Adaption in Dynamic Load Balancing: Potential and Techniques

Wolfgang Becker, Gerlinde Waldmann
Institute for Parallel and Distributed High Performance Systems (IPVR)
University of Stuttgart, Germany
email: {wbecker,waldmann}@informatik.uni-stuttgart.de

Abstract

In this paper, we investigate the benefits of automatic adaption in dynamic load balancing schemes. Three main problem sources are identified and two general approaches that can improve load balancing by adaption are introduced. Then we propose several approaches for adaption in centralized as well as clustered load balancing architectures in the area of coarse grain load balancing for shared nothing architectures. The potential benefits of adaption are examined by running real parallel applications concurrently in a heterogeneous workstation cluster with load balancing support.

Keywords: dynamic load balancing, adaptive load balancing, workstation farming.

1 Introduction

Dynamic load balancing is expected to optimize the average response time of single tasks and the overall application throughput in parallel and distributed systems. Task placement and migration decisions must be made at application runtime due to missing global work load estimations. Therefore, dynamic load balancing exploits instant informations about system state and application behavior.

Parallel computers and networks of workstations will be the architectures to solve most of the big computational problems in the near future. Important applications are being parallelized and there is an obvious need for application independent automatic load balancing. It is very efficient to distribute an application especially for certain system configurations, but if it is ported to other platforms, if several applications are using the system concurrently or if the actual load distribution is not predictable in advance, this does not work any longer.

Application independent dynamic load balancing optimizes the overall system throughput and stabilizes response times by collecting system state and application progress informations and possibly exploit task load predictions. Based on this information it commits task assignment and task migration decisions that yield equal load on the processing nodes and minimize synchronization and data communication delays between related tasks.

2 Adaptive Dynamic Load Balancing

Up to now, the terms *dynamic* and *adaptive* load balancing are not clearly defined in literature. For our literature review and the classification of adaption techniques we will use the following definition. There is no principal difference between dynamic and adaptive load balancing, since both include runtime parameters in decision making, usually current load measurement values. One idea towards adaptive load balancing is to switch dynamically between few different policies. Well known approaches identify the global system state as lightly, medium or heavily loaded and employ different strategies for each situation. Another approach to adaption is to learn from the outcome of previous decisions, i.e. include older measurement informations into the decision process. Typical approaches correct decision parameters according to long term observations. For example, the execution times or resource utilization of jobs or tasks can be logged and reused later as pre-estimations for the assignment of similar tasks. Since dynamic load balancing is performed at runtime, more complex learning algorithms are not feasible.

published in Tagungsband 3. Fachtagung Arbeitsplatz-Rechensysteme (APS‘95)
May 1996, Hanover, Germany

These two general approaches which can be found in literature in various styles, arose from different problems with non-adaptive dynamic load balancing. We will divide our approaches into three challenges, although they are not completely disjoint:

Adaptive Adjustment & Correction of Profile and Load Predictions

Dynamic load balancing strategies base their decisions on informations about the application requirements and system utilization, which are often not exact or not available at all. Although many researchers claim to use no advance estimations about tasks or system behavior, but just react on instantaneous observations, they implicitly use at least a steady-state or linear extrapolation of task and system load behavior. More sophisticated approaches enable the exploitation of applications' work load profile estimations submitted at runtime. In any case, the challenge for adaptive load balancing is to generate or improve estimations about near future work load and task profiles for assignment and migration decisions. By comparing the expected to the real system behavior, load balancing can rate the exactness and importance of the informations, or even correct them accordingly. For instance, one of the routing strategies described in [19] uses a feedback control mechanism that periodically analyzes the correlation between the estimated and the actual response time by regression analysis, and compensates the deviations accordingly.

Determination of Diffuse Factors in the Execution Model by Feedback

Another weakness of load balancing strategies is the inexact execution model on which the decision algorithm is based. In complex parallel and distributed systems for real applications it is very difficult to capture all activities, dependencies and various secondary effects by a general, compact and closed model. Dynamic load balancing decision making and information management incurs overhead at runtime, must be fast and hence base on very simple models. So, important side effects are neglected or, more generally, the correlation between the items considered by the load balancing strategy (e.g. processor run queue lengths) and the overall throughput, which is to be optimized (e.g. by simply equalizing these queue lengths), is not strong enough in various situations.

Adaptive Control of the Relationship Between Load Balancing Overhead and Profit

Dynamic load balancing suffers from its communication and computation overhead and the additional delays due to information acquisition and decision making. Non-adaptive strategies implicitly assume that their efforts are minimal and worthwhile, which is not true for all load situations and work load profiles. So the challenge for adaptive approaches is to minimize the efforts for load balancing itself or even to automatically optimize the relationship between cost and benefit. For example, consider a decentralized sender-initiated load balancing strategy searching for lightly loaded nodes in the system if the local load exceeds a certain threshold. This search can be very expensive and without success if all nodes in the system are heavily loaded. The information exchange between the nodes produces additional overhead. In situations of heavy system load and small load differences between the nodes, load balancing efforts should be reduced.

