A STUDY OF THE EFFECTS OF LOCKING GRANULARITY IN A DATA BASE MANAGEMENT SYSTEM

by

D. R. Ries and M. Stonebraker

Memorandum No. UCB/ERL M77/34

2 May 1977

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A STUDY OF THE EFFECTS OF LOCKING GRANULARITY IN A DATA BASE MANAGEMENT SYSTEM

by

DANIEL R. RIES

and

MICHAEL STONEBRAKER

Memorandum No. UCB/ERL-M77/34

2 May 1977

Electronics Research Laboratory and
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, California

ABSTRACT

Many data base systems guarantee some form of integrity control upon multiple concurrent updates by some form of locking. Some "granule" of the data base is chosen as the unit which is individually locked, and a lock management algorithm is used to ensure integrity. By a simulation model this paper explores the desired size of a "granule". Under a wide variety of seemingly realistic conditions, surprisingly coarse granularity is called for. The paper concludes with some implications of these results concerning the viability of so called "predicate locking".

Research sponsored by the Naval Electronic Systems Command Contract N00039-76-c-0022, the National Science Foundation Grant DCR75-03839 and the Army Research Office Grant DAAG29-76-6-0245.

I. INTRODUCTION

A commonly desired feature, in a multi-user data base system, is some sort of guarantee on the integrity of the data base when subjected to multiple concurrent update activity. Suppose, for example, the following two updates are executed simultaneously:

U1: "Give a 10% raise to all programmers"

U2: "Change all programmers to systems analysts"

With no controls some persons who were initially programmers would get the raise and some would not. Moreover, in general, the result would not be repeatable if the data base was backed up and two updates were rerun.

A common approach to this integrity problem is to define a "transaction", as a data manipulation program (whose allowed complexity varies from system to system). Then, a data base system guarantees that the outcome of a collection of simultaneous transactions is equivalent to the one produced by running the transactions sequentially in some order.

For example, System-R proposes supporting the above guarantee for a transaction consisting of a PL/1 program containing SEQUEL data manipulation commands [ASTR76]. INGRES proposes the above guarantee for a transaction consisting of a single QUEL data manipulation command [STON76].

Other systems propose supporting lesser sorts of guarantees. For example, System-R proposes several weaker levels of integrity control [ASTR76, GRAY76]. Moreover, both IMS [DATE75] and the

CODASYL proposal [CODA71, CODA73] suggest facilities which can only guarantee weaker conditions.

Regardless of what consistency conditions are supported, a concurrency mechanism must be present as an enforcement agent. Two general options appear feasible.

1) Physical locks on records, pages, segments, files, etc.

In this case, the data base system supports the notion of locking a "granule" of the data base. This "granule" is a record (CODASYL [CODA71], System-R [ASTR76], DMS-1100 [GRAY75]), a page (IMS [GRAY75]), a column of a relation (INGRES [STON76]), all records of a given type (LSL [LIP76]), a whole data base (SYSTEM 2000 [SPIT76]), etc. In addition, System-R proposes a granule whose size can be dynamically varied.

In all cases, a data manipulation command cannot proceed if a granule it needs is locked by someone else. Various strategies for requesting and releasing locks have been suggested [CHAM74, GRAY76, STEA76, MACR76], some with the necessity of detecting (and resolving) deadlock [COFF71].

Some systems (eg. CODASYL) leave the locking strategy effectively in the hands of the application programmer.

2) Predicate locks

Here a "logical lock" can be set on the exact portion of the data base which is required. The portion of the data base which is locked is determined by a predicate or qualification. The

predicate (eg. "all records with date field values in June, 1976") restricts the transaction to a logical subset of the data base. Such locks do not necessarily correspond to any "granule" of the physical data base. This approach is explored in [FLOR74, STON74, ESWA76].

In this paper we examine two questions:

1) If physical locking is used, what size should the "granule" be?

Fine granularity allows a higher degree of parallelism at greater cost in managing locks. For example, assume that a granule corresponds to a record in a data base. Then the transactions may run in parallel without conflict as long as they access distinct records. However, the data base system must be prepared to handle a lock table with the same number of entries as records in the data base.

Coarse granularity, on the other hand, inhibits parallelism but minimizing management of locks. If the granule is considered the entire data base, no transactions will run in parallel. The data base system will, in this case, only have to keep track of one lock.

In [SPIT76] the effect of scheduling on granule selection is examined for System 2000 by simulation. Our approach is also by simulation, but we examine a much larger class of alternatives. Also, our model is highly parameterized so it can hopefully yield insight into "granule" selection in a wide class of data base

systems.

2) How viable is predicate locking?

Our model, although it does not directly simulate predicate locking, is used to make predictions concerning the sort of predicate locking schemes that may be feasible.

The remainder of the paper is organized as follows. In the next section we indicate the model of a data base system which we assume and the model's inputs and outputs. Then in Section III we indicate and discuss the results of several test runs. Lastly, in Section IV we draw conclusions concerning the two above mentioned questions.

II. DESCRIPTION OF THE SIMULATION

The complete model simulates requests against a data base. A fixed collection of transactions are assumed to cycle continuously around Figure 1.

Initially, the transactions arrive one time unit apart and are put on the pending queue. A transaction then goes through the following stages.

a) The transaction is removed from the PENDING queue and required locks are requested. If the locks are granted, the transaction is placed on the bottom of the I/O queue. If the locks are denied, the transaction is placed on the bottom of a BLOCKED queue. The blocking transaction is recorded.

