Integration of batch and timesharing services

G. Houston* and R. Gillespie†

An attempt to add timesharing services to those already provided through a multiprogrammed batch operating system on the CDC 6600 computer provided lessons in both technical and strategic aspects of computing innovation. The technical problems associated with merging the conflicting functions are discussed in some detail. The problem of determining a strategy for properly introducing the new tool while not hindering the use of the existing ones is revealed through the choice of objectives for the project.

(Received September 1970)

1. Introduction and objectives

This paper provides a case study for:

1. An attempt to integrate batch and timesharing services on a single computer in an engineering environment.
2. An approach to the problem of building and attracting use of a new tool in a non-experimental environment.

Our belief in the utility of timesharing services was tempered by the difficulties we faced in building a reliable service and *then* interesting engineers in changing the reliable batch methods—which had an advantage of predictability—for what promised to be flow time improvements. However, the task facing us was not the design of yet another timesharing service; it was rather to usefully put an existing system into play in a production environment.

Thus our choice of technical goals for the system was heavily guided by such considerations as:

1. Could we avoid acquiring large amounts of equipment before the system was in production?
2. How could we avoid affecting the existing production service while undertaking the experimental development?
3. How could we provide the new services and yet take advantage of the massive investments in existing programs and procedures?
4. Which elements of timesharing services were most important to the problem space we were trying to span?
5. Which problems would benefit most from the new services?

The work described here was carried out at the Boeing Company, Commercial Airplane Division, Renton, Washington, where there was interest in finding economical ways to reduce the flow time for engineering problem analysis. Investigations showed that computer turnaround time was a major contributor to the overall time required to complete a single iteration of a major engineering design, typically a month or more for a process such as a wing analysis. Research experimentation with graphics and character-only displays had shown the flow time advantages of on-line operation, but the systems we had developed were not economically feasible for multiple terminal use.

In the latter half of 1967 the experiments were convincing enough to warrant the initiation of a development project with the overall objective of providing a timesharing service on the Control Data 6600 (the computer we were using for engineering analysis problems). The intent was to allow the service to support an economically viable collection of terminals (ranging from teletypes to high data rate alphanumeric and graphic displays). A timesharing system (SHARER) had been developed at the New York University Courant Institute by M. Harrison and J. Schwartz for an early version of the CDC 6600 operating system. While we considered other alternatives, including the development of a new operating system, the advantages of extending an existing system appeared to outweigh the time and costs of a new approach.

In order to meet the initial problems, an important goal was chosen: the service should support a mixture of batch and timesharing facilities that would provide reasonable transparency for the user, that is, convenient interchange between one class of service and the other. It was important to provide powerful facilities that would allow large and complex applications programs to run on-line without major change, since many of those programs represented 10 to 20 man-year investments.

2. Classes of timesharing services

Our developmental experience showed that some elusive points need to be made about timesharing services. The phrase 'timesharing' has become associated with two often-confused primary meanings:

1. A class of computing *services* available at a terminal.
2. A technical (or technological) *mechanism* of providing computing services.

The terminal user does not really care what mechanism is employed, as long as it provides the *service* he desires. The basic terminal services, regardless of mechanism, that we considered necessary to meet our objectives were:

(*a*) file editing;
(*b*) remote job entry, batch execution, and output retrieval;
(*c*) interactive execution of user programs.

Of these three, the last type of activity uses system resources in quite a different way than the first two. File editing and remote job entry activities typically require interactions characterised by short residency times, small memory space requirements, short CPU bursts, and minimal file activity, occurring at frequent intervals. Interactive execution is frequently the exact antithesis. In the design of a system providing the service, significant resource savings may be gained by providing a number of distinct mechanisms, each designed to handle a different class of terminal service. The original SHARER design did not really take advantage of this separation.

3. System development

In this Section we discuss the technical organisation of the timesharing subsystem and the most significant aspects of development.

3.1. *Basic operation*

Some technical aspects of the design of our system have already been discussed in Harrison and Schwartz (1967) and Harrison (1968). SHARER is a suboperating system which extends the batch multiprogramming capabilities of the manufacturer's standard 6600 SCOPE operating system to provide the additional functions required for timesharing. These functions

*The Boeing Company, Commercial Airplane Division, Renton, Washington, USA.
†Computer Center, The University of Washington, Seattle, Washington, USA.

