PROJECT MAC March 6, 1973

Computer Systems Research Division Request for Comments No. 10

S.M. THESIS PROPOSAL: AN EXPERIMENTAL ANALYSIS OF PROGRAM REFERENCE PATTERNS IN THE MULTICS VIRTUAL MEMORY

by Bernard S. Greenberg

Abstract: Recent experiments have shown that patterns of memory reference by programs operating in a virtual storage environment can be approximated by a remarkably simple model. The extents and limitations of this model are not known. An experiment is proposed to observe this behavior over previously unexplored domains. New models of reference behavior can then be derived.

Introduction

During the past ten years, the complexity of digital computer systems has increased beyond the point where their behavior may be easily predicted from careful study of their design. At this point, probabilistic and heuristic models of computer behavior must be devised, in order that hardware and software systems may naturally adapt to the services required of them. In this thesis, we shall concern ourselves with the formulation of a particular class of such models for the demands made by programs on computer memory systems.

In the class of computer systems possessing what are known as "multilevel memory systems", all information known to the system is stored in hierarchically ordered levels, ordered by the time required to retrieve information stored in that level. It is incumbent upon the operating system to effect the transfer of information between these levels. The operating system must anticipate the system's need for each data item, in order to manage the various levels of store. A popular heuristic, "least-recently-used", or "LRU", maintains that the more recently a data item has been accessed, the higher the likelihood of its re-use in the near future. Hence, multilevel memory systems managed by this technique strive to keep the most recently used data in the most readily accessible level of store. In most current such systems, the retrieval of data from successively lower levels (i.e., those having greater retrieval times) of the hierarchy is associated with an undesirable overhead.

Furthermore, data retrieval operations from these slower-responding levels usually implies the suspension of the requesting program. This causes slower response and decreased throughput, although multiprogramming attempts to minimize the latter. Hence, it is desirable to minimize the mean depth into the storage hierarchy of data retrieval requests. However, for the foreseeable future, the higher levels of multilevel storage systems seem considerably more expensive on a cost per bit basis than the lower levels, and an optimal cost/performance balance can be arrived at only with knowledge of the relative frequencies of access to each level. A simple model of this behavior, known as the "linear paging model", has been proposed by Saltzer (1) based upon measurements made on the Multics system at MIT. It states that the proportion of memory references which access data stored deeper than a given level in the hierarchy, taken in comparison to all memory references, is in inverse proportion to the total size of that portion of the memory system within that level. This is to say that doubling the main store of an LRU-managed system will halve the number of references extending out of this store. This model, which makes much intuitive sense, is essentially an empirical model, known to be true only within the limits of the experiments (which will be described) on which it was based. It would be of great use and interest to determine the nature of program memory reference pattern behavior beyond the measured scope of this model: to determine the range of applicability of the linear model, and what models hold beyond these limits. It is the intent of this thesis to do precisely that. By means of an experiment involving Multics, a large-scale analysis of program reference pattern behavior will be attempted. During the course of this experiment, a data-gathering operation will produce an extensive collection of data of types not gathered before, which may well suggest new and interesting observations, as well as help formulate a theoretical basis for the linear paging model, and provide grounds for future research.

Motivation

Sekino (2) has shown the importance of the mean headway (defined as the average number of total memory references by a program) between references to secondary storage. as a parameter in system throughput and response time calculations. For the class of computer systems under consideration, he expresses this headway as a function of primary memory size, and assumes the linear model as a basis for many of his further results. The nature of this function beyond a primary memory size of two million words has neither been observed nor analytically determined. The range of applicability of Sekino's results is thus limited to the two million word range over which this function is known. Over this range, direct measurements have shown remarkable adherence to the linear model. These measurements were made by J. H. Saltzer (1) on the Multics system, which has a three-level storage hierarchy, consisting of core, drum, and disk. This hierarchy is used by the software in such a way that it gives the appearance of a single, hardware-addressable entity, usually known as the "virtual memory" (3). Saltzer's measurements attempted to ascertain the mean headway between references extending between each of the levels of the hierarchy. These headways were approximated by noting the mean useful time between disk and drum references combined, and the mean useful time between disk references, for two different sizes of core and two different sizes of drum. By this method, four measurements were obtained, which seemed to fall well within the region predicted by the model. The approximation of the mean headway between exceptions (i.e., references past a given level) by the mean time between exceptions is an interesting one, which will be considered in this thesis.

The Multics memory hierarchy is managed by a practical approximation to the LRU algorithm. The deviation from the ideal is due to the exigencies of the minimization of overhead in a complex operating system, as well as the limitations of practical hardware to determine recency of use of a data item. Both core and drum are managed by these near-LRU algorithms, this management having the ultimate effect of determining what data is to be driven off each, down the hierarchy, (hopefully the least recently used on each), as processors making virtual memory references cause data to move up in the hierarchy. The disk requires no such management, as there is no lower level of store to which data must be moved. The implementation of both the core and drum management schemes takes the form of a list structure, wherein "pages", the 1024-word indivisible reference unit of Multics, are moved to the head of the list as they are used or found to be in use, or to have been in recent use. The result of this discipline is that the last item on the list for each level of store is the first candidate for replacement, when the storage management algorithm decides that pages must move down the hierarchy.

