Peter Bryant

Predicting Working Set Sizes

Abstract: Empirical analyses of data on working set size are reported. The data do not support the hypothesis that working set sizes are normally distributed. The data suggest various algorithms for predicting working set size based on the program's past history. Several representative algorithms are discussed and evaluated.

Introduction

Operating system schedulers and dispatchers must often make decisions about which programs may run together based upon some (implicit or explicit) prediction of the resources those programs will require if they are allowed to run. The quantities predicted may be working set size, resident set size, I/O activity, and so on. The usual technique is what is called here the "naive" technique, namely, to predict that the program will require those resources it required during its last execution interval.

Exactly which quantities should be considered in making scheduling decisions is a subject of active debate, and I do not intend to enter this debate here. At least one dispatcher [1], however, has been built that bases its decisions explicitly on predicted working set size, and the intent of this study is to propose and evaluate a few different prediction techniques for such a scheduler. The proposed techniques are suggested by the analysis of program trace data. The analysis procedure used was a crude version of techniques advocated by Box and Jenkins [2], and is described in some detail in the third section. It would apply equally well to predicting quantities other than working set size, and may thus be of independent interest here as a way of discovering possible models.

Many authors [3-7] have discussed the idea of a program's *working set*, defined as the resources a program requires during the execution interval $(T, T + \Delta T)$ as T varies; ΔT is known as the window size, T and ΔT are often measured in units of the number of instructions executed, and the resources are often taken to be the storage requirements of the program. In this study T is measured in instructions, and the working set is taken to be the 4096-byte *pages* of storage required by the program. The *working-set size*, W, in any interval $(T, T + \Delta T)$ is the number of distinct pages the program refers to in the interval.

The approach taken here is largely empirical and ad hoc; no particular model is suggested except incidentally as possible motivation for some of the predictive algorithms tested. The data used are described in the second section, and the analyses of them in the third section. The algorithms are described and evaluated in the fourth section. Some remarks and a possible extension of the techniques are given in the last section.

The data

The data for this investigation were derived from program traces of programs running under CMS version 3.1 [8] on CP-67 at the IBM Cambridge Scientific Center. The traces had already been reduced to SIMLOAD files for use with the CP-67 simulator [9]. This process divides a program's execution into segments defined as follows. A segment includes that portion of the execution of the program from the end of the previous segment until either a specified maximum number of instructions (usually 1000) have been executed or a specified maximum number (usually 10 or 15) of 4K pages (here K stands for kilobytes) have been referred to [10]. The SIMLOAD files consist of records summarizing the program's behavior in each segment, including in particular the number of instructions executed and a list of the pages referred to.

Because the processing of the traces themselves is quite lengthy, the working set data studied here were derived directly from the SIMLOAD files, by aggregating successive segments until the cumulative instruction

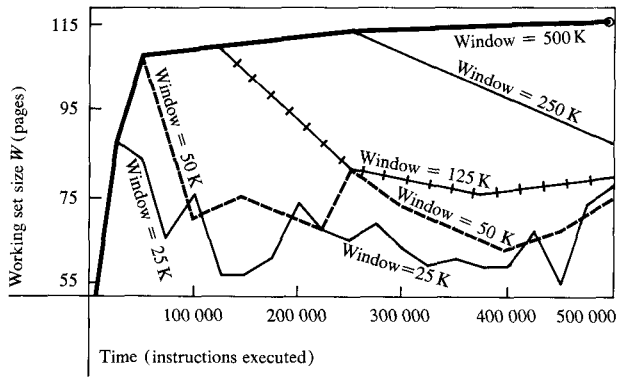


Figure 1 Working set size as a function of time for program SWP15. The window is the nominal window size as defined in the text.

