A Practical Method of Comparing Computer Bureau Costs

By K. Reed*

This paper presents a simple technique for comparing the computer usage accounting techniques on computer bureaux. The paper restricts itself to considering charges for cpu and I/O time. The technique allows different charging algorithms to be compared with varying precision, and allows assessment of the alternatives of hiring a complete installation and paying for each job according to the resources (cpu time, I/O time) used.

KEYWORDS AND PHRASES: Computing usage, accounting, cost comparison, charging algorithms.
CR CATEGORIES: 2.41, 2.44, 4.39, 4.6.

1. INTRODUCTION

The reduction of computing costs is a problem to which management does not seem to address itself very often. Many organisations live with their EDP departments, whether large or small, without adopting cost-control measures. A number of quite large organisations do not charge in-house users for EDP services; the computer and staff are there and if resources are available then the job required by department X will be done, otherwise it will not.

An entirely different situation will exist if the organisation is using a computer bureau. Most bureaux will provide costing information on a job-by-job basis and hard cash will be paid by the company at regular intervals. The costs become visible to management in a rather direct and forceful way.

The methods presented here allow bureau charging algorithms to be compared and the results are applicable to all computing situations.

The work reported was performed as part of the bureau selection procedure adopted by L.M. Ericsson Pty. Ltd. during 1971 when seeking computer time for the development of its AKE 131 stored Program Controlled Telephone Exchange project for the OTC(A) installation in Sydney. Some technique for comparing different computer charging schemes was needed since it was expected that Grosch's Law, (Adams (1962)), may have been applied to achieve cost savings. The packages concerned ran on the IBM 360/370 range and we had, therefore, machines of varying power to choose from. There was also the possibility that Grosch's Law may not apply, i.e. that a smaller computer may prove cheaper to use.

2. CHARGING METHODS AND COMPUTER USAGE

2.1 Charging Philosophies

Methods of charging for computer utilisation are extremely important to installation management. Their effects are critical in the highly competitive bureau market in which real profits must be made.

Charging algorithms must therefore ensure that the vendor obtains an adequate return on his investment and that the user is charged only for resources actually used.

The charging methods are also important to the computer bureau user who will want to limit his costs as much as possible. The literature is not short of papers on the philosophy of cost recovery (for this is what a charging method attempts), two recent papers, being Borovitz and Ein-Dor (1977), and Nolan (1977). There are also interesting attempts by Knight (1966, 1968), Arbuckle (1966) and Solomon (1966) to develop useful techniques for comparing computer performance.

The problem of bench marking is dealt with usefully by Curnow and Wichman (1977) and Walters (1977), while Legg (1972) contains a survey of charging algorithms. Equipment utilisation and procurement receives a deeper treatment in Drummond (1973) and Svobdova (1976), but comparing the charging techniques used by computer installations from the view-point of the effect upon the cost of running the same job on different installations has not been treated.

2.2 Charging Schemes

We discovered what we believed were four basic methods of charging. Only computer equipment charges are considered in what follows. Legg (1972) describes a number of charging algorithms, all of which fit the following categories:

2.2.1 Resources Utilisation Charging

Resource utilisation charging (RUC) attempts to ensure that the user is charged for actual resources used.

Such schemes, in some measure or other, charge the user for cpu time, core occupied and I/O transfers performed. The result is that many vendors make fixed I/O transfer time charges which approach the average access time for the most common device.

2.2.2 Process Time Charging

The RUC methods have some problems from the vendor's point of view since they may allow the user to optimise his expenditure to a very high degree. It may also make charging a little difficult, and, as will be seen, creates difficulties when comparing various types of RUC offers.

Process time charging simplifies the charging situation enormously. The user is charged only for cpu time and a measure of I/O time used – independent of core – at a fixed rate. The process time charging (PTC) method has the element of a gamble, and can be made to appear particularly attractive when benchmarking. Costs are easier to measure and are more predictable. This will tend to influence the inexperienced purchaser.

PTC charging consists of calculating the sum of cpu time and I/O times and multiplying by a suitable unit cost.

