own observations or from applications [17][21][24][32][34]. Comparing the previous estimations with the actual system behavior, load balancing can assess the accuracy and the value of the estimations and can correct them appropriately. For example, in the area of transaction routing [38] periodically perform regression analysis to determine the correlation between assumed and actually observed response times, and therefrom reduce the derivations for further routing decisions by correction factors.

  2. Regulation of difficult decision factors by feedback. Another weakness of dynamic load balancing is the vague execution model on which the decision algorithm is based. In real, parallel and distributed systems and large real applications it is extremely difficult to capture all effects and dependencies within a compact model. Because decisions have to be committed at runtime they must restrict to simple execution models. Hence, the correlation between the factors which load balancing considers (e.g. processor run queue lengths) and the overall throughput which is to be optimized, is not strong enough in all situations. The costs for remote data access, for example, depend on the size and complexity of the data, on the network utilization, on the lock wait times and on the utilization of the remote data owner - factors hard to merge into a compact formula. So, load balancing has to compare the actual effects of such decision factors in certain situations and globally adjust further decisions by appropriate correction factors, not knowing detailed reasons.

  3. Adaptive optimization of load balancing's cost - efficiency proportion. Load balancing profit is curtailed by its overhead, because it causes compute load, communication load and delays for information collection and decision making. Without adaption, load balancing assumes that its overhead is always appropriate and worthwhile which is not true for all situations and load profiles. Hence, load balancing must minimize its efforts, or even better, observe and regulate its cost - efficiency proportion. Decentralized load balancing, for example, should exchange less load informations in times of overall heavy system load, because there should not be many migrations anyway. Similarly, it should switch from the more expensive sender initiation to receiver initiation, i.e. not the many overloaded nodes should start searching for an underload node but the few idle nodes should pick up load from some overload colleague.

  The HiCon load balancing model employes a couple of adaption techniques, like adaptive correction of task size and data access profile estimations of the clients, adaptive determination of current remote access cost for certain data types in the current situation by feedback or adaptive observation of the load balancing component's compute load and decision delays with according simplification of the load balancing policy.

- *Critical Paths and Task Priorities*. To optimize the throughput of uncorrelated concurrent tasks, it is sufficient to consider each task isolated together with the load of the compute nodes. However, especially in large parallel applications there are dependencies between the tasks that should be considered by load balancing. Complex execution graphs in relational database queries are a good example. While several

static scheduling algorithms have been developed, the integration of such dependency considerations into dynamic load balancing decisions rises different problems. First, the precedence relationships are not given at system start-up but arrive every now and then during runtime and have to be integrated into the current situation. Second, there is uncoordinated multi user bustle, i.e. precedence relationships maybe known within task groups, but the groups as well as other unrelated tasks arise and run at arbitrary times. The third is the inaccuracy of the precedence relationships announced by the clients. Missing tasks, violated precedence relationships and wrong task size estimations should not severely destroy the profits of the load balancing decisions.

Despite these problems, dynamic load balancing should be able to profit from knowledge about inter task dependencies. E.g. the HiCon approach [5] decouples the priority calculation from the assignment decision and from the consideration of data affinities. Clients can dynamically announce arbitrary task dependency graphs containing precedence relationships and task size estimations. Load balancing therefrom computes and stores task priorities according to the critical path algorithms including a special prioritizing of tasks that entail large parallelism. The critical path is this path along a task's successor tasks which adds up the largest task sizes; it determines the response time of the whole task group. Later on, arriving tasks that are recognized as previously announced are sorted into the central task queue according to their priority. While residing in the central queue, the priority of tasks grows linearly. Tasks not belonging to a group just get a priority proportional to their estimated task size.

- *Consideration of Data Affinities*. In the HiCon model, tasks are queued centrally per workstation cluster, and load balancing assigns tasks in advance to servers for a certain amount of time. Selecting the best suitable server S for a task is guided by the minimum response time and the maximum system throughput, weighted depending on the overall system load.