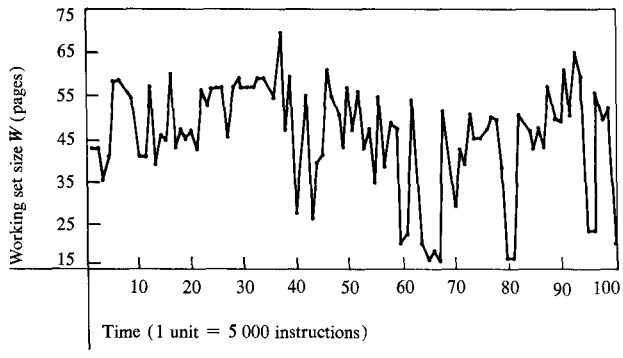


Figure 2 Working set size vs time for program SWP15 (window = 5 000 instructions).

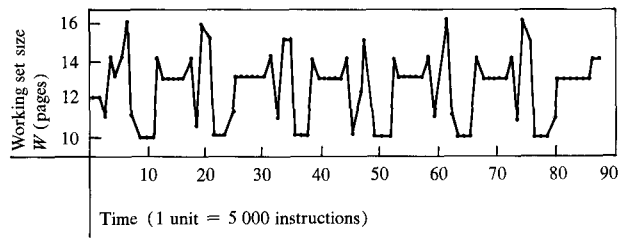


Figure 3 Working set size vs time for program TRVU (window = 5 000 instructions).

count equaled or exceeded the desired window size. (The corresponding working set was taken as the union of the sets of pages referred to in those segments aggregated.) Thus, for a nominal window size of 5 000 instructions, the actual window size would be at least 5 000 and might be as long as 5 999 (for a cutoff value of 1 000 instructions per segment), and would vary over the execution of the program.

In the data used here, the actual window sizes for a nominal 5 000-instruction window were generally uniformly distributed over the possible range. For larger window sizes, this effect is less important, in view of the 1 000-instruction cutoff for segments in the SIMLOAD files. The varying window size makes these data not strictly comparable with most published studies which use an exact, fixed, window size. On any particular machine, though, different instructions have generally different execution times. Further, a scheduler would probably not be equipped to (or want to) interrupt a running program after a specific number of instructions. From the point of view of developing scheduling algorithms, then, the approach used here seems no more unrealistic than using a fixed window size. Also, the data used here were examined to see if any particular bias was introduced by the varying window size. The only particular effects noticed were occasional *negative* correlations between actual window size and working set size, and these were apparently consequences of some clustering (see the discussion of bimodality below).

Data for five programs were analyzed. They included PERF92, an APL(CMS) execution; MAXMIN15, a PL/I compilation; MEDU, a job consisting of two assemblies and two FORTRAN compilations; and two other programs discussed below. Nominal window sizes of 5 000, 12 500, 25 000, 50 000, 125 000, 250 000, and 500 000 instructions were used.

Summary of data analyses

Figures 1 through 9 summarize the principal analyses performed on the working set data. Program SWP15, a PL/I compilation, is discussed in some detail and compared with the other programs. Essentially two basic patterns were observed. A perhaps typical random fluctuation in working set size is that illustrated by SWP15. Occasionally, though, programs display definite cyclical patterns over time, such as repetitive calling of the same subroutine. This may be only a portion of the whole execution. TRVU is discussed below to illustrate this sort of behavior; it is an artificial job, designed to simulate an editing session at a terminal. It is included here as an illustration of cyclical behavior, and is not intended to represent a realistic sort of job. Program PERF92, an APL execution, does display cyclical patterns in part of its execution, and it is discussed in a subsequent section.

Figure 1 shows the behavior of working set sizes (W) of SWP15 over the first 500 000 instructions for different nominal window sizes. The upper (heavy) line, or "envelope," shows the increase in W as the window size increases. The pattern shown in Fig. 1 is typical of those found in the other programs—a steep increase followed by a leveling off. The steepness and duration of the increase varied from program to program, but the general shape was as depicted in Fig. 1. In particular, this curve was always found to be concave. The fluctuations in the curves underneath it are also fairly typical.

