The Impact of Hardware and Software Alternatives on the Performance of the Gamma Database Machine

by

Robert H. Gerber
David J. DeWitt

Computer Sciences Department Technical Report # 708
July 1987

The Impact of Hardware and Software Alternatives on the Performance of the Gamma Database Machine

Robert H. Gerber*
David J. DeWitt

Computer Sciences Department
University of Wisconsin

* Current address: Digital Equipment Corporation, Colorado Springs, CO.

This research was partially supported by the Defense Advanced Research Projects Agency under contract N00039-86-C-0578. by the National Science Foundation under grants DCR-8512862, MCS82-01870, and MCS81-05904, and by a Digital Equipment Corporation External Research Grant.

ABSTRACT

In this paper, we present the results of using a simulation model to explore the performance of Gamma with varying hardware and software capabilities. Measurements taken from the Gamma prototype were used as the basis for the model, providing an accurate model on which to base our experiments. Using this model, a number of software modifications were identified that had the effect of significantly increasing the performance of the system.

1. Introduction

Soon after completing the initial prototype of Gamma [DEWI86, GERB86], we decided to implement a simulation model of the system using measurements taken from the Gamma prototype as the basis for the model. There were a number of objectives in constructing this model. First, a model of Gamma would enable us to conduct a number of experiments not possible with the actual prototype. For example, the prototype configuration consists of 20 VAX 11/750s (8 of which have disk drives). Having an accurate model would make it possible to explore how Gamma would perform as the size of the hardware configuration was increased. In addition, a model would make it possible to determine what impact faster CPUs and network interfaces would have on the performance of the system.

There were also a number of software issues that we wanted to explore. While the prototype could have been used to implement and evaluate the alternative designs, having a simulation model of the system would make it much easier to explore these alternatives. In addition, the simulation model enabled us to explore different software alternatives and hardware configurations simultaneously.

While the simulator could be used to explore a wide variety of issues including interactions of queries from different users, concurrency control and recovery alternatives, and alternative scheduling algorithms, in this paper we have focused our investigation on those software components that were identified as significant (or potential) performance bottlenecks by the single user benchmark tests reported in [DEWI86]. In addition to investigating these software issues, we intend to use the Gamma simulator to investigate the impact of varying the number of disks associated with each processor in Gamma and to determine the optimum ratio of processors with and without disks.

In Section 2, the multiprocessor simulation model of Gamma is described. Section 3 presents an analysis of the performance of selection queries in Gamma. The performance of join queries is discussed in Section 4. Section 5 contains our conclusions and suggestions for future research. Since completing this study, a number of the results obtained have been incorporated in subsequent versions of the Gamma software. A complete single user evaluation of Gamma can be found in [DEWI87].

2. Multiprocessor Gamma Model

The simulation model of Gamma was constructed using the facilities of the Starpak simulation package [GERB81]. The Starpak simulation package was originally written for use with the Starmod programming language

[COOK80] and has been modified for use in conjunction with UW-Modula [FINK83]. The simulation package was influenced by the designs of the earlier simulation packages of [KNUT64] and [ARMS68].

The simulation package provides tools for maintaining a sequenced set of future events. The package also provides queuing variables called stores that enable the modeling of contention for the shared, finite resources of a system. These features in combination with the process control structures of Modula provide an environment that enables the simulation to closely model a distributed system.

The simulation model of Gamma is composed of a number of distinct modules that represent the various hardware and software components of Gamma. These modules provide a functional interface to high level abstractions of the components of a database machine, and they encapsulate the implementation and policies that control the various components. The modules are, in turn, implemented using the facilities of the simulation package. The procedural interface provided by the modules enables the implementation of a module to be changed transparently with respect to the other modules in the system. For example, the one bit stop-and-wait protocol module could be replaced by a sliding bit window protocol module that exported the same procedural interface.

The hardware components that are represented in the model are intended to be examples of current, commercially available components such as those used by the Gamma database machine. Unless otherwise stated, all modeled systems have 2*N +1 processors where N is the number of disks in the system. As disks are assumed to be associated on a one-to-one basis with processors, each system will have N processors with disks and N processors without disks. The single additional processor in excess of 2*N processors is used to support the query scheduler processes.

The processors in the current simulation are loosely coupled via a token ring network. Processes transfer packets between main memory and the network interface of a processor using the FIFO services of a DMA controller. A limited number of read buffers are assumed on each network interface. All of the critical determinants of network access and performance are parameters to the model and can be altered for each separate simulation run.

A one bit stop-and-wait communications protocol [WATS1] provides for the reliable delivery of messages between pairs of processors. As in Gamma, ports are used as queuing and addressing primitives on each processor. The CPU costs, message acknowledgment strategies, and retransmission policies are modeled after those provided by the NOSE operating system[1] and the CRYSTAL nugget [DEWI84] with the exception that the simulation model

[1]The transmission of a reliable 2 kilobyte message across the network to a remote destination requires 12.4 milliseconds. A ''short-circuited'' message of the same size can be sent to another process on the same processor in

does not piggyback acknowledgements. Messages and acknowledgements are allowed to fail and retransmissions and backoff algorithms are used to guarantee the eventual delivery of a reliable message. Messages may fail when a receiver's network interface is not able to accept a message. This may occur when all the buffers on the receiver's network interface are full or when the network device has not yet been reset following a previous network access.

An operating system similar to the NOSE kernel used by Gamma is assumed. The processes on each processor are scheduled using a nonpreemptive scheduling policy. Only interrupt handlers are able to preempt control of a processor. The performance of the processors in the system is a parameter to the model and is used to scale the amount of time that a process will hold control of a specific processor.

Write operations to a disk are always blocking. Read operations may selectively block or alternately a single page read-ahead scan can be opened on a file. Latency times representing sequential or random disk I/O are attributed depending on the access pattern for a disk. Interleaved requests from different file scan instances are attributed with random latency times. Non-interleaved disk accesses on the same file scan are attributed with latency times representing sequential disk I/O. These employed disk times[2] represent the measured performance of Gamma using Fujitsu 8 inch disk drives.

All queries in the simulation study are assumed to reference relations of the Wisconsin Database [BITT83]. Tuples in all source relations are 208 bytes long. Source relations with either ten thousand, one hundred thousand, or a million tuples are used by the simulated queries.

In the analysis presented in Sections 3 and 4, the resource usages of various hardware components are presented in the context of the execution of individual queries. These resource usages represent the absolute amount of time that a resource was busy during the execution of a query. By comparing this time to the response time of a query, the relative utilization of that resource during the query can be calculated. CPU usage includes all time spent executing user or system routines, including interrupt handlers. Average and aggregate CPU usages are presented separately for processors with and without disks. Aggregate CPU usage is calculated by summing the resource usages of all similar CPUs, i.e. those CPUs with or without disks. Disks are considered busy from the time an I/O

4.4 milliseconds.

[2]The modeled, combined seek and latency times for sequential and random disk accesses are respectively, 12 and 28 milliseconds. An additional 2 milliseconds of CPU time is required to control the I/O operation. The disk provides a transfer rate of 1.8 Mbytes/second.

operation is initiated until the operation completes.[3] Both average and aggregate disk usages are presented. Aggregate disk usage is calculated by summing the disk usages of all disks in a system. The token ring network is considered busy between the time when a token is acquired until a message has been transmitted and the token has been released.

3. Analysis of Selection Query Performance

Selection queries that produce a result relation are the simplest query processing task that utilize most of the software and hardware components of Gamma. An analysis of the simulated performance of selection queries can thus provide a basis for identifying important determinants of the performance the query processing system. As indices are an effective way of avoiding some of the potential bottlenecks in a database system, indices would only make the simulated queries run faster and would not significantly help in the identification of performance bottlenecks. Thus, only selection queries on a non-indexed relation were analyzed. Also, all selection operations referenced non-horizontally partitioned attributes [RIES78, DEWI86] ensuring that each query accesses data on all disk drives.

In the following section, we first compare the performance of the modeled system with measurements taken from Gamma[4]. Then, the simulation model is used to explore alternative query processing and scheduling algorithms that provide enhanced performance. Finally, the impact of varying the number and capability of the available hardware resources is investigated.

3.1. Validation of the Simulation Model

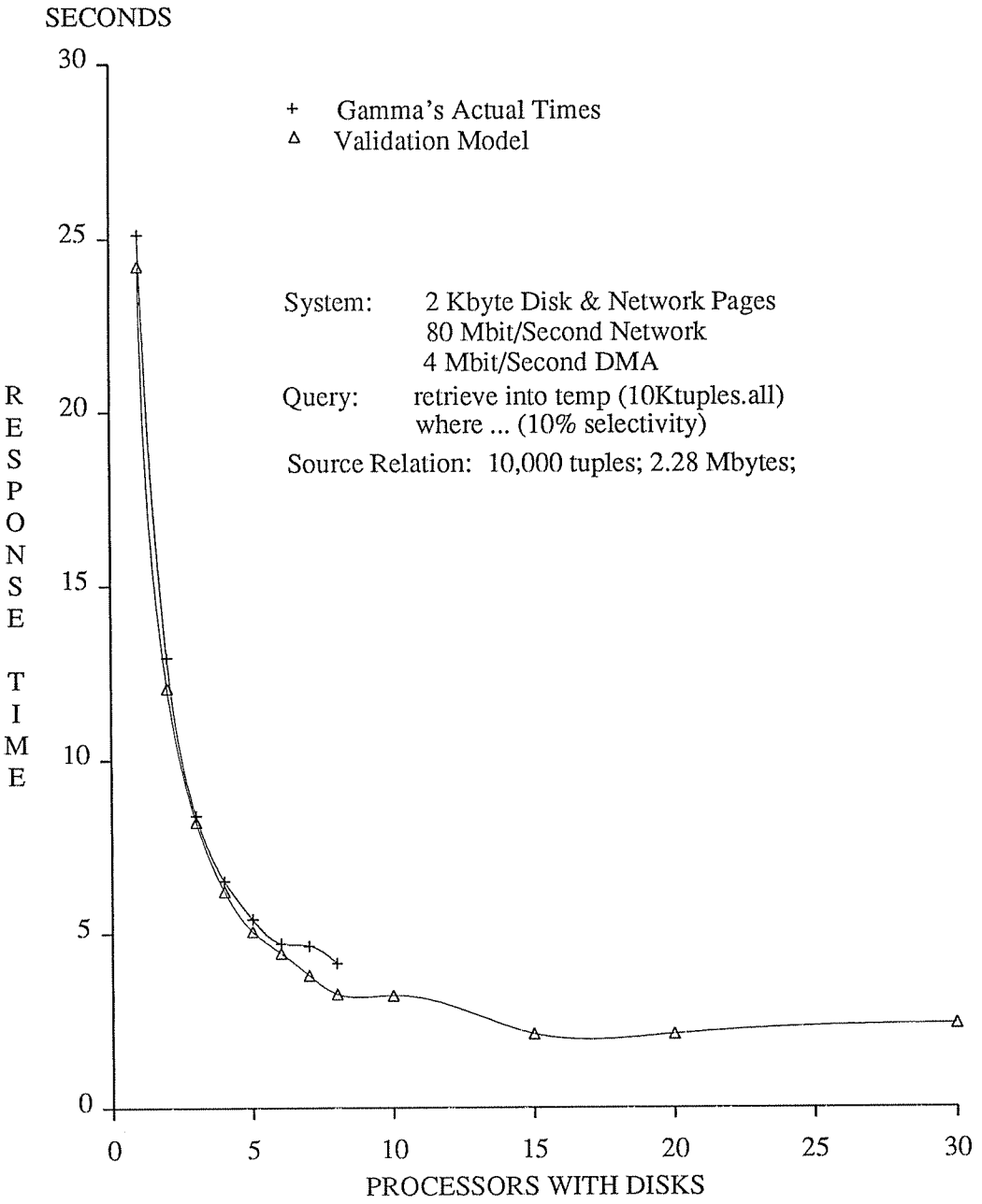
An initial task facing any simulation study is demonstrating the accuracy of the model. In the current simulation study, we are modeling discrete events, i.e. the response time of individual queries. For this reason, the accuracy of the model will be demonstrated by comparing the model with measured response times of identical queries run on Gamma.

In Figure 1, the measured performance of Gamma for a selection on a 10,000 tuple relation is compared with the performance predicted by the Gamma simulation model. This model, termed the **validation** model, has

---

[3]There is some overlap between CPU and disk usages because the CPU time required to control an I/O operation is included in both usage categories.

[4] These measurements, and all subsequent measurements, are based on the initial version of Gamma as described in [DEWI86]. The measurements reported in [DEWI87] are from the version of the system that incorporates the results obtained in this paper.



10,000 Tuple Selection Query Response Times
Figure 1

been designed to reflect as closely as possible the policies and algorithms of the initial implementation of Gamma. For systems with up to six disks, the performance of the validation model is within 93% agreement with the measured performance of Gamma. The anomalous performance of Gamma for the systems with seven or eight processors with disks is due to the use of slower, non-homogeneous disks [DEWI86]. As the validation model assumes

identical disks, the model does not reflect the anomalous behavior of Gamma for these cases.

