# LOCKING GRANULARITY REVISITED

by

D.R. Ries and M.R. Stonebraker

Memorandum No. UCB/ERL M78/71

6 October 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

# LOCKING GRANULARITY REVISITED

by
Daniel R. Ries
Michael R. Stonebraker

## ABSTRACT

Locking Granularity refers to the size and hence the number of locks used to insure the consistency of a database during multiple concurrent updates. In an earlier simulation study we concluded that coarse granularity, such as area or file locking, is to be preferred to fine granularity such as individual page or record locking.

However, alternate assumptions than those used in the original paper can alter that conclusion. First, we changed the assumptions concerning the placement of the locks on the database with respect to the accessing transactions. In the original model the locks were assumed to be well-placed. Under a worst-case and random placement assumptions, when only very small transactions access the database, fine granularity is preferable.

Second, we extended the simulation to model a lock hierarchy where large transactions use large locks and small transactions use small locks. In this scenario, again under the random-worst case lock placement assumptions, fine granularity is preferable if all transactions accessing more than 1% of the database use large locks.

Finally, we simulated database systems which support a "claim as needed" locking strategy together with the resultant deadlock. In the original study all locks were claimed in one atomic operation at the beginning of a transaction. The claim as needed strategy does not change the conclusions on the desired granularity.

Key Words and Phrases: concurrency, database management, locking granularity, locking hierarchies, multiple updates

CR Categories: 3.50, 4.33

This research was sponsored by the Air Force Office of Scientific Research, grant 78-3596; and the Naval Electronic Systems Command, Contract N00039-78-C-0013.

## 1. INTRODUCTION

In a previous paper [RIES77], we examined the effects of using locks of various sizes (or "granules") for concurrency control in a database management system. In this paper, we report three significant extensions that affect the conclusions of that study. We first briefly review some aspects of concurrency control in a database system and present an overview of the results in the original paper.

### 1.1. Database Concurrency Control

The concurrency control mechanism in a database management system is responsible for ensuring some level of consistency [ESWA76] during the processing of concurrent updates. A variety of concurrency control mechanisms have been suggested [CHAM74, GRAY76, MACR76, STEA76] which lock required portions of a database while a transaction is in progress.

Some mechanisms require the notion that certain units of the database can be locked. We shall refer to a unit of the database which can be locked by the concurrency mechanism as a "granule". The size of the granule in different database management systems varies. In some systems (CODASYL [CODA73], System R [ASTR76], DMS-1100 [GRAY75]) the granule may be as small as one record. Other systems (System 2000 [SPIT76], IMAGE [HEWL77]) support one granule covering the entire database. Still other systems (DBMS-11 [DEC77], LSL [LIPS76]) support intermediate sized granules such as files or areas.

### 1.2. Previous Results

In the previous paper [RIES77] we described a simulation model which was used to examine the effects of different granule sizes on the efficiency of the database management system. It is clear that fine granularity allows a higher degree of parallelism at a greater cost in managing locks. Conversely, coarse granularity inhibits parallelism but minimizes lock management costs. The simulation model gave insight into the tradeoffs between the increased parallelism and the locking overhead.

In the simulation, a fixed number of transactions were cycled continuously around the model shown in figure 1. In this model, the database was an abstract collection of entities. An entity can be thought of as the unit of data moved by the operating system into the database system buffers.

The number of entities "touched" or accessed by a given transaction completely determined the amount of I/O, CPU and lock resources required by that transaction. The I/O and CPU resources required for processing a transaction were respectively equal to the number of entities touched times an I/O cost per entity and a CPU cost per entity. The proportion of available locks required was equal to the percentage of the entities touched by the transaction.

Initially, the transactions arrived one time unit apart and were put on the pending queue. A transaction then went through the following stages.

a) The transaction was removed from the PENDING queue and all required locks were requested. If the locks were granted, the transaction was placed on the bottom of the I/O queue. If the locks were denied, the transaction was placed on the bottom of a BLOCKED queue. The blocking transaction was recorded. Note that no locks are held while on the blocked queue so deadlock was impossible.

b) After completing the I/O required, the transaction was placed on the bottom of the CPU queue.

c) After completing the CPU required, the transaction released its locks and joined the end of the PENDING queue. Note that each transaction went through one I/O phase and one CPU phase. Although they were sequential in the model, the result would be the same if each transaction went through many I/O - CPU phases in a single cycle. All transactions that were blocked by the completed transaction were placed on the front of the PENDING queue.

The major parameters in the model were:

1) The number of transactions.

2) The number of entities "touched" by a transaction.

3) The distribution of the entities touched by transactions.

4) The I/O cost of processing an entity.

5) The CPU cost of processing an entity.

6) The number of granules or locks.

7) The I/O cost of setting one lock.

8) The CPU cost of setting one lock.

The simulation was run varying the size of the granules while holding other factors fixed. Figure 2 shows a typical output from the model. In this figure, the average transaction "touched" 10% of the database, the I/O cost for accessing an entity was four times the CPU costs for manipulating the data within one entity (simulating I/O bound transactions), the I/O cost for setting a lock was equal to the I/O cost for accessing an entity, while the CPU cost for setting a lock was 1/5 of the CPU cost for manipulating an entity.

Note that machine utilization increases as the number of granules increases then levels off and falls. Also note that maximum utilization occurs at a relatively small number of granules and that utilization is within 1% of this optimum for 10 granules. The conclusion can be drawn that crude locking schemes with coarse granularity are nearly optimal. Since a crude locking system may be easier to implement than a sophisticated finer granularity scheme, it is preferred. This conclusion was tested against many changes in model parameters. These experiments included changing the ratio of I/O time to CPU time required by a transaction; changing the CPU and I/O resources required for locking; and changing the number of I/O channels. The model was also changed to reflect not keeping all of the locks for the entire duration of the transaction processing and to reflect the keeping of locks for a "think period" during which no useful I/O or CPU activity for the given transaction took place. In all cases curves similar to figure 2 resulted.

Three situations, however, did favor somewhat finer granularity than shown in figure 2. First, reducing the costs associated with locking, particularly keeping all locks in core (i.e. I/O lock cost was set to zero), significantly reduced the heavy overhead costs associated with fine granularity. The second case was was using very small transactions. In this case, the optimum granularity in figure 2 was moved to the right. Finally, a balance between the I/O and CPU costs for processing each entity needed by a transaction also moved the optimum to the right. Such a "balanced" system creates the greatest potential for I/O and CPU overlap. However, even in these cases coarse granularity would have served almost as well (i.e. within 5%).

## 1.3. Model Extensions

Several people have questioned the validity of some of our model assumptions. In particular our assumption concerning the number of locks that must be obtained for each transaction is sometimes suspect. Potentially, many more granules may be required to lock a given number of entities than allowed in the original model. Alternate assumptions concerning these numbers are explored in section 2. Also, a lock hierarchy [GRAY76] could have significantly changed our results. In a lock hierarchy, different transactions use different size granules for locking. With such a hierarchy, the costs of locking may be greatly reduced since it would be cheaper to set one large lock than to set many small locks. This alternative is explored in section 3. Finally, in our original model, all locks were requested at the beginning of a transaction. In some database locking mechanisms, locks are requested during the transaction processing. These "claim as needed" mechanisms can result in more concurrency since locks are not held during the entire lifetime of a transaction. However, the possibility of deadlock is introduced. This case is discussed in section

4. In section 5, we present our overall conclusions.

## 2. GRANULE PLACEMENT

In the original model, the number of locks required by a given transaction was exactly proportional to the percentage of the data base "touched" or accessed by the transaction. Hence, a transaction which touched half of the entities in the database would require half of the possible database locks. Note that this amounts to assuming that the granules are "well placed", i.e. that the entities needed by the transactions are packed into as few 'lockable' granules as possible. This assumption is reasonable for transactions which access the database sequentially. Although sequential processing in database applications has been observed [RODR76], actual transactions may require a combination of sequential and random accesses to the database. R. Fabry was the first to suggest that we explore the effects of assumptions other than "well placed" granules on the locking granularity.

