

Quasi-Static Load Balancing in Local Area Networks

Ulrich Hofmann
University of Salzburg
Austria
uho@cosy.sbg.ac.at

Marek Krajewski
Technical University Dresden
Department of Informatics
Germany

Abstract

A network of LAN-connected workstations provides the means for load balancing (LB) between computers. For a system consisting of N stations connected by a multiaccess network the optimal, quasistatic, iterative, distributed load balancing algorithm for a general, system-wide cost function is set up. The algorithm is based on the gradient projection optimization method using information about sensitivity of the performance measure with respect to the task flows. A simple heuristics to accelerate the convergence is proposed.

1. Load balancing in computer networks

1.1. Motivation

Personal workstations are currently mainly used for research, software development and engineering applications. These powerful stations are considered mostly private resources under control of their users. However, in order to provide access to common resources and to enable information exchange, these private resources are interconnected by one or more LANs to form an integrated processing environment. Such workstations-based, distributed systems (WDS) have increasingly replaced large mainframe-based, multi-user systems. Because high speed local area networks (with speed in the 80-100 Mb/s range) and high performance network interfaces are already emerging, efforts to use efficiently this excess computing capacity are currently being undertaken.

However, many researchers report that a large portion of the capacity of such networks of workstations is not utilized and could be used by users who need *additional cycles*. For example Theimer and Lantz [ThL89] reported that one third of machines were typically idle in a similar environment; Nichols [Nic87] reported that 50-70 workstations were typically idle during the day in an environment of 350 workstations total; and the measurements from Douglis and Ousterhout [DoO87] show 66-78% of all workstations idle on average. In results from Mutka and Livny [MuL91] the workstations were available more than 75% of the time observed. These capacities were available not only during the evening hours and on weekends, but during the busiest times of normal working hours. We expect this low load level to become more pronounced as faster processors (>10MIPS) and multiprocessor workstations come to the market in the next few of years.

On the other side Livny and Melman [LiM82] have shown that in a system of n independent processors modelled as M/M/1 systems [Kle75], the condition in which a job is waiting for service at one processor while another processor is idle occurs 70% of the time, for traffic intensities ρ (the ratio of arrival rate to service rate) ranging from 0.5 to 0.8. This idle-while-waiting probability (or probability of the load sharing success) indicates the possibility of reducing the average process delay. With a global load sharing strategy for a distributed system, the occurrence of this probability can be reduced and, consequently, the overall performance can be improved (a M/M/n coupled queue provides perfect sharing in this case).

Many interesting *parallel and distributed algorithms* require distributed scheduling facilities that distribute these tasks among processors to speedup the execution by taking advantage of system computation abilities and resources. A cluster of workstations can be used as a parallel machine to run parallel applications [Kle90], [Stu88], [MuL91]. Kleinrock and Korfhage [KIK89] examined the response time for a model of a distributed program which executes on a cluster of workstations, where the workstations alternate between intervals of availability and non-availability. Other researchers have developed applications specifically for exploiting workstation capacity. 'Optimistic make' is a facility for the V-System that has been implemented to take advantage of idle workstations. 'Make' is a tool used primarily to create up-to-date executable programs from their source files. 'Optimistic make' is identical in functionality to 'make', however the file system is monitored for out-of-date file, and idle workstations are used to bring the files up-to-date. Parallel 'make' exhibits coarse grain parallelism. More tightly coupled parallel algorithms have also been successfully run on workstation clusters [ChS88], [FiM87]. For instance, a number of

parallel optimization algorithms, such as branch-and-bound or mathematical programming, are well suited to run in this environment. Unfortunately, scheduling schemes developed for (shared memory) multiprocessors, such as those studied in [SqN91] do not apply to more loosely coupled distributed systems, since they schedule too fine a granularity (communication delays are substantial even in a high speed network).

This and other motivating examples show that while many workstation-based distributed systems (WDS) have the potential to deliver enormous computing capacity, often rivalling that of the most powerful supercomputers, much of their capacity is generally untapped because of the inability of the WDS system software to efficiently share computing resources among workstations. To make fullest use of this capacity, distributed schedulers must be developed that are appropriate to the unique features of these systems.

1.2. Models for load balancing in computer networks

Many research efforts have demonstrated that even simple load balancing algorithms produce considerable performance improvement. Theoretical studies have covered two kinds of algorithms, which we will denominate as *static* and *dynamic* strategies.

The *dynamic* control problem can be considered as a Markov decision problem (MDP) [EpVe89]. This means, for each task arriving at a single station a decision is made, based on the instantaneous network status knowledge. For some finite or infinite horizon cost function it determines where the current task have to be sent to for processing. In particular, much attention was devoted to a specific class of the dynamic policies: the threshold policies [KriLi90], [EaLaZa86]. The idea behind this class of policies is to send the task to the remote station, when the local queue length exceeds some (static or dynamic) threshold. By studying the dynamic methods, the queueing theoretic models as well as simulation experiments were used. The problems with the decentralized dynamic algorithms do arise from the time delay occurring by exchange of the state information. In [HsMa80] was proved that the knowledge of the last state of the last state of the system in all its decentralized components is a necessary condition for an optimal control strategy. However, there exists a computable solution of this problem for a linear system [Te81].

