Adaptive Load Control in Transaction Processing Systems

Hans-Ulrich Heiss
University of Karlsruhe
Germany

Roger Wagner
University of Karlsruhe
Germany

Abstract

Thrashing in transaction processing systems can be prevented by controlling the number of concurrently running transactions. Because the optimal concurrency level strongly depends on the workload characteristics which may change in time, two algorithms for adaptive adjustment of an upper bound for the concurrency level are proposed and compared by simulation.

1 Introduction

It is well known that transaction processing systems can be subject to thrashing. The term thrashing, coined by Denning [Denning, 1968] for overload effects in virtual storage systems, generally describes a phenomenon where an increase of the load results in a decrease of throughput (or another related performance measure). Systems with such a behaviour show a load-throughput function as in figure 1.

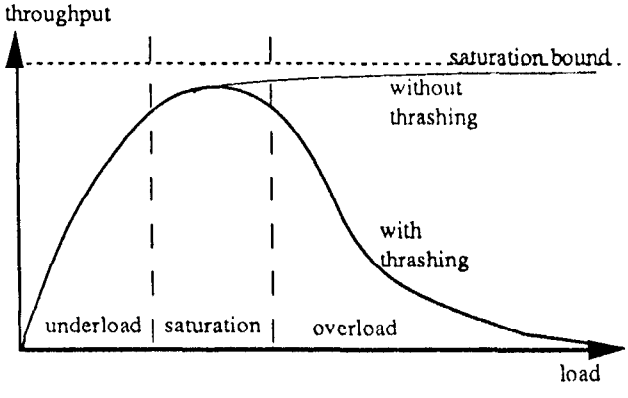

Figure 1: Typical shape of the throughput function with thrashing

Usually, three phases can be distinguished:

I. Underload
At light loads with sufficient resources available, the throughput grows almost linearly making use of possible parallelism in the system.

II. Saturation
When the finite capacity of the system becomes effective, the throughput function flattens out.

III. Overload (thrashing)
Further increasing the load will not lead to an asymptotic approach to the saturation bound but will cause a sometimes sudden drop in throughput.

Generally speaking, at least two classes of factors contribute to this overload effect:

- the management of the load units to be processed (e.g. transactions) causes an algorithmic overhead (e.g. list operations, sorting, searching etc.) that in many cases is overlinear with respect to the load.

- dependent on the type of the system, the load units start to hamper each other due to insufficient resource capacity.

In transaction processing systems, these mutual impediments are known to stem from contention for either physical resources (memory, processors) or logical resources (data granules). The former is usually called *resource contention* (RC), the latter *data contention* (DC). To describe the effect of contention in more detail, we have to distinguish between two major classes of concurrency control (CC) algorithms [Bernstein et al., 1987]:

1. Blocking CC algorithms (e.g. Two Phase Locking)
Analytic models show [Tay et al., 1985] that the mean number of blocked transactions b is a quadratic function of the total number of transactions n. This means that beyond a critical point ($db(n) / dn > 1$), the result of adding one more transaction ($n:=n+1$) will be that the number of blocked transactions b increases by more than one which in turn means that the number of active transactions a ($a = n-b$) will decrease.

2. Non-blocking CC algorithms (e.g. timestamp ordering, optimistic CC)
In non-blocking CC schemes, conflicts are resolved by aborting and restarting one of the involved

transactions. The mean number of restarts - which roughly corresponds to the number of blocked transactions in the blocking case - is also an overlinearly increasing function of the concurrency level n. With growing n, the fraction of rerunning transactions becomes larger and consumes a larger fraction of physical resources (e.g. processor capacity), and thereby tightens the contention. Because reruns cannot be regarded as useful work, more and more resource capacity is wasted resulting in a performance decrease. Roughly speaking, in non-blocking systems, data contention is resolved by increased resource contention which leads to a drop in throughput as soon as resource saturation is reached. Only in an ideal system with unlimited capacity, thrashing can be avoided.

Knowing that thrashing threatens in virtually every transaction processing system, we have to think about countermeasures that limit the load such that overload is prevented. Several solutions are possible:

1. Do nothing
   Rely on selfregulating market mechanisms: If the service (throughput, response time) becomes worse, fewer people want it. This approach, however, is not universally viable.