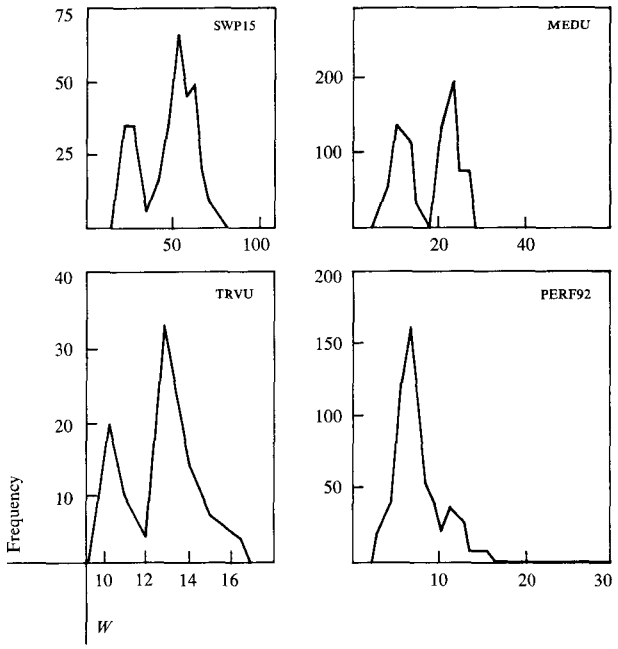


Figure 4 Histograms of working set size for various programs (nominal window size, 5K).

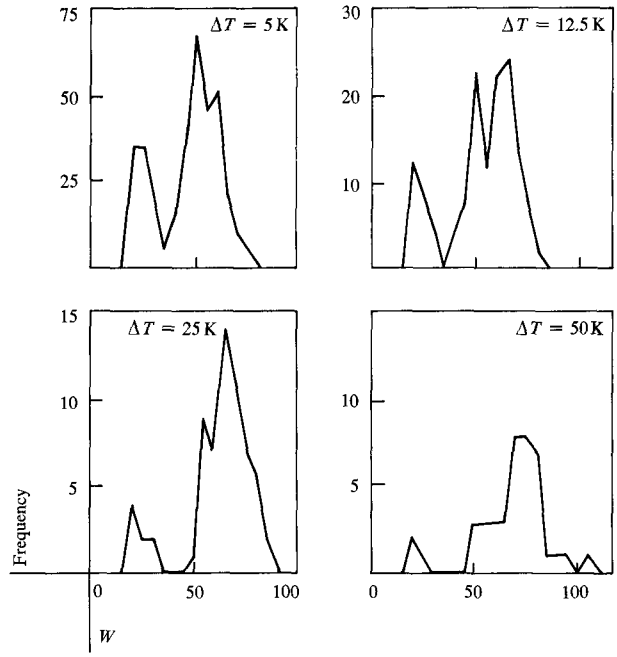


Figure 5 Histograms of working set size for various nominal window sizes ΔT (program SWP15).

Figure 2 gives a plot of W over time for a particular program, SWP15, and a particular nominal window size, 5 000 instructions. The corresponding plot for program TRVU is in Fig. 3, where its repetitive nature shows up clearly. There is perhaps a hint of a low-frequency cyclical pattern in the plot for SWP15, but analysis of the entire execution (Fig. 2 is for the first 500K instructions only) shows no such tendency.

Histograms of W are displayed in Fig. 4 for four programs. For both SWP15 and MEDU, the histograms are quite clearly bimodal. The histogram is also bimodal for TRVU though in view of the small number of distinct values of W involved, the bimodality might be considered artificial. PERF92 displays more nearly unimodal behavior. Bimodal distributions like these have also been reported by Rodriguez-Rosell [3, Fig. 6]. I found no evidence that such behavior was attributable to the varying actual window size, though that is a possibility. In any event it seems, on the basis of these data, that the often mentioned [4, 5] hypothesis that W is approximately normally distributed should not be accepted blindly. Compilers, particularly (as in SWP15 and MEDU), often consist of alternate executions of relatively large "phases" and relatively small "interludes," that update control blocks and determine which phase to call in next. In such cases bimodality is probably to be expected.