3 Related Work

Several publications propose control features for dynamic load balancing that can be seen as adaptive. Decentralized, sender-initiated dynamic load balancing is the most common approach, where each node in the system executes the same strategy. Tasks are executed locally as long as the load index of the node does not exceed some threshold value. Overload nodes search a destination node for remote execution. The decisions rely on informations exchanged between the nodes. Many approaches derive and encourage the necessity of adaptive switching between strategies and dynamic adjustment of decision parameters from extensive sensitivity measurements.

In [4], [5] the performance of a simple sender-initiated method, where overloaded nodes initiate task transfer, is compared with a receiver-initiated method, where lightly loaded nodes perform load balancing. The measurements confirm the necessity of switching between the two strategies. If the system load is light to medium, sender-initiated strategies show better performance, while at heavy load situations receiver-initiation is preferable.

Several load balancing strategies are investigated in [6] and measurements yield two important results: Both the size of the load information exchange time interval between the nodes and the threshold value for the local load index, upon which tasks are migrated away, should depend on the overall system workload.

A dynamic load balancing strategy that automatically adjusts its parameters according to the system load state, is described in [12]. The interval size for sending load informations to the central information collector is adjusted depending on the load variation. If the system is in a steady state, the exchange interval should be large to avoid unnecessary information message traffic. If system load is changing frequently, the exchange period should be chosen small, so that load balancing can react in time. The threshold value for the transfer strategy is adjusted dynamically using the average system load and a normalized constant that determines the aspired degree of load balance between the nodes.

[11] improve a Contracting-with-Neighborhood load balancing strategy by automatic adjusting parameters, to achieve maximum processor utilization with minimum overhead.

Database transaction assignment strategies are examined in [13]. Because the informations on which routing decisions base may not be readily available, load balancing strategies should give good results although they work with inexact informations. The minimum response time strategy, where each transaction is assigned to the processing system promising the minimum response time, is therefore improved in different ways. One enhancement employes a feedback process to adjust future response time estimates. It monitors the discrepancy between estimated and actual response time and introduces a correction factor based on regression analysis. According to the load differences between the processing systems, load balancing in [8] switches between assigning transactions to the preferred processing system, hosting most of the probably required data, or to the system with the smallest load index.

4 The HiCon Approach to Dynamic Load Balancing

The HiCon concept, developed by the IPVR, provides dynamic load balancing support for complex, parallelized applications on heterogeneous workstation networks. It is application independent and includes several automatic adaption techniques covered in this paper. Each workstation cluster is managed by a centralized load balancing component; The clusters share their load by a decentralized policy. Central load balancing manages a central queue of executable tasks, which can be assigned arbitrarily to the processors or migrated to neighbor clusters. The centralized approach provides sophisticated global load control and throughput optimization, while the decentralized coupling of clusters ensures scalability [2]. Load balancing reacts on instant system load and application state informations, and also exploits advance load profile estimations. Besides proper utilization of the aggregated processing power, load balancing also takes the data communication costs between cooperating tasks and remote access to common data into account. Load balancing assumes client - server structured applications, and communication takes place via shared data structures within applications or global persistent data, which are both migrated and replicated by enhanced operating system services. For more detailed descriptions of the HiCon project and its various features we refer to [1], [3]. Figure 1 gives a rough survey of the whole system.

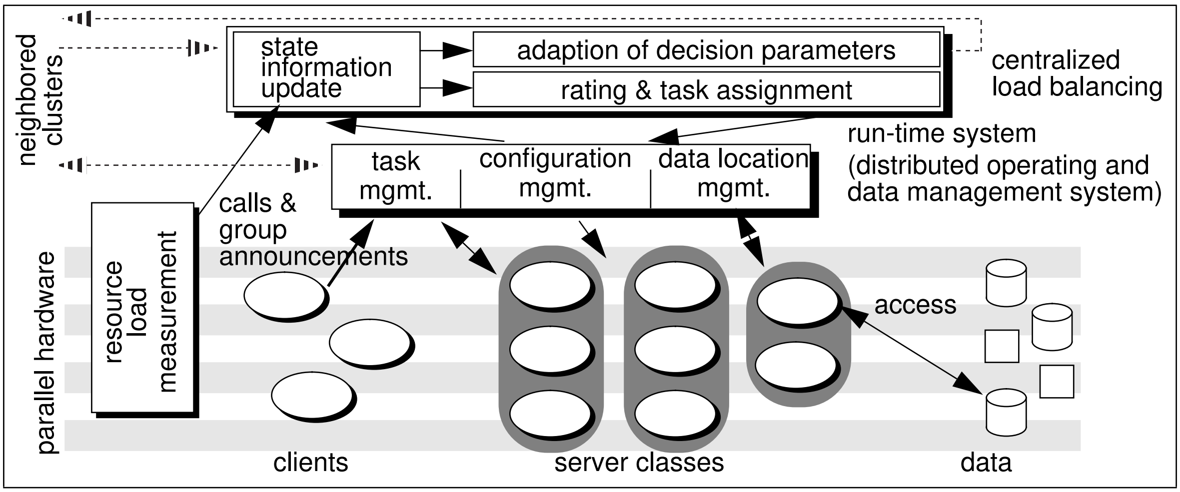


Figure 1: HiCon load balancing - system and application survey.