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

c) After completing the CPU required, the transaction releases its locks and joins the end of the PENDING queue. All transactions blocked by the completed transaction are placed on the front of the PENDING queue.

Note that all needed locks are requested initially. Hence, deadlock is impossible and there is no need to simulate it. Moreover, each transaction goes through one I/O phase and one CPU phase. Although they are sequential in the model, the result would be the same if each transaction went through many I/O - CPU phases in a single cycle. The costs to request and set locks includes the cost of releasing those locks. That cost is assumed to be the same even if the locks are denied. Lastly, the concurrency mechanism has preemptive power over the running transactions for the I/O and CPU resources. Thus when locking requests are being analysed, parallel running of transactions is not allowed. We comment in Section IV on the results to be expected if transactions request locks when actually required.

The following input parameters are required to drive the simulation.

dbsize- The number of accessible entities in the data base. An accessible entity is the unit moved by the operating system into data base system

buffers. Commonly this is a disk sector.

lgran- The number of granules into which the data base is divided. Each granule is assumed to be the same size. Hence, if lgran=1, a granule is the entire data base of dbsize entities. If lgran=2, a granule is dbsize/2 entities. If lgran=dbsize, each granule is 1 entity. A granule is the unit which is locked by a transaction.

ntran- The number of transactions in the system.

rad- Used to generate the distributions of the number of entities required by a transaction (see below).

ioqmax- Maximum number of transactions allowed on the I/O queue.

cpurate- CPU time required by a transaction to process one entity of the data base.

iorate- I/O time required by a transaction to access one entity of the data base.

lcpur- CPU time required to request (and set) a lock for one granule (CPU lock overhead).

lior- I/O time required to request and set a lock for one granule (I/O lock overhead, if any).

ioovlp- The I/O overlap possible. In one time interval

ioovlp units of time are equally distributed among the transactions on the I/O queue. This parameter is a surrogate for the number of independent paths used between main memory and secondary storage (and hence for how much I/O activity can go on in parallel).

cpovlp- CPU overlap allowed. In one time interval, cpovlp units of time are equally distributed among the transactions on the CPU queue. This parameter is generally 1 but can be set lower to simulate operating system overhead.

tmax- The number of time units to run simulation.

The following parameters are recorded for each simulation run.

tcpu- The number of time units in which the CPU is busy. During the tmax-tcpu remaining time units, the CPU is idle.

tio- The total I/O units of time used in tmax units of running the simulation. During the remaining time the I/O devices are idle.

lockcpu- The CPU units of time used in requesting, setting, and releasing locks.

lockio- The I/O units of time used in requesting, setting, and releasing locks.

usefulio- tio-lockio

usefulcpu- tcpu-lockcpu

totcom- Number of transactions completed at tmax.

avres- Average response time of transactions completed. Some transactions may have started but not completed. These are not included in the computation of totcom or avres.

We now describe the resources (entities) needed by each transaction and how 'lock conflict' is computed. The ith transaction (1<=i<=ntran) is described by:

NUi =i*rad number of data base entities to be accessed by the ith transaction.

LUi =ceil(NUi*lgran/dbsize) number of granules required by the ith transaction. (Note that this amounts to an assumption that the NUi entities are placed so as to require the minimum number of granules. Alternate assumptions can only increase the number of granules required.)

IOtimei=NUi*iorate I/O time for the ith transaction.

CPtimei =NUi*cpurate CPU time for the ith transaction.

LIOtimi =LUi*lior I/O time for locking by the ith transaction.

LCPtimi =LUi*lcpur CPU time for locking by the ith transaction.

Lock conflicts are computed as follows. Assume that enough granules are unlocked for the requesting transaction to potentially proceed. Let T1, T2, ..., Tk denote the k active transactions. Suppose each transaction Ti has Li granules locked. Divide the interval (0,1] into k+1 partitions.

P1 =(0 ,L1/lgran]

P2=(L1/lgran,(L1+L2)/lgran]

.
.
.

Pi=(sum(Lj,j<i)/lgran,sum(Lj,j<=i)/lgran]

for i=1,...,k.

and Pk+1 = (sum(Lj,j<=k)/lgran,1]

To determine if a given transaction must be blocked, choose a random number p, uniformly distributed on (0,1). If p is in Pj for some j<=k, then the transaction is blocked by Tj. Otherwise, the transaction may proceed. This amounts to the assumption that the first granule required by each transaction is uncorrelated with any of the granules already locked. If there are more granules to be accessed, they are assumed to be distinct from those already locked.

III. RESULTS AND DISCUSSION

The simulation was initially run with the parameters shown in Table 1 .

dbsize	ntran	rad	ioqmax	ioovlp	tmax
5000	10	50	10	1.000	10000

cpurate	iorate	lcpur	lior	cpovlp
0.050	.200	0.010	.200	1.000

TABLE 1: Input parameters

In this scenario, ten transactions were submitted to a data base of 5000 entities. The transactions required from 50 to 500 entities each (initially uniformly distributed). The I/O and CPU overlap parameters were set to one which results in only one transaction doing either at one time. Note that for this run the I/O rate is four times the CPU so that this simulates an "I/O bound" application. The CPU cost of a lock is 1/5 that required to process an entity. Lastly, the I/O cost of a lock is equal to the I/O cost of an entity. Hence, this initial run simulates a lock table kept on secondary storage.