The LRU algorithm is one of a class of algorithms known as "stack algorithms" which impose an ordering upon the pages managed by it. For the LRU algorithm, this ordering takes the form of an ordering by recency of use. The analysis of such orderings has been thoroughly explored by Mattson and others (4). In the Multics system, it may be noted that the core and drum management lists form part of this ordering. A second experiment, conducted by Steven H. Webber, but reported by Saltzer (1), took note of this. Each possible partition of the total ordering into a fixed number of discrete, contiguous regions may be viewed as the simulation of an LRU-managed multilevel memory system of that many levels. For, truly, in such a system, the first n most recently used pages (positions 1 to n on the list) are in the highest level of the store, the next m most recently used (positions n+1 to n+m on the list) are in the next highest level of the store, etc. Even though the identity of the pages in each level (or each list position) changes constantly, the mapping just stated remains invariant. Hence, measurents ascertaining the relative frequencies of reference to given regions on the list may be used to predict the frequencies of reference to the levels of multilevel memory systems whose levels are divided the same way (the same number of pages in each level as in each list region). Saltzer's second experiment consisted of observing the LRU list for the Multics drum, a 4000 page device, and deriving predicted mean headways between exceptions for main memory sizes up to this size. 32 more data points were obtained by this technique, which also were well within the region predicted by the linear model (1).

The Multics virtual memory is fairly unique among computer systems, in that all references to online data are made through it. That is to say, there are never any explicit requests made for data from any online storage device by a user program. Extensive files, source and object programs, scratch and temporary storage areas, and the supervisor itself are all referenced via hardware-generated addresses in an identical manner. Programs are written with this knowledge in mind, and reference themselves and the Multics environment as a single large collection of directly addressable data. Experience with Multics, as well as current trends in the computer industry, suggest the viability and increasing future use of this technique.

Due to the uniqueness of the Multics virtual memory system, other reported research with the reference patterns of multilevel memory systems has concentrated on those based upon another definition of virtual memory, which we will distinguish as "virtual core". With this technique, one or two processors operating with a fixed amount of core memory are made to appear to their programs as separate processors with a separate core memory for each. Programs are written with the limitations of this "virtual" machine in mind, and ask for data to be transferred in and out of the virtual machine explicitly. Another significant difference between Multics-like and virtual core systems is that the latter usually do not allow sharing of parts of address spaces, whereas in Multics sharing is a significant effect.

The analysis of the Multics virtual memory reference patterns that this thesis proposes will attempt to ascertain all relevant headways (i.e., relative reference frequencies for a multilevel storage system) for any hypothetical boundaries between levels within the entire size of the Multics storage system, providing a direct measurement of virtual memory reference patterns out to the limit of current-day on-line storage size. For the reasons just discussed, it is clear that these measurements, and the models which can be derived from them, are immediately useful only for Multics-like memory systems. Systems of the "virtual core" type may tend to produce different reference patterns, due to the fact that in such an environment, the total amount of information required by programs is limited to the size of the "virtual core". Hence, when real core is extended to the size of virtual core, programs cease to make secondary storage references. In fact, measurements by Fine (5), Belady (6), and others made on "virtual core" type systems show great variance with the linear model.

Thus, the work proposed in this thesis is to a large extent based upon a degree of confidence in the increasing future use of Multics-like virtual memory systems.

Approach

The method by which I intend to experimentally extend the basis of observation of virtual memory reference pattern behavior is to construct the entire ordering imposed by the LRU algorithm on the pages of the Multics system. The essence of this method is to dynamically construct and maintain a list of all pages in the system, ordered by recency of use, constructed from an exhaustive trace of selected data movement within the storage hierarchy. By the maintenance of such a list, processor references may be analyzed with respect to what position in the list is occupied by the page in question. Each reference to a page moves it to the head of the list, as it is now the most recently used. For a multilevel memory system, each region of the list represents a given level in the storage hierarchy. What is more important, the arbitrary partitioning of this list allows the simulation of the effect of the given reference pattern on any hypothetical LRU-managed multilevel system. For instance, had we such a list, we could predict the relative access frequencies to each level of system consisting of a 300-page core, a 1000-page drum, and arbitrarily much disk. The first 300 entries on the list represent the pages in core, the next thousand the pages on drum (assuming that pages in core are not duplicated on drum), and the remaining pages those on disk. The assumptions about the non-duplication of pages and the strict adherence to to the LRU algorithm are not critical, and can be easily accounted for.

Thus, it is easy to simulate any multilevel storage system by observing the reorderings of a least-recent-use ordering of all pages in the system produced by an extensive reference trace. The linear paging model may be expressed very easily in terms of these list interactions. It states that the ratio of references to position n in the list or beyond to all references is inversely proportional to n.