### 2.1. Alternative Granule Placement Assumptions

In order to study the effects of different lock placement assumptions two alternatives to "well-placed" locks were explored.

1) Worst case access:

Each transaction requires the maximum number of granules possible. If the total number of entities touched by a given transaction, say NE, is greater than the number of locks covering the entire database, say NL, then in the worst case, all of the locks might have to be set to access the needed entities. If NE is less than NL, on the other hand, the number of locks that have to be set is bounded by the number of entities, NE. Thus the number of locks required is the minimum of the number of locks for the entire data base and the number of entities touched by the transaction. This assumption simulates an "uncooperative" transaction; i.e. one whose access pattern is the worst possible from the point of view of the locking mechanism. This scenario is the opposite extreme of the "well placed" assumption.

2) Random access :

For each transaction, a mean-value formula is used to estimate the number of locks required. Let s be the number of records in the database; nlks be the total number of locks; and p be the number of records per block (=s/nlks). Then a transaction which accesses r records will require

$$\text{nlks} * \left[ 1 - \frac{C_r^{s-p}}{C_r^s} \right]$$

locks. The $C_r^{s-p}$ and $C_r^s$ represent the number of different ways r records can be selected from s-p and s records respectively. The number of locks required under the random access assumption is analogous to the number of blocks accessed when randomly selecting records from a blocked file. A mean-value formula for this number and its derivation are given in [YAO77]. This model accurately reflects random processing where the probability of accessing any entity is the same and independent of any previous entities accessed.

Which model is more accurate depends on the nature of transactions in a given application. However, our experience is that this will normally be somewhere between well-placed and random depending on how much sequential processing is done. Which model is chosen affects our original results. If the "worst case" is chosen, the following intuitive analysis applies. The graph in figure 3 assumes that all transactions touch the same number of entities, NE. The machine utilization measures decreased as the number of locks for the entire database increased from one to NE. The decrease is because each transaction will require more and more locks thus increasing the locking overhead. However, there is no additional parallelism because each transaction locked t e entire database.

The utilization increases, however, as the number of locks increases from NE to the total number of entities in the database because the cost of the locking overhead will remain constant while the allowed concurrency increases. The locking overhead remains constant since each transaction can never set more than NE locks. Consequently, the optimum number of locks is very dependent on the transaction sizes in the worst case placement lock assumption. Moreover, it will always occur at 1 granule or the maximum number of granules (corresponding to one lock per entity) if all the transactions are the same size.

The effects of having varying transaction sizes will be discussed below.

## 2.2. Simulation Extension

The simulation model was run for each of the three placement assumptions under a wide variety and combination of parameter values. Figures 4 and 5 diagram some of the results. In figure 4, the transaction sizes were determined by an exponential distribution with a mean value of 500 entities (10% of the database). In figure 5, the transaction sizes were also determined by an exponential distribution but with a mean value of 5 entities (0.1% of the database). For these runs, the locks were assumed to be in core (no lock I/O required) and the I/O and CPU time required by the transactions were equal. These conditions were chosen as the ones most favorable to finer granularity. The other parameters were similar to those described in the original paper. The top curve in both figures is consistent with the results in our original paper. The bottom two curves represent the worst case and random access assumptions.

For large transactions requiring about 10% of the data base (see figure 4) a smaller number of granules was still to be prefered to a lock for each entity. For small transactions requiring about 0.1% of the database (see figure 5) one lock per entity produced the greatest machine utilization under the worst case and random placement assumptions. However, even with small transactions, the degree of improvement was small as the granularity increased beyond a certain limit. With 200 locks for example, 90% of the maximum machine utilization was reached.

Next, the simulation was run with mixed size transactions (the size being generated by hyper-exponential distributions, also suggested by R. Fabry) using the best case, the worst case and random access assumptions. Intuitively, this simulates a few large transactions and many small ones. As in the original paper, under the well-placed assumption a small number of granules was best. A relatively flat curve relating machine utilization and the number of locks was observed for the worst case and random access assumptions. Thus, in these two cases, fine granularity did not interfere with useful I/O and CPU time, but more coarse granularity would have served as well. In fact, 98% of the maximum utilization was achieved by 10 granules. The basic problem with fine granularity is that the expense of running just a few large transactions seems to outweigh the gain due to the increased concurrency experienced by the small transactions.

## 2.3. Conclusions

These studies do change our original conclusions somewhat. Fine granularity may be best if the following two conditions are meet: 1) almost all of the transactions are small and 2) access patterns are random with no sequentiality. Under these conditions, the greater the number of locks, the greater the machine utilization. However, the rate of increase drops dramatically after a certain level of granularity is obtained (about 200 granules in our simulation). Hence "medium" granularity does almost as well as fine granularity; coarse granularity is unacceptable in this case.

If too many of the transactions access a large portion of the database, fine granularity produces too much locking overhead and coarse granularity is again to be preferred.

Regardless of the transaction sizes, in the data access patterns are primarily sequential, coarse granularity is still the most effective.

## 3. LOCK HIERARCHY

One way a large transaction can avoid the expense of locking many small granules might be to have the large transactions lock large granules while the small transactions continue to use the small locks [GRAY76]. An extension of our simulation model to study the effects of such a lock hierarchy on the desired granularity was first suggested by R. Karp.

## 3.1. The Model Extension

In the simulation extension a two level hierarchy was implemented. A transaction, depending on its size, either requested a set of small locks or one global lock which covered the entire data base. With this extension, we explored the interactions between any two levels of a more general hierarchy.

The simulation was modified by adding 'pending' and 'blocked' queues for the global lock. If a transaction is "small", it sets the global lock in shared mode and is placed on the original pending queue. From that queue the "small" transactions must compete for the small locks as in the original model. If the transaction is "large", the global lock is set for exclusive access and the transaction waits for all active transactions to finish. With the global lock set for exclusive use, new transactions, regardless of size would also wait in the blocked queue. Once the large transaction was allowed to proceed, it goes directly to the I/O queue bypassing the small lock control.

The simulation was used to study the effects of certain parameters of such a hierarchy on the desired granularity. One of the main areas of interest was the criteria for deciding whether the small locks or the global lock should be used. An input parameter was added to the simulation which specified the threshold percentage, tp, of the data base which must be touched by a transaction before it is declared "large". If a transaction used less than tp percent of the database, the small locks would be used. Otherwise, only the global lock would be set.

## 3.2. The Simulation Results

The simulation was run with threshold percentages of 0.1%, 0.2%, 0.5%, 1%, 2%, 5%, 25%, 50% and 100% for each of a large number of other parameter settings in order to find the value of tp which maximized machine utilization. The optimum threshold observed was dependent on the number of small locks, the assumptions concerning the placement of those locks, the number of entities touched by the transactions, and the size of the database.

Figures 6 and 7 compare the threshold percentages with the number of small locks. The depicted results represent simulation runs with a database of 5000 entities and transaction sizes determined by a hyper-exponential distribution with 90% of the transactions touching an average of 5 entities and 10% touching an average of 500 entities. Again the I/O lock costs were assumed zero (locks kept in core) and the CPU and I/O costs for a given transaction were equal -- two factors favoring finer granularity in our original study. The results in Figure 6, reflect the best case assumption that the transactions are sequential. Random access of the database is assumed for the simulation runs for figure 7.

Each of the graphs is divided into three areas based on machine utilization. The "optimum" line represents the threshold value, tp, at which the maximum I/O and CPU utilization was observed for a fixed number of small locks. With threshold values in area B, the hierarchical locking produced results within 2% of that maximum utilization. In area A, the utilization was less than in area B. In this case, too few transactions used the global lock, i.e. the threshold, tp, was set too high. In area C, the machine utilization was also less than in area B. In this case, however, too many transactions have used the global lock, i.e. the threshold, tp, was too low.