Figure 5 gives histograms of W for four different nominal window sizes for SWP15. They indicate that the bimodality is not due simply to a choice of window size. Of course, for very large window sizes, the number of data points becomes so small as to make a histogram of questionable value, but, within the range considered here, there seemed to be no particular evidence that changing the window size changed the general nature of the results. Most of the subsequent analyses used a window size of 5 000 instructions.

The main aim of this investigation was to evaluate possible predictive schemes, and the analyses depicted in Figs. 6 through 9 reflect this. They are more or less in the spirit of the approaches described by Box and Jenkins [2]. These approaches look at the autocorrelation functions of the quantities to be analyzed (in this case, W) and of the first, second, etc. differences of these quantities, until a recognizable pattern appears, which suggests a model to be examined. In the following analyses, no models beyond those suggested by examining the series of W values and its first differences were used very heavily, and the analyses themselves were not particularly formal. The interested reader may pursue these ideas in Box and Jenkins [2].

As remarked above, the histograms of W are often bimodal. But, as Fig. 6 shows, the distribution of the first differences $(\Delta W)_i = W_i - W_{i-1}$ is essentially unimodal and much more nearly normal. So are those for the second and third differences. This unimodality also held for various nominal window sizes, as Fig. 7 illustrates for ΔW, and this was typical of other programs as well. In view of Figs. 4 and 5, the joint distribution of the differences cannot be normal (for then that for W would

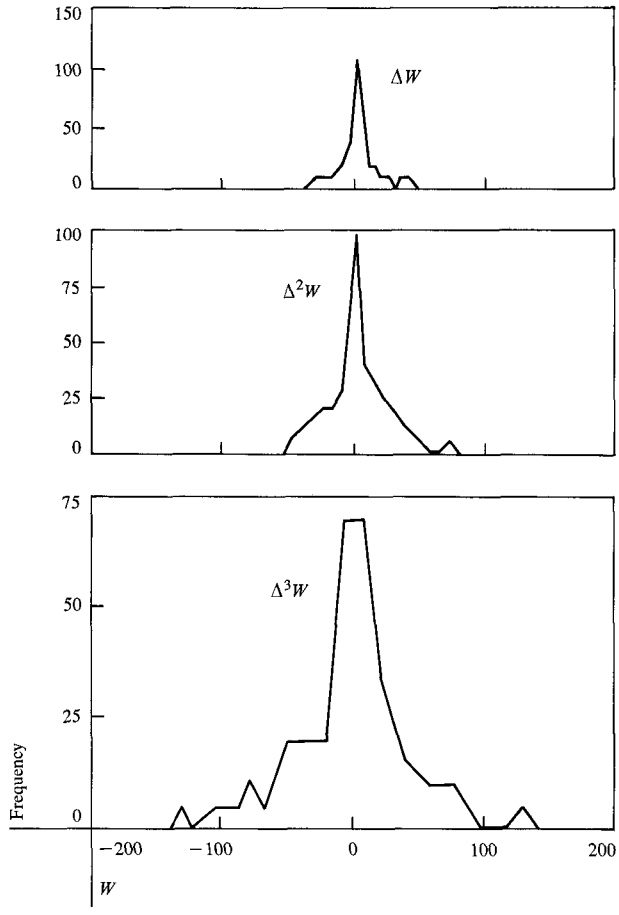
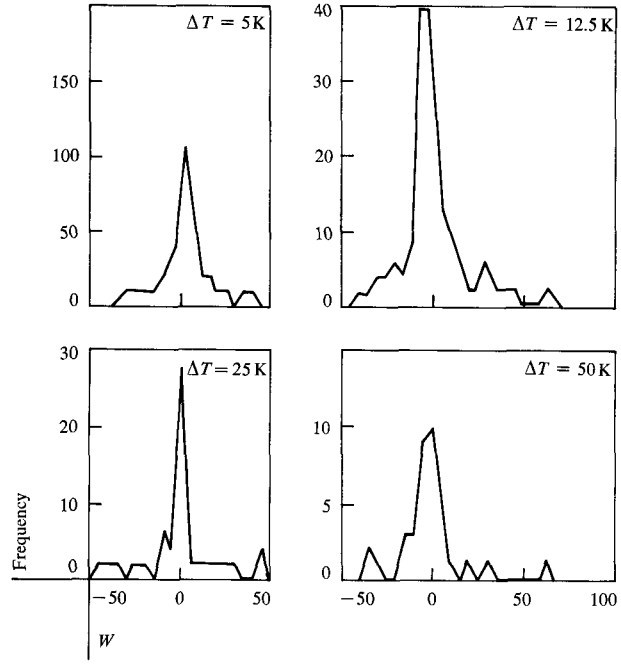