The partitioning of the time space into intervals of "non important changes" will be a practical way to model the system as a sequence of static intervals.

The *static* load balancing strategies base on the assumption, that the input traffic streams are completely characterized by some constant mean values. For solving problems of this type techniques as the network flow algorithms or mathematical programming can be employed [St77],

[TaTo86].

It seems to be clear that dynamic policies [EaLaZa86] may be more effective than static ones, where the former can exhibit a greater overhead than the latter. Unfortunately, only approximations of the optimal adaptive policies for light or hard load, applicable to various environments, are known [LeFi90].

In this contribution the static case is investigated. More exactly, we study a system with continuous task input stream. The system consists of workstations interconnected by a single multiaccess network (possibly a high speed LAN). We assume the mean values of input streams change much slower than our algorithm works (the *quasi-static* assumption [Ga77]), so the problem reduces to a mean value problem. Our task is then, to minimize some objective function for this system (f.e. mean service time of all tasks in the system).

The previous works on this area were begun by Tantawi and Towsley [TaTo86]. They gave the optimality conditions for a M/M/1 system connected by a LAN, where the cost function was the mean overall task processing time. An algorithm for computing the optimal load splitting from the given input streams (with regard to the communication costs) in a distributed system was given. This algorithm is not able to be performed distributed. Kim and Kameda [KiKa90] extended this algorithm to the multiclass job case. Kurose and Singh [KuSi86] have developed an iterative distributed static algorithm using gradient descent method, a methodology from the field of mathematic economics. The underlying distributed system model consisted of M/M/1 stations. Souza and Gerla [SiGe91] proposed an algorithm for static optimal centralized load balancing based on a closed-chain queueing model of the distributed system. The site constraints for tasks as well as multiple classes and station background load were considered. The optimal solution is obtained by means of MVA and the flow-deviation algorithm. The algorithm is not designed for distributed usage.

The algorithm presented here differs from the above mentioned ones and resembles in the basic idea of application of the projected gradient optimization method the Gallager algorithm [Ga77] for routing in wide area networks. The algorithm can be implemented in distributed manner, is independent from any theoretical assumptions upon input load characteristic and type of the cost function, as well as from the homogeneity or inhomogeneity of the stations in the system. Numerical examples are provided. The expression for the step size of the algorithm to guarantee the convergence could be obtained. Comparing our work with the Kurose-Singh algorithm [KuSi86], we

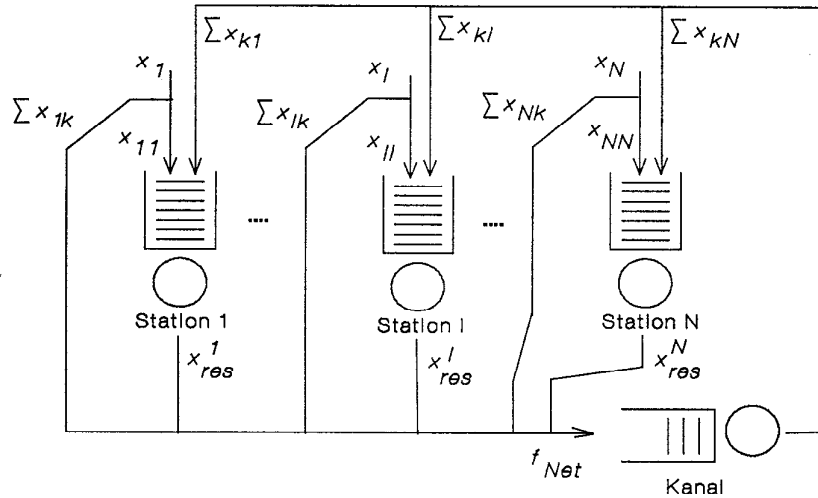


Fig.1: The model of the distributed system

must mention that the latter sets restrictions on the station model, and employs a different flows adjustment mechanism. Moreover, no results concerning the step size of the algorithm are given. The convergence proof holds only for infinitesimal step sizes.

2. System model

We consider a workstation based distributed system consisting of a local area network with N stations attached. We model the tasks arriving into the system with continuous input streams. The input task intensity at the i -th station is x_i tasks/s. We assume that the incoming tasks are completely independent from each other (this situation is common in networks of workstations). The input stream intensities obey to the quasi-static assumption. Each station i can decide to send a specific amount x_{ik} of the incoming traffic off to the station k for service (Fig. 1). After service the results are returned to the origin station.

We define the set of control variables

$$\bar{\varphi} = (\varphi_{ik}) : i, k = 1, 2, \dots, N, \text{ such that: } \varphi_{ij} \stackrel{\text{def}}{=} \frac{x_{ij}}{x_{ij}},$$

$$\text{where } \sum_{j=1}^N \varphi_{ij} = 1 \quad \text{and} \quad \varphi_{ij} \geq 0. \quad (1)$$

From the above definitions we can express the total stream of tasks entering for service at the station i as $f_i = \sum_{i \neq j} (\varphi_{ij} x_i + \varphi_{ji} x_j)$. The network load is: $x_{Net} = \sum_{i \neq j} \varphi_{ij} x_i$. The network traffic f_{Net} is however equal to the sum of the task stream and result return stream: $f_{Net} = x_{Net} + x_{ret}$. The returning result stream x_{ret} constitutes from the results of the task streams x_{ij} , where $i \neq j$. The intensity equals to the output intensity of the task results for that particular input stream at the station j . From the Burke's theorem [Kle75] we know that only for special case of a M/M/n system the output stream is equal to the input. In general case, the output intensity is a function of the total input stream f_j and mean service at the station j , and is not always explicitly known.