  It is instructive to take a closer look into the decision model, as far as it is concerned with data affinity consideration. Following formula shows the general cost model which is used to determine the best suited server instance for a task execution: $S = MIN_s(t_{compute}(s,a) + t_{dataComm}(s,a) \times d_{overrate} + t_{remWork}(s))$

  $t_{compute}(s,a)$ is the expected elapsed time to process task a at server s. It is computed by the estimated task size (instructions), the expected available processing power at server's node at the estimated task start time, and an adaptive correction factor. The available processing power is derived from the processor speed, weighted by task's floating point and integer demands, the number of SMP processors on the node and the number of concurrently executing tasks on the node, weighted by an adaptive factor estimating the cpu utilization of these task types. This factor considers the communication portion of the tasks' execution, and is updated by exponential smoothing from system level cpu utilization measurements.

  $t_{remainingWork}(s)$ is the expected time until server s will have finished all the tasks currently in its local queue. It is the sum of the expected calculation and communication times of the tasks queued at this server minus the time the server has spend on its current task up to now.

  While the main formula determines the server with minimum task response time, it can be shifted towards the server which gives optimum resource utilization by simply overrating the data communication cost. Factor doverrate overrates the costs depending on the overall system utilization by application tasks (which can be measured by the number of tasks queued centrally in the cluster), and on the business of the cluster load balancing component (which can be measured by the number of events waiting to be dispatched). Data communication overweighting achieves that less parallelism is exploited and data affinity becomes more important. In consequence, it reduces communication cost and ensures full cpu utilization, which are important issues in high load situations when global system throughput is the main optimization goal.

  $t_{dataComm}(s,a)$ is the expected elapsed time for data communication during task a's execution on servers. It basically sums up the access cost to non-local data records:

  $$t_{dataComm}(s,a) = dataTimeAdapt_{tt} \times \sum_{i=1}^{N} \sum_{j=1}^{N_i} t_{dataAccess}(s,a,i,j)$$

Data access patterns are provided by clients as set of N probable accesses to ranges Ni of data. Per announced data reference dataRefi,j(a) with given probability for write access, following costs are expected: $t_{dataAccess}(s,a,i,j)$ =

0	if *ownerServer(dataRefi,j(a))=s*,
dataRangeWriteProbi(a) x $dataCommCost_d$	if $s \in$ *copyHolders(dataRefi,j(a))*,
$dataCommCost_d$	if *ownerServer(dataRefi,j(a))* $\in$ *local cluster*,
$dataCommCostRemote_d$	otherwise.

Data access cost are differentiated between intra and inter cluster access. For read access a local copy is sufficient, for write access the original is required. These things depend on the underlying data management system. *dataCommCost* (and similarly *dataCommCostRemote*) is adjusted by observing real elapsed times per remote access to this data type, with exponential smoothing.

ownerServer and *copyHolders* are information tables telling the probable current data distribution over the clusters and among the servers within a cluster. Each time a task is assigned, the probable data distribution is updated according to the task's reference estimation, and every now and then the data management system sends partial updates of the real current data distribution.

While the formula for $t_{dataAccess}$ estimates the data access costs, the comparison between the servers uses a modification, where copy access costs are reduced by the factor dataReadWritecs to encourage the creation of copies for repeated data accesses. This factor is regulated by observing the average number of read accesses to a data type *d* between two write accesses.

$dataTimeAdapt_{tt}$ is another adaptive correction factor per task type, which is updated by observation and exponential smoothing. It represents the relationship between the total data access cost per task execution and the cost estimated by the formula above, based on the client's estimation and the estimated data distribution and the estimated network power (cost per remote data access).

For migration decisions between clusters, HiCon load balancing also considers the cost for moving the data to the remote cluster: Applications are migrated only (among other restrictions), if their expected remaining processing time is larger than the migration cost for the recently accessed data by the application. The costs are estimated similarly to the formulas shown above. This is important to avoid unworthy migration efforts.