For example, consider figure 6 with 1000 small locks. The machine utilization increased as the threshold percentage was increased from 0.1% to 5%, but decreased as the threshold increased from 5% to 100%. However, simulation runs with threshold percentages between 1% and 25% produced within 2% of the machine utilization observed with the optimum threshold.

In figure 6, "well-placed" granules are assumed. With more than 1000 small locks the optimum value of tp was between 1% and 5%. With the number of locks between 10 and 100, tp values of 50% to 100% were optimal. In this granularity interval, the 2% area included the case where all transactions used only the small locks. The overall maximum machine utilization occurs in figure 6 with 10 locks and tp values greater than 50%. In these cases, almost all of the transactions used the small locks. Hence, the value of a lock hierarchy under the well-placed locks assumption is very small.

However, in figure 7, the random access patterns for each transaction are assumed. With coarse granularity, the optimum threshold occured at 0.5%. With a higher threshold, more of the smaller transactions would use the small locks, and consequently would lock a large portion of the database. As a result, these transactions would expend more resources for locking than if the global lock were used without significantly increasing the concurrency allowed.

In figure 7, the differences in computer utilization between areas A, B, and C is small for coarse granularity. For 10 granules, for example, no matter what value of tp is used, the computer utilization is within 3% of the maximum observed for that granularity. Similarly, for 100 granules, the computer utilization was within 15% of the maximum observed for any value of tp. Thus even with random access transactions, a hierarchy with a small number of small locks can at best provide slight improvement over a single level locking system.

Under the random access assumptions, the overall maximum machine utilization occurred with 5000 granules and a tp of 1%. The cross-hatched area in figure 7 represents those combinations of tp and number of small locks which resulted in machine utilization within 2% of the overall maximum. Hence, fine granularity is to be preferred. The lock hierarchy effectively prevents excessive locking overhead for large transactions.

For fine granularity, the B areas in figures 6 and 7 have considerable overlap. For example, in figure 6, with 2500 small locks, the 2% of optimum interval occured with a tp between 0.5% and 10%. In figure 7, with the same number of small locks, the interval occurred with tp values between 0.5% and 5%. Thus, at this granularity, a tp between 0.5% and 5% can safely be chosen regardless of the randomness of the data access patterns.

Other simulation experiments used the worst case data access assumption and produced results very similar to those in figure 7.

In still other simulation runs, as the average transaction size decreased, the range of acceptable tp values (area B) also decreased. With fine granularity, regardless of the transaction sizes, a threshold between 1% and 2% always produced machine utilization within 2% of the maximum.

With coarse granularity, however, changes in the size of the transactions, created non-overlapped intervals of acceptable tp values. In other words, no one value of tp could be chosen that would be correct for vastly different sized transactions. Thus much greater care must be applied to a hierarchy with coarse granularity. Furthermore, a stable transaction size environment must be assumed.

The size of the database was also varied. For example, we ran the simulation with a database consisting of only 16 entities. In this scenario, we examined the possible interaction of a page/record hierarchy. An entity corresponds to one record in a page which holds 16 records. In our simulation, we were then modelling the effects of locking the whole page by the global lock, or locking individual records by the small locks. Some increase in machine utilization was observed with a threshold of 50%; but the increase over using no hierarchy at all was less than 4%. Again it appears that a lock hierarchy covering only a small number of smaller locks is not worth implementing.

We also ran the simulation with databases of up to 100,000 entities. The results were similar to the results produced with a database of 5,000 entities. For example, we ran cases where the average transaction size of most of the transactions was just 0.05% of a 100,000 entity database and the average size of a few large transactions was 1% of the database. In these cases, with the finest granularity (100,000 small locks), a threshold of 1% was still optimal.

## 3.3. Conclusions

A locking hierarchy should be implemented when the small locks are of a fine granularity; a low threshold is used to separate the large and small transactions, and random data access patterns are anticipated. With this model the increase in machine utilization over a single level locking scheme is potentially significant. Furthermore, a threshold of about one percent can be selected independent of the granule placement or transaction size assumptions.

With coarse granularity, on the other hand, a locking hierarchy should not be used. The potential gain is not significant and is only realized in certain cases. Another problem with the coarse granularity/locking hierarchy model is that the optimum value for the threshold percentage is extremely sensitive to the placement of the locks with respect to the transactions.

## 4. CLAIM AS NEEDED LOCKING

Another difference between our model and some database concurrency control immplementations deals with when the locks are actually acquired. In the original study, a "pre-claim" model was assumed where all of the locks were acquired before any transaction processing took place. In some database systems, a lock is not acquired until the related entities are actually needed by a transaction. These "claim as needed" schemes are used either to reduce the total time locks are held and/or because the locks to be acquired depend on data values of entities already accessed. In these cases, some locks may have to be held while other locks are requested, and deadlock can occur [COFF71]. In this section we examine the effects on the optimum granularity of a claim as needed scheme.

### 4.1. The Model Extension

The simulation was modified by cycling each transaction through the I/O and CPU queues (see figure 1) once for each lock required. The total I/O and CPU times required for a transaction are the same as in the original model and are equally distributed among each of a transaction's cycles.

Between each cycle, a transaction requests one lock. If the lock is granted, the transaction goes on the active queues. When a transaction has completed its last cycle on the active queues, all its locks are released as in the original model.

If the lock is denied, the requesting transaction is placed on the blocked queue. The lock can be denied due to locks held by other active transactions, or by blocked transactions. If the blocking transaction is on the blocked queue, a deadlock condition can exist. If deadlock occurs, a victim is picked for backout. The locks held by the victim are released, any blocked transactions are freed, and any time spent on the active queues by the victim is added to a "lost time" total.

## 4.2. The Simulation Results

The modified simulation was run varying the sizes of the transactions, changing the lock placement assumptions, and with and without a lock hierarchy. Again we assumed that there is no I/O cost associated with locking and that the transactions required equal amounts of CPU and I/O resources.

The results of these simulation runs are very similar to those in [RIES77] and those presented above for the preclaim strategy. In all cases, a claim as needed strategy does not change the granularity required for maximum machine utilization.

For example, figure 8, shows the results of running the simulation with no hierarchy, well placed granules and transaction sizes determined by a hyper-exponential distribution (see section 3.2). The lost time area represents the machine utilization by transactions that had to be restarted due to deadlock. The useful computing includes only the I/O and CPU resources used by successfully completed transactions.

The locking cost in the preclaim model is greater than the locking cost in the claim as needed locking model. This difference is because in the case of a lock request failure, in the preclaim model all of the locks must be requested again. In many cases, any decrease in lock costs, however, was more than offset by the lost time due to deadlock resolution. Thus the useful machine utilization was greater under the preclaim model than under the claim as needed strategy. Many other cases with different transaction sizes and lock placement assumptions were also tested and produced similar results.

For example, figure 9 compares the useful machine utilization between the two models under the assumptions that all transactions are small and that each transaction has random data access patterns. In both of these runs, the average transaction size was 0.1% of the database. Note that, with the possibility of deadlock, the machine utilization curve does not flatten out as the granularity increased in contrast to our observation in section 2.3. In these cases, we thus conclude that with a claim as needed model, the finest granularity may be manditory. Note, however, that the claim as needed scheme again produced less useful I/O and CPU utilization than the preclaim model.

However, as the average transaction size became even smaller, the last observation does not hold. With an average transaction size of less than 0.05% of the database, random data access patterns, and the finest granularity, the claim as needed scheme resulted in greater useful machine utilization. Under these conditions, the claim as needed strategy allowed the greatest concurrency since locks were held for a shorter period of time. In contrast to our other runs, very few transactions had to be backed out and the cost of rerunning such small transactions was insignificant.