Intuitively, these input parameters could be interpreted as followes:

dbsize is 5 million bytes (one entity is 1024 bytes)
Average transaction size is 250,000 bytes.
iorate of 100 msecs per entity (two disk accesses).
cpurate of 25 msec per entity.
lior of 100 msecs per lock.
lcpur of 5 msecs per lock.

The 10 transactions could correspond to 10 terminal users or application programs issuing commands against the data base. The simulation was also run with 20 transactions with no appreciable effect on the output parameters.

For these simulation runs, the value tmax=10000 was chosen after running all simulations for various smaller values including tmax =2500. In all cases, no change (except for scaling) was observed in the output parameters between tmax=2500 and tmax=10000. For some of the experiments discussed later, other values of tmax were required to guarantee convergence. Keeping these parameters fixed, the number of granules allowed was varied between 1 and 5000. The output from the simulations is presented in Tables 2 and 3.

no_of_granules	usefulio	usefulcpu	lockio	lockcpu
1	7041.957	1759.906	1282.000	12.820
2	8376.933	2091.914	970.000	9.700
3	9002.256	2237.415	777.000	7.770
4	9030.253	2258.925	671.000	6.710
5	9273.915	2304.927	604.000	6.040
7	9438.514	2309.940	474.000	4.740
9	9449.087	2337.442	428.000	4.280
10	9476.180	2324.941	437.000	4.370
15	9425.585	2358.445	403.000	5.210
20	9437.987	2354.943	396.000	5.280
30	9534.303	2377.449	371.000	6.720
40	9572.718	2354.949	360.000	7.900
50	9504.073	2339.950	360.000	8.790
75	9448.435	2332.452	454.000	13.290
100	9378.277	2324.951	482.000	15.430
125	9351.744	2316.457	547.000	20.890
150	9304.128	2279.960	618.000	23.700
200	9159.688	2259.959	753.000	30.000
250	9110.531	2249.964	806.000	36.740
300	8768.228	2177.465	1015.000	43.470
500	8517.211	2097.466	1390.000	69.499
750	7820.611	1919.974	1950.000	94.439
1000	7359.828	1814.976	2462.000	123.099
2500	4764.175	1189.980	4824.000	241.199
5000	3408.635	824.992	6120.000	305.998

I/O and CPU Costs
TABLE 2

Note that useful I/O peaks at 40 granules. Within 1% of this value is reached with only 10 locks and stays relatively constant until the lock I/O costs start to be a significant fraction of I/O time. For a small number of granules, high lock I/O cost results from lock conflicts which generate additional lock I/O. (In an actual implementation of a locking scheme, a small number of locks could easily be maintained in primary memory. This alternative is explored subsequently.) Similarly, the useful CPU time peaks at 30 granules, and again this value is almost reached (within 1%) with as few as 10 granules. These results are

portrayed graphically in Figure 2. The lock CPU costs reach a minimum at 10 granules. With fewer than this number, the request failure rate causes enough additional requests for locks that the overall CPU costs for locking is greater even though the total cost for each request should be smaller. With more than 10 granules, the reduction in lock request failures does not offset the costs of setting the additional locks required for each transaction.

no_of_granules	avres	totcom
1	751.914	128
2	557.232	168
3	534.399	178
4	523.082	182
5	490.297	195
7	506.667	189
9	515.117	188
10	472.330	203
15	484.214	196
20	462.678	208
30	472.732	205
40	454.189	212
50	441.537	218
75	430.543	223
100	420.416	231
125	463.255	208
150	460.429	210
200	435.748	222
250	504.021	192
300	447.065	215
500	472.088	204
750	570.089	168
1000	546.023	175
2500	815.784	115
5000	1054.988	86

Transaction throughput measurements
TABLE 3

The average response time and the total number of transactions completed at time tmax reached extremums at 100 granules. With this number of granules, the smaller transactions requiring less resources were able to run to completion and be recycled more

often. Thus a 'shortest job first' property was observed. Moreover, with finer granularity (>200 granules) the locking overhead increased the average response time.

For this particular situation, response time, throughput, useful CPU time and useful I/O time are all maximized with a small number of granules. Hence, a large number of granules (such as would be required to lock disk sectors or individual records) may be inappropriate.

To examine the effects of varying the input parameters on these initial observations, eight experiments were conducted. In each experiment, one or two parameters, in addition to the granularity, was varied.

EXPERIMENT 1: The effects of the ratio of the required I/O time to the required CPU time per entity was investigated. The CPU rate (cpurate) per entity for a transaction was held fixed at .05 units/entity. The simulation was run with I/O rates (iorate) per entity set at .01, .05, .1, .2, .3. For each setting of the I/O rate, the number of granules (lgran) varied from 1 to 5000. The lock I/O rate per granule was set equal to the I/O rate per entity in order to reflect the locks being on the same speed device as the data. Each simulation ran for 5000 time units. The other input parameters had the values indicated in Table 1.

The useful I/O curves for each setting of iorate was bell shaped and heavily skewed towards a small number of granules. As such they are similar to the curves is figure 2 and are not repeated

here. The maximum peak of these curves occurs with fewer number of granules as the iorate increases. Thus, as transactions become more I/O bound, the advantage of additional transactions running concurrently is outweighed by the additional locking overhead. Even with CPU bound transactions, however, within 5% of the peaks was reached with as few as 10 granules. Varying the iorate had little effect on the throughput measurements (average response time, and number of transactions completed) as a function of the number of granules allowed. The useful CPU time, as a function of granule size, showed a similar distribution as the useful I/O. The costs associated with locking were again minimized with less than 100 granules. For all values of iorate, within 15% of this minimum was reached with 10 granules.