Figure 6 Histograms of 1st, 2nd, and 3rd differences of working set sizes (ΔW, $\Delta^2 W$, $\Delta^3 W$); program SWP15; nominal window size, 5K.

Figure 7 Histograms of ΔW for various nominal window sizes ΔT; program SWP15.



be also), but Figs. 6 and 7 suggest that normality would be a much more useful first approximation for the differences than for the original values.

The *autocorrelation function* of a time series (in this case, the series W) is an important tool in developing predictive schemes or models. The *autocorrelation of lag k*, ρ_k, is the correlation between W_i and W_{i+k} and the autocorrelation function is defined to be ρ_k as a function of k. Values of ρ_k near 1 or -1 indicate a strong (linear) relationship between W_i and W_{i+k}, whereas values of ρ_k near 0 indicate little or no linear relationship. Alternatively, we can interpret this by saying that values of ρ_k near ± 1 indicate that W_i is useful in (linearly) predicting W_{i+k}. For example, if ρ_1 is large, but ρ_k is small for $k > 1$, it indicates that W_i is useful for predicting W_{i+1}, but W_{i-1}, W_{i-2}, etc. are not. If a series were cyclical, repeating itself every j intervals, this would cause ρ_j to be large. Examining the autocorrelation function, then, is one way of deciding which quantities to use in prediction methods.

In Fig. 8, the estimated autocorrelation functions of W for four of the programs are displayed, for lags 1 through 25. The cyclical, repetitive nature of TRVU shows up clearly in the large autocorrelation of lag 14. The other programs show a general trend of a large first-order autocorrelation, which tails away as the lag increases. PERF92 shows an increase at lag 20, corresponding to the first portion of its execution, which shows a cyclical pattern of period 20 (see discussion in next section).

The autocorrelation functions for the first and subsequent differences are all similar and are illustrated for SWP15 in Fig. 9. A strong negative autocorrelation of lag 1, with subsequent ones fluctuating around zero, is typical.

Results similar to these were obtained for other window sizes as well, although for larger window sizes there are fewer data, and this fact can be important in estimating the autocorrelation functions.

Predictive algorithms

• *Discussion*

The autocorrelation functions in Figs. 8 and 9 show high values of ρ_k for small k ($k = 1, 2, 3$) and lower values for $k > 3$. This suggests that the past behavior of a program may indeed be used to get some useful information for predicting its subsequent behavior, and that most of this information is contained in the two or three previous observations. For programs exhibiting characteristics like those of SWP15, a procedure that bases its prediction of the next value of W on the last two values seems reasonable. Alternatively, one might base the prediction on the last few differences ΔW. In fact, because the differences seemed so much more nearly normally distrib-