For the evaluation of the adaption concepts we investigated the real execution behavior of typical large important applications, within a prototype implementation of the HiCon load balancing concept on workstation clusters. The available applications are representatives for numerical simulations, image processing and commercial data processing. They are parallelized and nevertheless can run concurrently as homogeneous or heterogeneous load mix within the load balancing environment.

We restricted the measurements to a *Parallel image recognition* application. Image segmentation converts given pixel images into sets of homogeneous areas (polygons), usually as first step of a complete image recognition. The implemented algorithm takes four steps (Figure 2): Starting from an initial partitioning the square merge phase tries to combine neighbored squares as long as the resulting squares still build homogeneous areas. Parallel to this process square split operations refine squares that could not be merged, until each square represents a homogeneous area. Both processes are working concurrently with irregular parallelism on the image. The next step, polygon merge, merges as many as possible neighbored squares to arbitrary polygons. The last step, boundary extraction, yields the edges surrounding the polygons. The algorithm is typical for image processing and shows several computation stages, unstructured parallelism, differing task sizes and frequent data communication; load profiles depend heavily on the actual structure of the image. Because of the unstructured parallelism in varying degree and the intensity of data communication it is a non trivial problem for dynamic load balancing on shared nothing architectures.

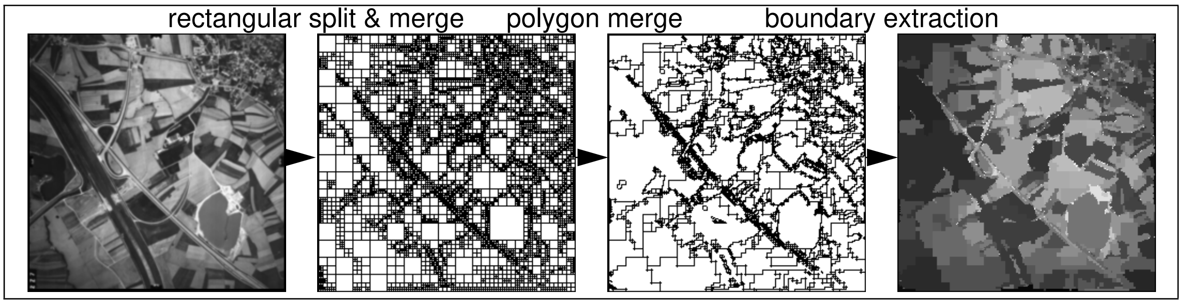


Figure 2: Steps of the image segmentation algorithm.

Further applications, like parallel finite element analysis, parallel breath first search in directed graphs, complex parallel database operations and database operations on geometric objects supported by spatial index structures, are available for the prototype environment.

5 Presentation and Evaluation of HiCon Adaption Techniques

Within the environment introduced above, we developed a set of adaption techniques, that cover the main well-known approaches and adapt or extend them for our load balancing approach and the target application types. In this section, the different advanced adaption techniques are described in detail. Measurement results prove the feasibility of the proposed concepts. The first adaption concept enables automatic optimization of the necessary load balancing efforts. The latter were developed to face missing or inexact load profile estimations and weaknesses in the decision model.

Automatic Avoidance of Load Balancing Congestion

This section concerns adaptive optimization of the relationship between load balancing overhead and profit. HiCon central load balancing provides substantial performance optimization for various applications and load situations by a rather sophisticated strategy [3]. Although we assume that this strategy is worth the effort, centralized load balancing can be congested by large systems and situations of heavy load [2] due to increasing costs for information aggregation and decision making. If clusters are configured too large and load balancing is overstressed permanently, then the clustering must be refined; But load balancing should cope with short term congestion situations by reducing its efforts from frequent information management and decision making.

In the HiCon approach, adaption reduces the complexity of the assignment strategy and the frequency of decisions, if the processor hosting the load balancing component becomes heavily loaded or load balancing becomes unable to make the required decisions in time. We identify congestion by the number of queued load balancing events that are currently not dispatched; The CPU utilization showed less correlation. Load balancing uses an instant counter as well as a long term, exponentially smoothed one. To react quickly on congestions, load balancing switches from the full strategy to a simple one as the instant counter exceeds a certain threshold. The simple strategy rates tasks just once at their arrival and does not update the ratings at each activation of the decision algorithm. It switches back to full load balancing, when the smoothed counter falls under a certain threshold. Second, load balancing assigns tasks to servers for some time in advance. This time should be reasonably small to achieve late assignments based on instant information, but has to increase according to the smoothed congestion counter, to get rid of centrally queued tasks (which must be rated several times) and to ensure that servers can work continuously.

For evaluation a cluster of 11 heterogeneous workstations with 22 servers was used. 15 parallel image recognition applications of differing problem size were thrown into the system. Two of them start immediately, causing moderate efforts for load balancing; After 50 seconds the other 13 applications start within a range of five seconds, generating lots of parallel tasks, that must be assigned carefully due to different task sizes and data affinities, hence able to completely overstress the complex load balancing strategy. The left of Figure 3 shows the load balancing cost without the adaption techniques for a period of time: The load balancing component has a rapidly growing queue of non-dispatched events, causing significant delays in task assignment, and idle servers. Load balancing consumes all the CPU time it gets on its host, instead of leaving as much as possible for the local servers. The right shows the behavior with integrated adaption, were the congestion could be recognized and defeated successfully. Load balancing still consumes much CPU time (profitable).