3.2. Identifying Design Flaws in Gamma

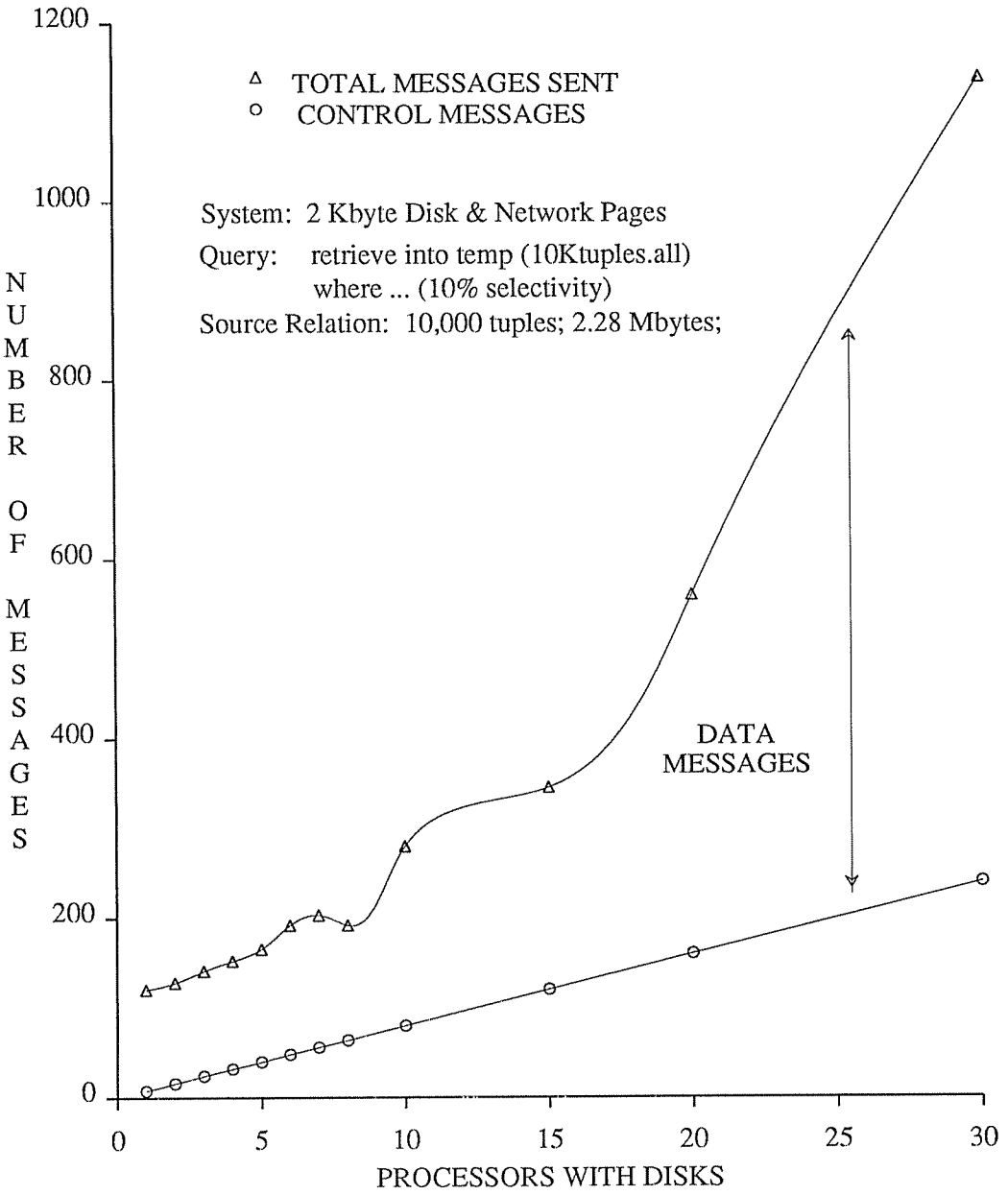
In Figure 1, the performance of the validation model is also presented for larger configurations of processors and disks than were available to Gamma. The periodic fluctuations in the performance of the validation model for these larger systems results from the policy that Gamma employs[5] when distributing tuples to a permanent result relation. Gamma distributes tuples of all result relations among output buffers on a tuple-by-tuple, round-robin basis. This can lead to a degradation in performance when, at the end of a query, all the output buffers are equally, but minimally filled. Consider, for example, a 10% selection on the 10,000 tuple relation. For a system with ten CPUs and disks, 100 tuples are produced by each participating processor. Each processor will use 10 output buffers, as the result relation is to be distributed among all of the disks in the system. Nine result tuples (208 bytes each) will fit in each of the ten buffers (2048 bytes each). After all the buffers have been filled and flushed once, exactly one tuple will be placed in each of the buffers (of which there are 100). When the selection finishes, each of these buffers must be written across the network, resulting in a minimum, total transfer of 200 data pages for the entire query.[6] Figure 2 illustrates the periodic fluctuations in the number of data pages that are sent during the 10,000 tuple selection query for systems of varying sizes.

Also, as illustrated in Figure 2, the tuple-by-tuple round-robin partitioning strategy used by Gamma has another, more serious drawback. As the size of the system is increased, Gamma transfers many more data pages across the network than necessary. As previously mentioned, this results from the fact that, at the end of a query, most of the selection output buffers will only be partially full. Nevertheless, these buffers must be flushed across the network. As the size of the configuration grows, the number of partially filled pages that are flushed at the end of a selection operation using tuple-by-tuple round-robin partitioning grows as the product of the number of producing and consuming processes associated with the output data stream of the selection. In this example, the number of flushed pages grows as the square of the number of processors with disks as a **store** operator process on each processor with a disk receives a fragment of the result relation. Furthermore, the receivers of these data streams (the

---

[5] This deficiency has been corrected in subsequent versions of the Gamma software.

[6] Actually, 1/10 of these data pages are transferred between **select** and **store** operators on the same processor. This results from the fact that tuples from one of the partitions being produced by each **select** operators are sent to a **store** operator located on the same processor. These local messages are not sent across the network, but do consume local resources as the messages are copied between local memory buffers.



10,000 Tuple Selection Query: Total Messages Sent
Figure 2

store operators) expect to receive an end-of-file message from each selection operator, so a message is required even if an output buffer is completely empty.

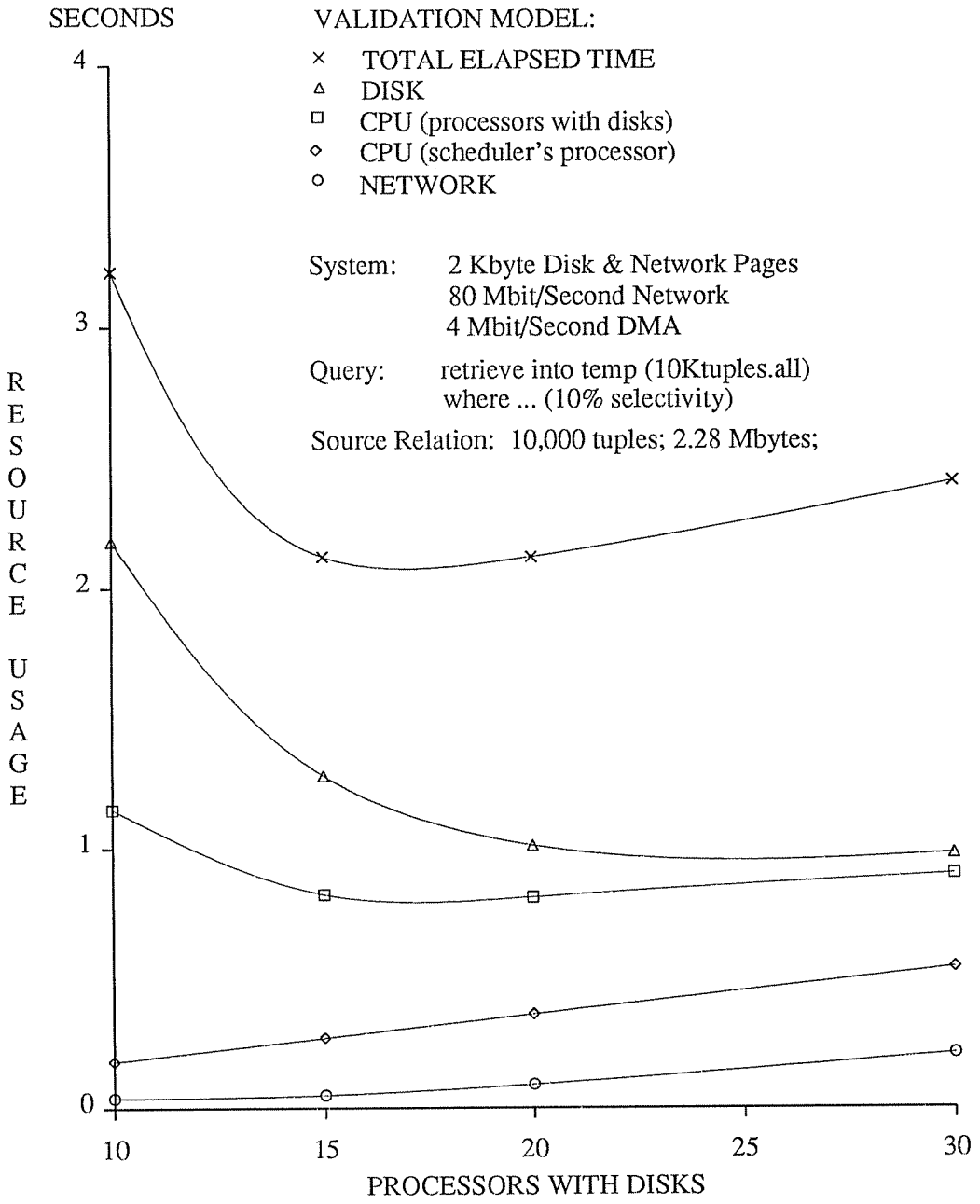
Figure 3 illustrates the impact that flushing these increasing numbers of partially filled buffers has upon the resource consumption of the various components of the system. As more processors and disks are added, the work-load of each component ideally should decrease. That is, in principle, the fixed processing requirements associated

with a particular query should be shared among all the components of the distributed system resulting in a smaller workload for each component. However, due to the processing requirements associated with the flushing of buffers at the end of this query, the CPU consumption of each processor with a disk levels off and actually begins to increase for systems with 15 or more disks.

In Figure 3, individual disk usage also does not decline proportionally for larger systems as additional disks are added to the system. This results from the fact that the flaw in the round-robin strategy used by Gamma is compounded by the way pages of result relations are written to disk. The initial version of Gamma (and the validation model) read pages of the result relation off the network directly into disk buffers. It was hoped that this strategy would avoid the extra overhead of copying tuples from network buffers to disk buffers prior to storing them on disk. However, this strategy fails when a large proportion of the pages received are only partially full. Since the tuple-by-tuple round-robin partitioning strategy creates just such a situation, the validation model stores an increasingly larger number of partially full pages of the result relation on disk as the size of the system is increased. For larger systems, this increased disk activity more than counterbalances the benefit of adding additional disks. In fact, for the validation model, when more than 25 disks are used, the addition of disk/processor pairs results in a net increase in the consumption of disk resources on every disk in the system.

Another serious complication occurs in the validation model due to the policy of reading pages directly from the network into disk buffers. This policy requires that the data page format used must satisfy the constraints of both the communications system and the secondary storage system. In Gamma, this resulted in the transmission of fixed sized data pages across the network. That is, in order to have the control information associated with a data page correctly aligned in a disk buffer, complete, maximum size data pages were always transmitted across the network. In Figure 3, the increase in the consumption of network bandwidth reflects the fact that full sized data pages are flushed at the end of a query regardless of the amount of data that is contained in the buffers.

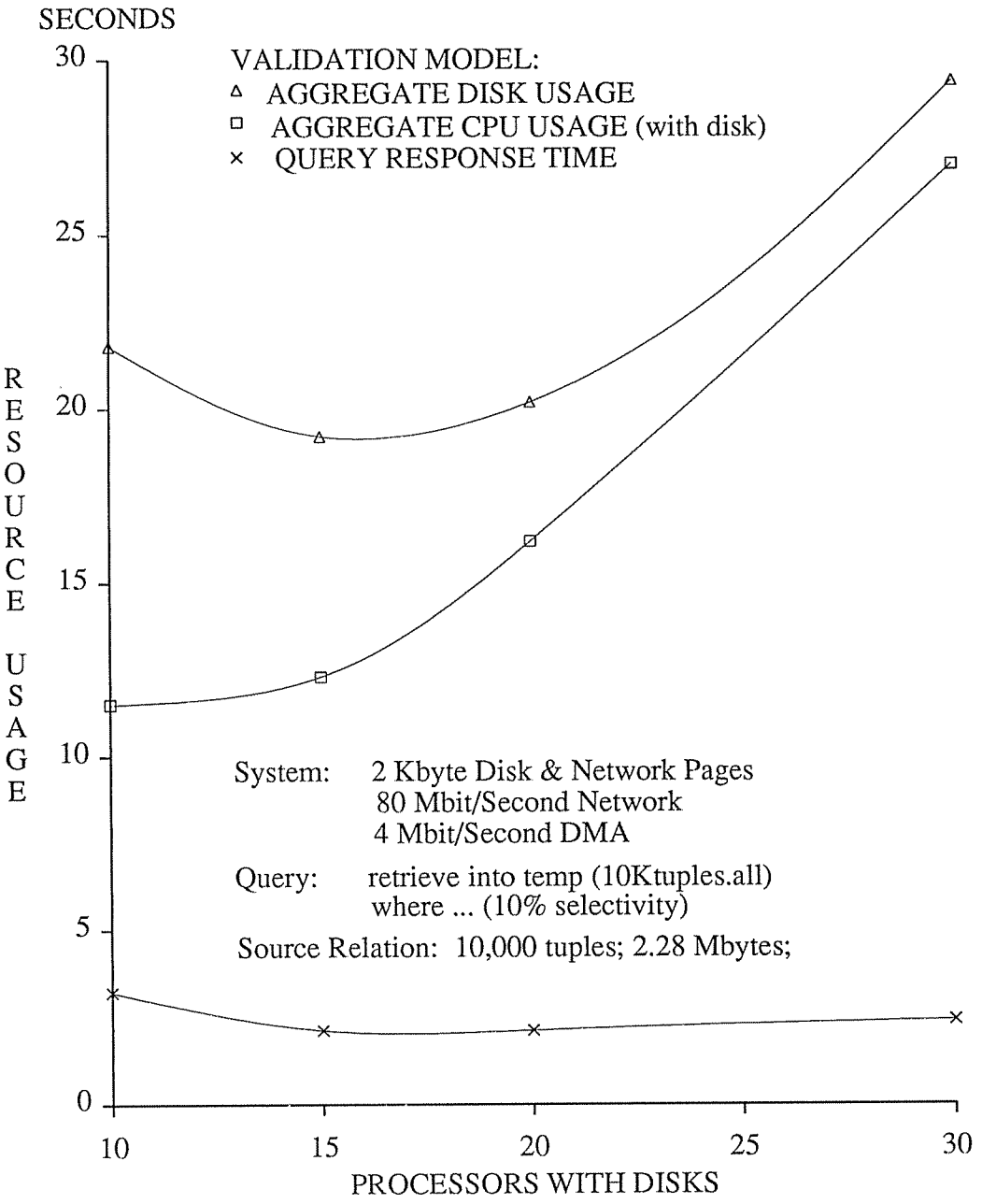
The increase in the CPU usage on the processor that supports the scheduler processes reflects the fact that schedulers in Gamma (and the validation model) communicate directly with each operator processor in the system. The query scheduler sends and receives a reliable message for each operator on each processor during both the activation and deactivation of the operator. Thus, a total of four control messages are exchanged by the scheduler and each processor with a disk in the process of scheduling each of the **selection** and **store** operations. Figure 2 illustrates the number of control messages that are exchanged in the course of the example selection query. Figure 3 illustrates the CPU usage that is associated with the processing of these messages by the scheduler process for this



10,000 Tuple Selection Query: Resource Consumption
Figure 3

query.