EXPERIMENT 2: In the preceding discussion, the lock I/O rate (lior) was set equal to the transaction I/O rate (iorate). In the next series of simulation runs, only the lock I/O rate and the granularity were varied. The simulation was run with other parameters as in Table 1. The useful I/O times (usefulio) are shown in figure 3.

AS the lock I/O rate decreased, a larger number of granules could be afforded before the advantages of more parallelism were out weighed by the locking overhead. Of particular interest is the situation where the lior was set to zero. This case was analogous to keeping all locks in main memory. Even with no lock I/O costs, there was a very flat extremum for usefulio between 10 and 200 granules. Having a granule correspond to fewer than 25 data

base entities (number of granules > 200) resulted in noticeably poorer performance. If the interpretation of an entity is a 512 byte page (or a 4096 byte sector) a data base management system should not 'protect' less than 13,000 (or 100,000) bytes of data with one lock.

EXPERIMENT 3: The CPU costs for setting one lock are dependent on the lock management algorithms. To investigate the effects of varying the CPU rate for locking on the desired granularity, the simulation was run with CPU lock (lcpur) costs per lock of .005, .01, .025, .05, .075, and .1. For this series lior and iorate were set to .2 and the cpurate was set to .05. Other parameters are as in Table 1.

For a small number of granules, the CPU lock costs were approximately linearly proportional to the CPU rate per lock. For a large number of granules, the CPU lock costs interfered with transaction processing, and CPU lock costs increased slightly less than linearly with lcpur. For all CPU lock costs tested, the minimum cost occurred at 10 granules.

The maximum amount of useful CPU time occurred with a data base of 20 granules regardless of the lock CPU rate. In all cases the graphs resembled the usefulcpu curve in figure 2 and are not repeated here. The lock CPU rate had little effect until the number of granules became large. Thus a fair amount of CPU time can be expended to manage a small number of locks.

EXPERIMENT 4: The number of entities required by a transactions

is determined by the rad parameter. For a rad of 25, the 10 transactions will vary from requiring 25 to 250 entities with an average number of 125 entities. The simulation was run with rad values of 1, 25, 50, 100, 250 and 500 on a data base containing 5000 granules. The first case results in an initial average transaction size of 1/1000 th of the data base. The last case on the other hand, results in an average transaction size requiring one half of the data base.

As the needs of the transactions increased, maximum machine utilization and throughput were obtained with fewer and fewer granules. Minimum response time behaved similarly. The optimum 1% and 5% intervals of Useful I/O are presented in figure 4. Note that even for very small transactions, 95% of the optimum was reached with as few as 10 granules.

EXPERIMENT 5: For the next series of simulation runs, the size of the data base was increased to 50000 entities. The transaction sizes ranged from using 50 to 500 entities and the simulation was run for 15000 time units. The 1% and 5% intervals were very similar to the results depicted in figure 4 for an average transaction size of .1% of the data base.

Of particular interest, the optimal number of locks did not appear to be linearly proportional to the size of the data base. The optimum number of granules did increase with the size of the data base which is consistent with the outcome of experiment 4. With both 5,000 and 50,000 entities, however, a small number of locks (<10) produced results within 5% of optimum.

EXPERIMENT 6: All of the previous runs had ioovlp values of one. These experiments thus simulated a system with one I/O path between main memory and secondary storage. In the next series of runs, this parameter was set to three and six to simulate, for example, a data base environment with three and six disk drives respectively. Other input parameters are the same as in Table 1.

Except for greatly increased magnitude, the output parameters had a similar distribution as those in Table 2. The useful I/O time (usefulio) versus the granularity for these simulation runs is shown in figure 5. Note, with 10 to 100 granules, the useful I/O increased by a factor of about 2.5 for three I/O paths as compared to one I/O path. (The best results possible would be increased useful I/O by a factor of 3.) Moreover, as the number of granules increased three drives became less and less effective. For 2500 granules, for example, only a 1.5 factor increase in useful I/O was realized. The results for six I/O paths were similar. Ten to one hundred granules tripled the increase in useful I/O. With 2500 granules, the increase in useful I/O was slightly less than doubled.

EXPERIMENT 7: For some data base systems all of the entities used need not be locked for the duration of the transaction. To reflect this property, the simulation was modified so that a transaction only held a given granule for one half of the time the transaction was active.

The results were again very similar to those in Table 2.

However, even fewer granules were required to achieve maximum machine utilization. The lock CPU costs were increased slightly for a small number of granules. For a larger number of granules, however, the lock CPU costs were decreased by 'lock releasing'. Thus the probability of lock conflict was reduced and the corresponding cost of multiple retrys for one transaction to run was also reduced.

EXPERIMENT 8: For still other data base systems, locks can be held while a user or application program pauses for some duration (often thought of as "head scratching"). The simulation was modified to reflect this effect by holding all locks for an idle period of 100 time units (say, for example, about 20 seconds in the scenario mentioned at the beginning of this section).

The results were also remarkably similar to Table 2. The useful I/O curve had slightly more variation than the curve if figure 2 with a peak occuring at 50 granules. Ten granules still produced useful I/O and CPU times within 5% of the optimum. Hence a small number of granules is still called for even with substantial pauses in the transaction processing.