To escape from this inconvenient situation we recall the spirit of the quasi-static assumption and observe, that for each task its result must be returned. Thus for observation period tending to infinity we can express x_{ret} as:

$$\lim x_{ret} = \sum_{i \neq j} \varphi_{ij} x_i = x_{Net} \quad (2)$$

If the above equation is applied for an observation period other than infinity, we must assume that within the observation time results of all tasks submitted for remote execution in this observation period will be returned to the origin station. It is not true, in general, because of the latency of the service stations and the transformation of its output stream, but for a sufficiently long observation period

it might be an acceptable assumption.

The cost function for the network transmission D_{Net} depends on mean service times for both task and result streams S_{task} , S_{res} and on its intensities. Assuming mean service times to be constant, the network cost function depends only on the network traffic. Our task is then to find minimum point $\bar{\varphi}^*$ for some general, system-wide cost function D :

$$\min D(\bar{x}, \bar{\varphi}) = \min \left[\sum_{i=1}^N D_i \left(\sum \varphi_{ji} x_j \right) + D_{Net} \left(\sum_{i \neq j} 2\varphi_{ij} x_j \right) \right] \quad (3)$$

where $D_i (f_i)$ is a general cost function related to station i depending only on total input stream intensity f_i . This means in practice that mean demanded service at each station is equal. The popular choice for D is weighted task delay, but we do not exclude other objective functions. Restating, our assumption for the overall objective function is its separability (what is the case in real distributed systems). We make an additional assumption, requiring the station-own cost functions D_i to be convex and differentiable. The differentiability of the functions D_i is not given for delay type functions: they are not defined in point $f_i = C_i$, i.e. in the saturation point. We can modify the function D in the way given in literature [KuSi86] to assure its continuity. The second cause for assuming that the differentiability is given, is the fact that properly designed load balancing algorithm should not operate with any station being saturated.

In the real world, there are two possibilities to implement the communication protocol: either the CPU is executing the protocol stack, or there is an auxiliary processor on the network interface doing this. It is possible to extend our model to incommode both the choices. The auxiliary processor could be modelled with an additional cost function $D_i^{prot}(\sum_{i \neq j} (x_{ij} + x_{ji}))$. Further on, protocol stack processing by the CPU diminishes the effective processor speed from the angle of the task being processed on the specific station. This could be modelled with additional, virtual task input for the processor, for example: $f_i = \sum_j x_{ji} + \alpha_{send} \sum_{i \neq j} x_{ij}$ with α_{send} being a multiplicative constant.

This two cases do not change significantly the results of the load balancing algorithm. So we do not consider them in this contribution, because they could obscure the

introduction of the essentials of the proposed load balancing algorithm.

3. Optimality criterions for the load balancing algorithm

The necessary optimality conditions for load balancing in a distributed system connected by a single multiaccess network were first given in [TaTo86]. They covered only a case of a specific objective function: the mean overall job processing time. Because we want to compute the optimal settings of φ_{ij} for the general case, we must investigate the necessary and sufficient optimality conditions for $D(\bar{\varphi}^*)$.

To minimize the general objective function $D(\bar{x}, \bar{\varphi})$ we will use the Lagrange multiplier method. We have the following task posed:

$$\min D(\bar{\varphi}), \quad \text{with} \quad \sum_{j=1}^N \varphi_{ij} = 1 \quad \text{and} \quad \varphi_{ij} \geq 0 \quad (5)$$

After the Kuhn-Tucker Theorem for a convex function [PsDa78] the gradient $\nabla D(\bar{\varphi})$ equals to the linear combination of the gradients of the restrictions $r_i \geq 0$, which are active in the optimal point $\bar{\varphi}^*$:

$$\nabla D(\bar{\varphi}^*) = \sum_i \lambda_i^* \nabla r_i \quad \text{with} \quad \lambda_i \geq 0 \quad \text{and} \quad \lambda_i r_i = 0, \quad (6)$$

where $\bar{\lambda}$ is the vector of Lagrange multipliers.

We get for the optimal point $\bar{\varphi}^*$ after (5)(6) the following system of equations:

$$\left\{ \begin{array}{l} \frac{\partial D^*}{\partial \varphi_{ij}} = \lambda_i + \lambda_j \\ \lambda_i (\sum_k \varphi_{ik} - 1) \\ \lambda_{ij} \varphi_{ij} = 0 \end{array} \right. \quad (7)$$

From (5b) (7c) follows:

$$\frac{\partial D^*}{\partial \varphi_{ij}} = \left\{ \begin{array}{l} = \lambda_i \quad \text{for} \quad \varphi_{ij} \neq 0 \\ \geq \lambda_i \quad \text{for} \quad \varphi_{ij} = 0 \end{array} \right. \quad (8)$$

This means straightforward, for optimal control variables set $\bar{\varphi}^*$, each station i must set its own controls φ_{ij} not equal to null only for these remote stations, which display the minimal values of the derivative $\partial D / \partial \varphi_{ij}$.