Figure 4 illustrates the total disk and CPU resources that are consumed by the example selection query as the size of the Gamma system is increased. The significance of the previously described design flaws is evident in Figure 4 as the total consumption of disk and CPU resources rapidly increases for larger systems. However, as Fig-



10,000 Tuple Selection Query: Aggregate Resource Consumption
Figure 4

ure 4 illustrates, Gamma and the validation model are still able to effectively use parallelism to offset the negative performance impact of the increased resource consumption. That is, the net response time of the selection query merely levels off and does not increase in proportion to the resource consumption of the query. These increased

levels of resource consumption would, however, certainly diminish the multi-user performance of the system.

3.3. Corrections to the Gamma Design

As the previous discussion indicated, a number of serious problems exist in the manner in which Gamma (and the validation model) manage data streams. The following discussion investigates solutions to these problems. The software design changes accompanying these solutions have been incorporated into a new model of Gamma that will be referred to as the experimental model.

The experimental model partitions result relations using a page-by-page round-robin strategy. That is, subsequent tuples of a result relation are assigned to the same output buffer until the buffer fills and is flushed across the network. At that point, the next buffer (associated with a different destination **store** operator) is assigned the subsequent tuples from the fragment of the result relation that is being produced by the local selection operator process. When an operation completes, the last remaining, partially full page containing tuples from the local fragment of the result relation is flushed across the network. For the selection query previously examined in Figure 1, this results in the transfer of a number data pages across the network that is linear with the number of selection operator processes employed.

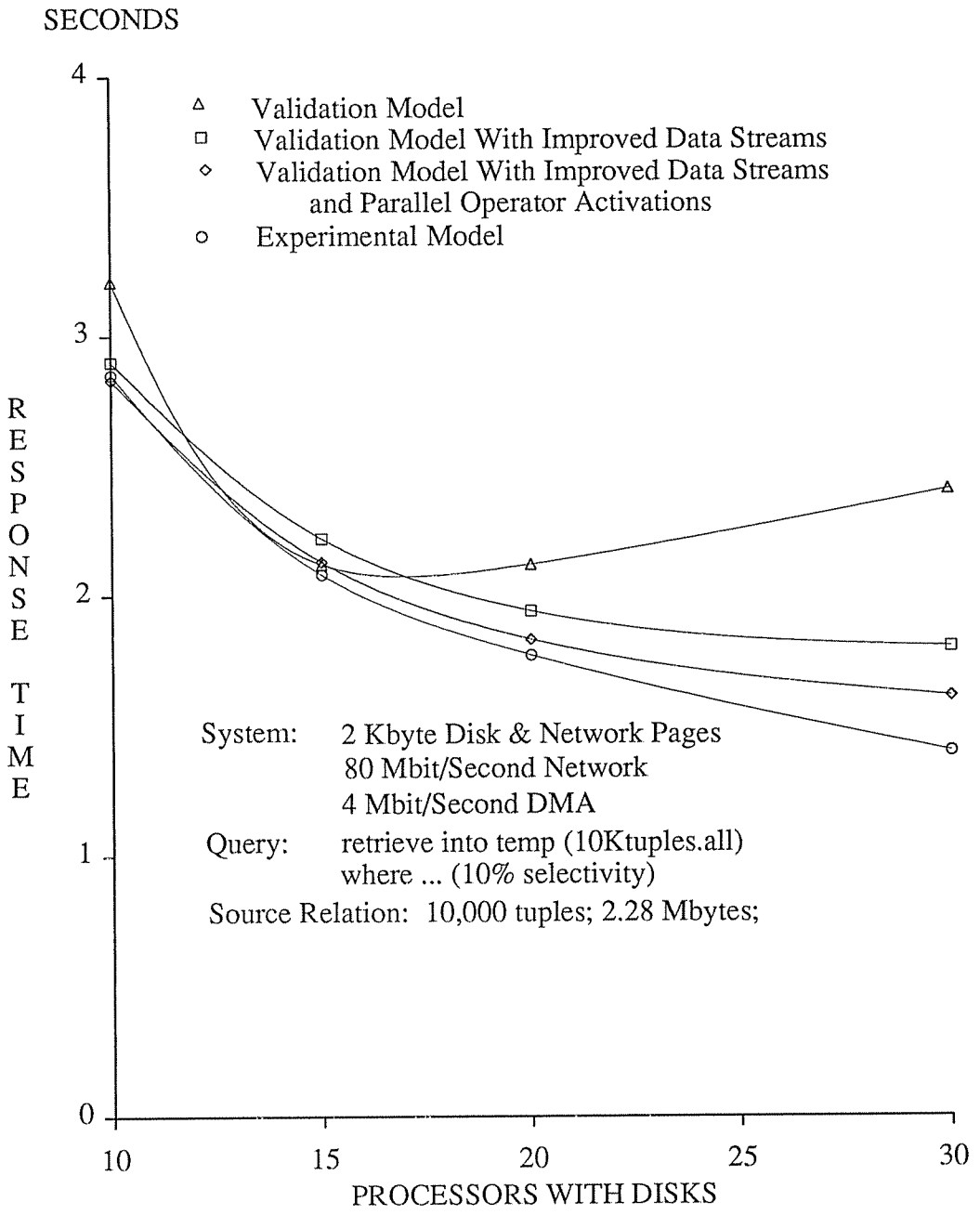
In the original Gamma design, an end-of-stream packet was transmitted across every pipeline at the end of a query, regardless if any data remained in the output buffer for the pipeline. In the experimental model, all destinations (**store** operators) do not expect to receive end-of-file messages from every selection operator. Instead, the scheduler assumes the responsibility of notifying the **store** operators when they have received all of their input data.

As previously described, in the original Gamma design, pages of a result relation were read directly from the network into disk buffers in order to avoid the costs of copying tuples between network and disk buffers. The experimental model, by contrast, reads these pages into a network input buffer. Tuples are then copied from the network buffer into a disk buffer. In this manner, the disk pages containing a result relation can be completely filled. Except for the last page of a local fragment of a relation, only completely full pages of tuples are written to disk.

Actually, with a page-by-page round-robin strategy, it would even be feasible to retain the policy of writing pages of the result relation directly from the network to the disk. However, this policy would prevent Gamma from changing the size of a disk page to be different from that of a network page. As will be demonstrated shortly, this would be a severe and costly restriction.

Figure 5 illustrates the performance impact of improving the management of data streams with respect to

the selection query previously used in Figure 1. The line labeled as incorporating improved data streams represents the performance impact of using page-by-page round-robin partitioning and collecting full disk buffers prior to writing the pages of a result relation to disk. While these improved data streams correct the periodic behavior of the validation model (caused by the relative fullness of the final pages produced by the selection operation), extra CPU



Performance Impact of Selection Query Control Improvements
Figure 5

overhead is incurred when copying tuples of the result relation from network buffers to disk buffers. This additional cost allows the validation model to perform slightly better than the improved model in systems with 15 or more disks.

As previously described, Gamma exchanges four control messages with each of the query processes that are assigned to a particular operation. Half of these messages are exchanged in a synchronous manner. Gamma (in particular, the scheduler assigned to a query) sequentially sends and receives a reliable message to each targeted processor when an operator is activated. That is, after sending an activation message, Gamma waits for a reply from the activated operator process before progressing to the activation of an operator process on the next processor. In contrast, during the deactivation of an operation, Gamma sends deactivation messages to all operator processes before waiting for the replies from any of the operator processes. Clearly, the asynchronous nature of the deactivation sequence is a more advantageous choice. In retrospect, the synchronous activation sequence of Gamma was a mistake and is not a requirement of the scheduling algorithm.

Changing the scheduling algorithm to allow activation messages to be sent to all nodes before accepting any reply messages produces performance improvement, as it shortens the time period necessary to start up query operations. This performance improvement is reflected in the third curve of Figure 5 labeled as using parallel operator activations.

Even with a more asynchronous reception of control messages, the centralized scheduler remains a potential bottleneck in the Gamma design. This results from the fact that the current design requires the scheduler to interact individually with each of the processes that are assigned to an operation. The experimental model contains a modified scheduling algorithm that applies increased levels of parallelism to the control of individual queries. Instead of requiring that the scheduler interact directly with each destination processor supporting a particular query operation, the experimental model passes the responsibility for distributing control messages to the query operators themselves. A binary tree structure is logically imposed upon the query operator processes for purposes of transmitting control information.[7] The scheduler sends a control message only to the operator process at the root of the control tree. When this operator process receives the control message, it sends the message on to the operator processes

[7]This binary control tree is completely independent of the compiled query tree that represents the operations that comprise a query. The control tree merely imposes an order among the various operator processes that have been assigned to execute a single operator. That is, the control tree contains no semantic information and is only used to determine the routing of control messages.

that are its immediate descendents in the control tree. These descendent processes continue the parallel propagation of the message through the control distribution tree. The processes at the leaves of the control tree terminate the propagation of the control message and initiate the reply messages. These replies are returned up through the control tree, preserving the control information that is specific to each operator process. The routing information for the control distribution tree is determined by the scheduler and is included in the initial control packet that is sent to each query operator process.

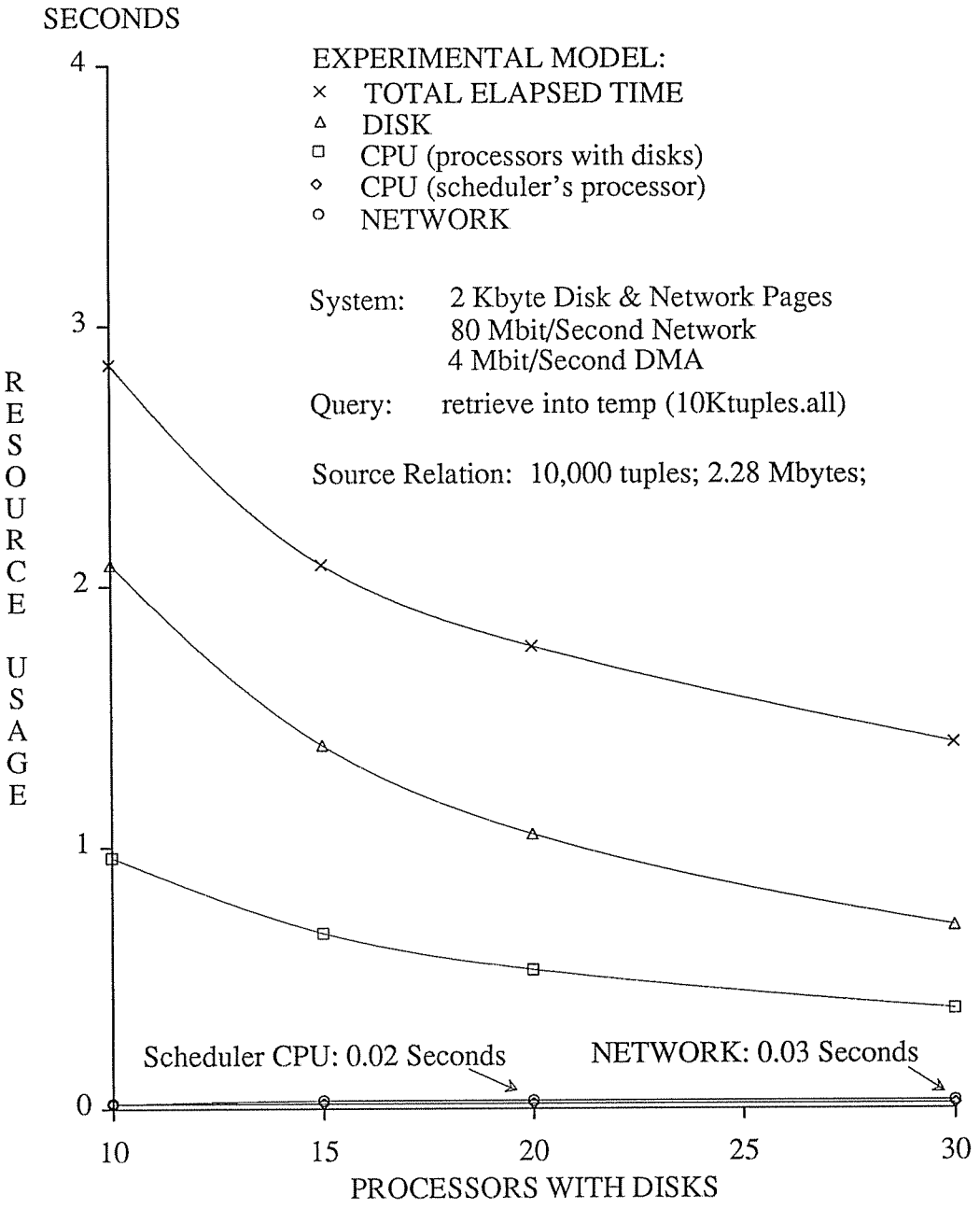
The experimental model includes all of the previously described data stream improvements plus the improved scheduling algorithm that uses binary, control distribution trees. In Figure 5, the performance of the experimental model for the example selection query is presented. The performance improvement produced by the experimental model increases with the size of the configuration because of the increased levels of parallelism that are applied to the distribution of control messages.

Figures 6 and 7 illustrate the resource usage that accompanies the experimental model for the 10,000 tuple selection query shown in Figure 5. The total consumption of disk resources increases only slightly in Figure 7 due to the fragmentation of the source and result relations across larger numbers of disks. That is, the last page of each fragment of a relation is only partially full and the number of these partially full pages increases linearly with the number of disks. While the disk remains the most heavily used resource in the experimental model, the workload associated with the use of the disks is effectively distributed as additional disks are added to the system.