2. Fixed upper bound
   The maximum number of concurrent transactions is a system parameter that is tuned by the system administrator when the system is installed or started up. This approach can usually be found in commercial database systems. When the transaction load is constant and the value is chosen appropriately, this solution may work. However, traces from real database systems often show large variations of the load, both quantitative and qualitative. They also show an overlay of variations of different periodicity, some of which are more regular and predictable, others not.

3. Theoretically derived 'rules of thumb'
   Analytical models sometimes suggest some conditions that must be satisfied to prevent thrashing. Tay et al. [Tay et al., 1985], for instance, claim that k^2n/D should be less than 1.5 where k is the number of data items accessed by each transaction, n is the concurrency level, and D is the database size. Iyer [Iyer, 1988] suggests that the mean number of conflicts per transaction should not exceed 0.75. Although the authors give some evidence for their results within the framework of their respective models, the question is whether these bounds actually apply to all possible load situations. If so, controlling the concurrency level to prevent thrashing could be easy. However, as long as no detailed examinations of these rules are available, they have to be considered with caution.

4. Feedback control mechanisms
   If we do not believe those rules of thumb or do not want to completely rely on them, we may look for more direct, i.e. model independent control mechanisms that only need a few and weak assumptions. This kind of solution can be provided by a control-theoretic approach. The dynamically changing optimal concurrency level requires an adaptive load control. Based on recent measurements of system quantities, an upper bound of the concurrency level has to be adjusted. This approach is the subject of our paper.

2 Related work

Thrashing as a phenomenon was first reported in [Denning, 1968]. A survey of different approaches for dynamic load control in virtual storage systems was given in [Denning et al., 1976] and in [Denning, 1980]. Thrashing phenomena in database systems with locking were brought to major attention by Tay et al. [Tay et al., 1985] using an analytical closed mean value transaction flow model. A similar model was used by Dan et al. [Dan et al., 1988] to analyze optimistic protocols. Franaszek and Robinson [Franaszek and Robinson, 1985] apply a random graph model that also reveals thrashing behaviour. A comprehensive analytical framework for data and resource contention in database systems was recently presented by Thomasian and Ryu [Thomasian and Ryu, 1990] who also report thrashing. Most of the results of these analytic models, including thrashing, were confirmed through simulation studies conducted by Agrawal et al. [Agrawal et al., 1987]. The main subject of all these contributions, however, is not to propose a mechanism for controlling the concurrency level, they rather are concerned with the possibility of modeling per se or with the comparison of different classes of concurrency control protocols. They just say that control of the concurrency level is necessary but don't say how that should be done. Only Tay et al. [Tay et al., 1985] and Iyer [Iyer, 1988] more concretely suggest criteria for such a control mechanism. While these two proposals are limited to blocking CC algorithms, our approach is more generally applicable.

3 Our Approach

We consider the problem of controlling the concurrency level in transaction processing systems as a dynamic optimum search problem [Heiss, 1989]. We are not concerned about any internal details of the system, we are solely interested in the functional relationship of the concurrency level n as the input and the resulting performance P as the output of the system. Generally, the throughput T is used as the performance index P. As we will see below, however, alternative quantities with similar shape are eligible. We assume that this function $P(n)$ at each time has a shape like figure 1 or - more precisely - that $P(n)$ is monotically increasing up to a

maximum at n_{opt}, and then decreasing. In other words, we assume the existence of a local maximum that is also a global one.

The performance is assumed to be also a function of the time allowing for almost arbitrary changes of the load characteristics (figure 2). The dynamic behaviour, however, should have some locality in the sense that the shape of the curve at time t_i is a good estimate for its shape at time t_{i+1}. In other words, the sample interval should be small enough that within an interval, stationary behaviour of a constant parameter stochastic process is a reasonable assumption. On the other hand, the sample interval should be large enough that the relevant quantities can be estimated with sufficient accuracy.

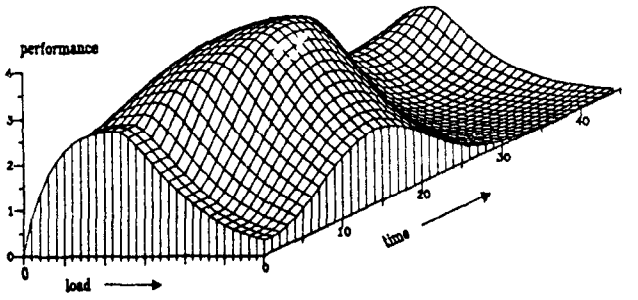

Figure 2: Dynamic behavior of a thrashing system

Looking at figure 2, the problem can be informally described as follows: Starting at time t=0 with an arbitrary load value, the algorithm has to find the 'ridge' of the 'mountain' and to track it along the time axis. Note that our moves in the control plane are limited to the direction of the positive time axis. Additionally, we do not know the shape of the mountain in total but all information we can obtain is the series of realized load/performance pairs from the past.