**Monash University Computer Centre. Manuscript received 25th December 1975 Revised version received 12th July 1977.*

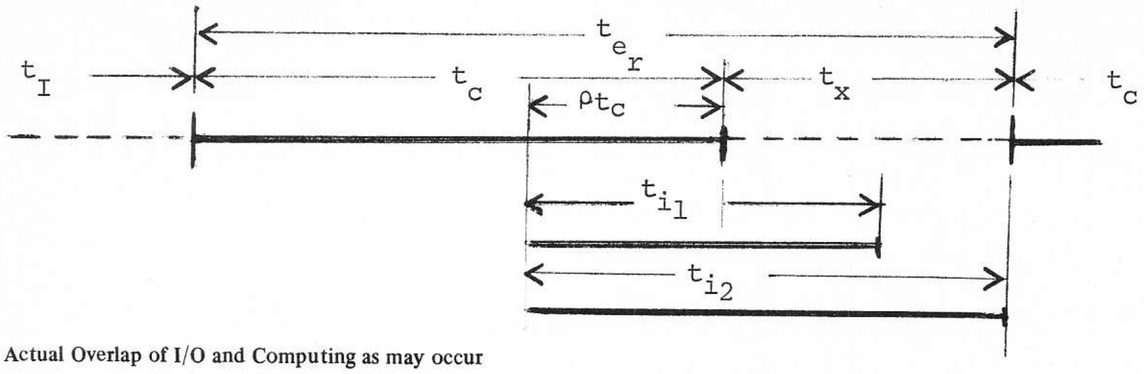

Figure 1 Actual Overlap of I/O and Computing as may occur

2.2.3 Elapsed Time Charging

Elapsed time charging (ETC) may apply in some circumstances. Charges are made using elapsed time as shown in the computer log with allowance for operator errors etc.

One might think that this method has little to recommend it, but the RUC and PTC approaches rob the user of savings due to overlapped I/O operations. This is because they involve the sum of cpu time and the notional time for *each* transaction. Good modern operating systems will provide automatic overlapping of I/O in applications involving blocked, buffered files, and some overlapping of I/O produced by swapping different jobs. This means that job times calculated for accounting purposes may exceed elapsed times when the job is run on its own. Figures 1 and 2 below show the effect upon real and "notional" waiting times.

In Figure 1, the idle time is:

$$t_I = \max(t_{i_1}, t_{i_2}) - \rho t_c$$

and real elapsed time is:

$$t_{e_r} = (1 - \rho)t_c + \max(t_{i_1}, t_{i_2}).$$

In Figure 2 however, the notional elapsed time is:

$$t_{e_n} = t_c + \bar{t}_{i_1} + \bar{t}_{i_2}$$

where $\bar{t}_i$ is the time measured for the I/O operations. The notional elapsed time t_{e_n} can exceed t_{e_r}, the real elapsed time for suitable values of the various I/O times.

2.2.4 Hiring A Complete Installation

The fourth possibility is to hire a complete installation. Many bureaux permit the user to do this provided that some minimum period of time is involved. The user can then arrange his own scheduling and, one would hope, reduce his costs because of the inherent difference between the elapsed time for the job mix and the sum of the individual elapsed times obtained by any of the previous methods. It should be noted that the previously presented accounting techniques are based upon a calculation of some form of elapsed time as mentioned in section 2.2.3.

2.3 Some Comments on Overheads

All jobs run on a computer incur initialisation overheads which are difficult to measure. Many resource charging systems add a fixed amount to the accounts, in money or resources, for each job (and job-step where appropriate). There are systems in use which charge the step overhead irrespective of whether or not a step is actually run.

It should be noted that jobs charged under ETC have the actual step overhead in elapsed time added to their accounts.

3. CHARGING FOR OTHER ITEMS

The discussion to date has been restricted to computer system charges, that is, cpu time, I/O time, core usage etc.

In practice, the user will be confronted with a variety of additional charges and will be asked to pay for all or some of the items in Table 1. The author has found the following items particularly pernicious and would

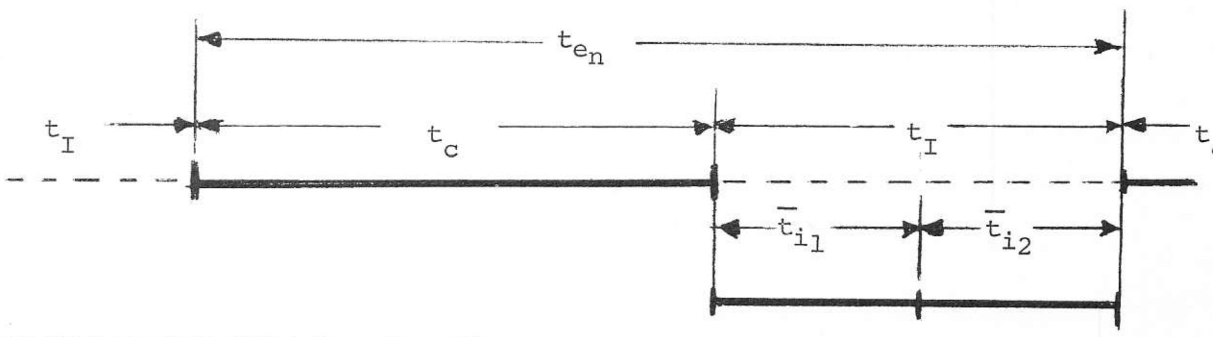

Figure 2 Effect of Counting I/O Transfers on Elapsed Time

recommend avoiding contracts which place overdue emphasis on them:

- unit record charges, particularly cards read and lines printed, if program development work is involved,
- on-line disk storage under similar conditions,
- stationery changes where these are likely to occur frequently,
- device mounting charges.