5 Experience

Concluding, the experiences from a number of real measurements that have been performed with the load balancing system, show the correctness and necessity of the techniques. Overall, the HiCon model was shown to be successful for different application domains like numerical simulation, image processing and database operations in various demanding configurations and load profiles.

Looking at parallel database applications executing dedicated on a workstations cluster, sequential execution, random task distribution and complex load balancing, e.g. according to the HiCon concept can be compared. In workstation clusters, communication and remote data access is rather expensive compared to parallel machines. Hence, the communication to processing power relationships will be valid for the near future and become valid for parallel systems too, because with growing processor power the data locality and caching issues are becoming increasingly important.

Random distribution across about five nodes can - compared to sequential execution - give significant acceleration for different applications: response time reduction to 40% for breath first recursive graph searches, reduction to 37% for complex database query execution. A load mix of database operations on R-Tree access structures - a sequence of parallel polygon inserts followed by a number of parallel spatial join operations can be accelerated to about 47% of sequential response time. Sophisticated load balancing, however can, by considering the different host speeds, the actual host loads the different task sizes and the expected data communication, reduce the response times to 27% (search), 21% (complex query) and 42% (R-Tree operations) compared to sequential processing. The additional 5% gain at the R-Tree operations, for example, results mainly from reducing the exploited parallelism in phases of high insert activities.

In multiuser operation, several of the respective applications were thrown concurrently into the system. Unbalanced execution means that applications executed sequentially on one of the nodes each. The response time then depends on the slowest node in the cluster, because all nodes got roughly the same

application load. Simple round robin load distribution even sometimes deteriorates the throughput, according to the common experience that in heavy load situations the system should not be disturbed unnecessarily by load balancing. Sophisticated load balancing however, can give enormous improvements (response time reduction) to about 57% (searches) and 60% (complex queries) compared to unbalanced execution. In the search applications, the data affinities are less important because the runtime system automatically distributes copies of the required data to the nodes, independent from the load balancing policy, which is a good idea for the large sets of read-only data. Hence, in the search application it was more important to avoid idle times by distributing tasks equally both looking at estimated task sizes and actual node loads. A closer look at the complex query executions shows that the concept of prioritizing tasks on critical paths is the reason for 10% throughput gain while the other earnings result from data affinity consideration.

In a very heterogeneous load mix, consisting of two parallel image recognitions, two parallel database query scenarios and two parallel finite element analysis calculations on a workstation cluster, load balancing has to face all different kinds of complexities as described in the first section. While “first free” load balancing (assign each task to the first server that becomes idle) yields 93% of the execution time compared to random task distribution, HiCon load balancing could reduce the overall response time to 42% of the random distribution. Detailed analysis told that the achievements mainly resulted from proper utilization of the resources in the overloaded system, and significant network delay reduction while all cpus could be kept busy.

If several clusters are coupled, communication is usually even more expensive and should be considered more carefully. As an extreme case, parallel applications were distributed across workstation clusters located in Stuttgart, Bonn, Toulouse and Belfast [7][9]. The wide area network imposed large latencies in the order of 100 ms per message, compared to 0.5 ms within a cluster. Seven parallel image recognition applications were started within one cluster with a distance of several seconds between each. When choosing applications randomly for migration between clusters, the overall elapsed time even increases to 105% compared to no inter-cluster load balancing. When selecting the best suited application, including the consideration of the data movements, migration can reduce elapsed time to 60%, and if migrations are limited to those where it could be worthwhile (with respect to the data movement cost and the expected remaining application run time), load balancing can reduce to 57% of the execution time.