Downloaded from https://academic.oup.com/comjnl/article/14/4/348/325080 by guest on 19 April 2024

include a command and communication facility for the various terminals; scheduling algorithms which allocate both central memory and central processor use for timesharing programs; and a considerable extension of the file management facilities available in the standard operating system. The system uses one SCOPE control point (a *control point* is similar to a partition or region in IBM terminology) for all timesharing activities. This one control point is multiprogrammed and timesliced amongst all on-line users. In most respects, SHARER appears to the operating system to be an ordinary batch job. Since there is no paging or segmentation hardware on the CDC 6600, programs are swapped in their entirety, using standard batch disk I/O functions. A single pair of base relocation and memory bound registers provides memory protection, but no ability to share programs or data.

Our basic machine configuration consisted of 131,000 (decimal) 60-bit words of central memory (roughly equivalent to 1 million eight-bit bytes), three fast disks of 70 million characters capacity each, up to 12 teletype lines, one central and 10 peripheral processors, and the usual batch I/O devices, including several remote printers and card readers.

3.2. *Conflicts and problems*

Our initial implementation of SHARER in mid-1968 provided a service to users which was characterised by three major problem areas:

1. Response times generally became inadequate when more than three to four users were logged in. Response was certainly judged to be inadequate if simple text-editing requests took more than five seconds; or if the compilation of a small FORTRAN program took two to three minutes.
2. The cost of using the terminal was very high: for example, \$50 to \$75 per terminal hour were typical, compared with \$15 to \$20 for competitive commercial services for doing the same job. This cost was inflated by two prime factors. Firstly, since response time constraints severely limited the number of active terminals, each terminal had to absorb a considerable fraction of the system overhead costs. Secondly, the timesharing subsystem dedicated for the complete period of its operation nearly half (51,000 words) of the central memory space (110,000 words) available for batch processing jobs.
3. Various user facilities were lacking in varying degrees. In particular, the response time problem led us to impose a limit of 500 lines (cards) on the size of a text-file that could be edited. The initial design required one copy of the editor for each user—each copy containing a text-buffer holding the entire file; and a particular user's copy was swapped for each editing transaction. Restricting editor file size served to limit the editor's impact on the system. A user's object program was additionally restricted to about 20,000 words.

It should be emphasised that in large part these problems were a consequence of our application of SHARER to our engineering workload in the environment of a heavily modified operating system. The problems do not necessarily reflect on the system's original design objectives at New York University, nor on the early version of the manufacturer's standard operating system we were using.

A diagnosis showed that the basic problems could be classified into four areas:

1. Central memory management.
2. Disk channel utilisation.
3. Peripheral processor contention; and
4. The editor and its interface to the system.

The original SHARER design had not adequately integrated the first three of these areas with the existing batch system: we shall outline briefly the redesign steps taken in each of these areas, and include a short description of some of the measurements taken to verify the validity of our corrective surgery.

SHARER'S utilisation of central memory was dramatically improved by allowing completely dynamic interchange of main storage (central memory) between timesharing activities and batch processing. Timesharing has absolute priority for memory space, and SHARER determines how much space is required to meet the on-line demand. If necessary, batch jobs (at other control points) are automatically swapped to and from disk. Improvements were also gradually introduced to the memory scheduling algorithm with the objective of giving fastest service to those on-line programs which use only a small amount of resources at each interaction, in particular for most activities supporting the file editing and remote job entry service classes. The major gains of our changes were to greatly reduce memory requirements to do a given task on-line, and also to improve response times for simple interactions.

Economic constraints forced us to use moving head disks for program swapping: Denning (1970) gives a good demonstration that such disks are virtually inadequate for swapping. We experimented with the allocation of files amongst the three independent disks available. By reserving one disk entirely for timesharing, we often recorded utilisation (disk channel activity) of around 70% (of elapsed wall clock time), clearly indicating much channel contention, much wasted memory-occupancy time, and consequently slow response times. We finally settled on allocating timesharing and batch files amongst all available disks, and thereby reduced activity on each disk channel to 25 to 35%. Response times were considerably improved, despite some batch competition for the disks.

To counter the impact of the long disk transfer times required to interchange two large (45,000 word) programs ($2\frac{1}{2}$ seconds in our operating system), the period such programs were allowed to remain in memory was increased from three to four seconds to around 30. This significantly reduced disk channel usage and total memory-occupancy times required for the interactive execution class of service.

Peripheral processor contention is a problem unique to the CDC 6600 architecture, and in our case was caused by both poor systems programming and a failure to effectively manage processors dedicated to particular I/O functions. Tedious corrective efforts in these two areas effectively removed the frequent system lockups and poor response times encountered in the early stages. SHARER itself seriously aggravated the problem by using one entire processor for teletype communication: a situation we were forced to accept.