TABLE 1

ITEM	METHODS OF CHARGING
Cards read/punched	Per thousand cards read, with reduction for bulk.
Lines printed	As above.
Forms changed	Charge for change based on whether or not standard stationery.
Devices mounted	Charge for mounting each device used.
Drives Used	May be charged by occupancy i.e. total number of drive-hours.
Device Rental	Monthly rental for devices e.g. disks.
Device Storage	Monthly charge for storage of disks, tapes, etc.
On-line Storage	Charge made for permanently mounted disk-space, or space used while on-line to a system.

4. DETAILS OF CHARGING METHODS

Without wishing to introduce a lot of mathematics, we must however, examine statements of charging techniques and formulate them into simple equations.

4.1 Resource Usage Charges

Let us consider the complete type of RUC charging system which was stated in the form of a table for two computers.

TABLE 2

Item		Computer	
		M_1	M_2
1)	cpu time	\$225.00 per hr.	\$645.00 per hr.
2)	Processing time – plus for each 2K bytes of storage in excess of 80K	\$120.00 per hr. \$3.00 per hr.	\$165.00 per hr. \$3.92 per hr.
3)	Number of I/O transfers	\$1.70 per 1000	\$2.90 per 1000

These figures are actually taken from a bureau offering services on two different computers in the same range where M_1 was less powerful than M_2. The figures are now some years old and are not applicable today, but are indicative of the type of situation a purchaser can encounter, even in one bureau.

The following accounting records are kept by the operating system – cpu time, I/O transfers, job core usage.

The quantity described as "processtime" was defined to be the sum of cpu time and 25 milliseconds for each I/O transfer. Therefore,

$$t_p = t_c + .025\ \ I \tag{1}$$

where

t_p is the process time for a job which used,
t_c seconds of cpu time
I I/O transfers performed.

If we let

C_c be the cpu time cost in dollars/sec.
C_p be the process time cost in dollars/sec.
C_k be the storage cost in dollars/2 kilobytes/sec. of process time,

and C_I be the fixed component cost of each I/O transfer, then the accounting charge C for a given job is:

$$C = C_c t_c + C_p\,(t_c + .025\ \ I) + C_k\,(t_c + .025\ \ I)\,(\tfrac{K-80}{2}) + C_I I \tag{2}$$

which reduces to

$$C = [C_c + C_p + C_k\,(\tfrac{K-80}{2})]\,t_c + [.025\,C_p + .025\,C_k\,(\tfrac{K-80}{2}) + C_I]\,I \tag{3}$$

where K is the nominal number of kilobytes used by the job in question. If the actual job size is k kilobytes then:

$$K = \begin{cases} k & ,k \geqslant 80 \\ 80 & ,k < 80 \end{cases} \tag{3a}$$

so that the smallest chargeable amount of memory is 80 kilobytes.

Substituting the constants from Table 2 we get:

$$C_{M_1} = [.0624 + .0000417\ K]\ t_c + [.00170 + .0000104\ K]\ I \tag{4}$$

and

$$C_{M_2} = [.182 + .0000544\ K]\ t_c + [.00296 + .0000136\ K]\ I \tag{5}$$

where $K \geqslant 80$ as defined in (3a).

Deciding what type of jobs to run on which machine is quite straight forward – we note that the cost C of a job taking t seconds and having used I I/O transfers has in this case the form:

$$C = Ht + BI$$

where the constants H and B are obtained from the costing data provided, and are *both* functions of core used.

4.1.1 Ratio of cpu speeds

The next point that we notice is that if the job was actually run on both machines, we would find that the metered cpu time used on machine M_1, might be t_1 and that on M_2 might be t_2. The term "metered" cpu time is stressed.

However, the number of I/O transfers (I) is going to be the same, since we have the same machine family, operating system etc. The cost of the job on M_1 is:

$$C_1 = H_1 t_1 + B_1 I \tag{6}$$

and on M_2:

$$C_2 = H_2 t_2 + B_2 I \tag{7}$$

This last equation can be altered, if we introduce the ratio $R_{21} = \frac{t_2}{t_1}$ then we have:

$$C_2 = H_2 R_{21} t_1 + B_2 I \qquad (8)$$

R_{21} is the ratio of cpu time on M_2 to that on M_1.

So the cost of the job on M_2 can be established in terms of measurements made on M_1 once a reasonable value of R_{21} has been agreed upon.

The equation of form (7), or rather a *pair* of such equations, are the key to the approach which we will develop further.