IV SUMMARY

Under the assumptions mentioned in the description of our model, it appears that a small number of granules is sufficient to allow enough parallelism for efficient machine utilization Furthermore, a large number of granules, corresponding to locking a page or record is extremely costly. Any advantages due to additional

parallelism are outweighed by this cost.

This conclusion is strengthened by several assumptions in the model that were designed to favor a large number of granules. For example, locks are "well placed" in that the minimum number of granules, that could possibly lock the number of required entities, are requested. Moreover, the first granule requested by a transaction is uncorrelated. Finally, no cost is added for the deadlock detection and rollback that is present in several systems with fine granularity. Changing these assumptions would only make a large number of granules even more costly.

Note, that a small number of granules is optimal whether the transactions are CPU or I/O bound. This result was expected for I/O bound transactions since, with one disk, not much overlap is possible. The surprising result was that even for CPU bound environments (experiment 1), a small number of locks was desirable. The simulation runs of three and six I/O paths (experiment 6) also indicated that, where overlap is possible, it can best be achieved with a small number of granules.

One might have expected that very fine granularity might be desirable when accessing a very small part of the data base. However, the simulation results (experiments 4, 5) indicate that a large number of granules interfere with the system throughput. Another interesting simulation result is that maintaining locks in main memory (experiment 2) appears to make little difference in the desired granularity.

All results point to a small number of granules for implementing concurrency control in data base management systems. In fact, in most cases 10 granules appears sufficient. Hence, a very crude concurrency control scheme seems most desirable.

Consider now the possibility of "predicate locking". Here a logical subset of the data base is locked based on the transaction and its qualification. Three results from the simulation support the potential viability of predicate locking.

Firstly, with predicate locking only a small number of locks must be maintained. The number of locks is proportional to the number of transactions active and not to the size of the data base.

Secondly, while predicate locking may require more CPU time per granule than physical locking would require, our simulation results (experiment 3) indicate that for a small number of granules, data base applications can afford considerable CPU costs for locking.

Finally, the parameter which had the most effect on the desired number of granules was the number of entities 'touched' by the transactions. As the transaction size decreased, the desired number of granules increased. Note, in predicate locking schemes, the portion of the data base locked is determined by the transaction, and not a prespecified granularity, effectively mimicing the above variable granularity.

It is left as a future project to support or refute these speculations on predicate locking by a more detailed study.

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.

CODA71 Data Base Task Group of the CODASYL Programming Language Committee. April, 1971.

CODA73 Data Base Language Task Group of CODASYL Programming Language Committee. February, 1973.

COFF71 Coffman, E., et.al., "System Deadlocks", ACM Computing Surveys, Vol. 3, No. 2, June 1971.

DATE75 Date, C. J.;An Introduction to Data Base Systems; Addison-Wesley, Reading, Mass., 1975.

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.

FLOR74 Florentin, J. J., "Consistency Auditing of Data Bases", The Computer Journal, Vol 17, No 1,

February, 1974.

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.

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

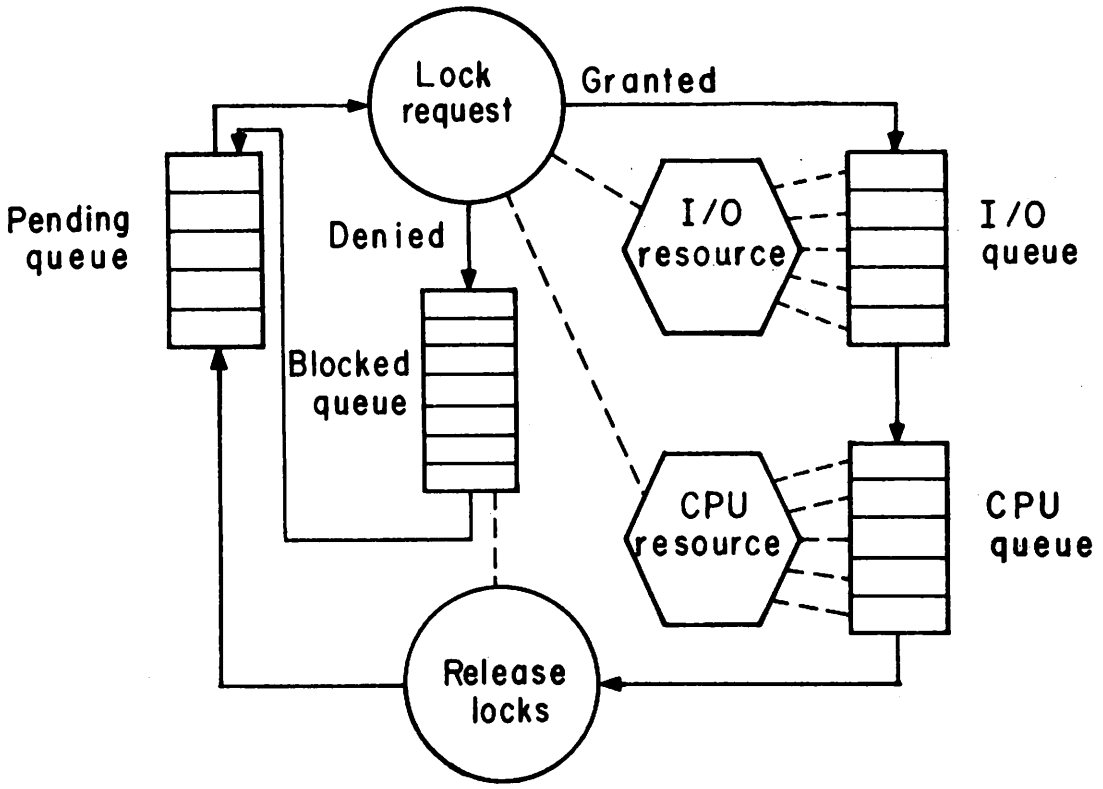
MACR76 Macri, P., "Deadlock Detection and Resolution in a CODASYL Based Data Management System," Proc. 1976 ACM-SIGMOD Conference on Management of data, Washington, D. C., June, 1976

SPIT76 Spitzer, J. F., "Performance Prototyping of Data Management Applications", Proc. ACM'76 Annual Conference, Houston, Texas, October, 1976.