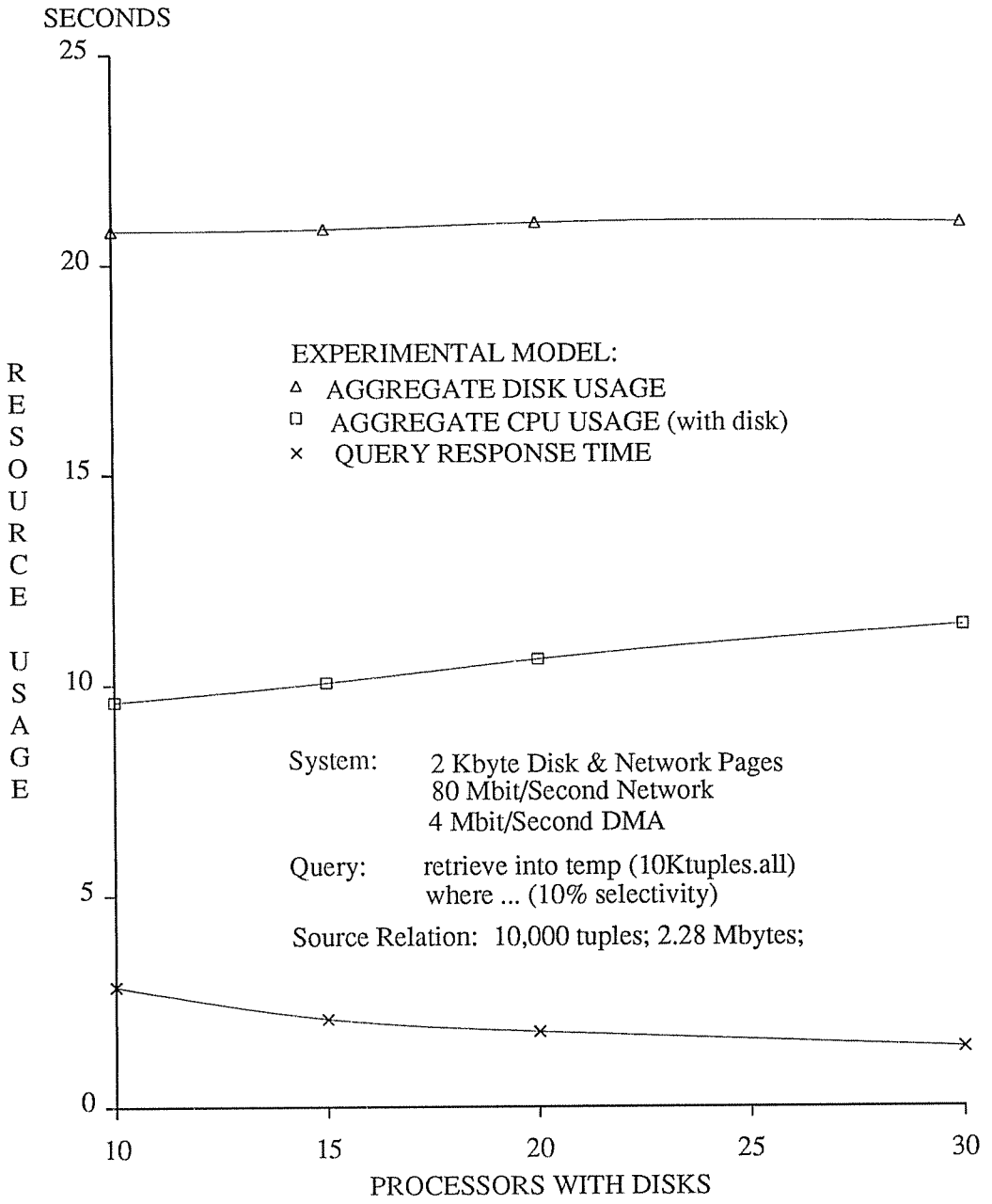
The increase in the aggregate level of CPU resources by the query operators reflects the fact that additional control messages are required as the size of the system is increased. In Figure 6, the success of removing the scheduler as a centralized bottleneck in the system is indicated by the fact that the scheduler consumes only a small fraction of CPU resources regardless of the size of the system. The scheduling algorithm evenly distributes the costs of managing the exchange of control messages to the query operator processes. Significantly, the CPU consumption and message passing latency accompanying the exchange of these control messages are incurred in parallel among the various operator processes.

Figure 6 demonstrates that the use of page-by-page round-robin streams and a fixed cost scheduler results in a consumption of network bandwidth that is highly independent of the size of the system. That is, the number of data and control messages that are exchanged by the experimental model increases at a low, linear rate as the size of the system is increased.

Finally, Figure 7 demonstrates that the experimental model can effectively use parallelism to reduce the

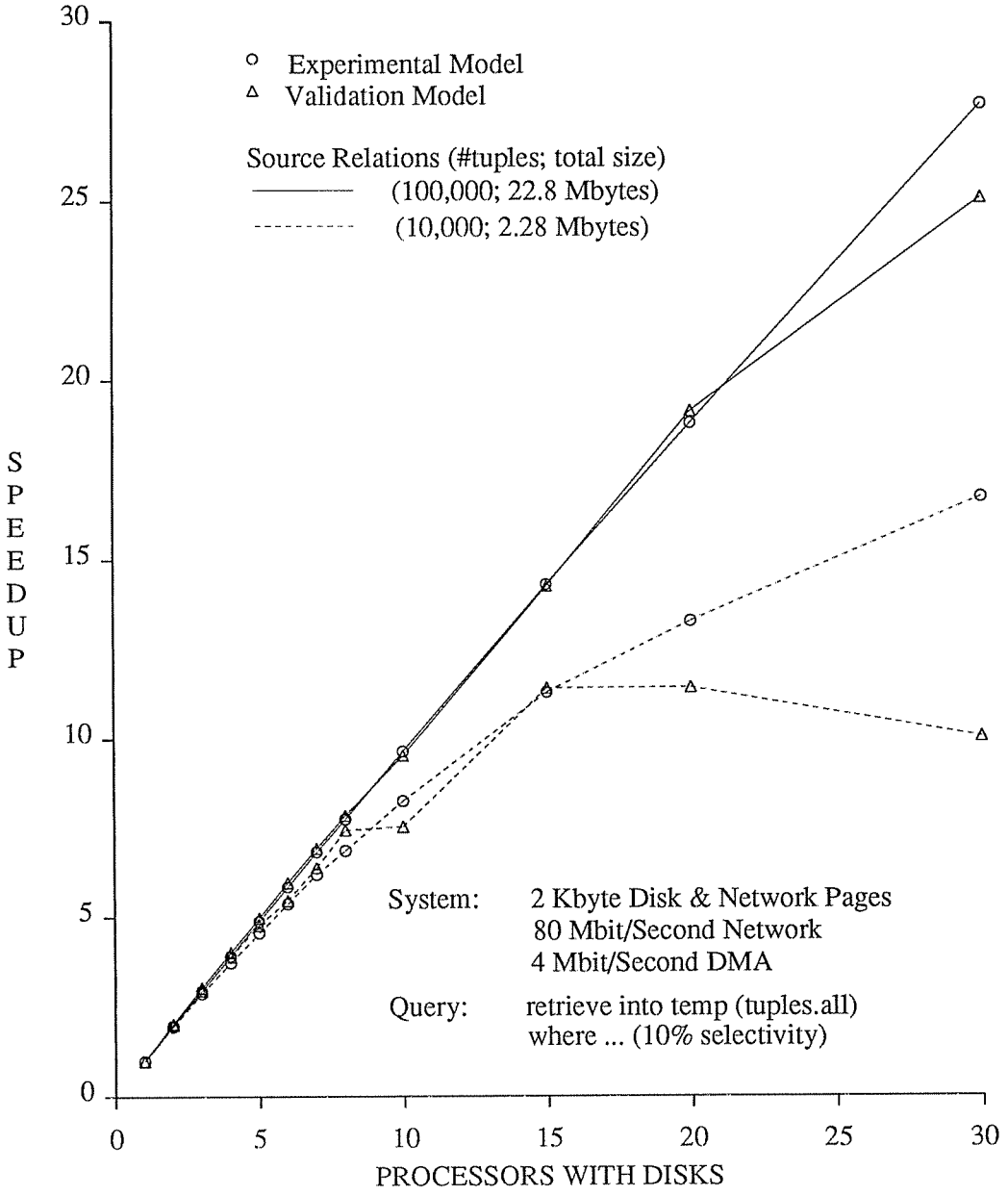


10,000 Tuple Selection Query: Resource Consumption
Experimental Model
Figure 6



10,000 Tuple Selection Query: Aggregate Resource Consumption
Experimental Model
Figure 7

response time that is associated with the example selection query. The aggregate disk and CPU consumption far exceed the net response time for the selection query. Figure 8 also reflects this fact by illustrating the performance speedup that accompanies the validation and experimental models for both the previously examined 10,000 tuple selection query and for a selection query that accesses 100,000 tuples. Both selection queries apply predicates with a 10% selectivity and are identical except for the sizes of the source relations.



Selection Query Speedup
Figure 8

Figure 8 demonstrates that increased levels of performance can be expected for selection queries that access sufficiently large amounts of data with respect to the number of resources that are used. One of the reasons for this result is that increasing the size of the system increases the number of control messages that are required for the management of a query. For the 10,000 tuple selection query, the performance speedup of both the validation and experimental models decline as the size of the system configuration is increased. This decline results from the

fact that the 10,000 tuple selection query produces only a small result relation (1,000 tuples; 228 Kbytes). For a configuration with 30 disks, each processor produces only 33 or 34 tuples of the result relation. Thus, a minimum of 120 data messages[8] must be exchanged in the course of the query for that configuration of the system. In contrast, as previously described, the query scheduler sends and receives a reliable message to each operator on each processor during both the activation and deactivation of the operator. Thus, a total of four control messages are exchanged by the scheduler and each processor with a disk in the process of scheduling each of the **selection** and **store** operations. For the 10,000 tuple selection query, therefore, twice as many control messages[9] (240) are sent as data messages (120). The experimental model is able to partially reduce the impact of this imbalance of control messages to data message via its scheduling algorithm that increases the parallelism that is applied to the flow of control messages.

In order to avoid an imbalance of control messages with respect to data messages, it may be advisable to store small result relations across a subset of the disks in large configurations. In this manner, subsequent selections or scans on the relation could be performed in a more efficient, albeit slower fashion.

3.4. Enhancements to the Gamma Design

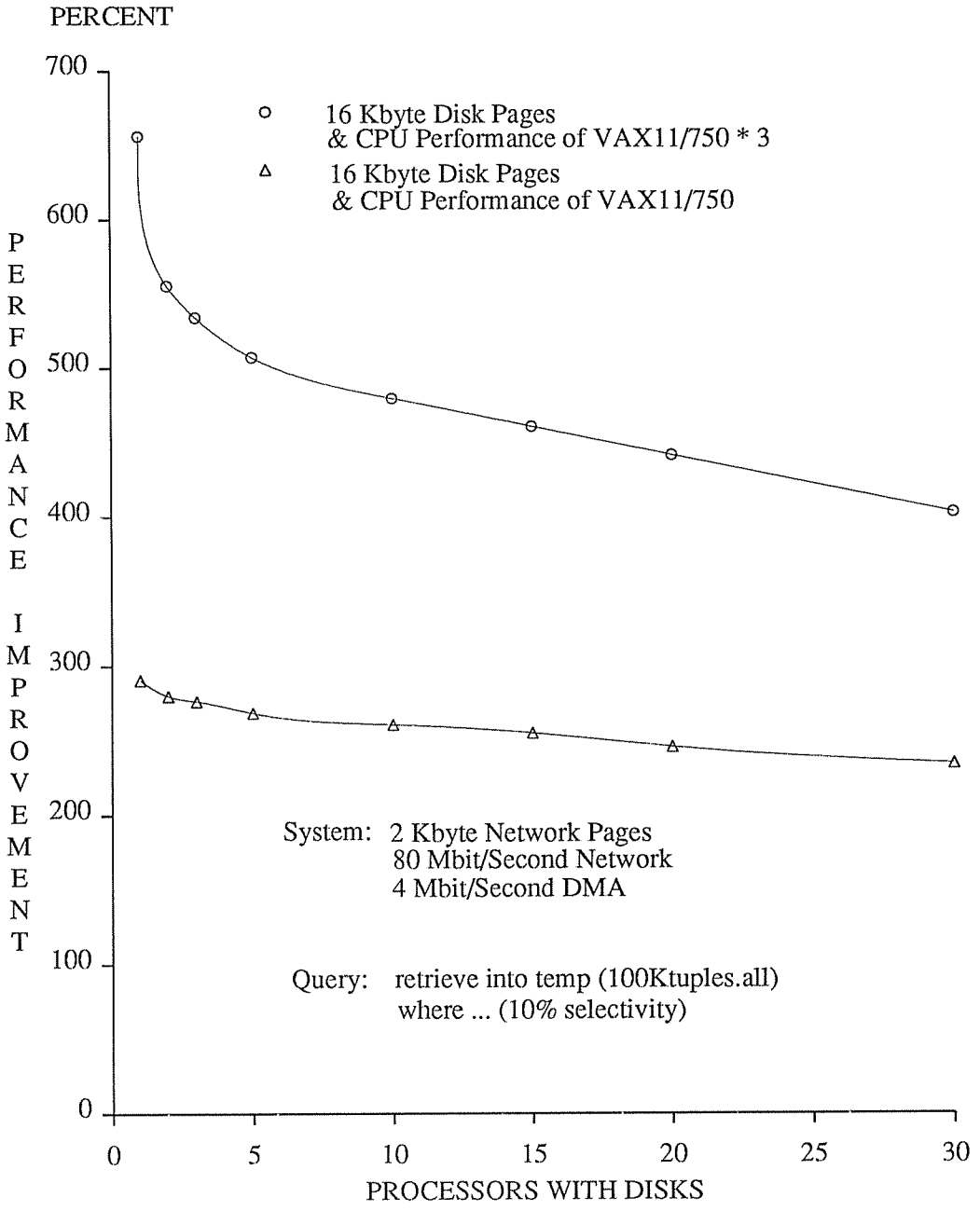
For the previously tested selection queries, the disks were the most heavily utilized resource in the system for Gamma. This bottleneck can be reduced by increasing the size of the blocks that are accessed by a single disk operation. Currently, Gamma uses 2 Kbyte disk pages. Figure 9 indicates that for selection queries, very significant performance improvements can be achieved by changing the disk block size from 2 Kbytes to 16 Kbytes. The figure presents the percentage improvement in performance[10]. Once the disk block sizes are increased, the performance of the processors themselves becomes a critical factor. The figure illustrates the further performance improvement that results from a 3-fold increase in the CPU performance for systems using 16 Kbyte disk blocks. The decline in the percentage of performance improvement for the systems with faster CPUs results from the use of short circuited messages. When the destination of a message resides on the same processor as a sending process, Gamma (and the validation model) do not send the message across the network, but instead short circuit the mes-

---

[8]Each processor uses 4 data pages to contain the 33 or 34 tuples that are produced locally.