4 The Algorithms

We describe two algorithms for this dynamic optimum search problem: (1) the *Method of Incremental Steps (IS)* and (2) the *Parabola Approximation (PA)* where the performance-function $P(n)$ is approximated as a polynomial of degree 2.

4.1 Incremental Steps (IS)

In its simplest variant, the *Method of Incremental Steps* works as follows: Starting with an arbitrary value of the load bound n^* as the control variable, we increase it by one at each time step and measure the resulting performance. If the performance also increases, we proceed. If it becomes worse, we interpret this fact as having exceeded the ridge of the mountain and therefore turn the direction until again the performance becomes worse. So we track the ridge in a zig-zag-fashion (figure 3).

More precisely, in each measurement interval $[t_i, t_{i+1})$ the actual concurrency level $n(t_i)$ and the performance $P(t_i)$ are measured. The new load bound $n^*(t_i)$ is adjusted according to the following rule:

$$n^*(t_{i+1}) := \begin{cases} n^*(t_i) + \beta\,(P(t_i) - P(t_{i-1}))\ \times \\ \qquad signum\,(n^*(t_i) - n^*(t_{i-1})), \\ \quad \text{if } |n^*(t_i) - n(t_i)| \le \delta \\ n^*(t_i) + \gamma, \\ \quad \text{if } |n^*(t_i) - n(t_i)| > \delta \wedge n^*(t_i) < n(t_i) \\ n^*(t_i) - \gamma, \\ \quad \text{if } |n^*(t_i) - n(t_i)| > \delta \wedge n^*(t_i) > n(t_i) \end{cases}$$

where

$$signum(x) := \begin{cases} 1 & \text{for } x > 0 \\ -1 & \text{for } x \le 0 \end{cases}$$

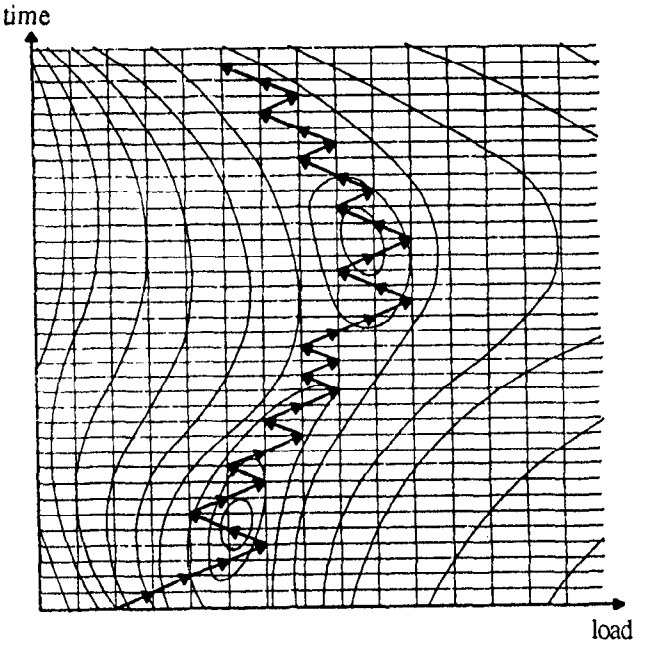

Figure 3: Example trajectory of the Method of Incremental Steps

As can be seen above, the algorithm has three parameters, β for changing the step size proportionally to the performance change, and γ and δ to prevent that the actual load $n(t_i)$ and the load bound $n^*(t_i)$ are drifting apart too far.

4.2 Parabola Approximation (PA)

The performance function $P(n)$ is approximated as $P(n) = a_0 + a_1 n + a_2 n^2$ (see figure 4.)