From the Kuhn-Tucker theorem follows, that (8) are the

necessary conditions for existence of an optimum. Concerning the *sufficient* condition it can be easily proved, that the following theorem holds [Kra94]:

THEOREM 1: For the optimization problem (5) the condition (8) is a necessary and sufficient condition for the existence of the optimum. ■

Now let us consider some important property of the optimal solution. Intuitive we feel, that the situation, when station i sends tasks to the station j and the station j sends in his turn tasks to station i is not optimal. Let us investigate this case more detailed. If we have $\varphi_{ij} \neq 0$ and $\varphi_{ji} \neq 0$ we can write after optimality conditions (8):

$$\partial D / \partial \varphi_{ij} \leq \partial D / \partial \varphi_{ji} \quad \text{and} \quad \partial D / \partial \varphi_{ji} \leq \partial D / \partial \varphi_{ij}$$

Using (9) and some simple algebra for rewrite those inequalities we come to the conclusion that: $D'(f_{Net}) = 0$. Thus the following holds:

Conjecture: If the overall objective function $D(\bar{x}, \bar{\varphi})$ depends on the network traffic f_{Net} , then in the optimal point from the fact that $\varphi_{ij}^* > 0$ it follows $\varphi_{ji}^* = 0$.

For example if the function $D(\bar{\varphi})$ represents the mean task delay, there must be always either $\varphi_{ij}^* = 0$ or $\varphi_{ji}^* = 0$.

4. The algorithm and its properties

4.1. Definition of the algorithm

Employing the insights in the problem given by optimality conditions (8) we can develop an algorithm with the following structure: in each iteration some ratio of the input load x_i will be taken off from all stations with nonminimal $\partial D / \partial \varphi_{ij}$ and redirected to the "minimal" station. Our task can be posed as follows now: minimize (3) with constraints (2) and under the requirement that the input traffic is to be taken away from the "non-minimal" stations and redirected to the "minimal" (in the before defined sense) ones, as an additional constraint. This means, we need an algorithm that computes merely new settings of φ_{ij} for $j \neq j_{\min}$. Once we have got this, the settings for the "minimal" station can be expressed with

$$\varphi_{ij_{\min}} = 1 - \sum_{j \neq j_{\min}} \varphi_{ij} \quad (9)$$

Normally, we pass here from the space of controls $\bar{\varphi} \in \mathbb{R}^n \times \mathbb{R}^n$ to the space of controls $\bar{\varphi}' \in \mathbb{R}^n \times \mathbb{R}^{n-1}$. We notice, the equality (9) assures additionally that every step of the algorithm produces feasible sets $\bar{\varphi}$. The feasibility is guaranteed too by the gradient projection method used further on.

Now we can take first the optimization in the new control space Φ' , without bothering about feasibility and then return to the original space Φ . To assure the convergence to the optimal point for this algorithm, we set the needed input load ratios for each iteration with the gradient projection optimization method. An iteration of the gradient projection method in the \mathbb{R}_+^N space is [PsDa78], [Wi88]:

$$\bar{x}^{n+1} = \max [0, \bar{x}^n - \eta \nabla f(\bar{x}^n)] \quad (10)$$

where η is a scaling factor, and ∇f the current gradient of the minimized function. We will use this iteration first in the space of φ' -controls (this assures feasibility and a step toward the optimality criterion), and then compute the complete settings for our origin control space.

But here we can act twofold: first we could perform the iteration in x'_{ij} space and then return to controls φ' or do it directly in the Φ' -space, and we *will not* get two identical algorithms. Discussion of this topic can be found in [Kra94] and for this contribution we concentrate on the *first* version of the algorithm.

Extracting the factors $\varphi_{ij_{\min}}$ from (3) with help of (9) we construct a new function $\tilde{D}(\bar{x}', \bar{\varphi}') : \Phi' \rightarrow \mathbb{R}$, with values equal to the values of $D(\bar{x}, \bar{\varphi})$. Its derivative is given as:

$$\frac{\partial \tilde{D}(\bar{x}')}{\partial \varphi_{ij}} = \sum_{i \neq j} \frac{\partial D_i(f_i)}{\partial f_i} \frac{\partial f_i}{\partial \varphi_{ij}} + \frac{\partial D_{j_{\min}}(f_{j_{\min}})}{\partial f_{j_{\min}}} \frac{\partial f_{j_{\min}}}{\partial \varphi_{ij}} + \frac{\partial D_{Net}(f_{Net})}{\partial f_{Net}} \frac{\partial f_{Net}}{\partial \varphi_{ij}}$$