[9]For each of the 30 processors, a total of eight control messages are required. Four control messages are exchanged when scheduling each of the **select** and **store** operators.

[10]Defined as the inverse of response time.



Percent Improvement in Performance of 100,000 Tuple Selection Query
Figure 9

sage within the confines of the local processor. The CPU overhead of making a copy of such messages and requeuing the messages to the destination port are more significant in smaller systems since a larger proportion of the total messages for a query will be short circuited. For the round-robin partitioned data streams of the selection query, the proportion of data messages that are short circuited is inversely proportional to the number of processors with disks

in the system. For example, the 10,000 tuple selection query results in the short circuiting of 100% of the total number of data messages for systems with a single disk. In contrast, for systems with four disks, 25% of the data messages for this query will be short circuited. Thus, smaller systems will tend to be more CPU bound than larger systems and the performance of these smaller systems will thus be enhanced to a greater degree by an increase in CPU performance.

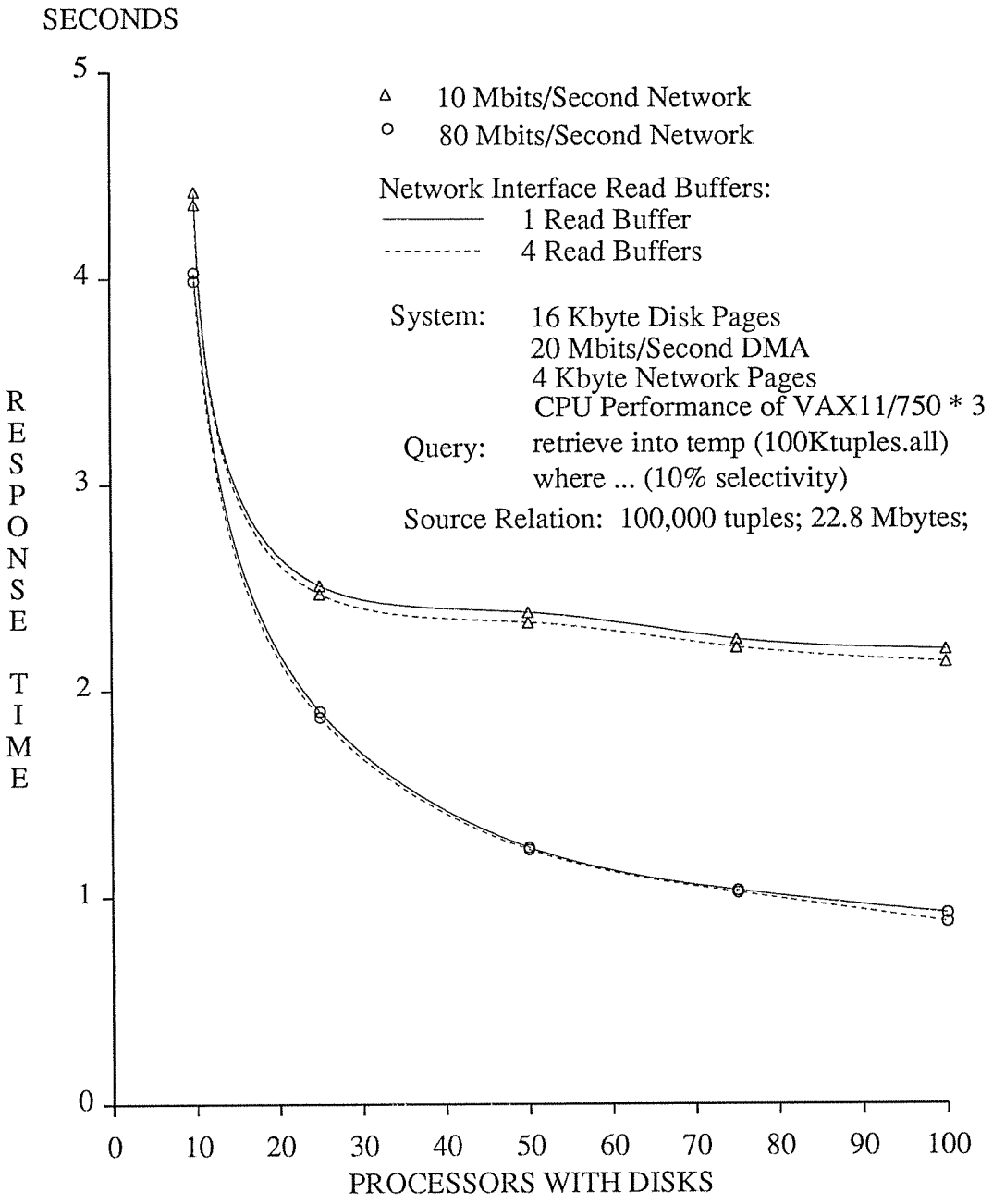
Figure 10 illustrates the ability of a single selection operation to saturate a token ring of moderate bandwidth. In this experiment, other potential bottlenecks in the system were removed by using large disk blocks, a fast CPU, a fast DMA controller (20 megabits/second instead of 4) and larger (4 kilobyte), maximum size[11], network packets. The number of read buffers on the processors' network interfaces had little effect in this experiment. Once the token ring became saturated, the existence of 4 read buffers only marginally improved the performance of the selection query in comparison to a system with network interfaces that only contained a single read buffer. If the data flow between producers and consumers of tuples streams is evenly distributed, it appears that the network bandwidth has more of a potential to become a bottleneck than does contention for read buffers at the destination sites. This observation is reinforced by the fact that all the selection query tests encountered very low levels of message retransmissions.

Figure 11 provides an indication of the extent to which a Gamma system with fast CPUs using large disk blocks can be effectively expanded. The indicated selection query accesses a source relation containing 1,000,000 tuples. The figure indicates that significant speedups can be expected for large configurations. However, performance does start to decline somewhat in systems containing 50 or more CPUs and disks.

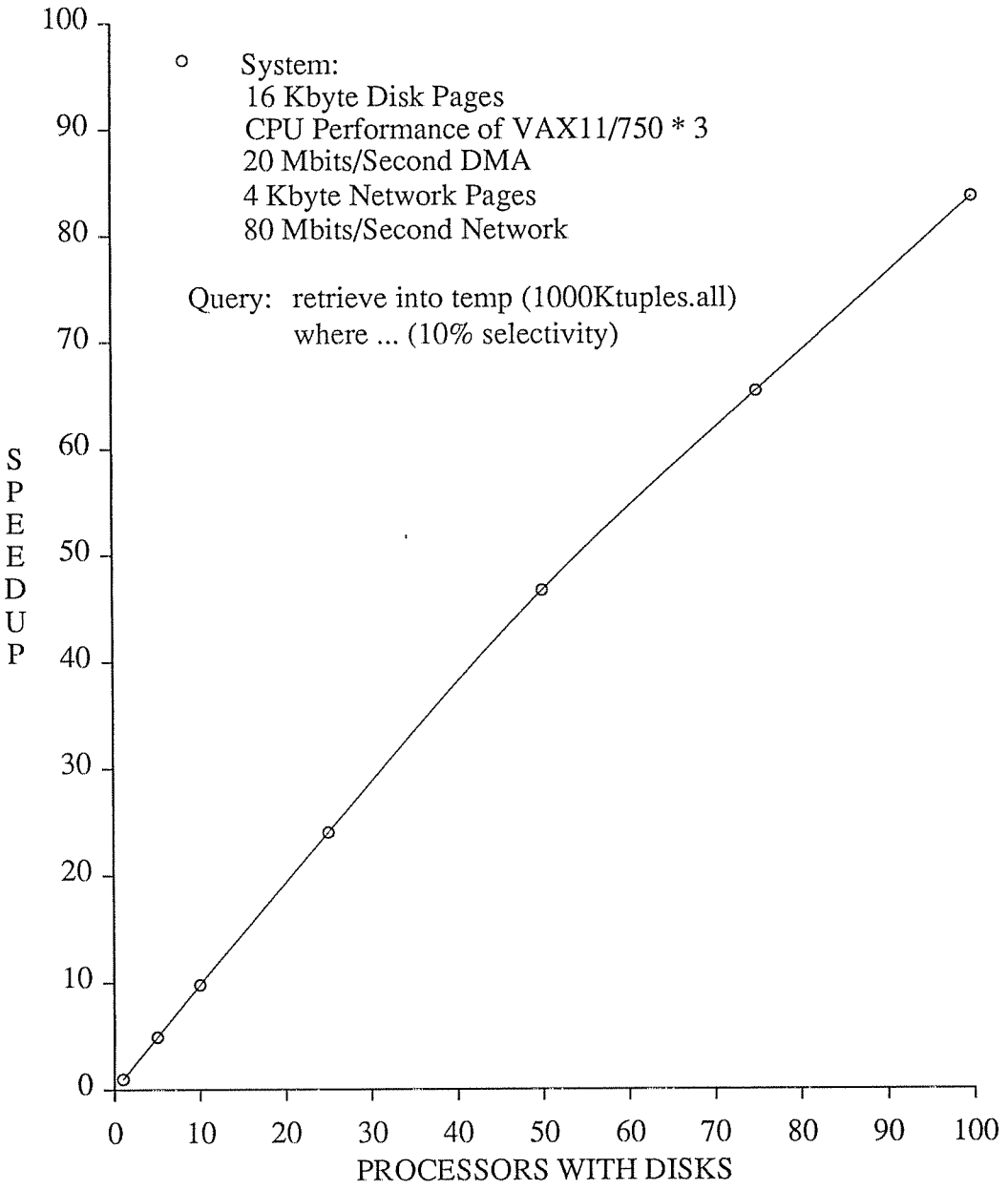
As indicated in the accompanying resource usage figures of Figure 12 and Figure 13, the workload associated with the 1,000,000 tuple selection query can be effectively distributed among systems containing up to 100 processors with disks. The cost of scheduling and controlling this query remains at a low level. That is, the synchronization and control of a large Gamma system does not become a bottleneck for the tested query.

In Figure 12, the ratio of the time that the network is busy with respect to the total elapsed time for the query indicates that the network is becoming more highly utilized. For systems with 100 processors/disks, the effective network utilization exceeds 50%. The effect of this increased utilization is that operator processes are more frequently blocked during a network access waiting for a token. However, for all systems tested, very few

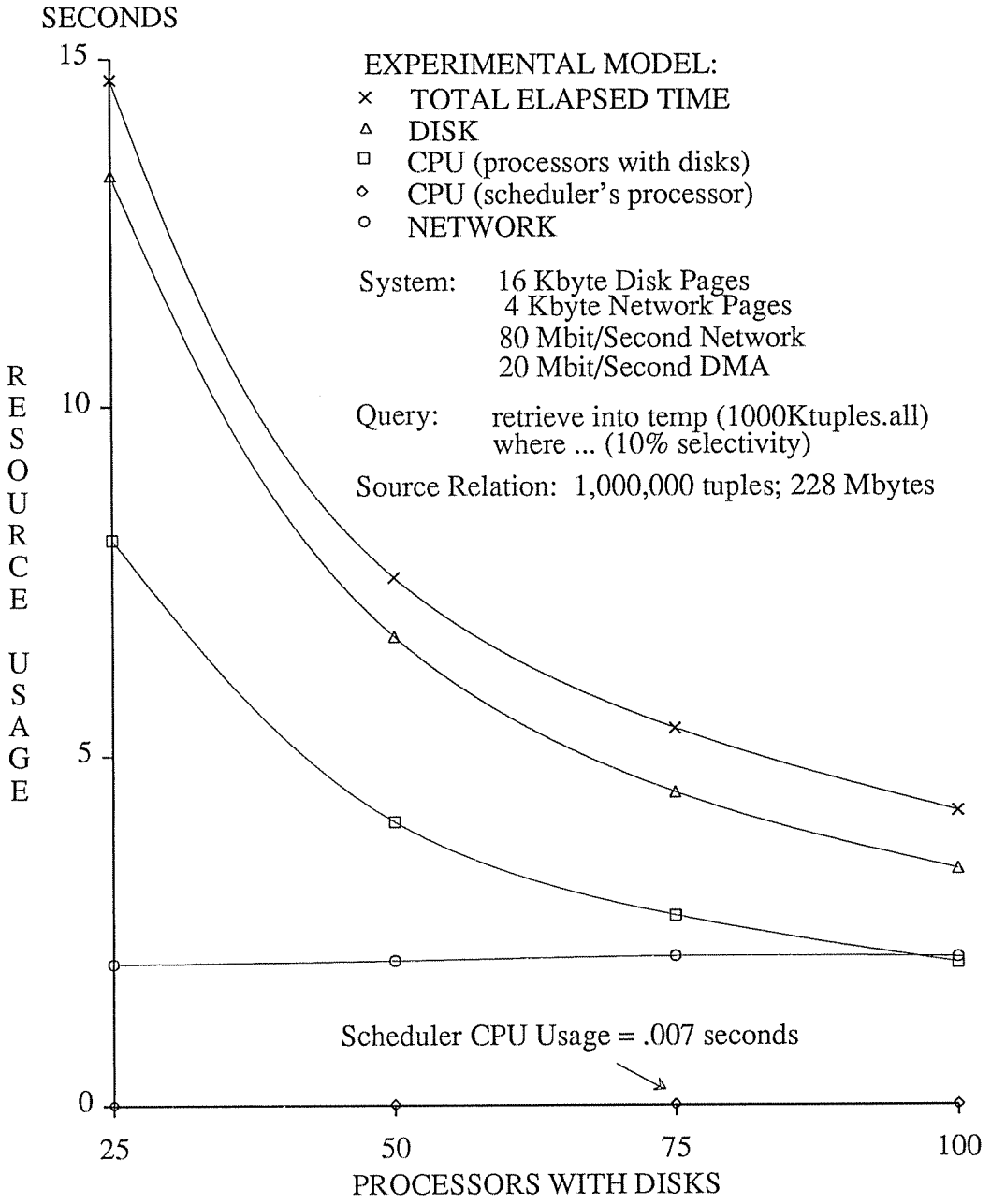
[11]The maximum size of packets is an attribute of the modeled network interface.