Metacomputing views coupled systems as one large computing resource. Measurements with distant workstations in Stuttgart and Toulouse configured as one cluster (note that load balancing still can distinct data communication cost between nodes in and between these clusters), gave interesting results [9]. While ignoring data affinity was a complete disaster, several complex parallel database queries could be fruitfully distributed across Europe. Load balancing stuck mainly to one cluster unless routing was switched to a faster ATM network. Sometimes it could be observed that load balancing first kept the data intensive tasks within one cluster. After the applications unfolded their parallelism and the overall load increased, load balancing first spread few tasks across Europe. With the time, however, these tasks pulled the execution towards the remote cluster, because successor tasks operate on the intermediate result data of their predecessor tasks, encouraging load balancing to perform further executions also at this site.

The benefits from task priority consideration where evaluated by executing complex parallel database query graphs on a workstation cluster [5]. Additional problems compared to most static scheduling setups are different node capacities and the fact that there are much more tasks than available processors. In summary, the trials showed that ignoring priorities or using priorities simply according to each task’s size is insufficient. Taking critical paths and entailed parallelism into account results in 5 to 10% increase of throughput. In a multiuser access environment, i.e. several query graphs executing concurrently, the dependency considerations yield even more improvement for following reason: In single user mode, simple load balancing can achieve a nearly ‘highest level first scheduling’ like distribution because the executable tasks arrive in this order. In multiuser concurrence however, load balancing has to prefer - and reserve capacities for - tasks from other graphs which are more urgent, although they may have arrived a bit later.

6 Conclusion

Dynamic load balancing for parallel and distributed database management systems is a key to efficient execution of large and complex queries that run dedicated or in multiuser concurrence. This paper introduced the data communication and remote data access issues for dynamic load balancing. It compiled the basic approaches and techniques to face these challenges. After this, the data communication related concepts of the HiCon system were presented in detail, and some measurement scenarios illustrated the need, applicability and the benefits of the approach. An important conclusion is that the consideration of data and communication is actually important for load balancing, however mostly ignored up to now. Load balancing facilities can benefit significantly by considering data affinities and communicating task groups appropriately. The HiCon concept was presented in detail as major representative for sophisticated and successful policies that are applicable for parallel database processing.

References

1. I. Ahmad, A. Ghafoor, G. Fox, Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers, Journal of Parallel and Distributed Computing 20, 1994
2. F. Anger, J. Hwang, Y. Chow, Scheduling with Sufficient Loosely Coupled Processors, Journal of Parallel and Distributed Computing 9, 1990
3. A. Barak, A. Shiloh, A Distributed Load-balancing Policy for a Multicomputer, Software-Practice and Experience Vol. 15 No. 9, 1985
4. K.Baumgartner, B.Wah, A Global Load Balancing Strategy for a Distributed Computer System, Workshop on the Future Trends of Distributed Computing Systems in the 1990's, 1988
5. W. Becker, G. Waldmann, Exploiting Inter Task Dependencies for Dynamic Load Balancing, Proc. IEEE 3rd Int. Symp. on High-Performance Distributed Computing (HPDC), 1994
6. W. Becker, J. Zedelmayr, Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures, Mitteilungen GI, Workshop Parallele Algorithmen und Rechnerstrukturen, 1994
7. W. Becker, G. Waldmann, Adaption in Dynamic Load Balancing: Potential and Techniques, Tagungsband 3. Fachtagung Arbeitsplatz-Rechensysteme (APS), 1995
8. W. Becker, Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen, Ph.D. Thesis, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
9. W. Becker, Fine Grained Workload Distribution Across Workstation Clusters of European Computing Centers Coupled by Broadband Networks, Faculty report 1995 / 9, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995
10. S. Bokhari, A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System, IEEE Transactions on Software Engineering Vol. 7 No. 6, 1981
11. N. Bowen, C. Nikolaou, A. Ghafoor, Hierarchical Workload Allocation for Distributed Systems, Proceedings Parallel Processing, 1988
12. N. Bowen, C. Nikolaou, A. Ghafoor, On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems, IEEE Transactions on Computers Vol. 41 No. 3, 1992
13. T. Chou, J. Abraham, Load Balancing in Distributed Systems, IEEE Transactions on Software Engineering, Vol. 8 No. 4, 1982
14. Y. Chow, W. Kohler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, IEEE Transactions on Computers Vol. 28 No. 5, 1979
15. J. Colin, P. Chretienne, C.P.M. Scheduling with Small Communication Delays and Task Duplication, Operations Research Vol. 39 No. 4, 1991
16. G. Copeland, W. Alexander, E. Boughter, T. Keller, Data Placement in Bubba, SIGMOD, 1988