Figure 4 compares overall execution times for different techniques. Load balancing without adaption clearly suffers from congestion, and a general increase of the assignment look-ahead time by factor 100 does not help. Additionally replacing the decision algorithm to a cheaper one as the queue exceeds 40 or even 4 events, and switching back if the smoothed

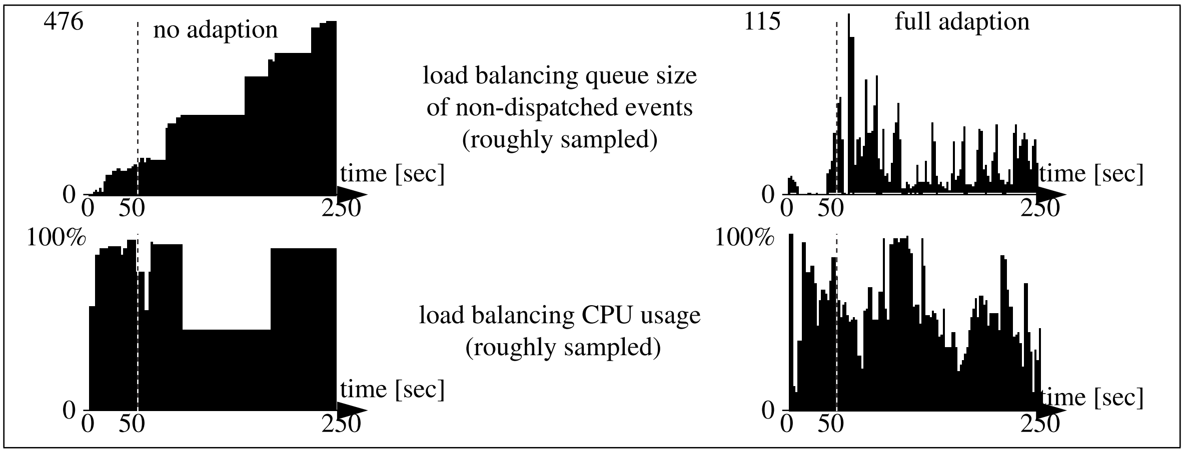


Figure 3: Congestion of a central load balancing component.

queue drops beyond 2, can successfully avoid the congestion. Adaption employed the simple strategy about half the time with switching periods of 3 - 15 seconds. The trials leads to following general conclusion: Complex centralized load balancing is worthwhile, as long as it can keep in time with information collection and decision making, but slight congestions can completely destroy the profits of costly strategies and must be avoided absolutely. Simple strategies must be chosen carefully, for the load balancing problem does not become easier, e.g. ignoring data affinities in this scenario can yield tenfold task execution times. In our approach load balancing just contents with decisions based on older informations.

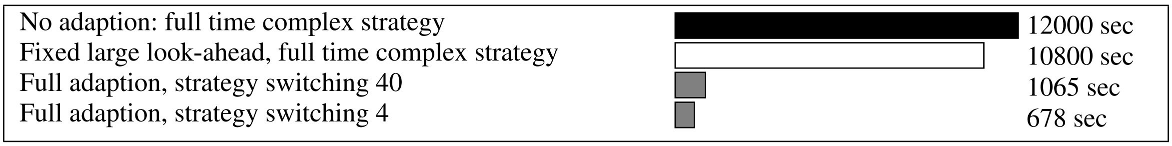


Figure 4: Comparison of execution times in a congestion prone scenario.

Adaptive Adjustment of Inter Cluster Migration Threshold

This concept concerns decentralized inter-cluster load balancing which is very simple compared to the sophisticated centralized load balancing policy. As the network connection between clusters can be very slow [12] and information is sparse compared to accurate intra-cluster balancing, inter-cluster balancing simply routes whole (parallel) applications from overstressed clusters to neighbor clusters which appear to be lightly loaded. Therefore, neighboring cluster load balancing components exchange aggregated load information. Adaption of the information exchange frequency turned out to be not necessary, because the overhead is negligible if only significant changes are multicasted.

Since workload migration between different clusters is expensive and entails a corresponding portion of data migration, load balancing agents try to distribute work load as far as possible among the servers within their clusters. Hence, a fixed threshold *T* ensures, that applications are migrated only in situations of significant load skews, i.e. if $activeAppls_N/\Sigma p_N < (activeAppls_{me} - (T+1))/\Sigma p_{me}$, where the number of applications running in the clusters are normalized by the clusters' aggregated processing power. Both values are informations exchanged between the clusters. A second, complex threshold requires that applications spend at least a minimum amount of processing time in the cluster before being migrated again, and that they have a minimum amount of time left to process. The latter condition also prevents the migration of small applications. In loosely coupled systems with communicating data intensive applications, the data transfer efforts following an application's migration are most critical, so the threshold must adapt dynamically to the network capacity and load and the applications' active common data volumes. HiCon load balancing chooses the best suited local application *a*, i.e. which was not processed yet or has maximum $w = \frac{processingTimeSpentLocal}{\Sigma AC_a}$ using $\Sigma AC_a = \sum_{d \in activeGlobalData_a} interClusterAccessCost_d$, and migrates *a* to least loaded neighbor *N* (see above), if the threshold criterion $w > 1 \wedge \mathit{remainingProcessingTime}_a > \Sigma AC_a$ holds. The currently expected migration cost for data types between clusters are regulated dynamically as presented below in the 4th technique. Estimation of applications' current global data volume can be obtained from the data management, while the clients give estimations of the applications' remaining amount of work.