4.2 PTC

Typically, for a given machine (say in fact M_1) we might find another bureau offering to charge as follows:

TABLE 3

Item	M_1
1) Process time	\$220.00 per hr.

where process time is calculated as

cpu time + 45 ms per I/O transfer

$$t_p = t_c + .045\ I \qquad (9)$$

Adopting the same approach as that used earlier, we find that a job taking t seconds of cpu time (as measured) and using I I/O transfers, would cost:

$$C = .0611\ t + .00275\ I \qquad (10)$$

which clearly can be written in the same form as equation (6), i.e.

$$C_3 = H_c t + B_3 I \qquad (11)$$

4.3 Elapsed Time Charges

The elapsed time charges cannot be related so readily to the above equations and costing methods for a number of reasons among which are:

1) The user's elapsed time gets the full benefit of any overlapping amongst I/O operations and between I/O operations and computing in a stand-alone environment.
2) Elapsed times will show the full effects of the variability of I/O operation times relating to file positioning, that is, seek times.

We can however, relate elapsed times for stand-alone jobs to the two charging methods above as we will show later – although the accuracy of the comparison is weakened.

4.4 Other RUC Formula

Most of the charging methods that the author has encountered can be reduced to the:

$$C = Ht + BI \qquad (12)$$

(where t is metered cpu time and I the number of I/O operations) form provided that each individual I/O operation, not the actual channel time is measured.

To show this, we will consider two other charging methods.

4.4.1 I/O Scaled Charges

One bureau charges in the following manner:

Cost for system second = K_1 dollars

System second is: 1 second of cpu time,

or 10 I/O transfers to disk multiplied by the fraction of core used.

System seconds are also obtained by:

Channel time on magnetic tape multiplied by fraction of core used.

Using the following nomenclature:

t	= metered cpu time
K_1	= cost of system second in dollars
I_d	= number of I/O transfers to disc
t_I	= nominal I/O time per disk transfer
T_m	= channel time for magnetic tape
f	= ratio of core used to core available

then we have:

$$C = K_1\ [t + f(t_I + I_d + T_m)] \qquad (13)$$

$$C = K_1 t + K_1 f t_I I_d + K_1 f T_m \qquad (14)$$

which admittedly is not quite in the required form!

This particular system has real possibilities for cost optimisation since the "f" used varies as the amount of core varies during the job – and the system seconds due to I/O alter therefore during the job. Strictly then, if T_{m_j} seconds of channel time are used and I_{d_j} disk I/O transfers are made when the core fraction is f_j the job cost will be:

$$C = K_1\ [t + t_I \sum_j f_j I_{d_j} + f_j\ T_{mj}] \qquad (15)$$

$$I = K_1\ [t + \sum_j f_j\ (t_I I_{d_j} + T_{mj})] \qquad (16)$$

Obviously, a program which can confine its I/O activity to periods when f_j is small will have a lower cost than a similar program which cannot.

4.4.2 I/O Rates Charged Differently

Another bureau charges at separate rates for different I/O devices, differentiating between those used for unit record and other I/O. This will give us an equation of the kind

$$C = C_4 t + H_{41} I_f + H_{42} I_s \qquad (17)$$

where I_f is the number of "fast" device I/O transfers, and I_s the number of unit record I/O transfers.

5. COST COMPARISON TECHNIQUE

5.1 General

The approach used, is to consider only those I/O transactions *not* due to unit record activity – these can be estimated from the blocking factors of the input/output queues which are generally available, and subtracted from the total I/O transfer figure.

Our equation is of the form:

$$C = Ht + BI \tag{18}$$

where t is actual *recorded* cpu time, and I *recorded* I/O transactions.

We can consider that each cpu second had a number of I/O transfers associated with it – dividing by the cpu time, we obtain:

$$c = H + Bi \tag{19}$$

the cost of *one* second of cpu time during which i I/O transfers were performed.

Alternatively, the number of I/O transfers per cpu second determines the cost of the job. This function has a graph which is a straight line and involves only two constants which will be fixed for the job in question.

Equation (18) has two independent variables – we have just dropped one of them.

Returning to the equation pair (6), (7)

$$C_1 = H_1 t_1 + B_1 I \tag{20}$$

$$C_2 = H_2 R_{21} t_1 + B_2 I \tag{21}$$

where $R_{21} = \frac{t_2}{t_1}$

we obtain, on normalisation

$$c_1 = H_1 + B_1 i_1 \tag{22}$$

and

$$c_2 = H_2 R_{21} + B_2 i_1 \tag{23}$$

and it is easy to see when either machine is cheapest, without even running a job! (If the cpu time clocks are accurate!)

5.2 Details of Comparison

We wish to know the conditions under which machine M_1 is cheaper to use than machine M_2, even though M_2 is the more powerful of the two.