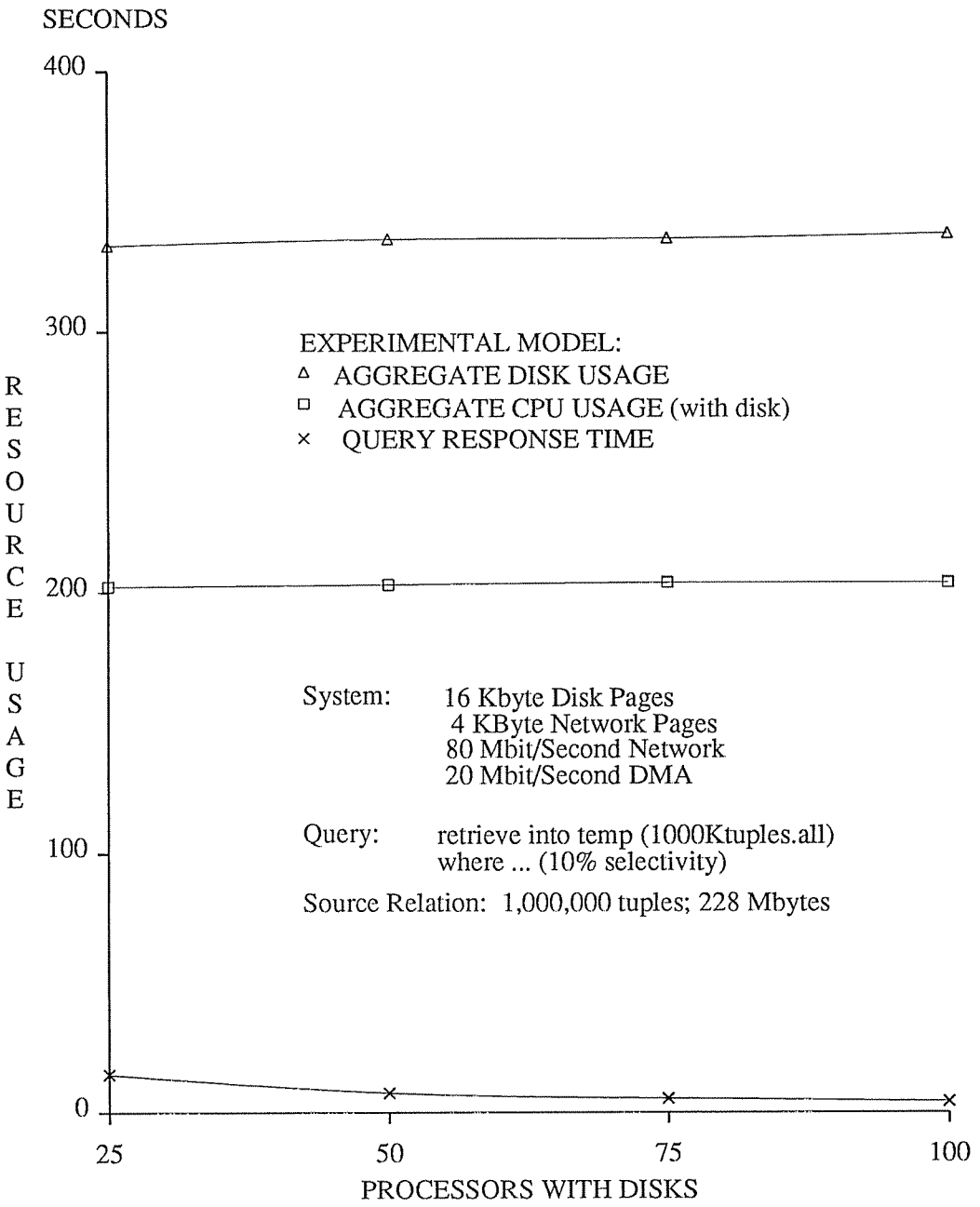
100,000 Tuple Selection Query with Varied Network Capabilities
Figure 10



1,000,000 Tuple Selection Query Speedup
Figure 11



1,000,000 Tuple Selection Query Resource Usage
Figure 12



1,000,000 Tuple Selection Query: Aggregate Resource Usage
Figure 13

retransmissions occurred. That is, acknowledgements were delivered before senders timed out and retransmitted messages. Also, contention for the network buffers of the receivers did not develop. The availability of a non-blocking network write facility might be expected to reduce the negative impact of increased network utilizations on the performance of queries in large systems by reducing the latency incurred in waiting for tokens. A non-blocking network write facility might also enhance query throughput in a multi-user environment as a query would not be

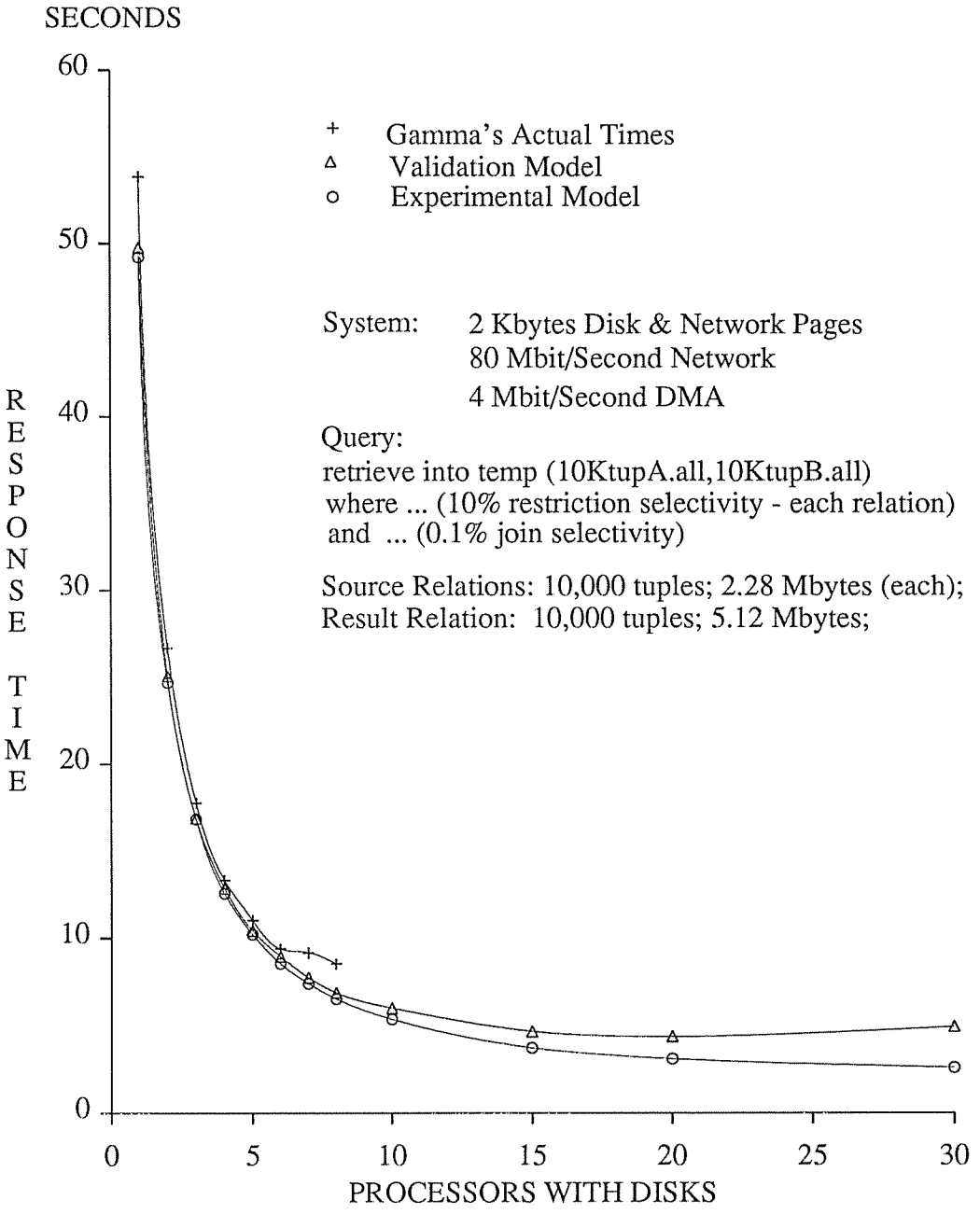
blocked by the write operations of other concurrent query processes.

4. Analysis of Join Query Performance

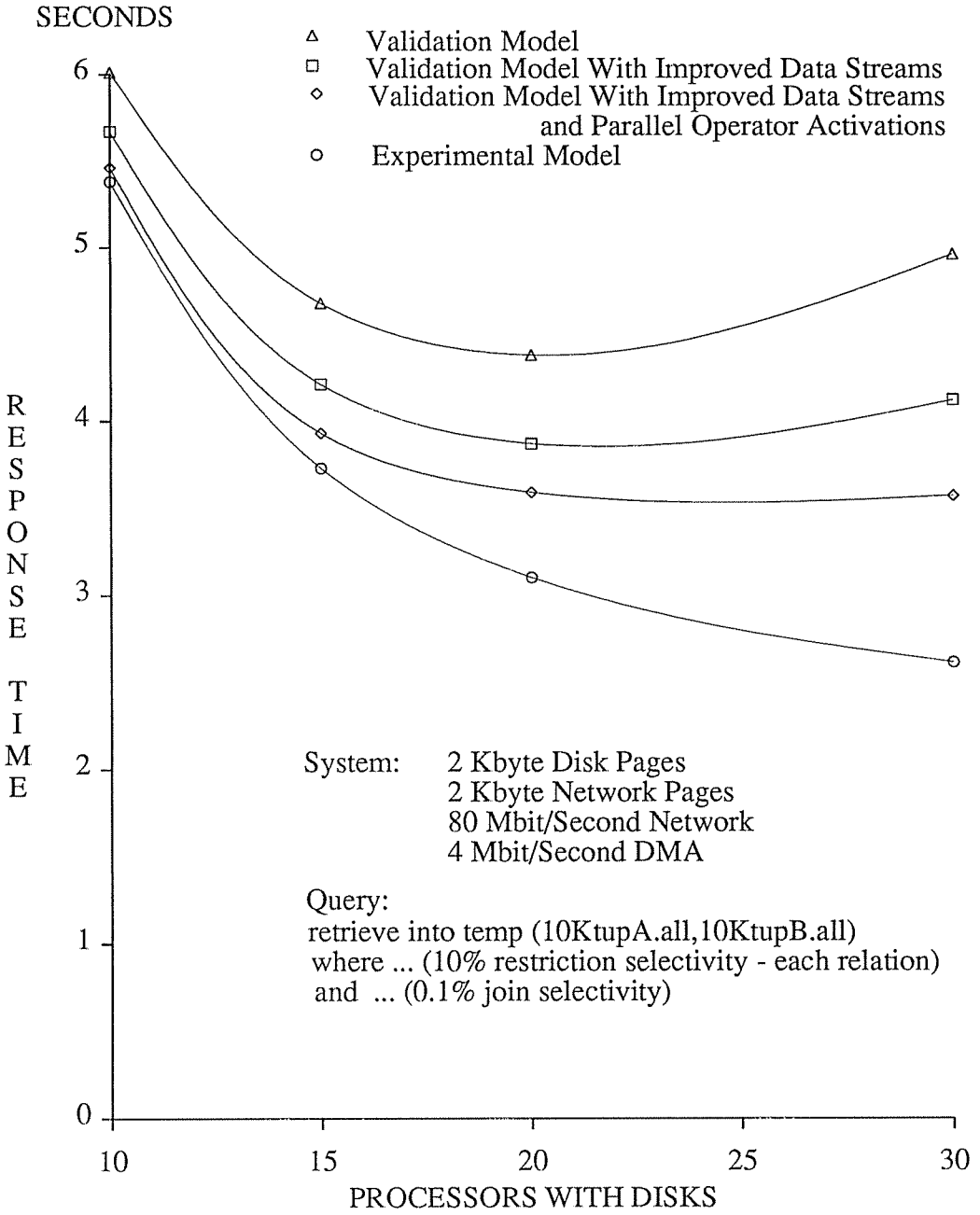
One of the promising features of Gamma is the ability to execute join operations efficiently on processors that are remote from the sites where the source relations are produced. In this section, we examine the impact of applying the design enhancements of the experimental model described in Section 3 to the execution of a join query. The simulation model is then used to explore the level to which the parallelism of a large multiprocessor Gamma system can be effectively applied to a join query that accesses large amounts of data.

Figure 14 illustrates the ability of the validation model to accurately simulate the measured performance of a join query. Each of the source relations of the join operation are produced by a 10% selection on a 10,000 tuple base relation. For both source relations, each of the 1,000 tuples qualified by the selection participate once in the result relation of the join operation (which thus has a join selectivity of 0.001). The performance of the validation model ranges between 90% and 96% agreement with the measured performance of Gamma for systems with up to twelve processors (half of which have disks). As in the case of the simulated selection query, the validation model does not attempt to model the performance anomaly caused by Gamma's non-homogeneous seventh and eighth disks. The periodicity evident in the modeled selection query of Figure 1 is less obvious in the join query, as the hash-partitioned data streams produced by the selection operators flush pages across the network at more irregular intervals than the tuple-by-tuple round-robin partitioning strategy. However, at the end of the selection operation on the building and probing relations, the hash-partitioned strategy still flushes an increasing large number of partially filled pages as the size of the system is increased. However, in contrast to the selection query, these partially filled pages do not result in any additional disk I/O, as they are consumed by join operator processes and not **store** operator processes.