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

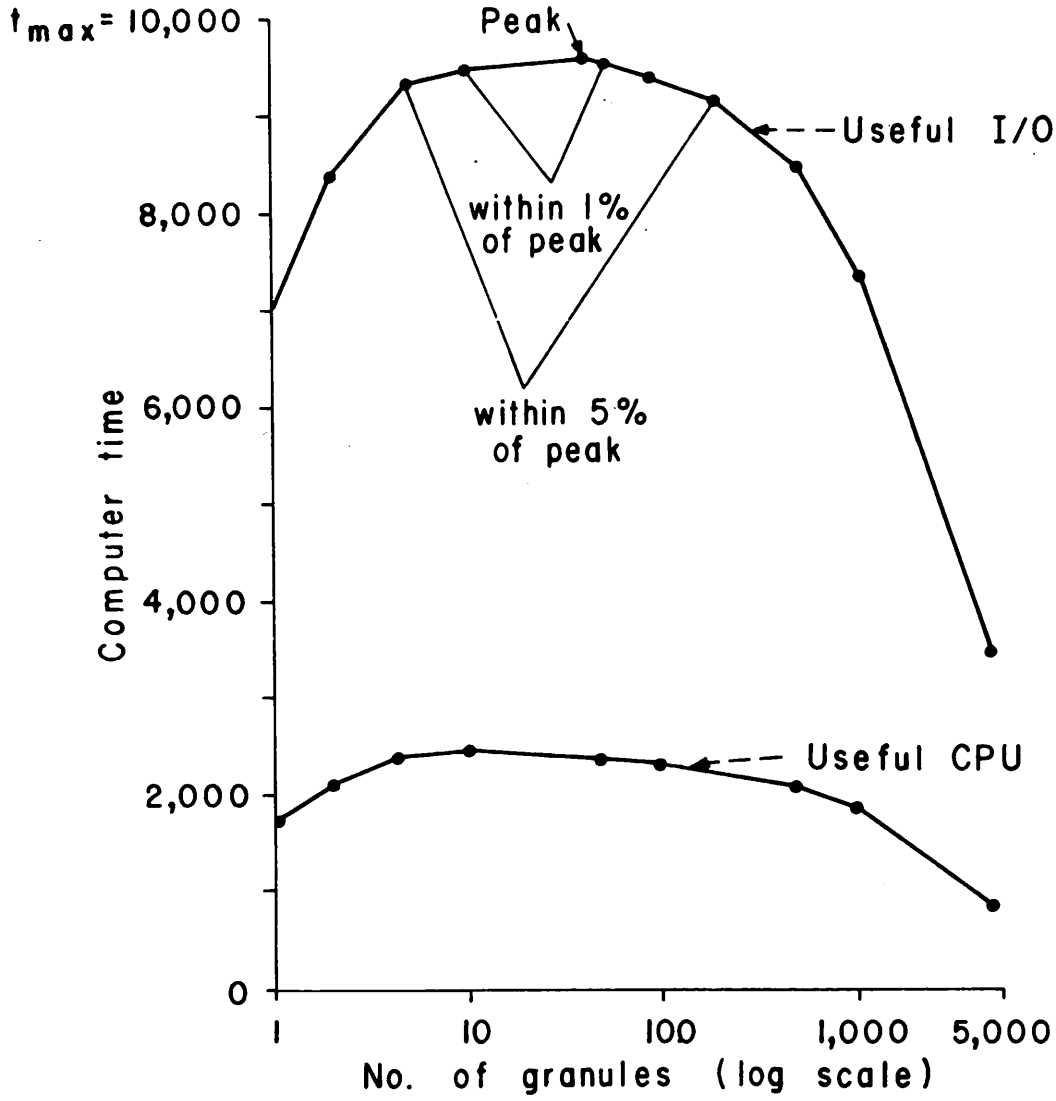
STON74 Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Systems", University of California, Electronics Research Laboratory, Memo. ERL-M473, August, 1974.

STON76 Stonebraker, M. et al "The Design and Implementation of INGRES", ACM Transactions on Data Base Systems, Vol 1, No 3, Sept. 1976.