uted, with mean 0, the first method investigated was derived by assuming that $[(\Delta W)_i, (\Delta W)_{i+1}]$ was distributed according to a bivariate normal distribution [11] with mean vector $(0, 0)$, common variance σ^2, and correlation coefficient ρ. Using some estimator of ρ, say $\hat{\rho}$, and predicting $(\Delta W)_{i+1}$ by its expected value given $(\Delta W)_i$, we have the estimator

$$(\widehat{\Delta W})_{i+1} = \hat{\rho}(\Delta W)_i,$$

and we estimate W_{i+1} by

$$W_i + (\widehat{\Delta W})_{i+1} = W_i + \hat{\rho}(W_i - W_{i-1}).$$

A similar estimate can be based directly on the W_i. Suppose (W_i, W_{i+1}) is distributed with mean vector (μ, μ) and covariance matrix

$$\sigma^2 \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}.$$

Then we can predict W_{i+1} given W_i by $\hat{W}_{i+1} = \hat{\mu} + \hat{\rho}(W_i - \hat{\mu})$, using some estimators $\hat{\mu}$ and $\hat{\rho}$ of μ and ρ.

There is no a priori reason to limit the "memory" of the algorithm to just the immediately preceding observation, but including more would increase the complexity and storage requirements of the algorithm. The autocorrelations in Figs. 8 and 9 seem to indicate that the immediate predecessor gives most of the information, too. Some experiments using up to four previous observations were run, and while occasionally they did better than the ones using a memory of one observation, it was rare. Usually, it appeared that including more W's or (ΔW)'s simply added more noise without improving the prediction. These results are thus not included below.

Similarly, one could estimate and keep track of many autocorrelations, in the hope of detecting periodic behavior such as that shown for TRVU, but this procedure would quickly become unwieldy. Aside from the problems of estimation and the storage requirements, there is the question of how many to include. As PERF92 shows, it is also possible for a program to show periodic behavior for only one part of its execution, yet have that part affect the entire autocorrelation function. The extra effort seems unlikely to result in much improvement, and such algorithms were not evaluated.

• *Algorithms*

Many variations on the kinds of algorithms discussed above are possible. Parameters such as μ and ρ could be either given once and for all or dynamically estimated for each program. Various dynamic estimation schemes are possible. It was not the intent of this investigation to try to pin down exactly which variant was "best," because that might well depend on the particular operating system and the program load. Rather, five algorithms