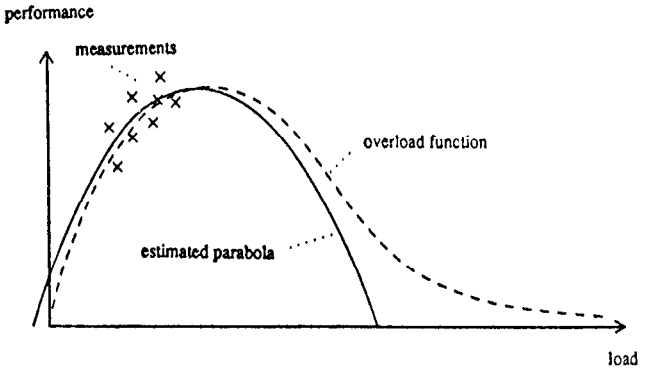

Figure 4: Principle of the Parabola Approximation

Based on recent measurement pairs (P,n) the coefficients a_i are estimated using a recursive least-square estimator with exponentially fading memory [Young, 1984]. The fading is controllable by a weighting parameter α. The recursive way the algorithm works makes it both space- and time-efficient. Having found the parabola, its maximum is used as the new load threshold. The control law is (roughly):

$$n^*(t_{i+1}) := \begin{cases} -a_1/(2\,a_2), \text{ if } a_2 < 0 \\ n_{alt}, \text{ otherwise (see section 5.2)} \end{cases}$$

4.3 Realization of load control

Once the controller has determined a new optimal load bound n^* it has to be enforced in some way:

- Admission control
  The admission to the transaction processing system is controlled by a 'gate' that accepts an arriving transaction if and only if the actual load n is below the current threshold n^*. Otherwise the transaction has to wait in a FCFS-queue. Waiting transactions are admitted as soon as $n<n^*$ holds again.

- Displacement
  Changing transaction behavior may lead to a situation where the controller suggests a new n^* well below the current load n. Here we have two options: (i) We merely use admission control and hope that by normal departures the load n will drop below n^* soon. (ii) In addition to admission control, we instantaneously enforce the new threshold n^* by aborting as many active transaction as necessary. (Victim selection may be based on the same criteria as for deadlock breaking.) Because aborting transactions always means wastage of system resources this approach is justified only if the responsiveness of the controller cannot be achieved otherwise.

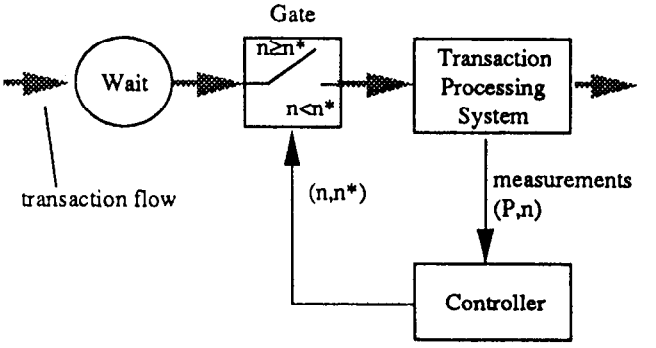

Figure 5: Structure of load control

Our experiments showed that admission control alone was responsive enough to prevent thrashing even with dramatically changing workloads. In addition, not displacing transaction has a smoothing effect on system behavior that supports controller stability. We therefore employed the load control in the way depicted in figure 5. Nevertheless, aborting transactions may be an additional measure as a last resort.

5 Controller Performance and Stability

Although the algorithms worked fine in a broad variety of cases, we can think of 'pathological' situations that would 'fool' the algorithms and lead to poor performance. There are also a few parameters associated with each algorithm that have to be tuned carefully. This tuning requires some knowledge about the statistical properties of the measured quantities which can be regarded as stochastic processes. Tuning does not necessarily mean manual adjustment, it can also be done automatically by an overlaid, outer control loop that takes long-term measurements to adjust the parameters of the inner control loop.

A general problem is the choice of an appropriate measurement interval length. Taking the departures as a stochastic process and assuming stationarity, it is possible to calculate the necessary duration of measurements to estimate the throughput with a given accuracy and for a given confidence level [Heiss, 1988]. This interval length clearly depends on the parameters of the departure process, especially its second moments. The problem is that we have to strike a balance between the stability (not to react to stochastic events ('noise')) and responsiveness (quickly respond to actual changes in the workload). For that reason, the measurement interval should not be longer than required to filter out stochastic noise. To name some figures, an estimate should comprise rather hundreds of departures than some tens.

5.1 Incremental Steps

The simplicity of this algorithm makes it generally prone to failures in specific situations. While it is relatively stable with regard to changes of the optimum's position, it may fail when the height of the optimum is growing without changing the position. Because all steps lead to an improvement, the algorithm 'thinks' to be on the way to the top, but actually goes astray. To prevent a performance breakdown and to help the IS algorithm to recover, a static lower and upper bound for the threshold n^* should be provided.