We require:

$$c_1 < c_2 \tag{24}$$

i.e.

$$H_1 + B_1 i_1 < H_2 R_{21} + B_2 i_1 \tag{25}$$

which can be re-arranged to give

$$H_2 \left(\frac{H_1}{H_2} - R_{21}\right) < i_1 \left(1 - \frac{B_1}{B_2}\right) B_2 \tag{26}$$

We can make the following assumptions, since M_2 is the more expensive machine, which means that the vendor attempts to charge more for resources. These are:

$$H_1 < H_2 \qquad A_1: \tag{27}$$

reflecting a more powerful cpu, and

$$B_1 < B_2 \qquad A_2: \tag{28}$$

reflecting relatively higher costs of I/O activity.

We note that condition A_2 means that:

$$i_1 \left(1 - \frac{B_1}{B_2}\right) B_2 > 0 \tag{29}$$

So that, provided $R_{21} > \frac{H_1}{H_2}$ it will always be cheaper to use M_1 since:

$$H_2 \left(\frac{H_1}{H_2} - R_{21}\right) < 0 \text{ if } R_{21} > \frac{H_1}{H_2} \tag{30}$$

Similarly, if $B_1 > B_2$ which can happen on installations with very high speed drums (e.g. 5 mS drums on some local utilities), we have the possibility that:

$$i \left(1 - \frac{B_1}{B_2}\right) B_2 < 0 \tag{31}$$

so that M_2 may always be cheaper than M_1.

Taking the particular example chose, and assuming

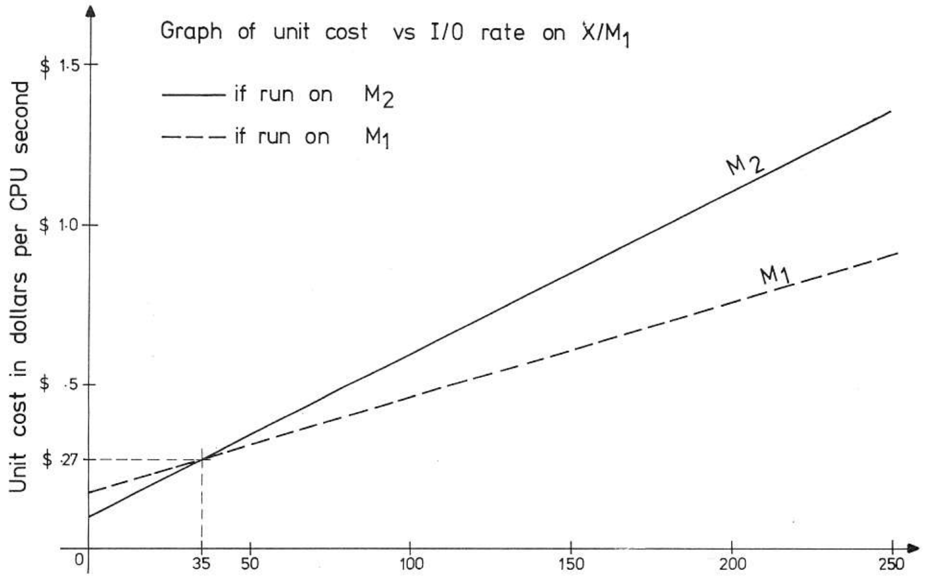

Figure 3 I/O Rate in transfers per CPU second M_1

that 220K bytes are required, and that R_{21} is .286 (i.e. M_2 is 3.5 times faster than M_1), and doing the arithmetic we see that provided $i > 35$ I/O transfers per cpu second, then machine M_1 will be cheaper to use than M_2!

This is shown in the graphs of the unit costs for each machine (Ref. Fig. 3).

Note that we can introduce a ratio, or increment so that we can require M_1 to be cheaper by a certain amount, e.g.

$$\frac{C_1}{C_2} < D \tag{32}$$

where D can be greater or less than 1, which alters our basic equation to:

$$i > \frac{H_2\left(\frac{H_1}{H_2} - DR_{21}\right)}{B_2\left(D - \frac{B_1}{B_2}\right)} \tag{33}$$

So that knowing the constants H_1, B_1, H_2 and B_2 we can plot the straight lines corresponding to the values of i and R_{21} for which (30) holds for fixed values of D (or vice versa).

5.3 Relationship with Elapsed Times

5.3.1 Single Job Elapsed Times

Let us assume that each I/O transfer uses an average of $\hat{t}_i$ seconds perhaps this is an actual constant in many cases.

For a single job taking T seconds we have:

$$T = t_c + \Sigma \text{ I/O time} \tag{34}$$

If I I/O transfers occurred at an average of $\hat{t}_i$ seconds each, we have:

$$T = t_c + I\hat{t}_i \tag{35}$$

where t_c seconds of cpu time are charged to the job.

We can define the ratio of cpu time to total time, the "compute boundedness", to be:

$$\beta = \frac{t_c}{T} \tag{36}$$

$$\beta = \frac{t_c}{t_c + I\hat{t}_i} \tag{37}$$

We can convert this back to a form involving i, the number of I/O transfers per cpu second:

$$\beta = \frac{1}{1 + i\hat{t}_i} \tag{38}$$

Relating this back to our case of supplier X, charging 25 mS per I/O transfer, we find that jobs for which $\beta < .533$ should be run on the slower machine.