Considering (9) for computing the $\partial f_{Net} / \partial \varphi_{ij}$ factors and then substituting the gradient in (10) [Kra94] we get the following algorithm:

$$\varphi_{ik} = \begin{cases} \varphi_{ik} - \Delta_{ik} & , k \neq k_{\min} \\ \varphi_{ik} + \sum_{k \neq k_{\min}} \Delta_{ik} & , k = k_{\min} \end{cases} \quad (13)$$

where η stands for the algorithm's step size and:

$$\begin{aligned} \Delta_{ik} &= \min [\varphi_{ik}, \eta a_{ik} / x_i] \\ a_{ik} &= D'_k(f_k) - D'_{k_{\min}}(f_{k_{\min}}) - 2(\delta_{ik} - \delta_{ik_{\min}}) D'_{Net}(f_{Net}) \\ D'_{k_{\min}} &= \arg \min_k \{ D'_k(f_k) + (1 - \delta_{ik}) D'_{Net}(f_{Net}) \} \end{aligned} \quad (14)$$

4.2. Convergence of the algorithm

It was shown, that the gradient projection method converges to the optimum when the Lipschitz assumption for the cost function $D(\bar{\varphi})$ holds, and the step size η fulfills: $\eta \leq L/2$ (i.e. η sufficiently small) [Wi88]. So we can see, that the step size is crucial for the convergence, and we will give a more strictly bound for it in case of our algorithm [Kra94]:

THEOREM 2: For the step size η_k obeying:

$$0 < \eta_k \leq 2(4N(N-1)M_{Net} + N^2M)^{-1} \quad (13)$$

where $M \geq \max D_i''(f_i^{\lambda_k}), f_i^{\lambda_k} \in (f_i^k, f_i^{k+1}),$ and

$M_{Net} \geq \max D_{Net}''(f_{Net}^{\lambda_k}),$ the algorithm (11) results in decreasing in the step k the overall cost function $D(\bar{\varphi})$. Moreover, with $k \rightarrow \infty$ the algorithm converges towards an unique minimum point described by equations (8). ■

Gradient hill-climbing optimization methods do not converge exorbitant quickly. It is possible to show, that the convergence rate is linear [Wi88]. Moreover we can prove the following theorem:

THEOREM 3: Let D^* be the minimum value of the cost function D , and φ^* the optimal set of controls. Then for consecutive iterations of the algorithm (11) holds:

$$D_{k+1} - D^* \leq q(D_k - D^*)$$

where $q < 1$ and $\|\varphi_k - \varphi^*\| \leq Cq^{k/2}$. ■

The proof of this theorem is a modification of the standard steepest descent method convergence proof [PsDa78].

It is possible to prove the convergence of an asynchronous version of the algorithm [Kra94]. The proof is similiar to the proof of theorem 2, additionally using Lipschitz assumption for the cost functions D_i , to upper bound the error of the estimation of D_i -derivatives at a single station i . From the proof it is visible, that the asynchronicity in general can not destroy the convergence, but can only slow it down.

5. The Stepsize Considerations

Scaling factors are often a plague in the use of the iterative algorithms. As we could see foremost, choosing it too great, we can not assure convergence. But on the other side, the step size guaranteeing convergence may result in extremely low convergence speed because the underlying mathematical method (projected gradient) has a rather poor convergence properties.

In the nonlinear programming [Wi88], the standard procedure to accelerate the convergence and remove the scalability problems is the use the inverse of some approximation of the Hessian matrix $H = \nabla^2 D(\bar{\varphi}, \bar{x})$ instead of the step size η in (10). This would result in a new algorithm based on the so called Newton method, which would be able to detect automatically the input flow and choose the appropriate step size. The Newton method displays a very good quadratic convergence rate. Observe, that in (13) we nevertheless use the inverse of the second derivatives, thus acting somewhat in spirit of the Newton method.

We can widen our narrow range of the step sizes, still guaranteeing the convergence. The idea is, to make first a great step which brings us near to the solution, and then apply the relatively small step sizes guaranteeing convergence. We can do it, by taking first step two times greater than the permitted one, and the next steps proper after (13). Generally, we can take the first step multiplied with K , the next with $K-1$ and so on. Formally:

$$\eta(n) = \begin{cases} \eta(M) * (K-n) & \text{for } n < K \\ \eta(M) & \text{otherwise} \end{cases}$$

In the implementation we must find the value of the upper bound M . We propose for this purpose to take the greatest from the second derivatives observed during the preceding iterations. To be more adaptive, the algorithm can choose for M the maximal second derivative from the last l iterations, but our results are obtained using the more simple technique stated above.

Our heuristics, compared to the one given in [KuSi86], does not need more information (only the maximum second derivative) and is much simpler in use: instead of guessing the maximal allowable step size out of an unknown range, we take as parameter an integer, which maximal value (according to the experiments) should not be very much larger then N^2 .