The modified simulation was also run with a lock hierarchy and various threshold percentage values. A similarity in the shapes of the curves between the preclaim and claim as needed strategies was also observed. Under the random access assumptions, for example, the maximum machine utilization is again reached with the finest granularity and a threshold value of 1 to 2 percent.

### 4.3. Conclusions

The acquisition of locks throughout the processing of a transaction does not significantly change the other conclusions of this paper. However, two observations should be made. First, deadlock detection and resolution appears to be generally more expensive than the release and rerequest used in the preclaim strategy. When locks are known at the start of a transaction a preclaim algorithm is suggested.

Second, if all of the transactions are small and a random access pattern is present (see figure 9), an algorithm allowing for deadlock makes the finest possible granularity essential.

## 5. SUMMARY

In a previous simulation study, we concluded that for concurrency control locking in a database management system, coarse granularity, such as file, area or database locking, was almost always preferable to fine granularity such as individual page or record locking.

In this extension of the study, several alternative assumptions showed that there are cases where this conclusion doesn't hold. In particular, if ALL transactions are randomly accessing small parts of the database, then finer granularity is to be preferred.

However, if several of the transactions are large, a lock hierarchy must be used if the fine granularity is still to be supported. Such a hierarchy was shown to be beneficial mainly for fine granularity and random data access patterns. In these cases, all transactions touching more than 1% of the database should use a few large locks rather then many small locks. If the data access patterns were sequential, however, single level locking with coarse granularity was still to be preferred.

Both of these results hold, regardless of whether a preclaim or claim as needed strategy is used for lock acquisition. In general, with a few exceptions, if possible, a preclaim strategy produced better machine utilization than the claim as needed model.

Our overall conclusions are that the optimum locking granularity is somewhat application dependent. In many cases, coarse granularity, such as file or relation locking, with a preclaim strategy is to be preferred. Such a coarse granularity can be augmented with a lock hierarchy using very small granules, such as record or page locking within a file, at the lower level of the hierarchy. Then the concurrency control can be tuned to allow for a wider class of transactions. In particular, this class includes those transactions that randomly access just a few records of a given file. However, any gains of such a hierarchy are still application dependent and must be weighed against increased implementation and locking overhead costs.

## REFERENCES

ASTR76 Astrahan, M. et.al, "System-R: Relational Approach to Database Management," ACM Transactions on Data Base Systems, Vol 1, No 2, June 1976.

CHAM74 Chamberlin, D. et. al, "A Deadlock-Free Scheme for Resource Locking in a Data Base Environment",IBM Research Report, San Jose, Ca., June, 1974.

CODA73 CODASYL Programming Language Committee. CODASYL COBOL Data Base Facility Proposal, March 1973

COFF71 Coffman, Jr. E.G., Elphick, M.J., Shoshani, A., "System Deadlocks" Computing Surveys, Vol 3 No 2, June 1971 pp 67-78

DEC77 Digital Equipment Corparation, "DBMS-11 Data Base Administrator's Guide", DEC-11-ODABA-A-D, 1976.

ESWA76 Eswaran, K. P., Gray, J. N., Lorie, R. A., Traiger, L. I.; "On the Notions of Consistency and Predicate locks in a data base System ", CACM Vol 19, No 11, November, 1976.

GRAY75 Gray, J.N.,Lorie, R.A., and Putzolu, G.R. "Granularity of Locks in a Shared Data Base", Proc. 1975 VLDB Conference, Framingham, Mass., Sept., 1975.

GRAY76 Gray, J. N., Lorie, R. A., Putzolu, G. R. and Traiger, I. L.; "Granularity of Locks and Degrees of Consistency in a Shared Data

Base." Proc. IFIP Working Conference on Modelling of Data Base Management Systems; Freudenstadt, Germany; January 1976.

HEWL77 Hewlett-Packard Corporation, "IMAGE Reference Manual", 1977.

LIPS76 Lipson, W. and Lapezak, "LSL User's Manual"; Computer Systems Research Group, University of Toronto, Technical Note No 9, August, 1976.

MACR76 Macri, P., "Deadlock Detection and Resolution in a CODASYL Based Data Management System," Proc. 1976 ACM-SIGMOD Conference on Management of data, Washington, D. C., June, 1976

RIES77 Ries, D. R., Stonebraker, M. "Effects of Lock ng Granularity in a Database Management System", ACM Transactions on Database Systes, Vol. 2, No. 3 September, 1977

RODR76 Rodriquez-Rosell, J., "Empirical Data Reference Behavior in Data Base Systems" Computer Vol 9, No. 11, November 1976 pp 9-13

STEA76 Stearns, R. E. et al, "Concurrency Control for Data Base Systems " Proc 1976 ACM Symposium on Foundation of Computer Science, October 1976.

YAO77 Yao S. B., "Approximating Block Accesses in Database Organizations", CACM Vol. 20, No. 4, April 1977, pp 260-261