5.2 Parabola Approximation

Due to the larger amount of information used by this algorithm, it is generally more stable. This amount of information is controlled by two parameters: measurement interval length Δt and aging coefficient α. The selection of these parameters shapes the memory of the estimator. Figure 5 shows two different estimation approaches that use the same amount of information: the dotted line characterizes a long measurement interval and $\alpha=0$, i.e. older measurements are not considered, the solid line means an interval length five times smaller but an aging coefficient of $\alpha=0.8$. The area below the lines can be interpreted as the amount of information used. Because the algorithm is based on a least squares approach, it needs some variations in the measurements to get useful estimates. It is therefore better to choose a small Δt and a large α instead of a large Δt and small α.

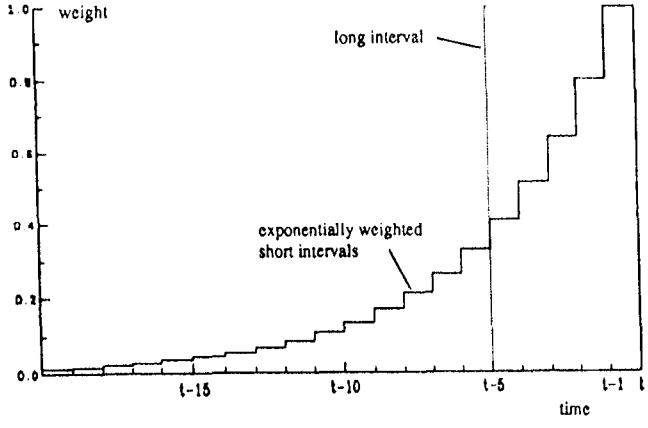

Figure 6: Alternative shapes of the estimator's memory.

It is essential for the parabola approach that the estimated parabola opens downward. There are, however, situations were the algorithm may find an upward open parabola:

a) the true performance function has a broad, flat hump and the collected measurements suggest a convex course of the function (figure 7).

b) the true performance function changed its shape abruptly in a way that the current load bound is now deep in the thrashing region beyond the inflexion point where the shape of the function is actually convex (figure 8).

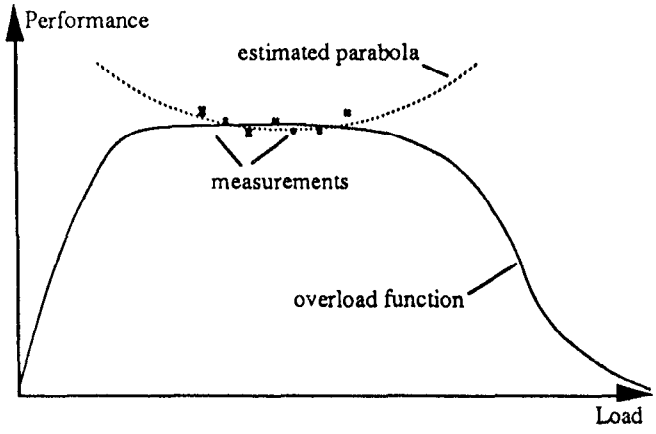

Figure 7: Performance function with a flat hump

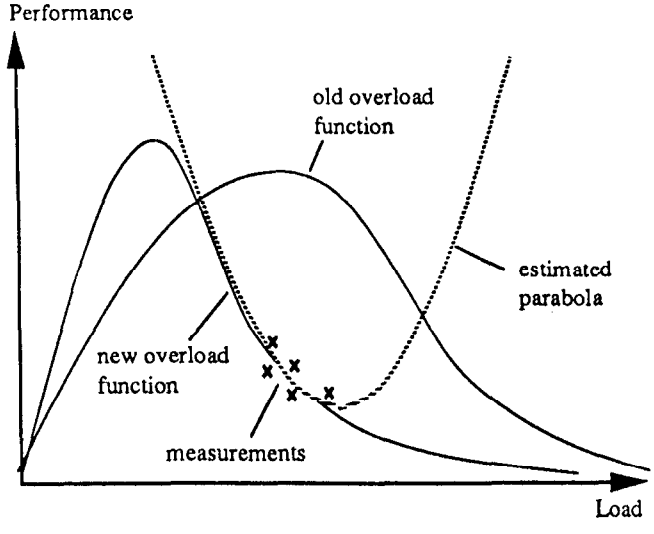

Figure 8: Abruptly changing shape of the performance function

The upward opening parabola is characterized by a positive value of the coefficient of the quadratic term a_2. If this is the case, the result of the estimation is obviously unreliable and useless. We have to provide some countermeasures to recover from this situation. There are several options:

Besides the fact that these different measures define slightly different optimal loads, it may also be a matter of numerical stability of the algorithms, which criterion should be used. With regard to that, the function with the most distinct extremum should be choosen. In our experiments, the throughput T turned out to be the most significant indicator for overload situations. We therefore concentrate on T as our performance measure.