Our last major problem was that of editor interface. Our editor redesign goal was to allow editing of text files of up to 10,000 lines, whilst minimising system impact. We considered in some detail the merits of a re-entrant, serially reusable, or multi-user editor, and dismissed the practicality of implementing them because of the system complexity involved and manpower required. We thus remained committed to swapping about 9,000 words of editor for each user. Editing is carried out by software-simulated paging of a virtual text-buffer. Additionally, an input stacking method was developed which eliminated about half of previous editor swapping; an efficient terminal output buffering scheme added; and a range of additional editor capabilities were implemented, including some powerful context editing functions. In short, we took some pains to provide special mechanisms to handle the class of service typified by editing.

3.3. *High data rate devices*

Many timesharing systems are able to support a large number of terminals by virtue of the very slow input/output speed of teletypes and similar devices—a maximum rate of 600 to 900 characters per minute. As noted earlier, we were particularly interested in also supporting some alphanumeric displays and a large graphics display. Some measurements showed that the alphanumeric displays produce mean data rates of about 3,000

Downloaded from https://academic.oup.com/comjnl/article/14/4/348/325080 by guest on 19 April 2024

characters per minute, with peak data rates of up to 20,000 characters per minute; i.e., in terms of total characters transferred, each display is equivalent to between five and 30 teletypes. An example of the application problems which lead to these requirements is that of airline route allocation which requires the examination of a relatively large amount of output on each of several iterations in order to determine optimum routing. We anticipated the same rates for the large graphics display, and preliminary work (described in Gillespie, 1970) confirmed this, but economic conditions prevented us carrying the study through to production use.

We initially found some difficulty in integrating these devices into a system designed for teletypes. The application programs driving these displays used a considerable fraction of available resources, and substantially impacted response times for teletype users. A striking improvement was made by taking the obvious course of considerably reducing the display application program size since the programs require considerably more activity to support the displays than teletypes.

3.4. *Measurement and performance analyses*

It has been amply demonstrated (for example, Campbell *et al.* 1968) that measuring techniques should be considered at an early stage of system development. In our initial implementation, SHARER was not equipped with any measuring tools. We added various simple software measuring devices on a piecemeal basis. Some simple counters verified the value of the corrective actions described earlier—for example, the ratio of swapping I/O to user I/O activity gradually moved from about 1:1 to 1:4, presumably indicating more efficient usage of the disks.

A persistent management requirement was for an adequate measurement of system response time, to determine whether the system was overloaded or had capacity to spare. To this end, we developed a statistic, Equivalent Response Time (ERT). ERT is the mean execution time of a series of 10 test programs normalised to the number of logged in users at the time the tests are run. The tests cover a range of activities, from the (trivial) rewinding of a disk file to the (rather non-trivial) manipulation of several hundred records. Thus they attempt to recognise that response time is a function of the particular operation. ERT must be interpreted with care, since it is highly empirical, and can vary, for example, from six to 40 from one day to the next without any other observable change in timesharing activity. We can attribute this to the extent that varying batch workloads affect timesharing performance. Evidently, we failed to effectively decouple the interaction of one class of service on the other.

In the hope of developing more concrete concepts than ERT, we surveyed some of the literature. In our experience, the major shortcoming of almost all studies, e.g. Stimler (1969), is that they assume that system overhead problems are so small that they can be ignored. Our early experience was that system performance was almost completely limited by the disk I/O swapping overhead, and that even after our development efforts, this factor is still sufficiently large to invalidate most of the analyses available. We conducted both practical and theoretical experiments with an extended (bulk) core storage device, and showed that timesharing capacity could be increased by 100 to 200%, depending on the sophistication of utilisation of the device. Baskett *et al.* (1970) contains an interesting analysis of a similar situation.

3.5. *Results*

These various problems were diagnosed and corrected by mid-1970. As a result, satisfactory timesharing performance has been increased from two or three to the limit of the 12 terminals. Editor requests are handled within two or three seconds in about 90 to 95% of occurrences, and small programs can be compiled and executed in 20 to 30 seconds. The maximum timesharing program size was raised to 45,000 words, and the performance figures just quoted are maintained with several users running programs up to the maximum size. Hourly terminal costs were reduced to a more competitive $20 to $30 per hour, while user facilities are for the most part more powerful than available competition.

4. Application restructuring

As noted, a major objective of our development was to capitalise on the massive investment (e.g., several thousand man-years) of development of existing batch applications. We thus paid some attention to the problems of shaping existing applications to interactive usage. Thus it turned out that all of our significant interactive applications come from modifications of batch-processing programs. The applications were typically those where the user (not usually a computer professional) makes several runs to converge on a solution. Each run is provided with a set of input data; the input data is modified in each successive run based on the results of the previous run, based on the user's judgement of the situation. The applications are very much of the man-in-the-loop kind; they are often problems for which so far adequate mathematical models have not been developed, and the user must play with the inputs based on his skill and knowledge to arrive at the right results.