If we assume that this figure is realistic, allowing for overlap of I/O, then we find that the machine M_1 will be cheaper for all but real number crunching jobs, and certainly for any utility type functions!

These remarks apply, of course, to the characteristics of jobs as they run on machine M_1.

It may be necessary to introduce a scaling factor in (38) to allow for the situation where a fixed figure $\hat{t}_i$ does not correspond to elapsed time.

However, even assuming that a suitable constant cannot be found, we can relate an equation of our standard form $C = Ht + BI$ to elapsed time charges so that the conditions under which equality may be achieved can be examined.

Assume that the unit charge for ETC is C_L per unit time. A job taking T units costs therefore:

$$C = C_L T \tag{39}$$

Define actual degree of computer boundedness to be:

$$\beta_a = \frac{\text{cpu time}}{\text{Elapsed time}} = \frac{t}{T} \tag{40}$$

We wish to examine the conditions under which:

$$C = C \tag{41}$$

i.e.

$$C_L T = Ht + BI \tag{42}$$

which reduces to:

i.e.

$$C_L \frac{T}{t} = H + Bi \tag{43}$$

$$\frac{C_L}{\beta_a} = H + Bt \tag{44}$$

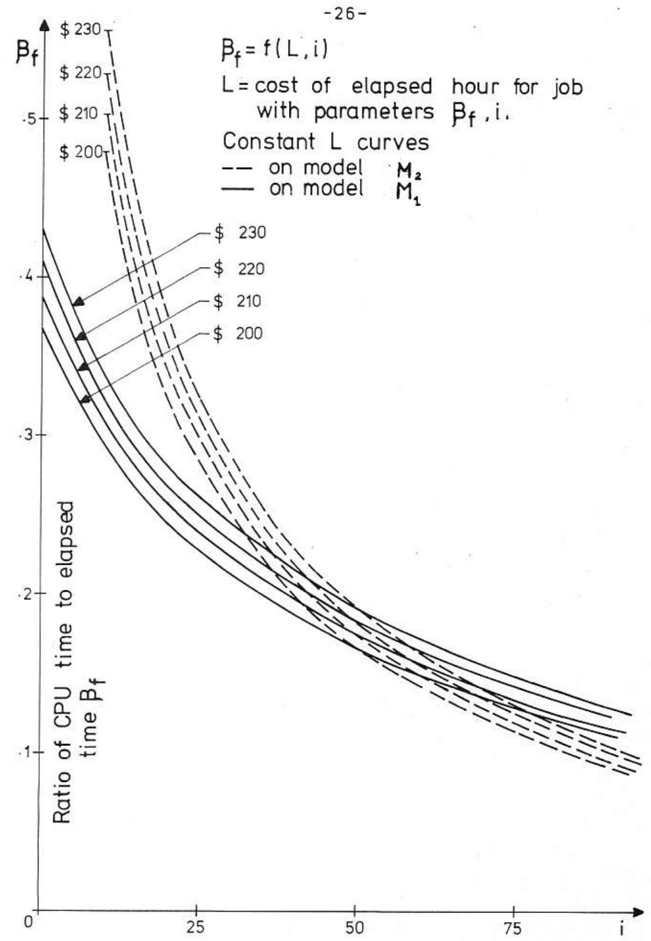

Figure 4 I/O Rate, no EXCPS per second on model

Assuming an interest in constant costs, we get:

$$\beta_a = \frac{C_L}{H + Bi} \qquad (45)$$

and we can plot the required hyperbola, which show the conditions under which the two charging schemes are equivalent, see Figure 4.

Obviously, more bench-marking may be needed – *except* that if two identical machines are involved, all data may be obtained from one benchmark, if the job is run *"stand-alone"*.

We note that the ratio $\frac{C_L}{H + Bi}$ must be less or equal to 1 otherwise β_a exceeds 1 and this is impossible – it shows that:

$$\text{if} \qquad \frac{C_L}{H + Bi} < 1 \qquad (46)$$

then ETC is dearer than either RUC or PTC!

5.3.2 Hiring the Complete Installation – Job Mixes

In this case we seek a comparison between running a number of jobs independently and charging for them independently, and hiring the installation and seeking the gains due to multi-programming etc. by running jobs concurrently.

If we have n jobs, then the total cost is:

$$C = \sum_{i=1}^{n} (Bt_{cpu_i} + HI_i) \qquad (47)$$

$$C = B\tau + H\psi \qquad (48)$$

where τ is the sum of all cpu times
and ψ is the sum of all I/O transfers.