7 The Simulation Model

The simulation model is composed of two parts, a physical model and a logical model. The physical model depicted in figure 11 is a closed one where N statistically identical transactions are circulating. It consists of a homogeneous multiprocessor system serving a shared queue, a disk subsystem with constant service times and no contention, and a set of N terminals where the transactions are started. The logical model represents the data access behavior of the transactions. Each transaction accesses a constant number k of data items. The execution of a transaction therefore consists of k+2 phases: an initialization phase, k phases with gradually increasing data set size, and a final phase for commit processing. The data items are selected randomly (i.e. no hot spots). As CC algorithm we use a timestamp certification scheme [Bernstein et al., 1987], because an optimistic protocol is more interesting due to its relationship between data contention and resource contention. The parameters used are roughly the same as in [Yu et al., 1987] that where derived from customer workload traces.

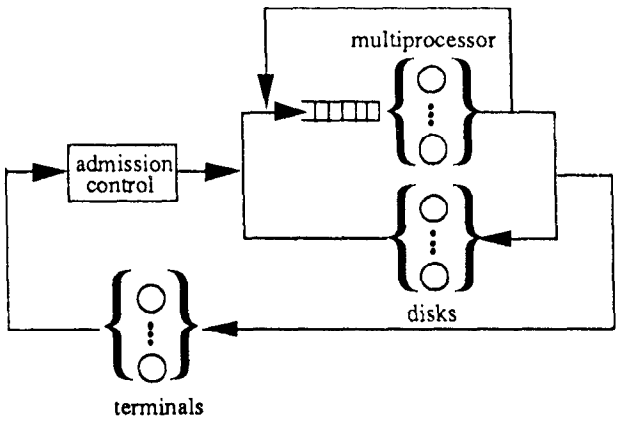

Figure 11: Simulation model

The dynamic change of the load charactistic was carried out by varying one of the following parameters:

- k, the number of locks per transaction
- fraction of queries
- fraction of write accesses for updaters

Variation of all these parameters showed significant impact on both height and position of the optimum throughput.

9 Simulation results

We first tested the two control algorithms under stationary conditions. All parameters were kept constant. For different levels of concurrency a stationary simulation run was conducted.

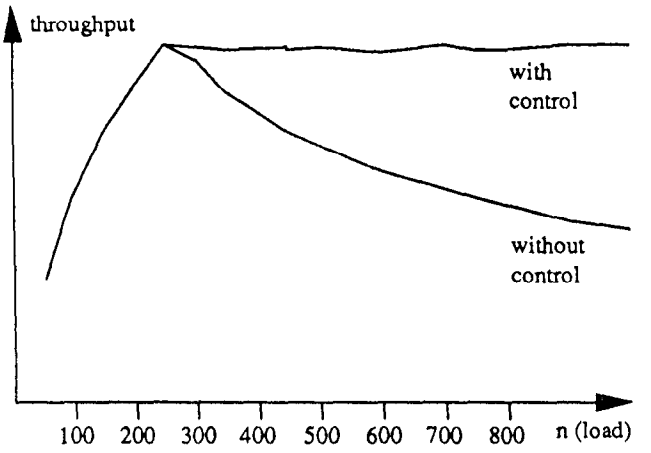

Figure 12: System throughput with and without control in the stationary case

Figure 12 shows that both algorithms had the desired property to keep the load at the point of optimum throughput and thus prevented thrashing. (Actually, figure 12 shows the resulting throughput for the PA algorithm. The difference between PA and IS was insignificant in this case.)