Figure 15 reflects the relative performance impact of applying the improvements to the data stream and scheduling algorithms that were analyzed for the selection queries in the previous section. The join operator processes apply round-robin partitioning to the generated result relation. The curve representing the system with improved data streams illustrates that the performance of the join is significantly improved by using a page-by-page round-robin strategy and collecting full pages of tuples before writing pages to the result relation. The beneficial impact of using a parallel activation strategy is also significant. However, most noticeable is the improvement pro-



10,000 Tuple Join Query Response Times
Figure 14



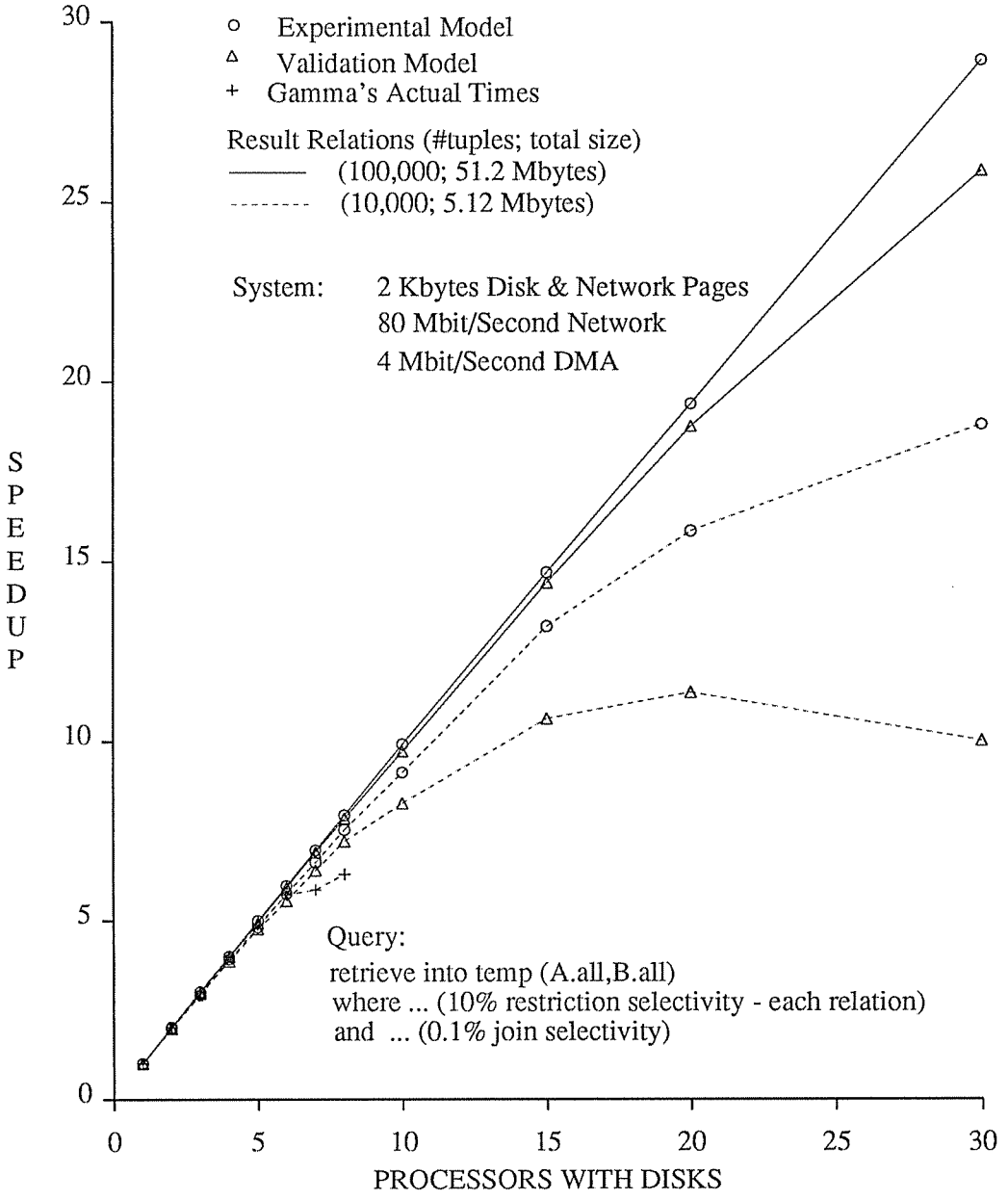
Performance Impact of Query Control Improvements
Figure 15

vided by the distributed scheduling algorithm[12] of the experimental model. The increased number of operators in the join query magnifies the relative performance benefits of the control strategy of the experimental model for large systems. Five separate operators have to be activated in the course of the query (2 selections, the build and probe phases of the join, and the store operation) whereas only two operators were activated for the selection query.

[12]Parallel distribution of control messages through a binary control tree.

Therefore as the size of a system is increased, the distributed scheduling algorithm of the experimental model become increasingly important

The speedup factors for the 10,000 tuple join queries are shown in Figure 16. For small configurations, Gamma and both of the simulation models produce linear speedups. However, as was the case for the 10,000 tuple selection, the problems caused by the design flaws of the validation model are compounded in larger configurations



Join Query Speedup
Figure 16

because the ratio of control to data messages becomes significant. The performance of the validation and experimental models is also presented for a join of two 10%-selected 100,000 tuple relations. The high data volumes of this query allow the experimental model to provide 100% linear speedup across the entire spectrum of the presented systems. The high data volumes of this join query also mask most of the negative performance aspects of the validation model.

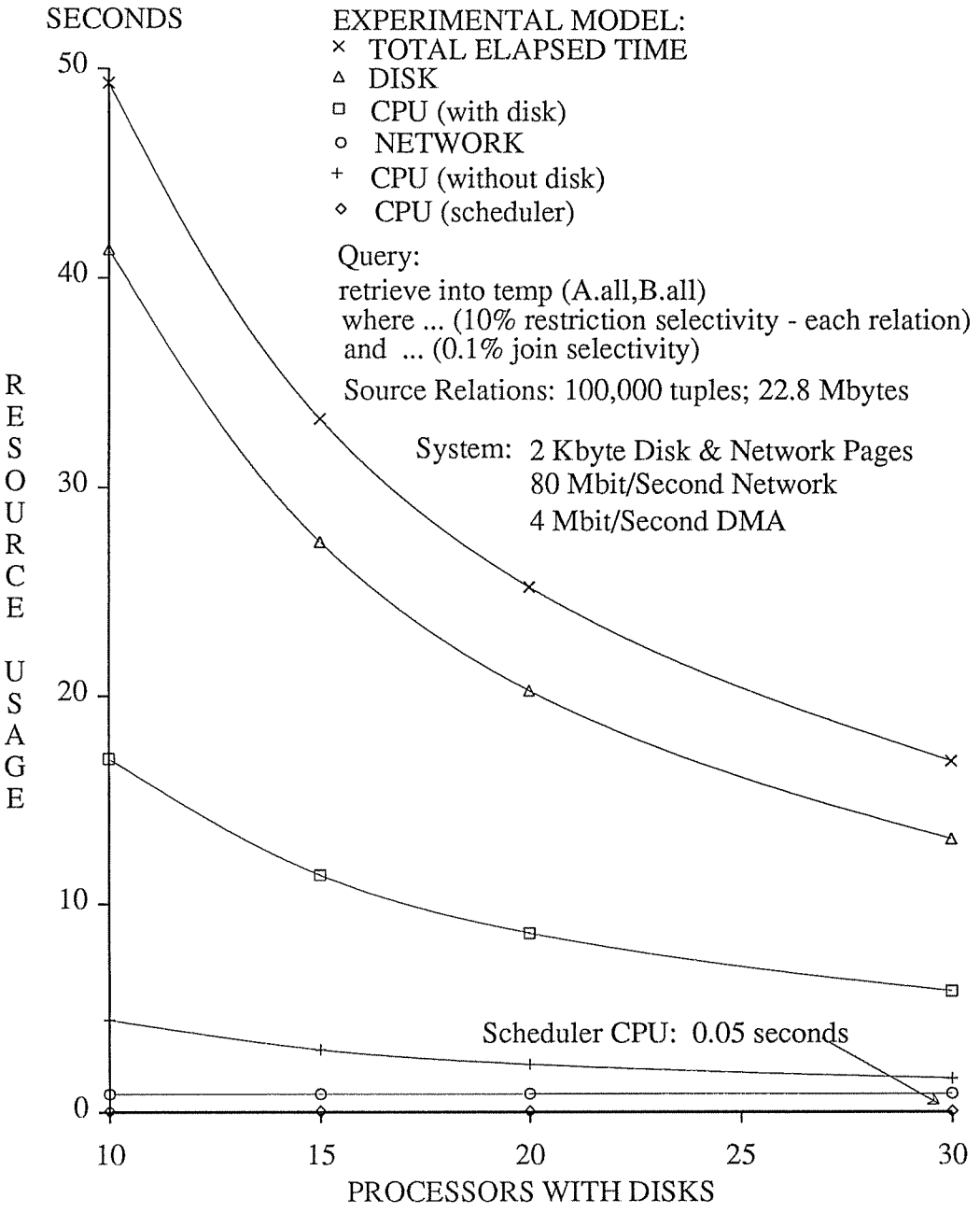
Figures 17 and 18 present the resource usage of the 100,000 tuple join query that provide the basis for the 100% linear speedup in the performance of the query. As the size of the system is increased, the disk and CPU workload are evenly shared. That is, as resources are added to the system, Figure 17 shows a proportional decrease in the resource consumption of the individual disks and processors. Similarly, Figure 18 does not show any increase in the total disk or CPU workload as the size of the system is increased. Also, as was the case for the selection queries, the CPU resources consumed by the scheduler are minimal.

The slight decrease in the aggregate disk usage in Figure 18 is due to the fact that in larger systems a higher proportion of pages of the result relation reside in the buffer pools of the processors with disks at the end of a query. Before the **store** operation completes, these remaining pages of the result relation are flushed to disk. However, at the end of a query the flushing of these pages does not interfere with the disk accesses of the selection operation. That is, for this query, as the size of the system is increased, there is less movement of the disk arm as the relation scans of the **selection** and **store** operations conflict less frequently. It should be noted, however, that this effect is an artifact of the single user benchmark query and would not be expected to persist in a multi-user environment.

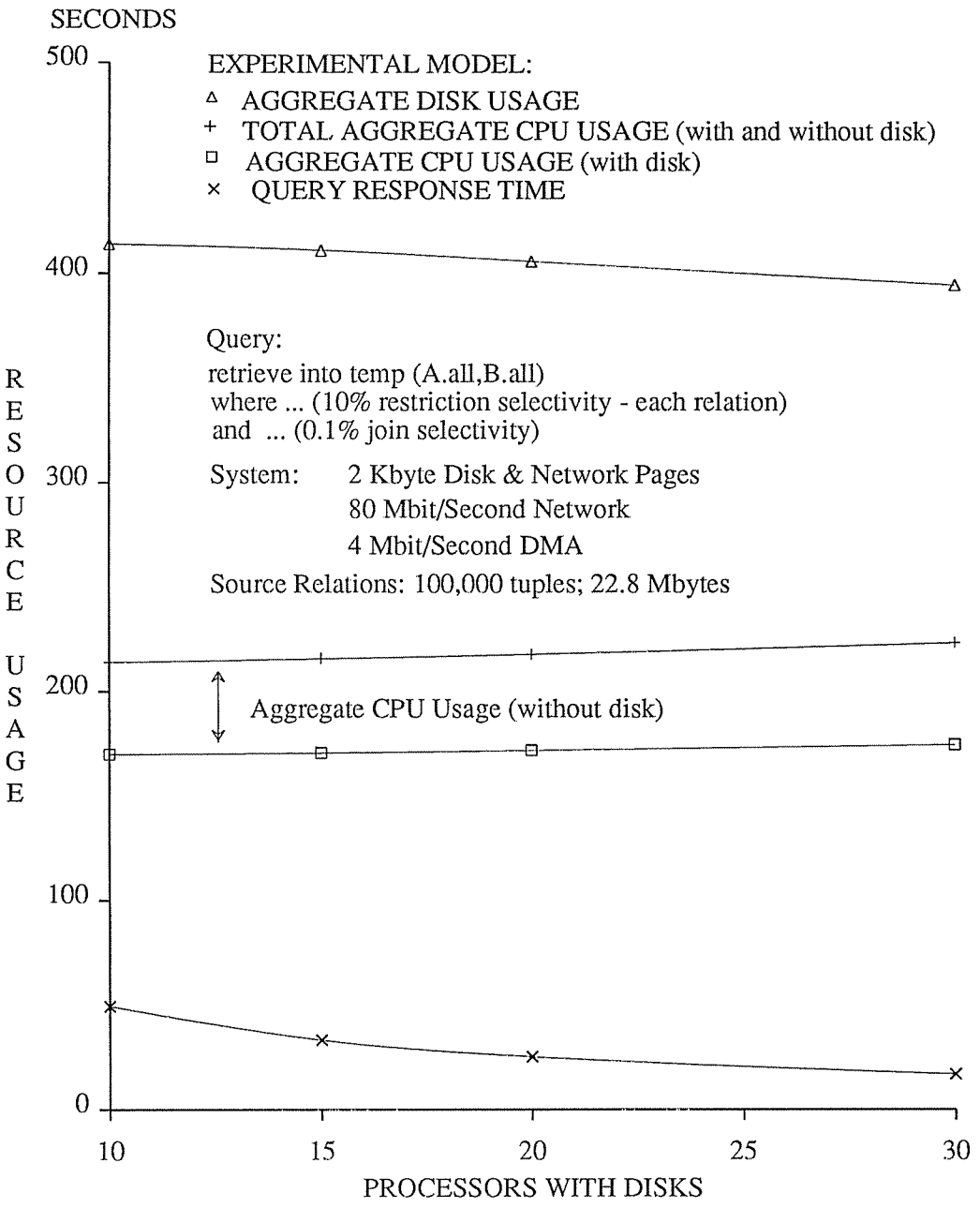
Figure 19 presents the performance speedups that accompany a join of two 10%-selected 1,000,000 tuple relations. For systems with larger disk blocks, etc., and up to 50 disks (and 101 processors), the speedup factors for this join query are linear, with almost a 100% speedup for each additional disk[13] However, beyond this point the speedups for the query fall off dramatically. As Figure 20 illustrates, there are still significant levels of available network bandwidth.[14] In Figure 20, the relatively uniform consumption of network bandwidth reflects the fact that the decreased response times for larger systems are not caused by significant numbers of retransmissions. Instead,

[13]With each additional disk, two processors are also added to the system. Only one of these processors is associated with the additional disk. That is, the size of the system is increased by adding a processor with a disk and another processor without a disk.