For evaluation, a network of 4 heterogeneous clusters located in Stuttgart, Bonn, Toulouse and Belfast with 4 workstations per cluster, was set up to roughly evaluate this approach. 7 image processing applications entered one cluster in 5 seconds distances, causing significant load skew. Figure 5 shows the elapsed times for different approaches. The first approach without adaption ① migrates an arbitrary application to equalize the load, while the second approach without adaption ② always migrates the best suited application. The adaptive threshold can increase the throughput by another 7%.

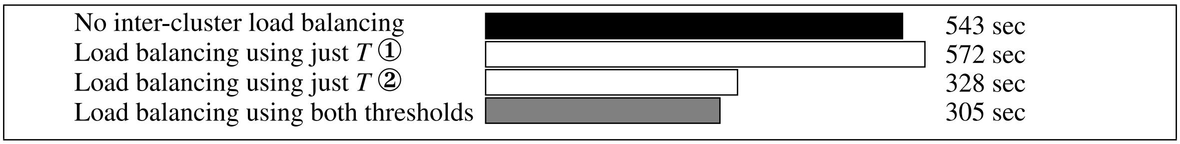


Figure 5: Effect of data communication sensitive migration threshold.

Adaptive Adjustment of Task Size Estimations

The decision algorithm uses task size estimations to select the best server for assignment. However, these informations, provided by clients at call time, are just rough estimations. Following adaption technique tries to improve the clients' declarations and the therefrom derived task processing requirement estimations for the different task types.

After task completion load balancing can determine from the real execution time $|T|$, how exact the client has estimated the task size $|T|_e$ and adjust a correction factor f_{cs} for further tasks of this type by exponential smoothing: $f_{cs} = (1-\alpha) \cdot f_{cs} + \alpha \cdot |T|_e / |T|$, where α is the smoothing weight. The clients' declarations for further tasks of class c, task type s will be corrected by $|T|_c = f_{cs} \cdot |T|_e$. This strategy works best, if the deviation between estimated and real execution time is approximately linear, which turned out to be a quite realistic assumption for different applications.

For evaluation of the dynamic adjustment of task size estimations, we executed two parallel image recognitions, partially overlapping in time. Centralized load balancing had to distribute the tasks across four heterogeneous workstations. Image recognition consists of four phases (task subclasses) as explained in section 4; For each of these phases a separate correction factor is managed.

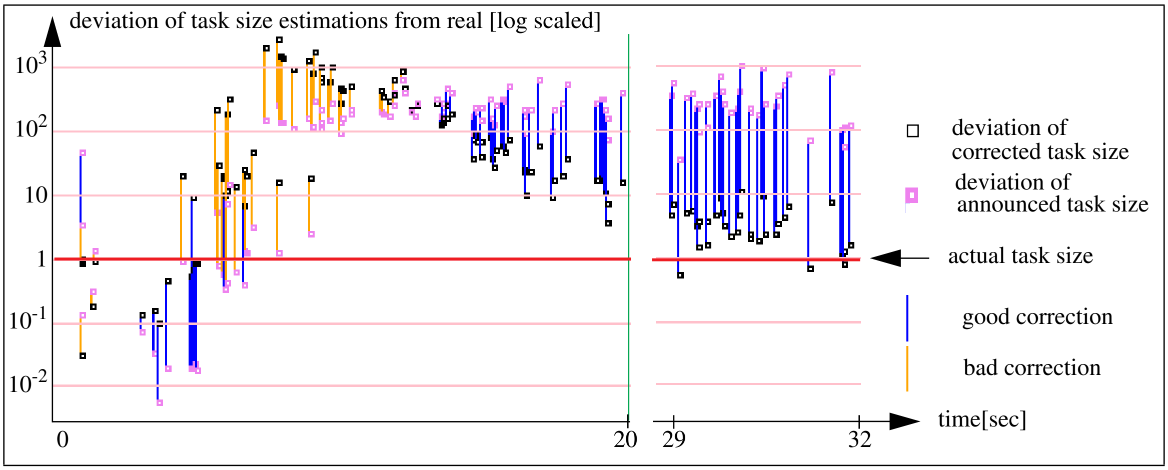


Figure 6: Adaptive correction of task size estimations by feedback.