6. Numerical results

The algorithm will be tested first in the example system form [TaTo86]. The example given in this article considers a distributed system consisting three nodes (modelled as central server systems with CPU and I/O channels), connected via a single channel with fixed, independent of network traffic, communication delay. The nodes are modelled by a central server model and consist of one CPU and several I/O channels (see Fig.2). The mean node delay functions for central server models used, can be obtained by means of the BCMP-theorem and are:

$$D_1(f_1) = \frac{3}{20-f_1}, \quad D_2(f_2) = \frac{3}{10-f_2}, \quad D_3(f_3) = \frac{5}{10-f_3}, \quad D_{Net} = t.$$

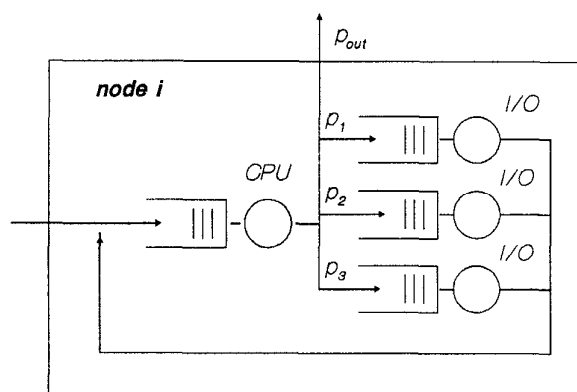


Fig. 2: Model of a system node

For details we refer to [TaTo86]. We note that in the used delay functions we do not allow any temporary overload, because we find that a good load balancing can not cause short term overloads at any station. So we do avoid the losses due to overload at the cost of a perhaps slower convergence.

First we state that our distributed algorithm finds exactly the same optimal point the Tantawi-Towsley algorithm does. Next we want to observe how the algorithm converges in our example system. From the Fig.3 we notice the typical behaviour of the gradient descent algorithm: fast convergence first, which settles down afterwards, and converges slowly in the neighbourhood of the optimum. Next, we see that for the step size being too great ($\eta=2.0$)

no convergence takes place and the algorithm oscillates around the optimum.

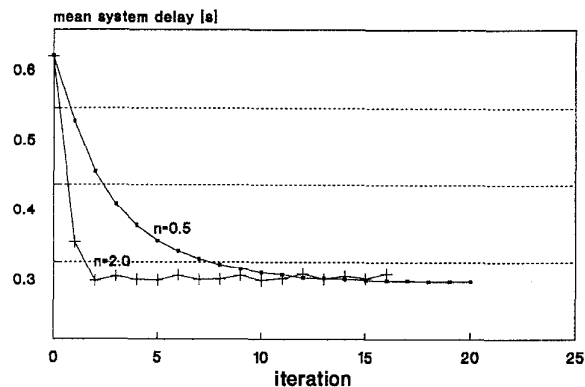


Fig. 3: Convergence of the algorithm ($t=0$) according to the step size

Knowing the behaviour of the algorithm we see that it would not make much sense to wait for the exact convergence. The problem is that it is hard to decide whether the algorithm decreases very slowly or converges, when we are in the flat part of the curve in Fig.3 In the real applications we do not really need the exact convergence: it is sufficient when the algorithm gives us a *substantial benefit*. So we will investigate how quickly the algorithm achieves 90% of the improvement which is maximal possible, and in the following the word convergence will be used only in this meaning.

Let us now proceed to the discussion of the proposed step size heuristics. We have two aims developing it: first to accelerate convergence, and second to ensure that for a specified step size the algorithm converges quickly for a quite wide range of system loads, optimally for all possible loads. Thus we want discuss these two topics in the following.

First, we have examined the influence of the step size on the convergence speed for various system loads. For this purpose we have tested the heuristics using different values of the parameter K in systems of different sizes (3, 9 and 15 stations), with different input rates and networks of different capacity. Systems consisted of stations of three types described in the previous experiment and of equal number from each type. The input rates for each station of a specific type are equal. The network connecting the

stations is modelled as a single M/M/1 channel with a capacity C .

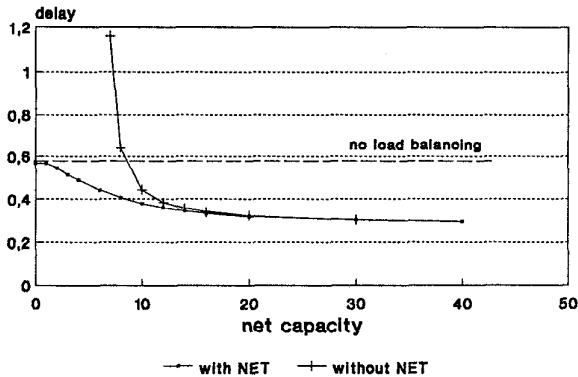


Fig. 4: Network influence on the load balancing

The network capacities used in tests were obtained by testing the influence of the network delay on the load balancing for our basic system. In the Fig.4 we can see the results for the algorithm which considers the network delays by establishing the optimal control and its version neglecting the influence of this factor. For our tests we have chosen three capacities: 4,10 and 16 representing great, middle and no influence of the network transmission delay on the load balancing as seen in the Fig.4.

The results of our investigation are visible in Fig.5. Here, the typical dependance between the number of iterations needed to obtain the convergence and the system load is shown. The loads are arranged due to the maximal percent improvement possible. We used both high and low loads as well as homogenous and inhomogenous (in respect to the station's ρ coefficients) ones. The optimal values of the objective function (maximal improvement possible) were obtained with the precision of six digits.

First we can see, that the convergence speed does not depend as much on the input pattern, as it depends on the greatest possible gain in the objective function. So the optimal choice of the parameter K depends mainly on this possible gain. We can state, that for gains under 50% we observe that the setting $K=2N$ results in reasonable behaviour, i.e. convergence in about 5 steps and in small systems even in 1 or 2 steps. Here N denotes number of stations in the system.