We can now go direct to equation (43) and re-write it using:

$$i^* = \frac{\psi}{\tau} \qquad (49)$$

i.e. the average I/O transfers/cpu seconds over all jobs.

Of course, we are relating all of this back to the degree of compute-boundedness, β_a, and are introducing $\beta_a{}^*$, the ratio of the sum of cpu times for the jobs in the mix to the elapsed time for mix.

Returning to equation (45) we get:

$$\beta_a{}^* = \frac{C_L}{H + Bi^*} \qquad (55)$$

which is a lot more useful, since an average value for $\beta_a{}^*$ will often be available for a given installation! It is after all, only the degree of cpu utilisation.

As demonstrated by Dorwick (1975), this enables the user to get the benefit of the overlapping between cpu and I/O across the whole job mix. As already remarked, other charging methods assume no overlap of any kind.

The availability of this figure may enable a first prediction to be made concerning the economics of creating a job mix.

Certainly, an analyst would be well advised to examine closely any job mix producing values of $\beta_a{}^*$ which differed significantly from the installation norm to establish the reason for the variation and the effect on his costs.

Of course, if:

$$\beta_a{}^* > \frac{C_L}{H + Bi^*} \qquad (51)$$

then elapsed time charging is cheaper.

6. COMPARING COSTS ON DIFFERENT COMPUTERS

The method described is extremely accurate when different machines in one manufacturer's range are being compared. It is also useful as a means of choosing between different charging techniques offered on identical machines.

It can, however, be used to estimate the effects of moving to a completely different computer although the results will not necessarily be as accurate. The accuracy will depend upon the extent to which the I/O activity can be predicted on the new installation. If, for example, we can assume that the logical I/O activity is unchanged, then the new physical I/O rate can be obtained with a knowledge of the operating system input/output subsystem. Allowance must be made for I/O techniques such as indexed sequential and hash addressed access methods in which subsidiary I/O transfers may be generated at differing rates.

The result will be some scaling factor for the I/O rate.

A similar situation exists with cpu time. A value of R_{21} applicable to the particular program needs to be obtained. This figure is best obtained by writing a small program with similar characteristics and running it on both systems. This technique may help in obtaining an I/O scaling factor. If S_2 is the I/O scaling factor then:

$$c_2 = \frac{H_2}{R_{21}} + S_2 B_2 i \qquad (52)$$

is the cost per unit cpu time on the first machine, where H_2 and B_2 are the cost factors for the new machine.

Again, we require that:

$$c_2 < c_1$$

i.e.

$$\frac{H_2}{R_{21}} + S_2 B_2 i < H_1 + B_1 i \qquad (53)$$

and we know H_2, B_2 and i so that we can produce constant contour plots of S_2 as a function of R_{21} for values of $c_2 = Dc_1$ where $D < 1$. It will then be possible to see whether or not the values of R_{21} and S_2 which give $c_2 < c_1$ are likely to occur. This approach will not work in marginal situations and needs to be used with considerable care.

7. CONCLUSION

A technique for comparing computer resource charging proposals has been presented. This technique works well in situations where different computers in one manufacturer's range running identical software are being considered. It can also be used to estimate the likely effect upon running cost of moving a program from one machine to another provided that various parameters can be estimated.

The technique is simple to use and requires only basic algebra.

It is also possible to make such management decisions relating to computing procurement as when to rent a complete installation and when to pay for resources used on a per job basis.

8. ACKNOWLEDGEMENTS

I am grateful to the management of L.M. Ericsson (Australia) Pty. Ltd., for their patience, tolerance and encouragement in this work which was undertaken between August 1971 and January 1972. My immediate supervisor at the time, Mr. D.W. Clarke, encouraged me greatly and I acknowledge the permission granted by L.M. Ericsson to publish this material.

I could not have completed this work without the assistance of technical and sales staff of IBM (Australia), Computer Resources Co Pty Ltd and Computer Sciences Australia, who entered into the spirit of the investigations. Messrs. Peter Howell and Peter Richardson detected significant errors in a preliminary draft and are gratefully acknowledged.

An unknown referee is to be thanked for reading the original paper and suggesting the addition of section 2.2.3 which clarifies subsequent material. The same referee (I presume) carefully read the revised paper and fielded numerous small, but important errors.

Ms. C. Davern of the Computer Centre, Monash University, is also to be thanked for her work in typing the revised paper.

References

ADAMS, C.W. (1962): "Grosch's Law Repealed". *Datamation,* Vol. 8, No. 7, pp. 38-39 July 1962.

ARBUCKLE, R.A. (1966): "Computer Analysis and Thruput Evaluation". *Computers and Automation,* January 1966, pp. 13-16.

BOROVITS, I. & EIN-DOR, P. (1977): "Cost/Utilisation: A Measure of System Performance". *CACM,* Vol. 20, No. 3, March 1977, pp. 185-190.