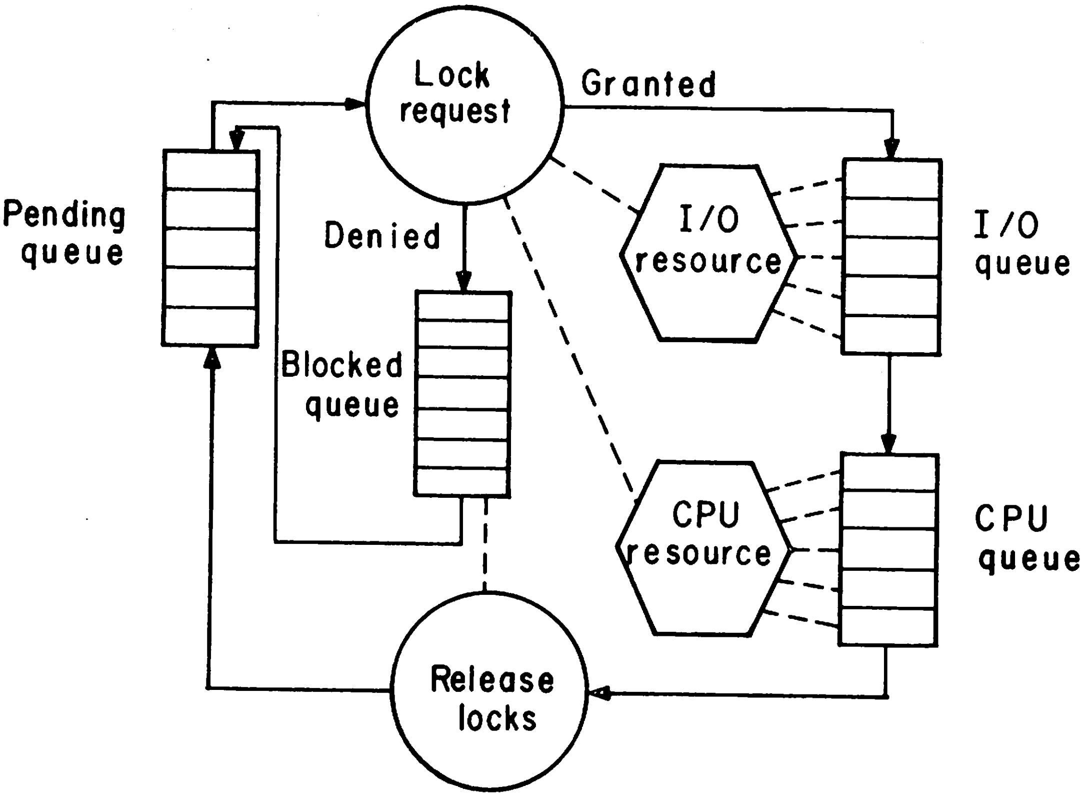


Figure 1: Simulation model

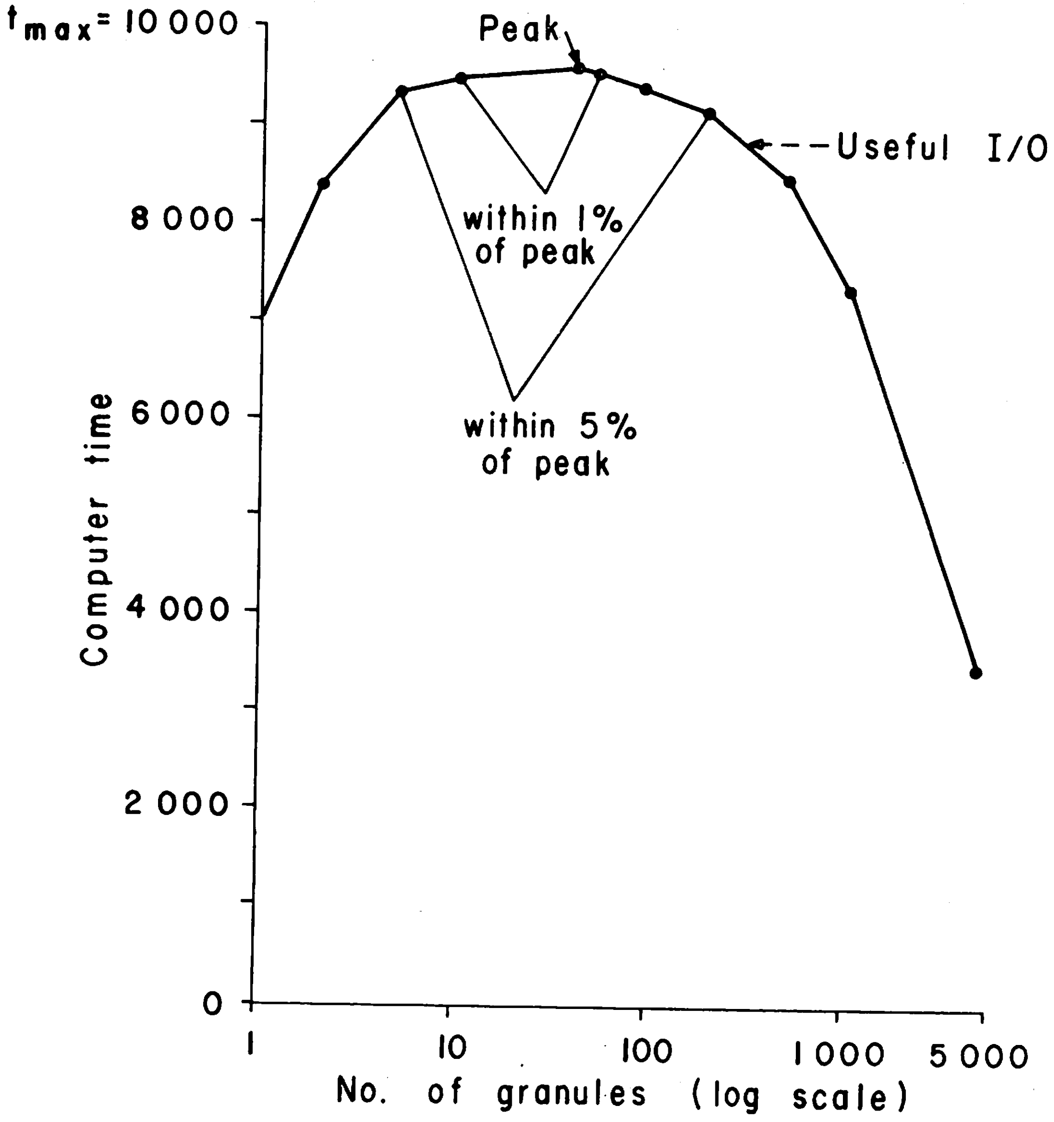


Figure 2: Computer time versus Number of Granules