17.M. Devarakonda, R. Iyer, Predictability of Process Resource Usage: A Measurement-Based Study on UNIX, IEEE Transactions on Software Engineering, Vol. 15 No. 12, 1989

18.K. Efe, B. Groselj, Minimizing Control Overheads in Adaptive Load Sharing, Proc. 9th Int. Conf. Distributed Computing Systems, 1989

19.D. Evans, W. Butt, Load balancing with network partitioning using host groups, Parallel Computing 20, 1994

20.P. Gopinath, R. Gupta, A Hybrid Approach to Load Balancing in Distributed Systems, Symposium on Experiences with Distributed and Multiprocessor Systems USENIX, 1991

21.K. Goswami, M. Devarakonda, R. Iyer, Prediction-Based Dynamic Load-Sharing Heuristics, IEEE Transactions on Parallel and Distributed Systems Vol. 4 No. 6, 1993

22.B. Indurkhya, H. Stone, L. Cheng, Optimal Partitioning of Randomly Generated Distributed Programs, IEEE Transactions on Software Engineering Vol. 12 No. 3, 1986

23.L. Kale, Comparing the Performance of two Dynamic Load Distribution Methods, Proceedings Parallel Processing, 1988

24.T. Kunz, The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, IEEE Transactions on Software Engineering, Vol. 17 No. 7, 1991

25.C. Lee, D. Lee, M. Kim, Optimal Task Assignment in Linear Array Networks, IEEE Transactions on Computers, Vol. 41 No. 7, 1992

26.T. Lewis, H. El-Rewini, Parallax: A Tool for Parallel Program Scheduling, IEEE Parallel and Distributed Technology, 1993

27.F. Lin, R. Keller, The Gradient Model Load Balancing Method, IEEE Transactions on Software Engineering Vol. 13 No. 1, 1987

28.H. Lin, C. Raghavendra, A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC), IEEE Transactions on Software Engineering Vol. 18 No. 2, 1992

29.V. Lo, Heuristic Algorithms for Task Assignment in Distributed Computing Systems, IEEE Transactions on Computers Vol. 37 No. 11, 1988

30.R. Lüling, B. Monien, F. Ramme, Load Balancing in Large Networks: A Comparative Study, IEEE Symposium Parallel and Distributed Processing, 1991

31.P. Ma, E. Lee, M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, IEEE Transactions on Computers Vol. 31 No. 1, 1982

32.W. Osser, Automatic Process Selection for Load Balancing, Master Thesis, University of California, Santa Cruz, 1992

33.R. Pollak, A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer, Proc. Int. Symp. Parallel and Distributed Supercomputing, 1995

34.E. Rahm, R. Marek, Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems, Proc. 19th VLDB Conference, 1993

35.C. Shen, W. Tsai, A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion, IEEE Transactions on Computers, Vol. 34 No. 3, 1985

36.B. Stramm, F. Berman, Communication-Sensitive Heuristics and Algorithms for Mapping Compilers, ACM SIGPLAN Vol. 23, No. 9, 1988

37.A. van Tilborg, L. Wittie, Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers, IEEE Transaction on Computers Vol. 33 No. 9, 1984

38.P. Yu, A. Leff, Y. Lee, On Robust Transaction Routing and Load Sharing, ACM Transactions on Database Systems Vol. 16 No. 3, 1991

39.S. Zhou, X. Zheng, J. Wang, P. Delisle, Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Canada, 1992