CURNOW, H.J. & WICHMAN, B.A. (1977): "A Synthetic Benchmark". *Computer Journal,* Vol. 19, No. 1, 1977, pp. 43-49.

DORWICK, P.W. (1975): "Comparing Computer Usage: A Case Study" *Australian Computer Journal,* Vol. 7, No. 1, March 1975, pp. 12-14.

DRUMMOND, M.W. Jnr. (1973): "Evaluation and Measurement Techniques for Digital Computer Systems". Prentice-Hall Inc. Englewood Cliffs N.J. 1973.

KNIGHT, K.E. (1966): "Changes in Computer Performance". *Datamation,* Vol. 12, No. 9, September 1966, pp. 40-54.

KNIGHT, K.E. (1968): "Evolving Computer Performance". *Datamation,* Vol. 14, No. 1, January 1968, pp. 31-35.

LEGG, M.P.C. (1972): "Allocation and Costing of Computer Resources". *Proc. 5th Australian Computer Conference* pp. 572-577, Australian Computer Society Inc. 1972.

NOLAN, R.L. (1977): "Effects of Chargeout on User/Manager Attitudes". *CACM,* Vol. 20, No. 3, March 1977, pp. 177-184.

SOLOMON, M.B. (1966): "Economics of Scale and the IBM System/360". *CACM,* Vol. 9, No. 6, June 1966, pp. 435-440.

SVOBDOVA, L. (1976): "Computer Performance Measurement and Evaluation Methods: Analysis and Applications". American Elsevier New York, 1976.

WALTERS, R.E. (1977): "Benchmark Techniques: A Constructive Approach". *Computer Journal,* Vol. 19, No. 1, 1977, pp. 50-55.

Book Review

Chess Skill in Man and Machine, ed. P.W. Frey, 1977 (published Springer-Verlag). 217 pp, cloth, US$16.00.

In 1957 Professor H.A. Simon (a computer scientist) predicted that a computer would be world chess champion within the following ten years. Shortly after the time limit expired David Levy (later to become an international chess master) made a wager with a number of computer scientists that he (Levy) would not lose to a computer program before the 31st of August, 1978. There is an outside chance that Levy may lose that wager judging from the most recent battle between him and CHESS 4.5 (running on a giant CYBER 176). Levy won the match when CHESS 4.5 resigned on the 43rd move, but it was a hard fight (see SIGART Newsletter, No. 62, April, 1977). Incidentally, CHESS 4.5 was entered *and won* the 1977 Minnesota Open Chess Championship, defeating along the way players rated as Class A and Expert on the USCF scale.

How can a program achieve such high standards of play; can improvements to Grand Master capacity be expected; what results of a more general nature can be learned from chess programming? Such questions, and others, are tackled by Frey and the contributors to this book. The volume opens with a description of interesting games played up to 1975 in tournaments where two programs or a human and a program were pitted against each other (some games from the 1976 ACM Championship appear in the Appendix). This is followed by a chapter reviewing the literature dealing with the question of how people make decisions and perceive positions during chess games. Inevitably the question arises as to whether a program should try to play like a human being or use algorithms that are peculiarly suited to machine implementation.

The next two chapters might convince the reader that machine oriented algorithms are better. Chapter 3 looks at the basic problems any chess program must resolve: board representation, move generation, position evaluation, creation of move trees, pruning move trees and so on. Chapter 4 is a detailed description of the CHESS 4.5 program by its designers. We find that their program uses "brute force" generation of all possible moves down to a given depth, although some moves are followed a little deeper to make sure no sudden captures or checks were just hidden from "view". What a contrast from the method adopted by human players who typically generate fewer than fifty moves during a search for a best move while CHESS 4.5 may generate as many as 500,000 before making its choice. But CHESS 4.5 designers close their chapter with some doubts about the ultimate value of "full width" searching and subsequent contributors in Chapters 5, 6 and 7 amplify those doubts with descriptions of move selection procedures which prune away (hopefully) useless moves and their continuations so that more computer time will be available for a deeper analysis of the remaining promising moves.

Finally in Chapter 8 the optimism of the earlier chapters is balanced by a discussion of the enormity of the task and the naivety of today's programs. Included in that chapter is a section aimed at both those who consider chess programming a waste of resources, and those who condone it.

Overall the book is well worth reading by anyone interested in chess. A little effort was put into the text to make the book accessible to non-programmers, but three out of the eight chapters would make very difficult reading without a programming background. References in the back of the book provide a good cross section of the literature up to mid 1975.

P.M. Herman,
Monash University

CORRIGENDA

In the November 1977 issue, Volume 9, Number 4, two names were incorrectly cited on the title page of their respective articles. They were:

Page 145 E.C. Cook should have read B.G. Cook

Page 159 H.W. Hodaway should have read H.W. Holdaway

The initials of Mr Cook were also incorrectly shown on the index page.

The editor apologises for these mistakes.