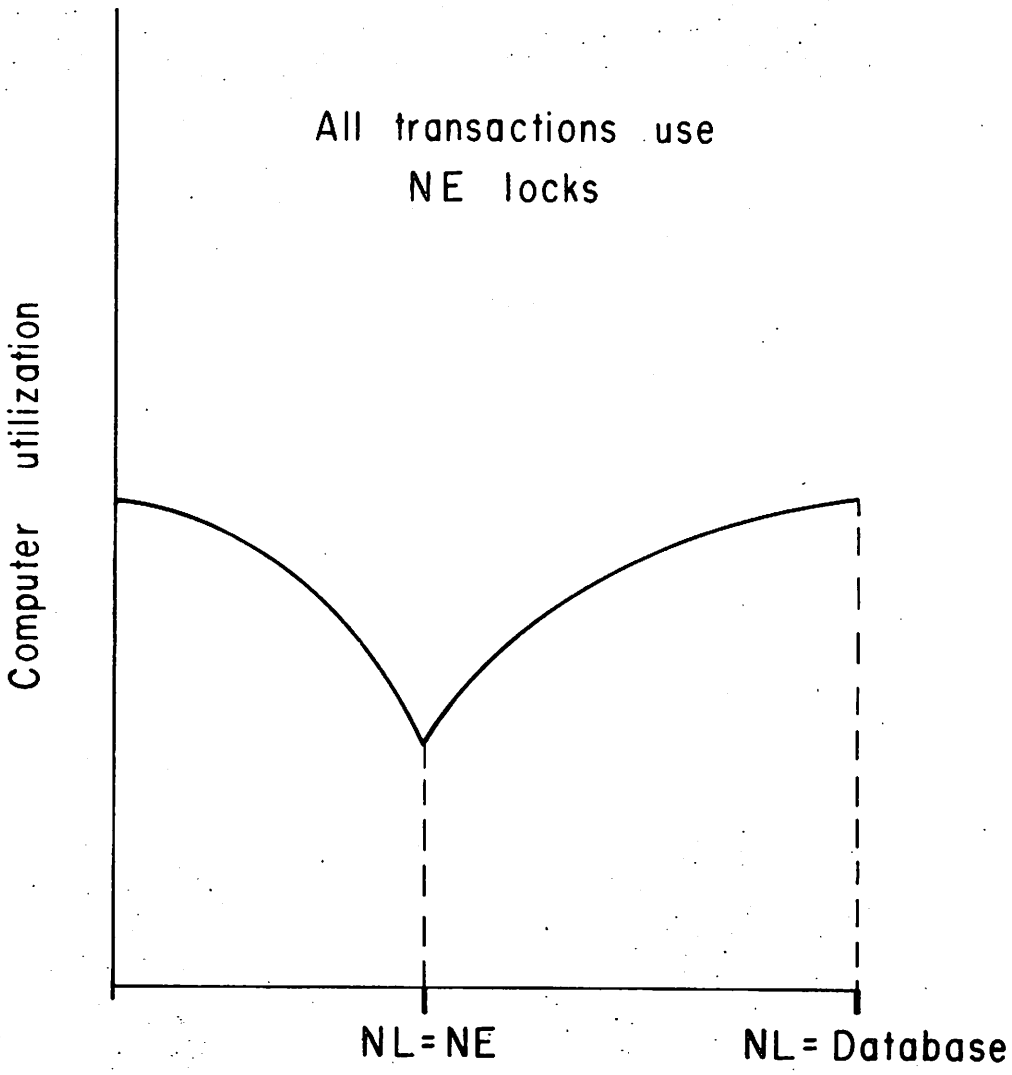


Figure 3: Worst Case Lock Placement

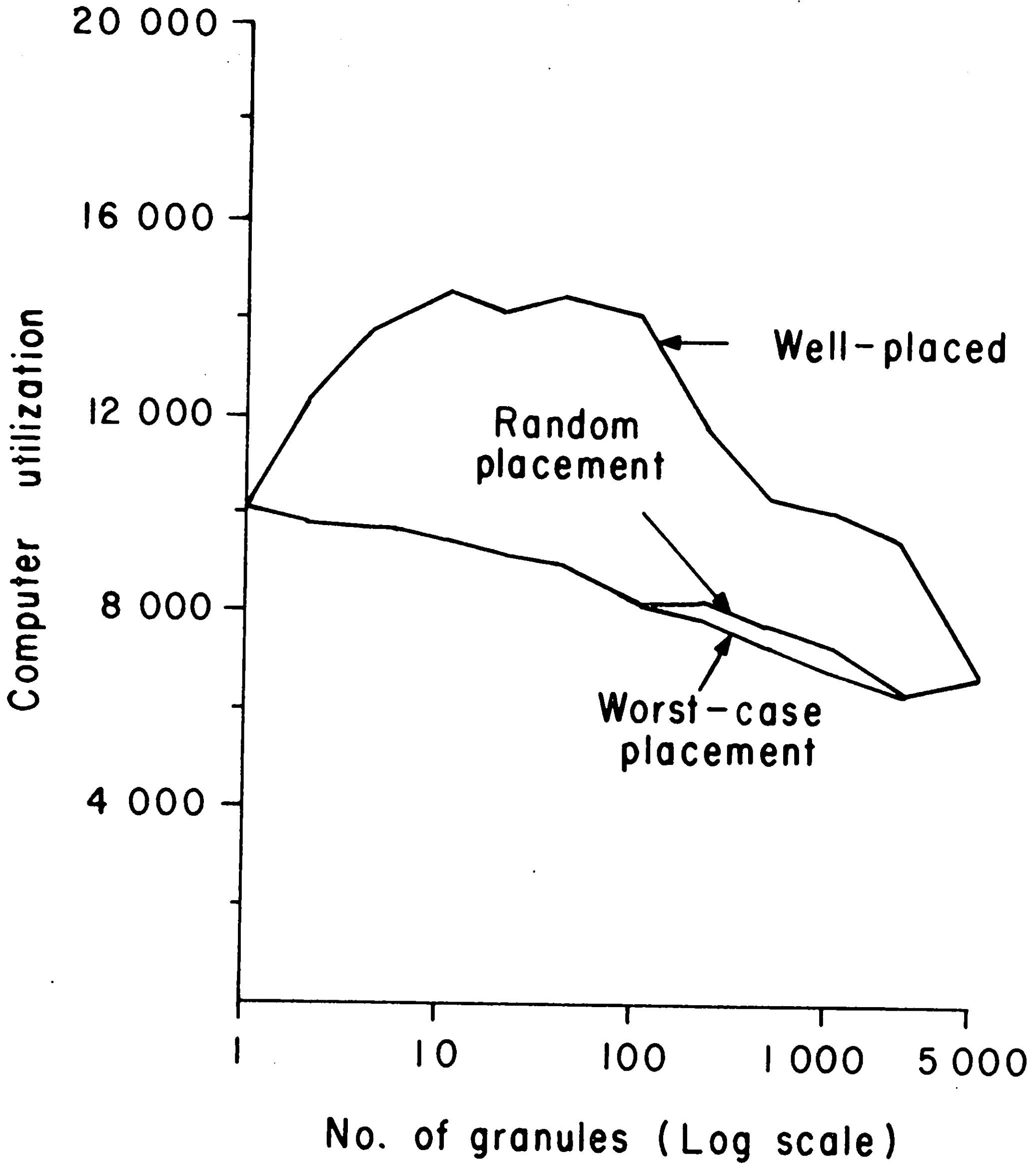