Figure 6 shows, how load balancing adjusts task size estimations in the square split phase. The deviations of clients' task size estimations and the corrected estimations from real task size are drawn over the time in logarithmic scale. Altogether, the proposed adjustment strategy achieved maximum estimation improvements of factor 1000. At the beginning of the split calls the task sizes were underestimated by factor 100; Later on clients' estimations were too high. So the adjustment was occasionally worse, until the factor has been fit. Such effects depend on the structure of the recognized picture, so dynamic adjustment of the task size estimations is necessary. The adaptive correction of the task size reduced the execution time of the scenario by 8% from 333 to 276 sec. If a task type (e.g. a parallel loop) bursts all tasks, there is no chance for adaption within a single execution of the application, but further runs of this application type can benefit.

Adaptive Determination of Data Communication Cost

The cost for exchanging and copying data items between servers depends on dynamically changing factors that are hard to acquire and to consider appropriately, e.g. the current network utilization due to multiuser concurrence or application internal parallelism, current data sizes and lock duration times. Hence, the HiCon load balancing concept provides an adaption technique to determine data access cost per data type from recently observed costs by exponential smoothing, and uses this information to estimate the penalties for a task's execution on servers that usually don't have all probably requested data items locally available. The usage of instant values is useless due to huge fluctuations, but, as shown below, significant long term changes should be recognized.

For evaluation we again executed two parallel image recognitions, partially overlapping in time. Centralized load balancing had to distribute the tasks across four heterogeneous workstations. First a larger, more complex image is computed, and after 400 sec image recognition of a smaller picture is started additionally. To evaluate the benefits of adaptive communication cost estimation, the scenario was run with and without adaption. The measurement without adaption used the final values from the adaptive run, i.e. it got the best possible fixed values. Adaption reduced the overall elapsed time of the scenario by 9% from 950 to 860 sec: Load balancing recognized increased communications costs in application specific phases of high concurrent data access or high network load, and hence reduces the exploited parallelism or preferred servers with higher data affinity despite their CPU power and CPU load. Figure 7 shows the evolution of the data communication costs for the main data structures of the application during the execution:

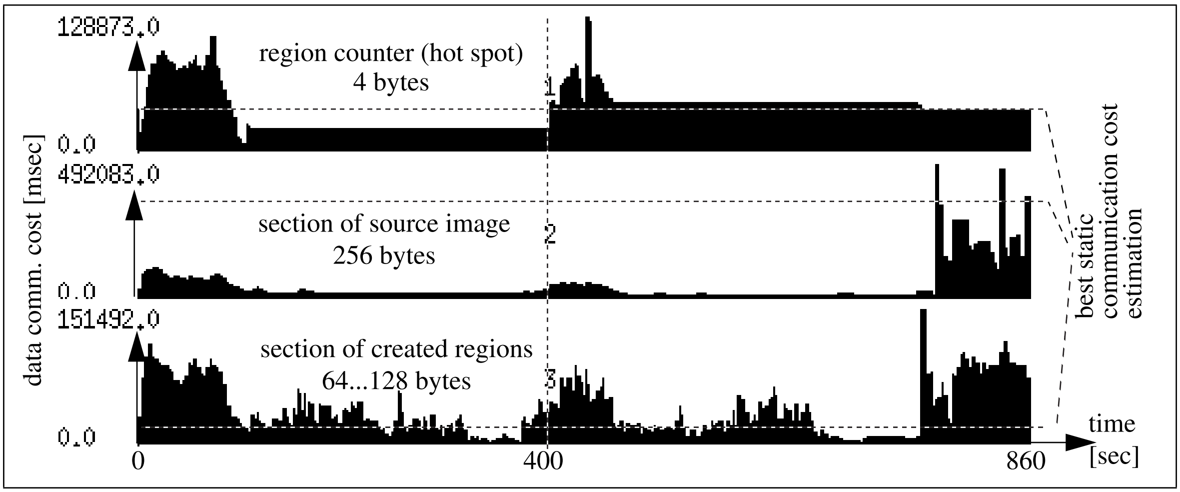


Figure 7: Dynamic adaption of remote data access cost estimations per data type.

During the quad split&merge phase of the image recognition the data access delays due to locks and transmission rose to 50 msec, because of high frequency of concurrent updates.

Data access cost also increased in the quad split&merge and in the boundary trace phases due to high degree of parallelism. The access costs during the smaller application (started after 400 sec) are lower, especially in the quad split&merge phase. Increasing communication cost due to multiuser parallelism cannot be obtained from this scenario. However, a similar scenario, overlapping the quad phases of starting two identical image recognitions, data cost increased by factor 1.5 on average.

Adaptive Correction of Tasks' Data Communication Portion Estimations

When choosing the best suited server for a task, the decision algorithm also considers arising data communication costs. Clients' estimations of data references are rough estimations and often faulty. It is useless for load balancing to keep track of the data parts actually accessed, because this will certainly change for the following tasks, e.g. think of the similar tasks within a parallelized loop, which usually work on disjoint data sets. However, from the elapsed wait time for non local data during a whole task execution and the average costs for the remotely accessed data (see above), the portion of missing declarations can be determined for future estimations. Non-adaptive load balancing implicitly assumes that data reference pattern announcements are correct and complete.