The major focus of the study, however, was on dynamic behavior. To that end, the above mentioned parameters were changed during the simulation runs in two fashions: (1) a jump-like variation to model abrupt changes in the workload and (2) a sinusoidal variation modelling more smooth and gradual changes. While both algorithms were able to follow gradual changes, the more sophisticated PA algorithm was clearly superior to IS in the case of jump-like changes: Figures 13 and 14 show examples of their respective behavior. The broken line indicates the position of the true optimum n_{opt}, and the solid line is the trajectory of the load threshold n^* adjusted by the respective controller algorithm. IS in figure 13 reacts very quickly to the jump of the optimum's position but has serious problems to adjust correctly to the new load situation. The PA algorithm (figure 14) needs some more time to respond but tracks the optimum more accurately and reliably. The oscillations of the trajectory in figure 14 are enforced by the algorithm as explained in section 4.2.

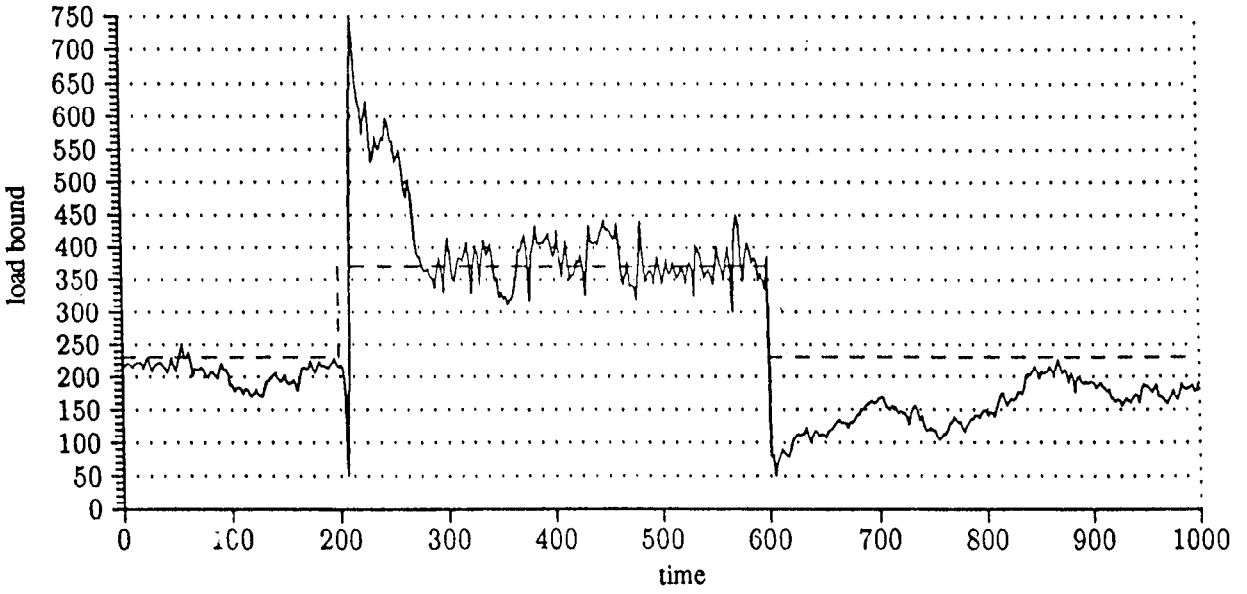

Figure 13: Trajectory of the Incremental Steps when the position of the optimum changes abruptly