Figure 4: Large Transactions

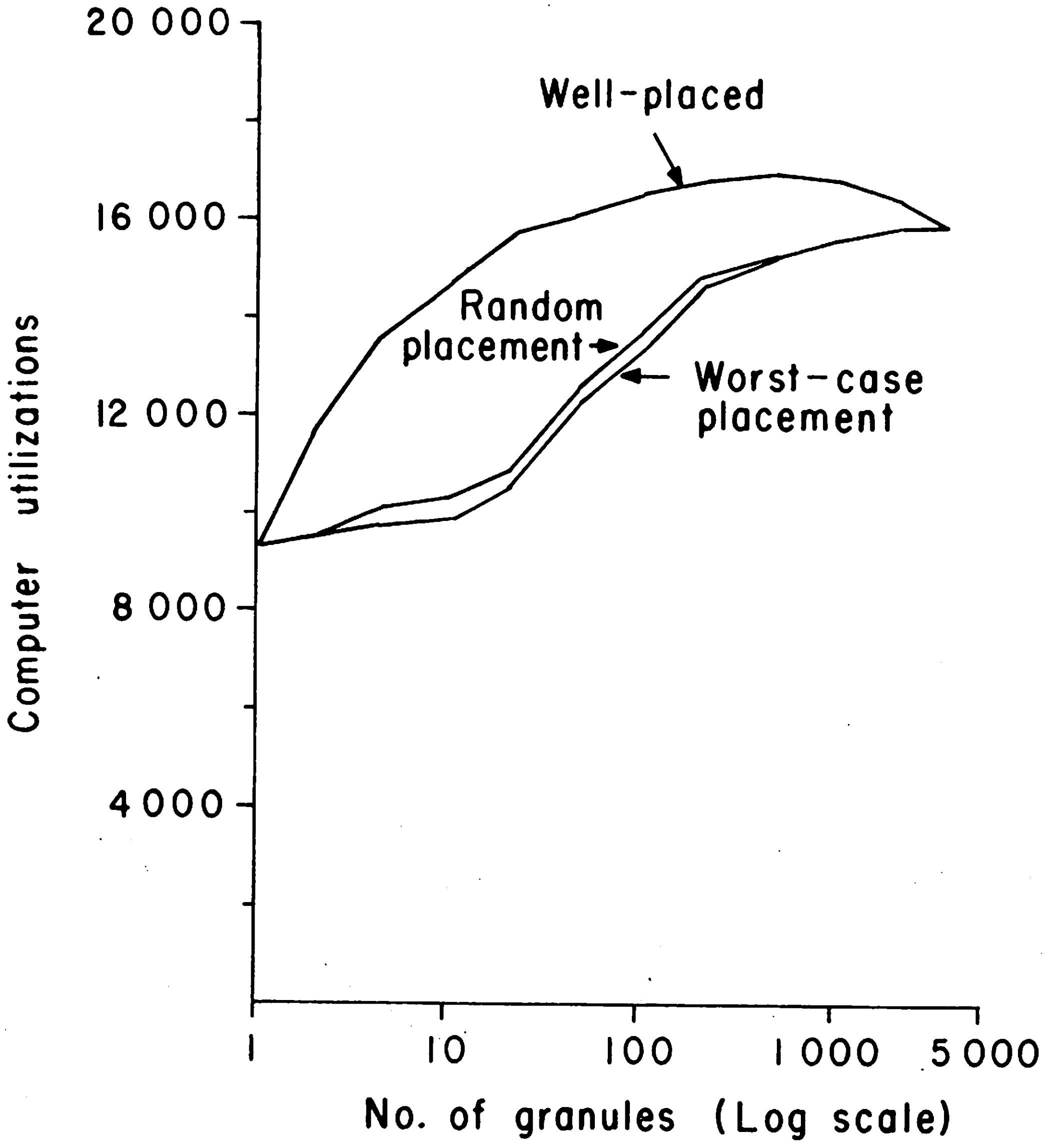
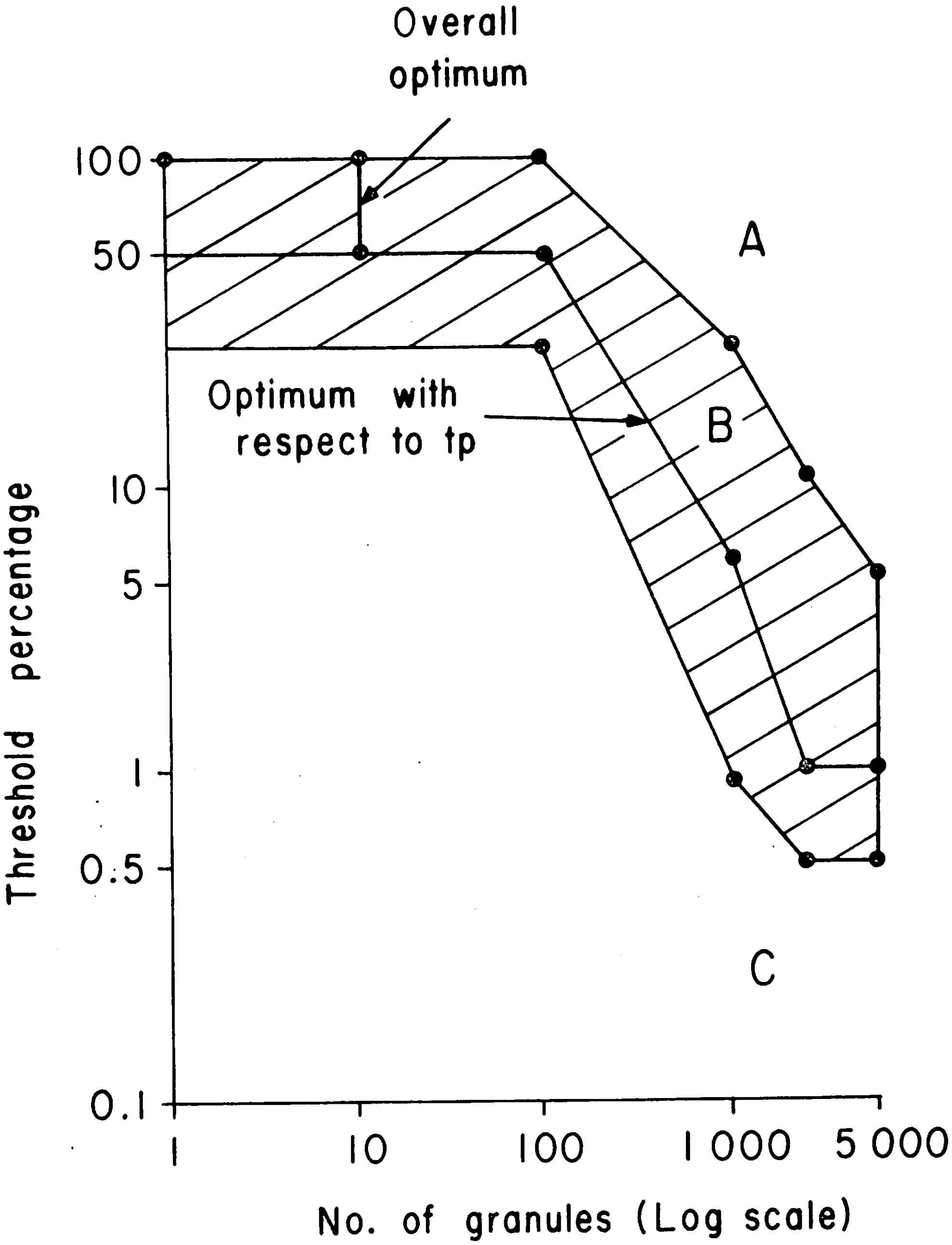
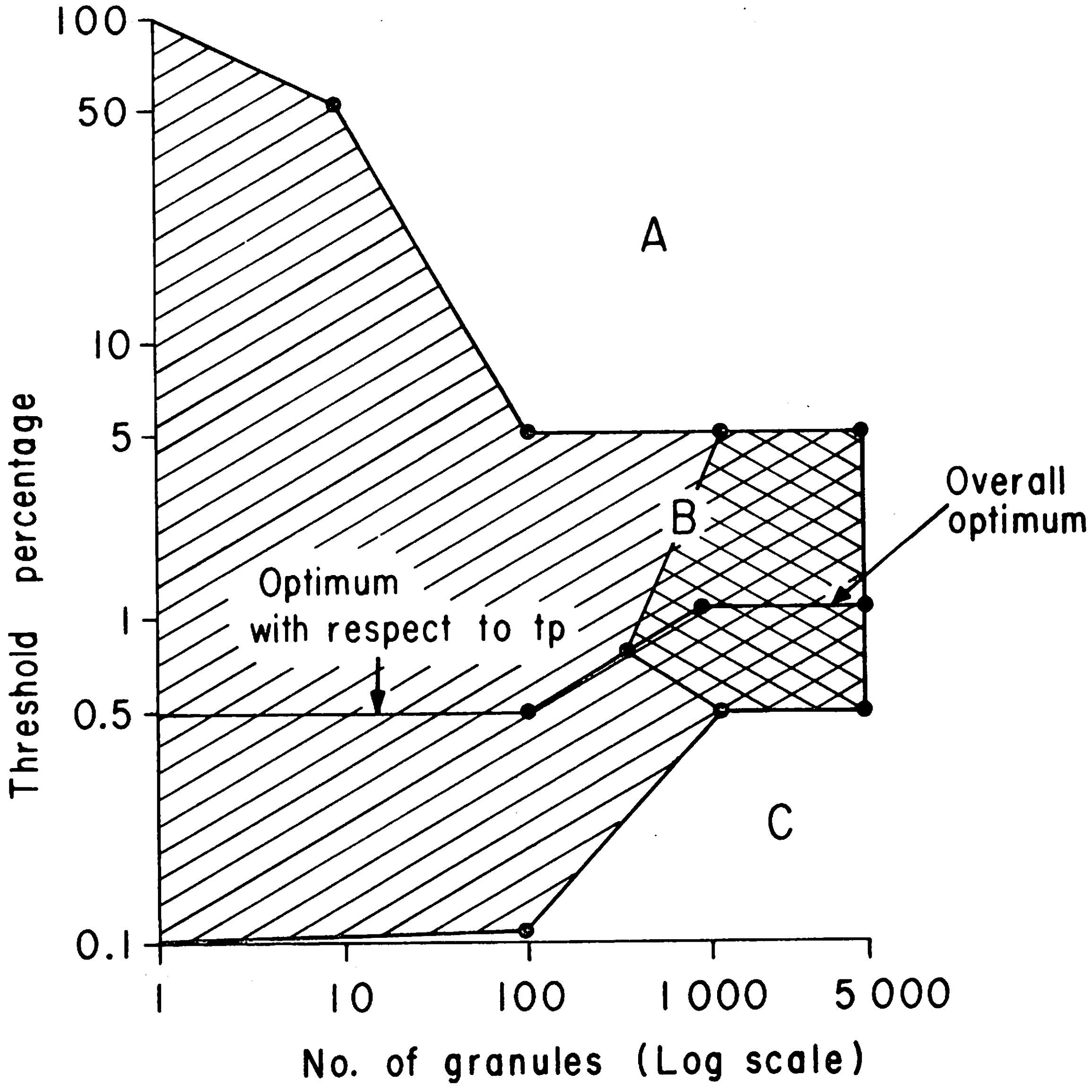