We introduce a correction factor to improve the estimated data wait time (which was derived from clients' announcements). This factor is updated per class and task type each time a task has finished, according to the formulas for the task size correction factor. For each task T the client may declare data ranges $p^1, \ldots, p^m$ that will probably be accessed during T's execution. The client may attribute each range with the assumed write access probability $wp(p^k)$. The communication cost for server S to access data item p_i^k of range part p^k is estimated by $d_{com}(S, p_i^k) = \overline{CO}(S, p_i^k) \cdot \overline{wp}(p^k) \cdot accessCost(p) + \overline{O}(S, p_i^k) \cdot wp(p^k) \cdot accessCost(p)$, where *accessCost* is regulated by feedback (see above), $CO(S,p)=1$ iff server S currently has at least a copy of data item p and $O(S,p)=1$ iff server S owns p. So, server S is expected a delay for remote data access of $D_{com}(S, T) = \sum_{k=1}^{m} \sum_{i=1}^{n_k} d_{com}(S, p_i^k)$. This D_{com} is corrected by a factor which is adapted much like the execution time estimations.

For evaluation we employed the same scenario as above. Figure 8 shows, how load balancing adjusts the estimations of data communication in the square split phase. The measurements for square split tasks show that deviation could be reduced by factor 20. There are some exceptions where the clients' estimations yield drastic underestimated communication cost (factor 10^6), but the adaption strategy does not - and should not - react on these exceptions. In the other phases of the image recognition an improvement of about factor 100 could be achieved.

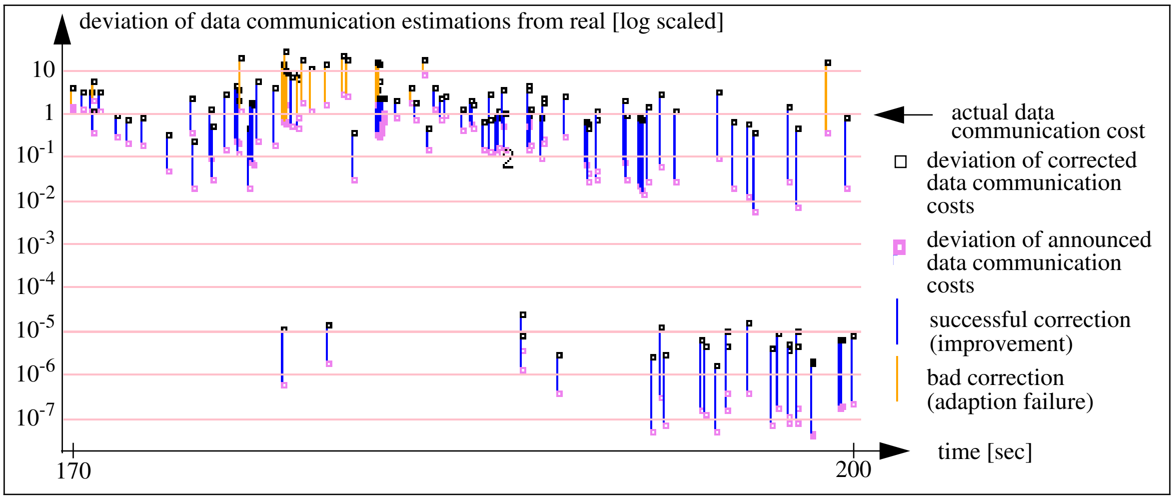
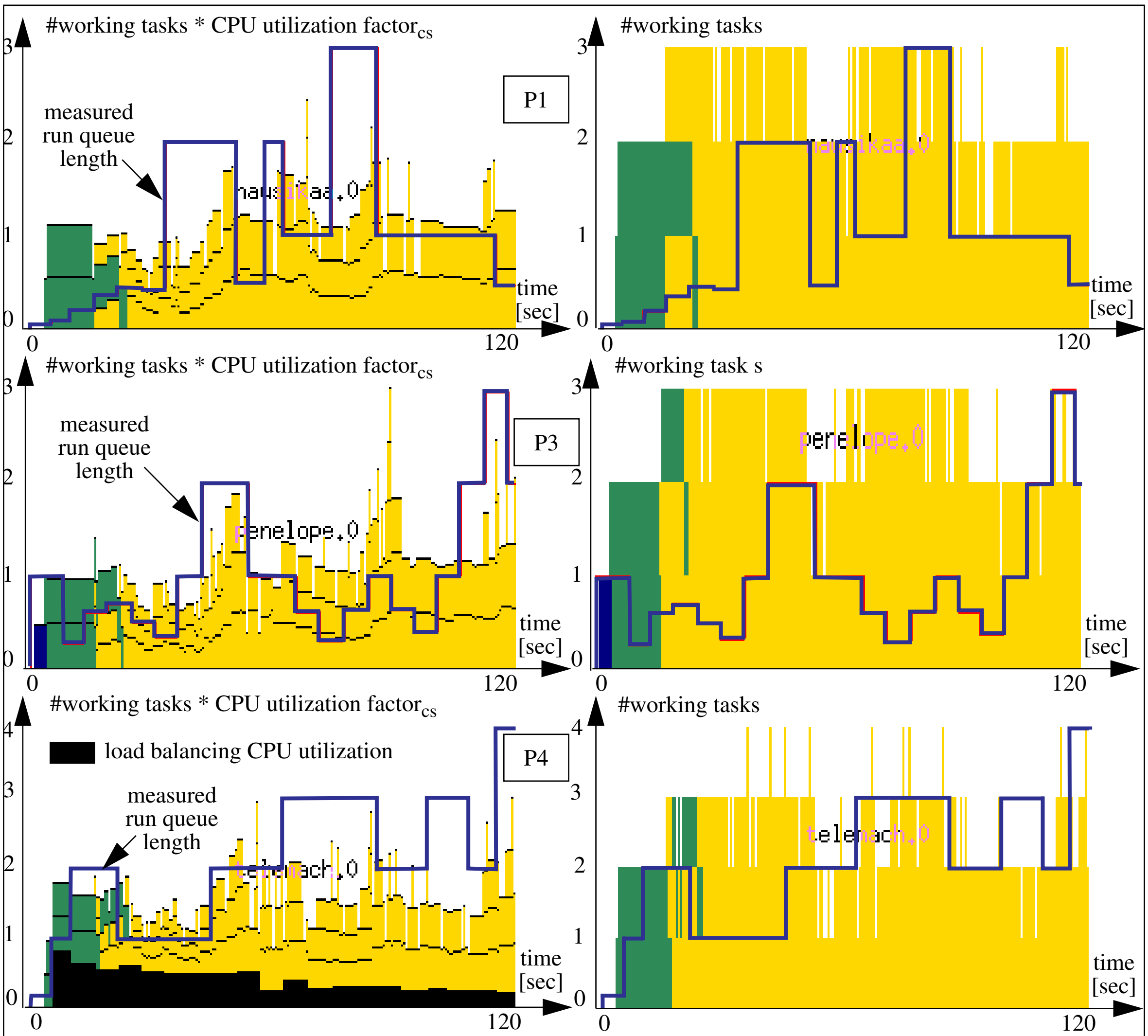