Given this type of background, the applications require some restructuring; basically it is more or less possible to leave the computational heart of the application program very much as it was in batch mode, but to remodel the input-output end to communicate with a display or teletype instead of the card reader and line printer. It is necessary to develop a simple but flexible means of editing the input data, and experience has shown that the most effective way to do this is to develop special-purpose methods for each application; while it is possible to provide general-purpose editing software for images of decks of cards, the process is slow, cumbersome, and quite difficult, compared with editing software tailored to the application and user. The use of a FORTRAN interface for our alphanumeric displays makes it possible for the application-programmer to provide the editing functions himself directly. Typically, it appeared to take two or three man-months to do a good job of the interactive interface for a moderately complex application.

A particularly useful way of using a display is the 'menu' technique: much used with interactive graphic systems, it appears to be equally applicable to character-only displays. For the non-computer-professional, it is a very direct way to communicate with the machine, and it is an important technique in breaking the communication barrier. Our experience has shown that it it almost impossible to teach some types of non-computer professional to use the teletype, though they can readily adapt to the display.

From a system standpoint, the result of the restructuring is that when the application program executes, it exhibits two distinct types of behaviour during execution. One is where the user is editing his input, or scanning his output: there is very small usage of CPU; a certain amount of disk-file I/O, and sometimes heavy demands for new displays. The other type of activity is when the user has made all the modifications he wants to his data: he then pushes the 'compute' button, and then follows a period of considerable CPU and I/O activity; users are usually prepared to wait several minutes for its completion.

5. Conclusions

These observations are those we feel important.

1. The psychology of introduction of a new service is crucial. If you wish to attract users to a new tool or service before

Downloaded from https://academic.oup.com/comjnl/article/14/4/348/325080 by guest on 19 April 2024

it is clear that there will be a long-term commitment, both inducements and illusion are needed—inducement through 'free' time and ability to use the system on useful work; illusion through good user documents and manuals which help dispel the chancy nature of any such development. The development group must change gears at some point in order to devote at least 50% of its time to users in order to understand their problems. Ways to use new tools are often obvious to the tool builder but not the mechanic (and later on vice versa).

2. Conflict between the objectives of maintaining production reliability and avoidance of new equipment hindered development. The treatment of the timesharing service as just another job could not be sustained without compromising performance. Measurement of interactive system performance was far more complex than expected and some performance measures need to be designed into the operating system.
3. Observation of service use points to more investigation of editing and file development as being more important than full interaction.
4. The application restructuring strategy worked, and thus we successfully built upon the massive investment in batch program development.

References

Baskett, F., Browne, J. C., and Raike, W. M. (1970). The management of a multi-level non-paged memory system, AFIPS Proc. Spring Joint Computer Conference, p. 459.

Campbell, D. J., and Heffner, W. J. (1968). Measurement and analysis of large operating systems during system development, AFIPS Proc. Fall Joint Computer Conference, p. 903.

Denning, P. J. (1970). Virtual memory, *Computing Surveys*, Vol. 2, No. 3, p. 153.

Gillespie, R. G. (1970). The anatomy of an interactive graphics display project: an engineering tool, Graphics 70, Brunel University.

Harrison, M. C. (1968). Implementation of the SHARER2 timesharing system for the CDC 6600, *Comm. ACM*, Vol. 11, p. 845 (December).

Harrison, M. C., and Schwartz, J. T. (1967). SHARER, a timesharing system for the CDC 6600, *Comm. ACM*, Vol. 10, p. 659 (October).

Stimler, S. (1969). Some criteria for time-sharing performance, *Comm. ACM*, Vol. 2, p. 47 (January).

Correspondence

To the Editor
The Computer Journal

Sir,

The recent article by McConalogue (1970) describes a method for curve drawing which appears to combine accuracy and simplicity, especially when the tangents are already known. When they are unknown, McConalogue employs a 'complementary algorithm', which expresses the x and y co-ordinates at P_{k-1}, P_k and P_{k+1} as separate Lagrange quadratics in a parameter equal to the chord length $D_k = \{(\Delta x_k)^2 + (\Delta y_k)^2\}^{\frac{1}{2}}$. This produces tangents which are invariant under axis rotation.