The crux of this analysis is the production of this list, and observation of how the reference trace which produces it interacts with it. This analysis consists specifically of the keeping of a total for each list position of references to this position, and reordering the list as each reference is processed. In the Multics system, constraints of time and space forbid the construction of this list in real time. This approach would involve the maintenance of this list as the system runs, with references causing list searches and reorderings as they happen. There are three reasons why this is not a feasible technique. First, processor references cannot be traced in any meaningful way, other than simulated running of the entire system. The Multics hardware is capable of determining only if a given page has been used in a given interval. Only those references which cause missing page exceptions can be easily traced by the software. Second, the searching of this list requires extensive time. Experimentation suggests that every way of organizing this data structure to determine the position on this list of a given page requires either a linear search or a large amount of data movement. Several techniques have been suggested to reduce to amount of work required for most searches, but they either degenerate with time or require periodic updates, consuming excessive time. For a system which generates 100 page faults a second, it appears that there is no feasible way to search this list in real time. Third, the specific implementation of the Multics virtual memory control software is such that it requires that all relevant data and procedures be in core when they are needed. That is to say, the page-fault routines are non-recursive. The storage of this list in core at once would require approximately one-fifth of Multics core, which in addition to being an unacceptable degrading of a functional computer utility, would have a noticeable effect on the behavior of the data being measured.

These problems are all capable of solution. As the experiments of Saltzer produced data describing memory systems up to the size of the Multics drum, (the second level in the Multics storage hierarchy) only the portion of the list describing the disk (the third level) is of interest. References to any data not in core are trapped by the hardware, and easily traced by the software. The problems of time and space are solvable by simply accumulating enough data to be able to reconstruct the portion of the list corresponding to the disk. This data must include not only references, but enough data to reconstruct motion down the hierarchy, i.e., pages moving from higher levels to the disk. The creation and maintenance of the list representing the total ordering by recency of use by a program operating on this previously collected data allows the entire power of virtual memory to be used as a tool rather than a constraint, and presents no space problems. The problem of the time required to process this data becomes surmountable when it is not required to be contemporaneous with real time, in which case it is simply an example of a time-consuming list-processing program.

Anticipations

It is anticipated that the principal result of this thesis will be the determination of the range of applicability of Saltzer's linear paging model for LRU-managed Multics-like multilevel storage hierarchies. It is hoped that a more general model can be formulated, of which the linear model can be found to be part. The accumulated data amassed as both by-product and result of this experiment represents a history of data movement of a type and extent not previously documented on any system. The use of this data to model other than LRU algorithms and/or disk storage allocation and accessing algorithms can be foreseen. Another critical aspect of the problem at hand is to relate relative reference frequencies to mean time between references, and mean useful work between references (to a given level), measures of great practical importance. A related set of measurements being carried out as part of this work concerns inter-arrival time between page faults, which may well help to clarify some issues in this regard.

Status and Resources Required

As of this writing, (2/20/73) all of the software for obtaining the necessary trace of Multics disk traffic has been written, debugged, and installed in the Multics system. The programs for producing relative reference frequencies have also been developed and debugged. The entire experiment has been run on a trial basis on a small test configuration of Multics, wherein the total number of pages at issue was fewer than those considered by the drum experiment of Webber and Saltzer. The results have been quite encouraging, producing the previously observed "linear" reference pattern. This trial experiment served only to verify the validity of the technique, and not to derive any result.

At this point, the majority of work remaining to be done is the actual running of the experiment, which is awaiting a Multics administrative decision to begin, and the reduction and analysis of the derived data. This reduction, as stated before, will most likely require a large amount of computer time to carry out, which constitutes the only resource which has not yet been allocated for this thesis. The most feasible approach toward obtaining this resource appears to be the use of the Multics development machines (current and follow-on), small systems regularly deconfigured from Multics for system test purposes. The computer time required to perform statistical analyses on the reduced and raw data does not seem to be excessive, and is probably well within the budgetary limitations of the Computer

Systems Research group, which has provided the funds for most of the effort related to this project until now.

A reasonable prediction of the completion of this thesis would be completion by June, 1973, but certainly by September, 1973. The majority of the preparatory and developmental work for this thesis is already complete. The first run of this experiment will likely be within a week of this writing, and it is only the detailed analysis of this experiment and its result which remain to be done.

Bibliography

(1) Saltzer, J.H., "A Simple Linear Model of Demand Paging Performance," Project MAC memo M0131, November 3, 1972.

(2) Sekino, Akira, "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems," Ph.D. Thesis, MIT, Dept. Of Electrical Engineering, September, 1972.

(3) Bensoussan, A., Clingen, C.T., and Daley, R.C., "The Multics Virtual Memory," Communications of the ACM 15, 5 (May, 1972), pp. 308-318.

(4) Mattson, R.L., et al., "Evaluation Techniques for Storage Hierarchies", IBM Systems Journal 9, 2 (1970), pp. 78-117.

(5) Fine, Gerald H., et al., "Dynamic Program Behavior under Paging," ACM Proceedings of the 21st National Conference, P-66, Thompson Books, Washington, D.C., 1966, pp. 223-228.

(6) Belady, L.A., "A Study of Replacement Algorithms for a Virtual-storage Computer," IBM Systems Journal 5, 2 (1966), pp. 78-101.