Figure 8: Adaptive correction of tasks' remote data access delays by feedback.

Adaptive Estimation of CPU Utilization from Task Executions and Load Balancing

For task assignment decisions, load balancing also considers the processor load factor caused by other currently active servers on that processor. The processing capacity can be divided by the number of active servers or by the average CPU run queue length. While the latter measurement, provided by the kernel every five seconds, is too rough and includes negligible load peaks, the first assumption often fails because most task executions are interrupted by wait intervals for disk access or remote data. So we use the CPU run queue length just to adapt the estimated average CPU utilization per server class and task type. Then, the expected processor load is estimated more up-to-date and accurate as the sum of active tasks' utilization factors. This also enables better near future extrapolation than the steady state approach. Further, the processor utilization by the load balancing component can be subtracted from the CPU run queue length, when adjusting the utilization factors.



(a) CPU load estimation with adaption (b) CPUload estimation without adaption

Figure 9: Adaptive processor load estimation.

For evaluation we employed a cluster of four workstations each hosting three servers and each executing an image recognition with different problem sizes. Figure 9 shows the estimated CPU utilization and the measured run queue length for some processors. On the left, processor load is estimated using the utilization factors, which are long term adapted by run queue length observation. On the right, load is estimated assuming that each active tasks fully utilizes the CPU, i.e.increments the run queue length by 1. Quad-split tasks (light gray) consumed only 30% CPU, because they spend most of their time for data communication. Hence, without adaption processor load is overestimated. The run queue length provided by operating systems is rough (integer) and shows strange peaks although already smoothed, and the arbitrary intervals do not reflect task execution starts or completions. So it should not be applied immediately for accurate processor load estimations. The effort of the load balancing component on P4 also changes: It decreased slowly, because the data distribution stabilized and less data movement events occurred.

6 Conclusion

This paper presented a comprehensive introduction to adaptive dynamic load balancing, reviewed the main potential and proposed generally applicable techniques. The proposed concepts were validated by real measurements within a shared nothing parallel computing environment. Integrating automatic adaption techniques into dynamic load balancing turned out to offer significant throughput improvement. While adaptive distributed load balancing has been investigated in several publications, we focussed on techniques for centralized load balancing. The evaluation was restricted to a few interesting scenarios and cannot cover all relevant aspects and situations. For example, adaption is not only able to adapt to values or behavior which are initially unknown, but also dynamically adapts to changing load situations or application behavior, which did not become evident from all scenarios.

References

1. W. Becker, *Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen*, Informatik Forschung und Entwicklung 1994-3, Springer Verlag, 1994.
2. W. Becker, J. Zedelmayr, *Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures*, Mitteilungen GI, Parallel Algorithmen und Rechnerstrukturen, GI/ITG Workshop Potsdam, 1994.
3. W. Becker, *Lastverteilung in Workstation-Netzen*, BI Special Issue on Parallel Computing, RUS, University of Stuttgart, 1995.
4. D. Eager, E. Lazowska, J. Zahorjan, A Comparision of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing, Performance Evolution, Vol. 6, 1986.
5. D. Eager, E. Lazowska, J. Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 12, No. 5, 1986.
6. D. Ferrari, S. Zou, *A Load Index for Dynamic Load Balancing*, Proceedings Fall Joint Computer Conference, Dallas Texas, 1986.
7. X. He,