However, for greater gains we must choose much

greater K values and, as we can see from Fig.5, these values do not cover a broad range of possible gains, because for smaller gains the step size is *too great* and leads to increasing instead of decreasing the objective function. This results in great change in the undesirable direction and, what is much worse, can produce great second derivative which in our heuristic results with a very little consecutive step sizes. We see: the greater the possible gain is, the greater value of K we must chose, if we want the algorithm to converge quickly. So it is difficult to cope with such an situation: the algorithm is simply not adaptive enough. We did not observe overloads except by very high values of $K (>N^2)$.

Finally, we investigated dependency between the convergence speed and the system size. We compared the number of iterations needed to achieve convergence in systems of various sizes, for optimal K choice under high and low load. This numbers were obtained for two different system loads: one that was unhomogenous and the other that was homogenous in respect to the stations's ρ coefficients. We found that there is a clear dependance of the convergence speed and system size: the greater the system, the slower the convergence. This fact hangs together with the nature of the algorithm: we allow only one "minimal" station k_{\min} the task streams are redirected to, for each iteration at the station i . In our experiments there are more such stations, because in the greater test systems we simply replicate the basic three station system. This leads to significant deterioration of the convergence rate. To improve the algorithm we could allow more than one "minimal" station and then divide the traffic redrawn from the "nonminimal" stations equally among them. For very light and homogenous load (1, 0.5 and 0.5 tasks/s) the effect described above is not very significant.

7. Conclusions

The minimization method used in our algorithm (steepest descent method) do not show the best convergence rate, but it has two very important advantages: simplicity and guaranteed convergence, the other methods not always have. The Newton method for example converges only if the initial point is choosen sufficiently near to the minimum. If we want the algorithm to converge for any starting point choice, we must introduce the direction optimization [W188]. We can not do it unless we know

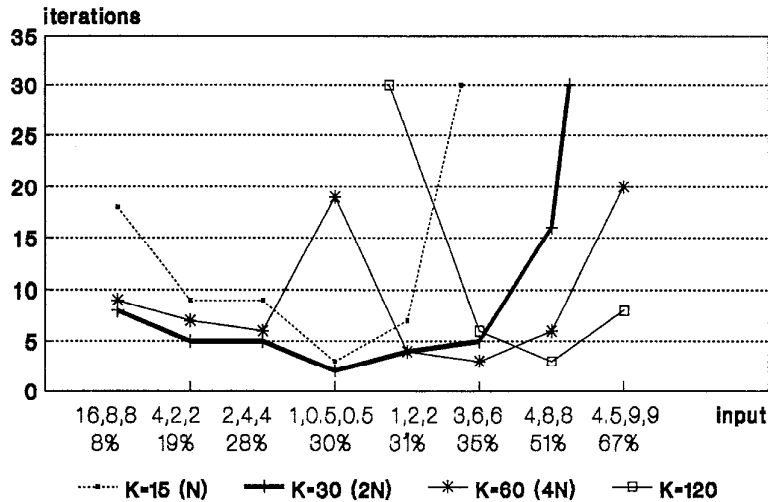


Fig. 5: Typical behaviour of the algorithm

exactly the cost function, and worse: this optimization is best performed in a centralized manner. Thus the computations are more complicated even not considering the inverse of the Hessian matrix (used in each step) and every station needs much more information to perform an iteration.

Other optimization method that could be applied is the conjugate direction method. It uses only objective function derivatives and the vectors ϕ , but it converges in the best case after n steps, where n is the number of variables and involves complicated computations. In our case, for an system of 15 stations $n=256$!

Using the simple steepest descent method together with the proposed step size heuristics, we can achieve a substantial improvement within several iterations, involving only small amount of information.

Further we have seen, that the algorithm first converges rapidly to the neighbourhood of the optimum and then considerably slows down the convergence rate. We could then expect, that the algorithm do not over-react to the changes in the environment where the quasi-static assumption is violated and will behave good in such case. Moreover, we can expect, that the algorithm automatically detects differences in station speeds (due to station derivatives) and thus can be best employed in systems consisting of nonhomogenous stations. For an homogenous system, the greatest benefit will be clearly achieved for

substantial differences in input stream intensities at particular stations.

A serious problem for the practical use of the here introduced algorithm seems to be the on-line estimation of needed derivative values. The further research on this problems has been started. A perturbation algorithm for fast computation of gradients was presented in [CiHo93].