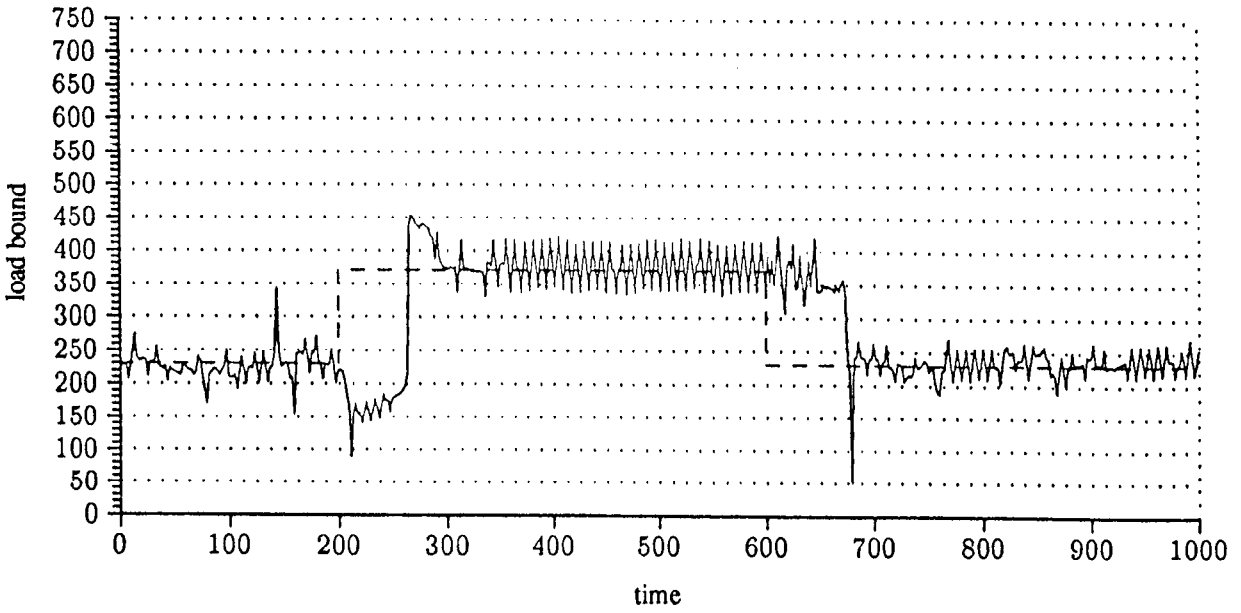

Figure 14: Trajectory of the Parabola Approach when the position of the optimum changes abruptly

10 Conclusions

Transaction processing systems need a control of the concurrency level to prevent thrashing effects. This control mechanism should be adaptive to cope with the dynamically changing load. The control problem can be approached as a dynamic optimum search problem for which heuristic algorithms of different complexity and quality are available. The reasonable assumption that the only local maximum is also a global one excludes the problem of getting stuck in a local optimum which other

hill climbing problems are faced with. The problems remaining are those of stability and reliability. We showed how these problems can be overcome by the two presented algorithms. The simulation experiment was designed to reveal the behavior of the algorithms in difficult situations. The more sophisticated PA algorithm outperformed the simpler IS algorithm in all cases examined and always avoided thrashing.

References:

R. Agrawal, M.J. Carey, M. Livny: Concurrency Control Performance Modeling: Alternatives and Implications. *ACM TODS* 12,4 (Dec.1987), pp. 609-654.

P. A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems. *Addison Wesley*, 1987.

A. Dan, D. F. Towsley, W. K. Kohler: Modelling the Effect of Data and Resource Contention on the Performance of Optimistic Concurrency Control Protocols. *Proc.4th Conf. on Data Engineering*, Los Angeles (Feb.1988) pp. 418-425.

P. J. Denning: Thrashing: Its Causes and Prevention. Proc. *AFIPS FJCC* 33, 1968, pp. 915-922.

P. J. Denning: Working Sets Past and Present. *IEEE TOSE* 6,1 (Jan.1980) pp. 64-84.

P. J. Denning, K. C. Kahn, J. Leroudier, D. Potier, R. Suri: Optimal Multiprogramming. *Acta Informatica* 7 (1976) pp. 197-216.

P. Franaszek, J. T. Robinson: Limitations of Concurrency in Transaction Processing. *ACM TODS* 10,8 (March 1988) pp. 1-28.

H.-U. Heiss: Overload in Computer Systems. *Springer-Verlag*, Heidelberg, 1988 (in German)

H.-U. Heiss: The Generalized Thrashing Effect and its Prevention. *IBM Research Report* No.RC14667 (June 1989), IBM Res. Div., Yorktown Heights, NY10598

H.-U. Heiss: Overload Effects and Their Prevention. (to appear in *Performance Evaluation*)

B. R. Iyer: Limits in Transaction Throughput - Why Big is Better. *IBM Research Report* No.RJ6584 (Nov. 1988), IBM Res. Div., Yorktown Heights, NY10598

Y. C. Tay, N. Goodman, R. Suri: Locking Performance in Centralized Databases. *ACM TODS* 10,4 (Dec. 1985) pp.415-462.

A. Thomasian, I. K. Ryu: Analysis of Database Performance with Dynamic Locking. *JACM* 37,3 (July 1990) pp.491-523.

R. Wagner: Adaptive Load Control in Transaction Processing Systems, *Diploma thesis*, University of Karlsruhe, Faculty for Informatics, 1990 (in German).

P. Young: Recursive Estimation and Time-Series Analysis. Springer-Verlag, Berlin, 1984.

P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, D. W. Cornell: On Coupling Multi-Systems Through Data Sharing. *Proc. IEEE* 75,5 (May 1987) pp. 573-587.