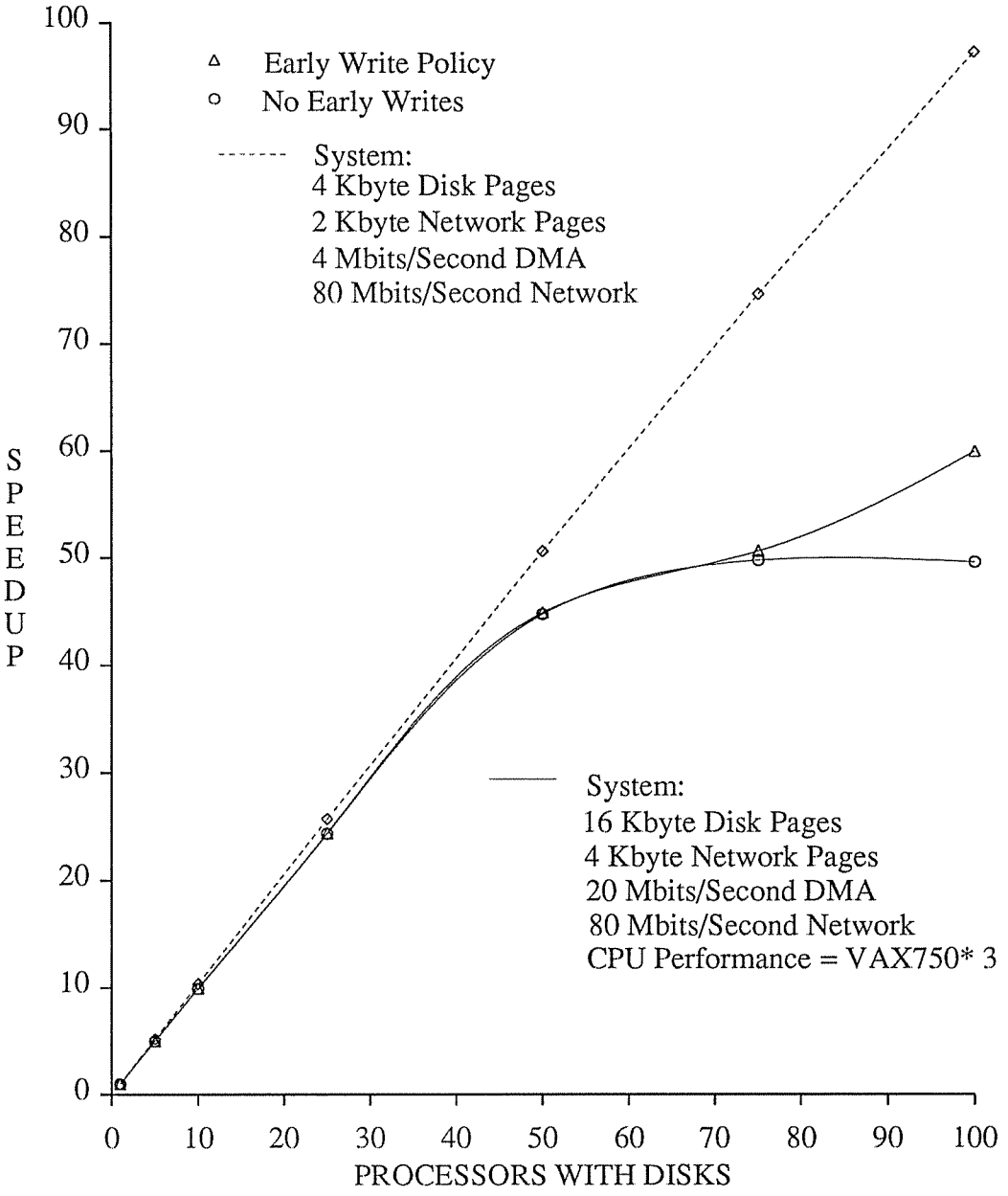
[14]The ratio of network usage to the response time of the query for a system with 50 disks indicates that the network is 54% utilized.



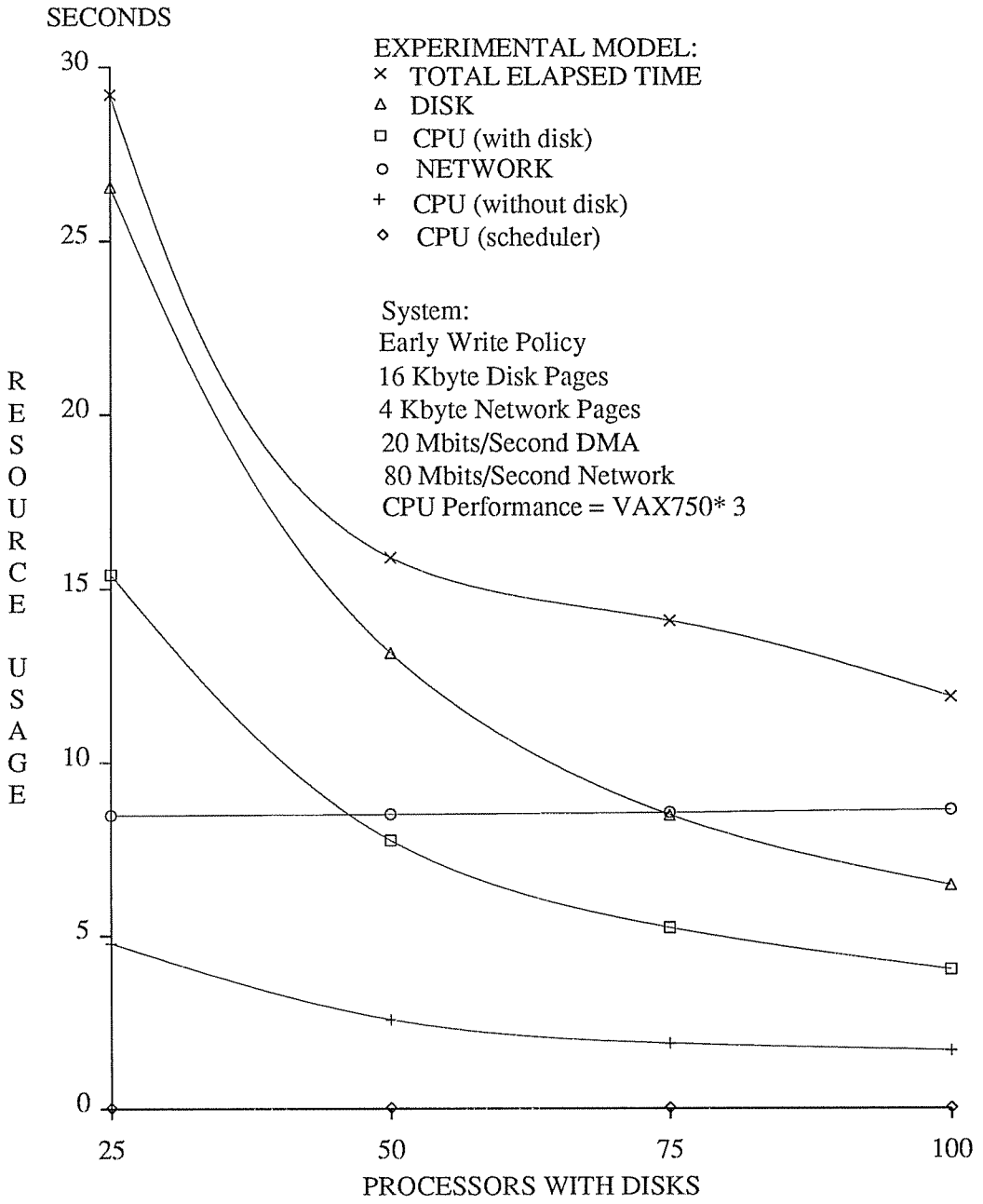
100,000 Tuple Join Query: Resource Consumption
Figure 17



100,000 Tuple Join Query: Aggregate Resource Consumption
Figure 18



1,000,000 Tuple Join Query Performance Speedup
Figure 19



1,000,000 Tuple Join Query: Resource Usage
Figure 20

the problem appears to be related to the highly periodic pattern[15] with which the join query loads the network. Convoys of processes end up waiting for tokens for significant amounts of time. For the 100 disk system, the average time spent waiting for a token when writing a message was 1.27 milliseconds. In contrast, systems with less than 50 disks seldom waited for a token at all. For a fully saturated network, such waits would be expected. However, the

[15]This is a artifact of the single-user operation.

network is not completely saturated by this configuration.

In an attempt to break up the periodic nature of network accesses generated by this join query, a policy of writing some of the output buffers before they completely were filled was implemented. In this **early write** policy, each operator process writes out one half of its output buffers initially at the time when they become half full. As Figure 19 illustrates, this policy is mildly successful.

Figure 19 seems to indicate that a more extensive policy of early writing or randomizing the production of hash-partitioned data pages is required.[16] The current policy of flushing partially full output buffers (early write policy) is only applied the first time the buffers are filled. However, this policy is only effective in those cases when the output buffers are only filled and flushed a small number of times.[17] In smaller configurations (40 to 80 disks), the output buffers are filled and flushed a greater number of times and the current, limited early write policy is ineffective in preventing the eventual formation of convoys. That is, for the tested selection query using 16 kilobyte disk pages, source tuples for the join operation can be produced quickly enough to negate the initial randomness introduced by the writing of partially full output buffers. In the future, a more aggressive policy of randomly writing out partially full output buffers will be tested. Potentially, the performance benefits of introducing additional randomness into the pattern of network access for large systems using large disk pages will outweigh the additional CPU resources that will be consumed transmitting a larger number of partially full pages.

In Figure 19, the performance speedup for a system using smaller, 4 kilobyte disk pages and more modest hardware is also presented. The speedup for this system demonstrates that linear speedups for join operations can indeed be maintained for very large systems. That is, the scheduling algorithm does not become a bottleneck when join operations are executed on configurations with large numbers of processors.

Despite the decline in speedup, both systems using 16 kilobyte disk pages still provided better absolute performance than the system using 4 kilobyte disk pages. For configurations containing one hundred disks, the system with 16 kilobyte disk pages and early writes of partition buffers performed the 1,000,000 tuple join query in 11.9 seconds, whereas the system with 4 kilobyte disk pages executed the query in 28.1 seconds.

While the system using 4 kilobyte disk pages also generated a highly periodic load on the network, the effect was less severe than for the system using 16 kilobyte disk pages. The reason for this difference results from

---

[16]However, this problem may not exist in a multi-user environment due to the random effects of competing for resources with other concurrently executing queries.

[17]For the system with 100 disks the output buffers of each processor with a disk are flushed at most twice.

the fact that systems with larger disk pages fill and flush partition buffers at a faster rate. That is, a larger disk page will yield more qualified tuples, each of which may potentially fill a partition buffer and trigger a network access. Furthermore, since systems with larger disk pages have lower response times, they generate higher levels of overall network utilization. With higher network utilizations, the convoys of processes waiting for tokens are formed more frequently and persist for longer periods of time.

5. Conclusions

The simulation model of Gamma identified a number of software modifications capable of significantly increasing the performance of the system. Specifically, the following changes were found to provide significant improvements in the response time of individual queries:

- Page-by-page round-robin partitioning for distributing pages of result relations.
- Collecting and writing full pages of tuples to disk.
- Larger disk pages.
- Parallel scheduling algorithms.

Gamma has been modified to partition result relations on a page-by-page round-robin basis. In the revised implementation, a single output buffer is used when a result relation is partitioned. When the buffer fills, it is written to the destination process associated with the current partition. The partitions receiving pages of tuples are alternated on a round-robin basis. Also, each of the processes producing fragments of a result relation use different initial partitions. While this method of round-robin partitioning does not provide as uniform a distribution of tuples as the earlier tuple-by-tuple method, it consumes fewer system resources and reduces network congestion. With page-by-page round-robin partitioning, only a single page has to be flushed across the network when a operator process that is producing a permanent result relation completes.

A second modification was also made to the way data streams are handled by Gamma. Previously, the **store** operators read pages of a result relation directly from the network into a page of the buffer pool. Later, these buffer pages were flushed to disk. This method of reading pages of a result relation directly from the network into disk buffers saved the CPU resources that would otherwise be spent copying tuples from network buffers to disk buffers. The simulation study indicated, however, that this strategy resulted in large numbers of partially full pages being appended to a file.

To solve this problem, the **store** operator was modified to collect a full page of tuples before appending a page to a result relation on disk. While this change requires copying tuples between network and disk buffers, it results in at most one partially full page being appended to a file. In addition to the savings in time associated with fewer disk I/O operations, both for storage and later retrieval of the data, this change also makes it possible to separate the size and the format of the network and disk pages from one another.

We have also modified Gamma to use larger disk page sizes. While the prototype currently uses 4K byte disk pages, it is now possible to recompile the system to use any page size up to the size of a disk track. The impact of the modified scheduling algorithm will be investigated in a future revision of the Gamma software.

REFERENCES

[ARMS68] Armstrong, J., Ulfers, H., Miller, D., Page, H., SOLPASS, A Simulation Oriented Language Programming and Simulation System, Technical Report, 1968.

[BITT83] Bitton D., D.J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems - A Systematic Approach," Proceedings of the 1983 Very Large Database Conference, October, 1983.

[COOK80] Cook, R. P., "*MOD - A Language for Distributed Computing", IEEE Transactions on Software Engineering, 6, 6, November, 1980.

[DEWI84] DeWitt, D. J., Finkel, R., and Solomon, M., "The Crystal Multicomputer: Design and Implementation Experience," IEEE Transactions on Software Engineering, August, 1987.

[DEWI85] DeWitt, D., and R. Gerber, "Multiprocessor Hash-Based Join Algorithms," Proceedings of the 1985 VLDB Conference, Stockholm, Sweden, August, 1985.

[DEWI86] DeWitt, D., and R. Gerber, "GAMMA, A High Performance Dataflow Database Machine", Proceedings of the 1986 VLDB Conference, Tokyo, Japan, August, 1986.

[DEWI87] DeWitt, D., Ghandeharizadeh, S., Schneider, D., Jauhari, R., Muralikrishna, M., and A. Sharma, "A Single User Evaluation of the Gamma Database Machine," to appear, Proceedings of the 5th International Workshop on Database Machines, October, 1987.

[FINK83] Finkel, R., Cook, R., DeWitt, D. J., Hall, N., Wisconsin Modula: Part III of the First Report on the Crystal Project, University of Wisconsin -- Madison, Technical Report 501, April, 1983.

[GERB81] Gerber, R., Cook, R., and P. Clancy, The Starpak Modula Simulation Package, University of Wisconsin -- Madison, 1981.

[GERB86] Gerber, R., "Dataflow Query Processing using Multiprocessor Hash-Partitioned Algorithms," Ph.D. Dissertation, University of Wisconsin Technical Report #672, Madison, WI, October, 1986

[KNUT64] Knuth, D. E., and J. L. McNeley, SOL -- A Symbolic Language for General Purpose Systems Simulations, IEEE Transactions on Electronic Computers, August, 1964.

[PROT85] Proteon Associates, Operation and Maintenance Manual for the ProNet Model p8000, Waltham, Mass, 1985.

[RIES78] Ries, D. and R. Epstein, "Evaluation of Distribution Criteria for Distributed Database Systems," UCB/ERL Technical Report M78/22, UC Berkeley, May, 1978.

[WATS81] Watson, R. W., "Timer-based mechanisms in reliable transport protocol connection management", *Computer Networks 5*, pp. 47-56, 1981.