Figure 5: Small Transactions



Area A: tp too high
Area B: Highest Machine Utilization
Area C: tp too low

Figure 6: Well-Placed Locks in a Lock Hierarchy



Area A: tp too high
Area B: Highest Machine Utilization
Area C: tp too low

Figure 7: Ramdom Lock Access in a Lock Hierarchy

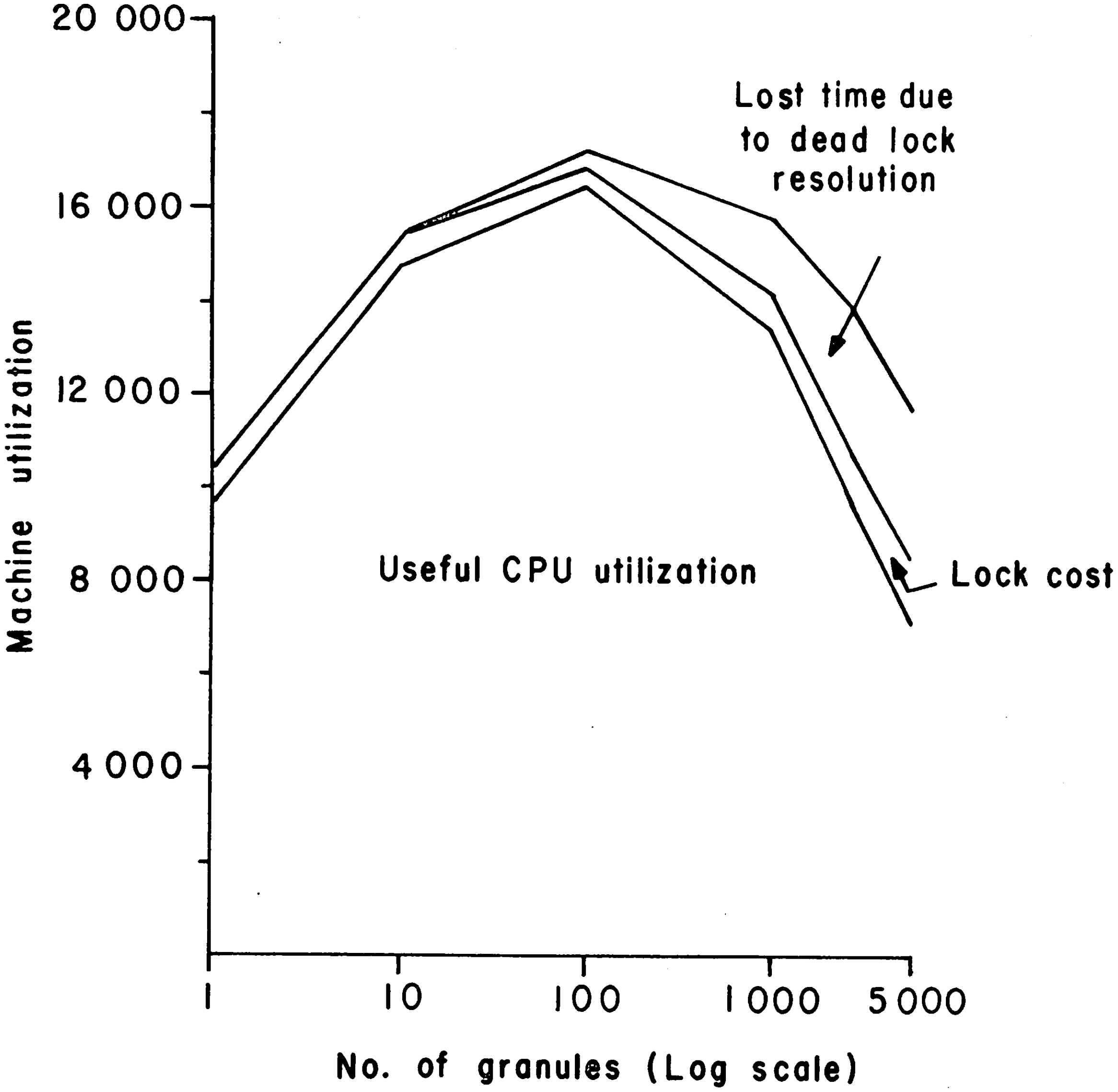


Figure 8: "Claim as Needed" Locking

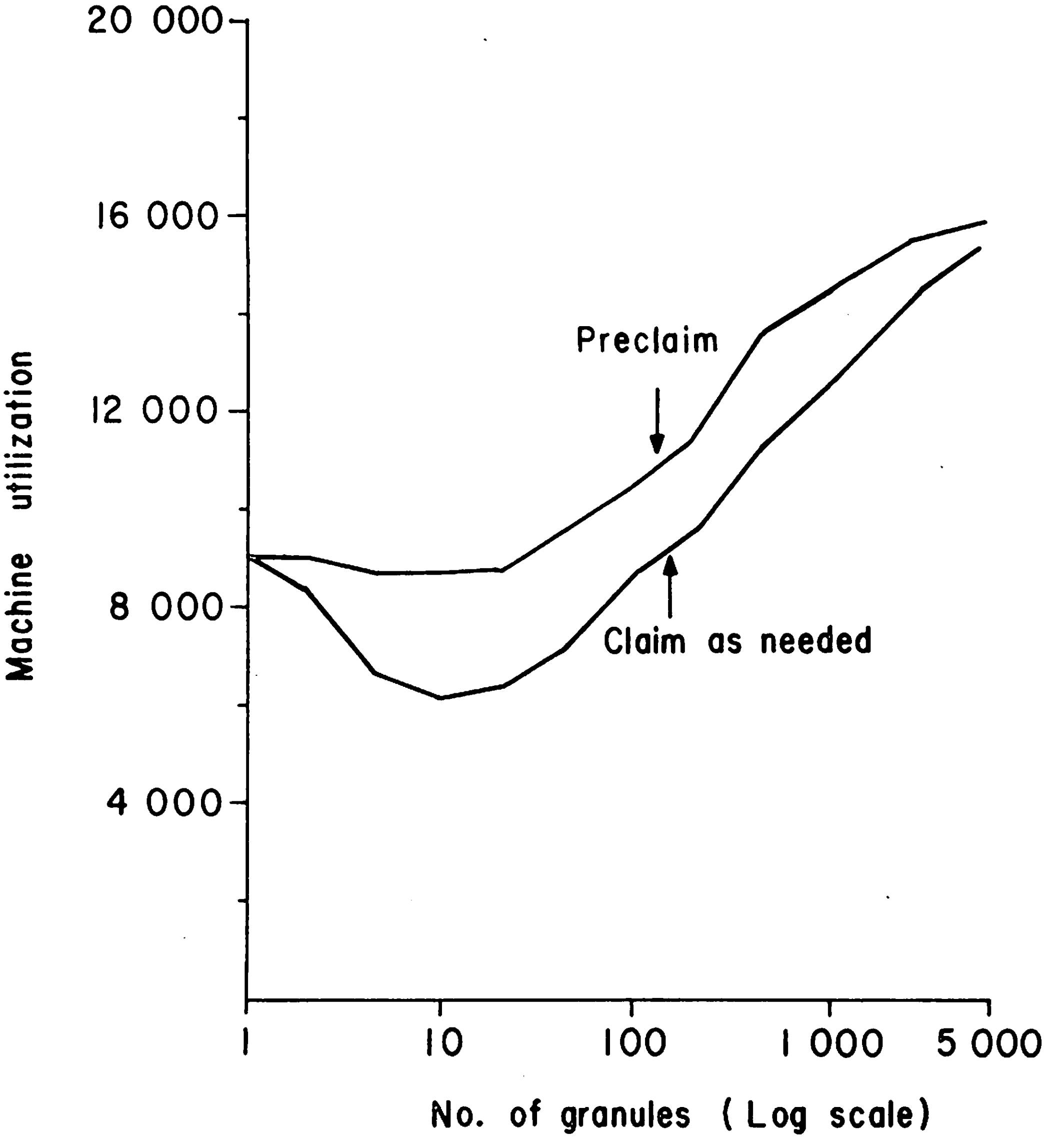


Figure 9: "Preclaim" versus "Claim as Needed"