8. References

- [ChS88] D.R. Cheriton, M. Stumm: "The Multi-Satellite Star: Structuring Parallel Computations for a Workstation-Cluster", *Distributed Computing*, 1988
- [CiHo93] S. Ciereszko, U. Hofmann: "Adaptive Load-Sharing with On-Line Gradient Estimating in Network Environments", *Proc KiVS 93*, München, Springer-Verl., pp.188-202
- [Do087] F. Douglis, J. Ousterhout: "Process Migration in the Sprite Operating System", *Proc. 7th Int. Conf. on Distrib. Comp. Syst.*, Berlin 1987, pp.18-25
- [EaLaZa96] D.L. Eager, E.D. Lazowska, J. Zahorjan: "Adaptive Load Sharing in Homogenous Distributed Systems", *IEEE Trans. on Softw. Eng. Vol. SE-15*, No.5, pp.662-675, May 1986
- [EphVe89] A. Ephremides, S. Verdú: "Control and Optimization Methods in Communication Networks

- Problems", *IEEE Trans. on Autom. Contr.*, Vol.AC-34, No.9, pp.930-942, Sept. 1989
- [FiM87] R. Finkel, U. Manber; "DIB - A Distributed Implementation of Backtracking", *ACM Trans. Progr. Lang. and Syst.*, Vol.9(2), pp.235-256, 1987
- [Ga77] R. Gallager: "A Minimum Delay Routing Algorithm Using Distributed Computation", *IEEE Trans. a Comm.*, Vol.COM-23, No.1, pp.73-84, Jan. 1977
- [HsMa80] R.A. Hsu, S.I. Marcus: "Decentralized Control of Finite State Markov Processes", *Proc. Conf. on Decision and Ctrl.*, Albuquerque, 1980, pp.143-148
- [KiKa90] Ch. Kim, H. Kameda: "Optimal Static Load Balancing of Multiclass Jobs in a Distributed Computer System", in: *Proc. of the 10-th Int. IEEE Conf. on Distr. Comp. Syst.*, Paris 1990, pp.562-569
- [Kle75] L. Kleinrock: *Queueing Systems, Vol.1: Theory*, N.Y. Wiley 1975
- [Kle90] L. Kleinrock: "On Distributed Systems Performance", *Comp. Networks and ISDN Systems*, 1990, pp.562-569
- [KIK89] L. Kleinrock, W. Korfhage: "Collecting Unused Processing Capability: An Analysis of Transient Distributed Systems", in *Proc. of 9-th Int. IEEE Conf. on Distr. Comp. Syst.*, Newport Beach, Calif., June 5-9, 1989, pp.482-489
- [Kra94] M. Krajewski: *Quasi-Static Load Balancing in Distributed Systems*, Thes. Diss. TU Dresden, Inst. of Informatics, 1994
- [KrLi90] Ph. Krueger, M. Livny: *A Cost-Benefit Approach to Robust Distributed Scheduling*, Technical Report DSU-CISRC-2190-TR5, The Ohio State University, Feb. 1990
- [KuSi86] J.F. Kurose, S. Singh: "A Distributed Algorithm for Optimum Static Load Balancing in Distributed Computer System", *Proc. of the 5-th IEEE INFOCOM*, Miami, 1986, pp.458-467
- [LeFi90] A. Levine, D. Finkel: "Load Balancing in a Multiserver Queueing System", *Computer Ops. Res.*, Vol.17, No.1, pp.17-25, Jan.1990
- [LiM82] M. Livny, M. Melman: "Load Balancing in Homogenous Broadcast Distributed System", *Proc. of the ACM Comp. Netw. Perf. Symp.*, April 1982, pp.47-55
- [MuL91] W.M. Mutka, M. Livny: "The Available Capacity of a Privately Owned Workstation Environment", *Perform. Eval.*, Vol.12, July 1991, pp.269-284,
- [Nic87] D.A. Nichols: "Using Idle Workstations in a Shared Computing Environment" in *Proc. of 11th ACM Symp. Operat. Syst. Principles*, Austin, Texas, 1985, pp.5-12
- [PsDa78] B.M. Pshenichny, Yu.M. Danikin: *Numerical Methods in Extremal Problems*, Mir Publishers, Moscow 1978
- [SiGe91] E. de Souza Silva, M. Gerla: "Queueing Network Models for Load Balancing in Distributed Systems", *J. of Parallel and Distr. Comp.*, Vol. 12, pp.24-38, 1991
- [SqN91] M.S.Squantale, R.D. Nelson: "The Design and Implementation of a Decentralized Scheduling Facility for a Workstation Cluster", *Proc. 1991 ACM SIGMETRICS Conf. on Measurement and Modelling of Comp. Syst.*, San Diego, May 1991
- [Stu88] M. Stumm: "The Design and Implementation of a Decentralised Scheduling Facility for a Workstation Cluster", in *Proc. of 2nd IEEE Conf. Computer Workstations*, March 1988, pp.12-22
- [St77] H.S. Stone : "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", *IEEE Trans. on Softw. Eng.*, Vol.SE-3, No.1, pp.85-93, Jan. 1977
- [TaTo86] A. Tantawi, D.Towsley: "Optimal Static Load Balancing in Distributed Systems", *Journal of the ACM*, Vol.32, pp.445-465, April 1986
- [Te81] R.Tenney: "On the Concept of Decentralized Control", *Infrom. and Ctrl.*, Vol.50, pp.1-12, 1981
- [ThL89] M.M. Theimer, K.A. Lantz: "Finding Idle Machines in a Workstation-Based Distributed System", *IEEE Trans. on Softw. Eng.*, Vol.SE-15, No.11, Nov. 1989, pp.1444-1457
- [Wi88] R. Wit: *The Methods of Nonlinear Programming*, Warsaw 1988, in polish