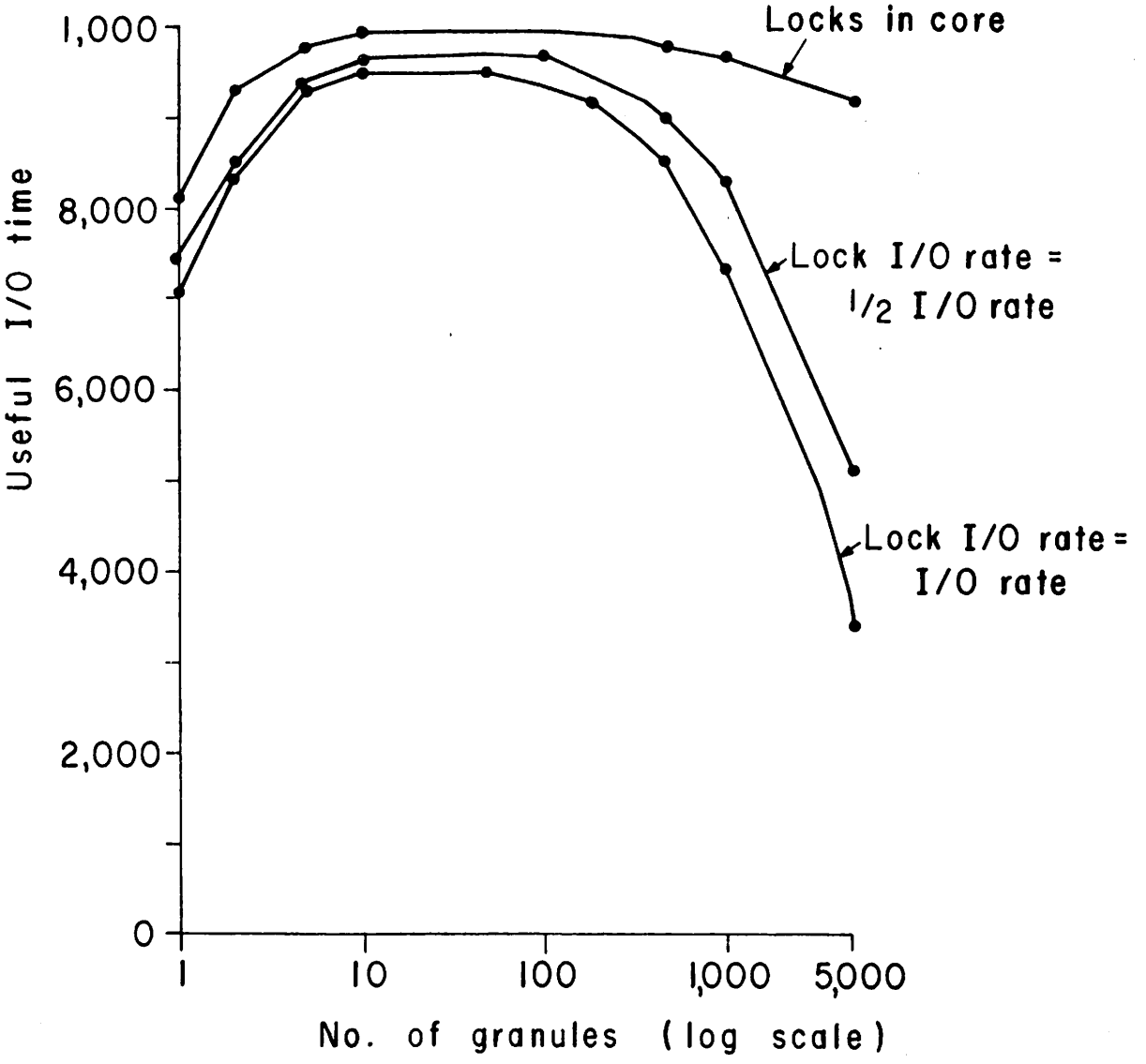
Simulation Overview

Figure 1



Computer time versus no. of granules

Figure 2



EFFECTS OF LOCK I/O RATE

Figure 3

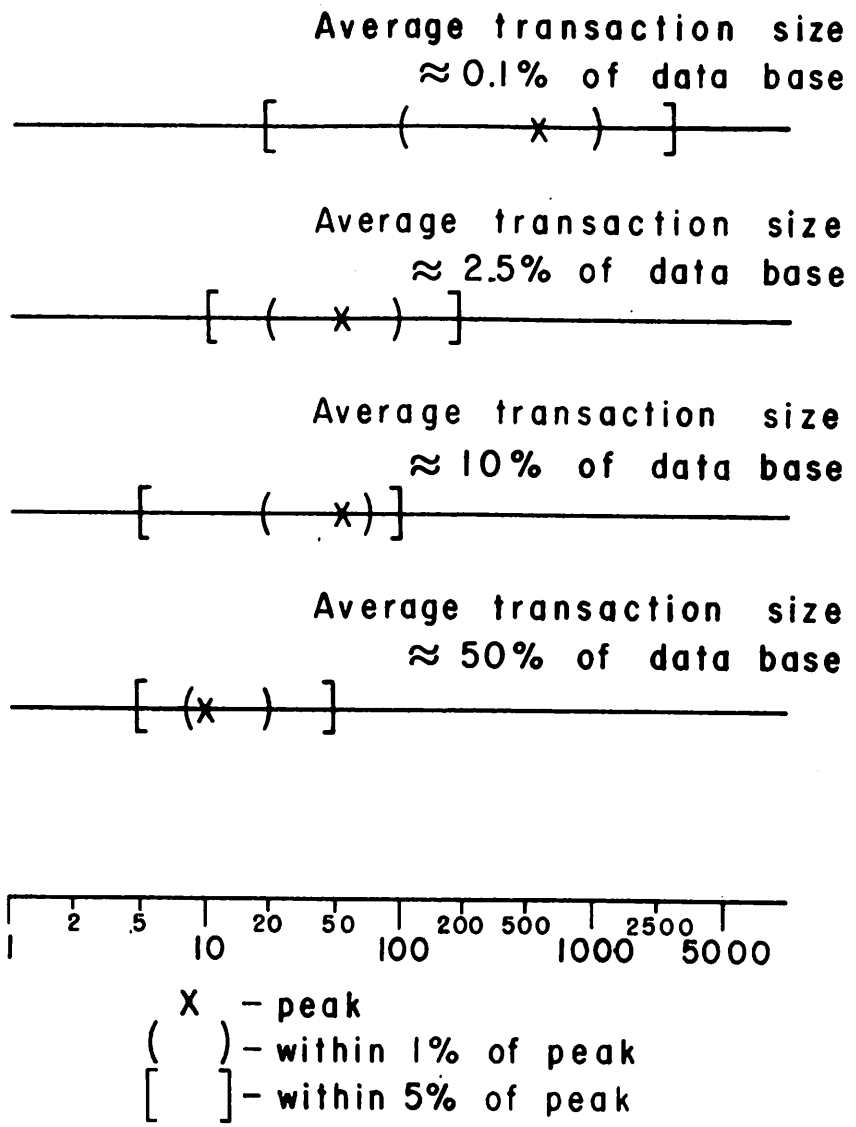
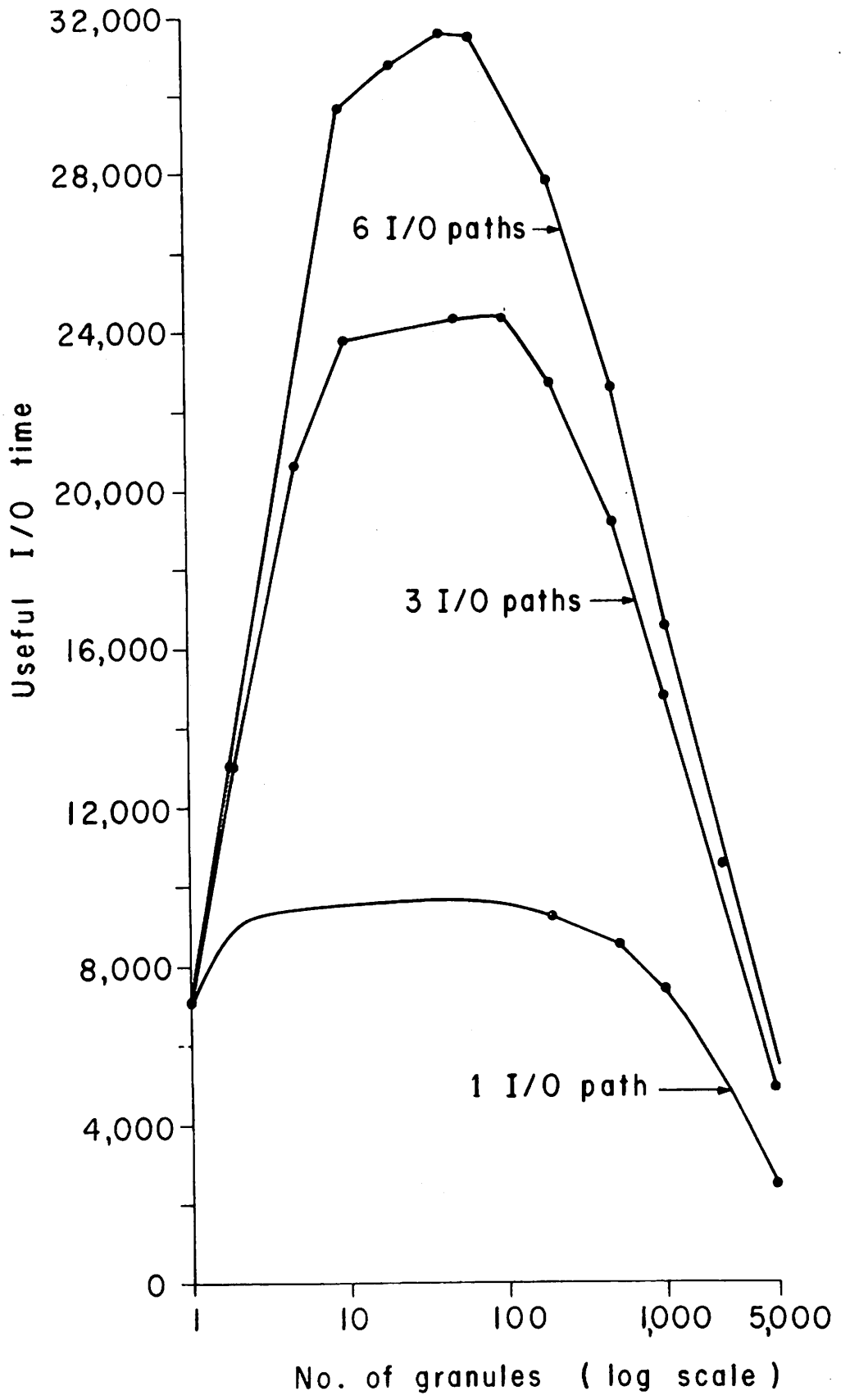

Average transaction size
≈ 0.1% of data base
Average transaction size
≈ 2.5% of data base
Average transaction size
≈ 10% of data base
Average transaction size
≈ 50% of data base
1
2
5
10
20
50
100
200
500
1000
2500
5000
X - peak
( ) - within 1% of peak
[ ] - within 5% of peak
Transaction size versus no. of granules

Figure 4



EFFECTS OF MULTIPLE I/O PATHS

Figure 5