An alternative procedure, with the same invariance, utilises the circular arc defined by the same three consecutive points. The co-ordinates $(\bar{x}\bar{y})$ of the centre of this circle are found from

$$D\bar{x} = (x_{k-1} + x_k)\,\Delta x_{k-1}\,\Delta y_k - (x_k + x_{k+1})\,\Delta x_k\,\Delta y_{k-1} - (\Delta y_{k-1} + \Delta y_k)\,\Delta y_{k-1}\,\Delta y_k$$

$$D\bar{y} = (y_k + y_{k+1})\,\Delta x_{k-1}\,\Delta y_k - (y_{k-1} + y_k)\,\Delta x_k\,\Delta y_{k-1} + (\Delta x_{k-1} + \Delta x_k)\,\Delta x_{k-1}\,\Delta x_k$$

where

$$D = 2(\Delta x_{k-1}\,\Delta y_k - \Delta x_k\,\Delta y_{k-1})$$

and

$$\Delta x_k = x_{k+1} - x_k,\ \Delta y_k = y_{k+1} - y_k\,.$$

Thus the angle between the x-axis and the tangent at an arbitrary point (ξ, η) on the circular arc is $\theta = \tan^{-1}\{-(\bar{x} - \xi)/(\bar{y} - \eta)\}$; hence

$$N\sin\theta = \Delta y_{k-1}(\Delta x_k^2 + \Delta y_k^2) + \Delta y_k(\Delta x_{k-1}^2 + \Delta y_{k-1}^2) + D(\xi - x_k)$$

$$N\cos\theta = \Delta x_{k-1}(\Delta x_k^2 + \Delta y_k^2) + \Delta x_k(\Delta x_{k-1}^2 + \Delta y_{k-1}^2) - D(\eta - y_k)$$

where N is a normalising factor.

Setting $\xi = x_k$, $\eta = y_k$, the tangent at P_k is found to be precisely the same as that derived by McConalogue. This somewhat surprising result shows that McConalogue's algorithm is exact for points which lie on circular arcs.

However, if P_{k-1} is the first point on an open curve, the slope there must be estimated from the above formulae with $\xi = x_{k-1}, \eta = y_{k-1}$; and if P_{k+1} is the last point, $\xi = x_{k+1}, \eta = y_{k+1}$. In general, the slope at a point P is most simply calculated from

$$N\sin\theta = \alpha\Delta y_{k-1} + \beta\Delta y_k - \gamma\Delta x_{k-1} + \delta\Delta x_k$$
$$N\cos\theta = \alpha\Delta x_{k-1} + \beta\Delta x_k + \gamma\Delta y_{k-1} - \delta\Delta y_k$$

where $\alpha = (\Delta x_k)^2 + (\Delta y_k)^2$, $\beta = (\Delta x_{k-1})^2 + (\Delta y_{k-1})^2$ in all cases, and $\gamma = D$, $\delta = 0$ at P_{k-1} (if this is a starting point), $\gamma = 0$, $\delta = 0$ at P_k, and $\gamma = 0$, $\delta = D$ at P_{k+1} (an end point). Incidentally, the radius of curvature at P_k is N/D.

These results for the end points of open curves differ from those found by McConalogue, which may be written

$$N\sin\theta = \alpha'\Delta y_{k-1} + \beta'\Delta y_k$$
$$N\cos\theta = \alpha'\Delta x_{k-1} + \beta'\Delta x_k$$

where

$$\begin{array}{lll} \alpha' = \alpha + 2\sqrt{\alpha\beta}, & \beta' = -\beta & \text{at } P_{k-1}, \\ \alpha' = \alpha, & \beta' = \beta & \text{at } P_k, \text{ and} \\ \alpha' = -\alpha, & \beta' = \beta + 2\sqrt{\alpha\beta} & \text{at } P_{k+1}. \end{array}$$

It is suggested that the alternative formulation is preferable in that it is exact for circular arcs, does not require the use of square roots in finding the direction ratios, and avoids the somewhat arbitrary selection of cord length as a parameter.

Yours faithfully,
M. J. Ecclestone

Computer Branch
CEGB North Eastern Region
St. Mary's Road
Leeds 7

1 January 1971

Reference

McConalogue, D. J. (1970). A quasi-intrinsic scheme for passing a smooth curve through a discrete set of points, *The Computer Journal*, Vol. 13, No. 4, pp. 392-396.

Erratum

In the paper 'The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration' by M. Clint and A. Jennings (this *Journal*, Vol. 13, No. 1, p. 76) there was an error in Table 1. The element of the fifth row and twelfth column should read 997·30 and not 977·30. We are grateful to Terry G. Seaks, Department of Economics, Duke University, Durham, North Carolina 27706, USA for pointing out the error.

Downloaded from https://academic.oup.com/comjnl/article/14/4/348/325080 by guest on 19 April 2024