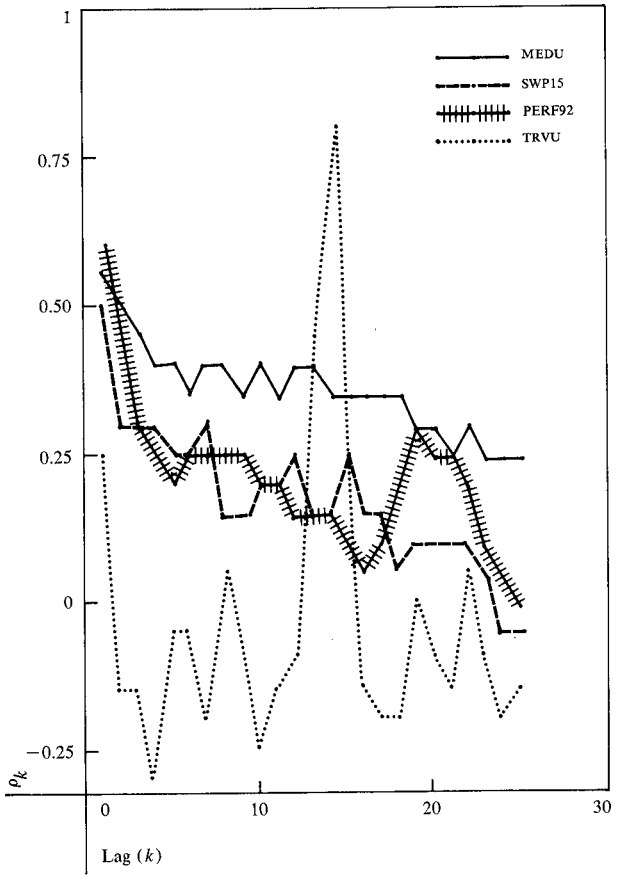
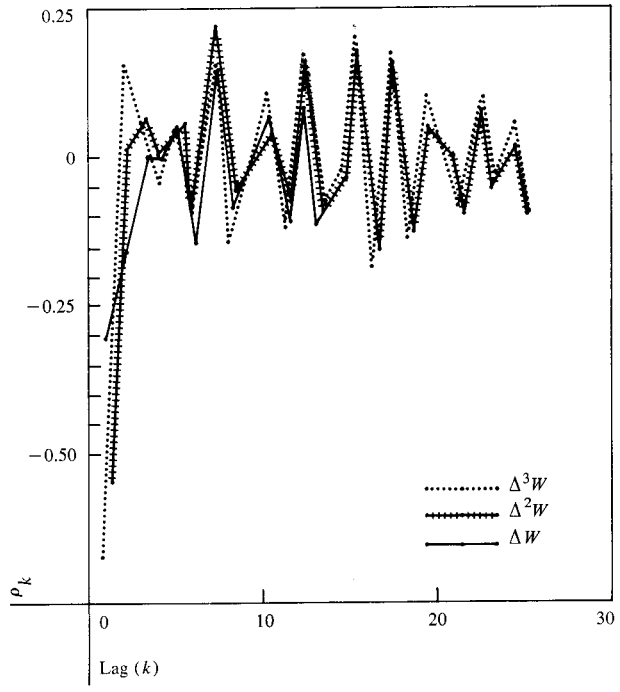


Figure 8 Autocorrelation functions ρ_k of working set size for various programs; nominal window size, 5K.

Figure 9 Autocorrelation functions ρ_k for ΔW, $\Delta^2 W$, $\Delta^3 W$; program SWP15; nominal window size, 5K.



were chosen as representative of the algorithms suggested by the data, and they were evaluated by testing them against data from five programs. The details of the algorithms are given in Table 1. Algorithm 1 is the "naive" predictor, $\hat{W}_{i+1} = W_i$. Algorithm 2 predicts $\hat{W}_{i+1} = \mu$, the mean of the W_i. Algorithm 3 is based on ΔW: $\hat{W}_{i+1} = W_i + \rho_\Delta(W_i - W_{i-1})$, where ρ_Δ is the correlation between $(\Delta W)_i$ and $(\Delta W)_{i+1}$. Algorithm 4 is based directly on W: $\hat{W}_{i+1} = \mu + \rho_W(W_i - \mu)$, where μ is the mean of W and ρ_W is the correlation between W_i and W_{i+1}.

The motivations for Algorithms 3 and 4 were discussed earlier. Algorithms 1 and 2 were included as a basis for comparison. Algorithm 1 is typical of the usual approaches in today's systems. Algorithm 2 would be appropriate if W_i and W_{i+1} were independently normally distributed with a common mean μ.

Table 1 Algorithms tested.

Algorithm 1	$\hat{W}_{i+1}^{(1)} = W_i$
Algorithm 2[a]	$\hat{W}_{i+1}^{(2)} = \mu_i$
Algorithm 3[b]	$\hat{W}_{i+1}^{(3)} = W_i + (W_i - W_{i-1})\rho_{u,i}$
Algorithm 4[a,c]	$\hat{W}_{i+1}^{(4)} = \mu_i + (W_i - \mu_i)\rho_{w,i}$
Algorithm 5[d]	Choose k_0 to minimize $\mathrm{MSE}_i^{(k)}$ $(1 \le k \le 4)$. Predict W_{i+1} using Algorithm k_o

[a] $\mu_i = A_i/B_i$, where

$A_1 = \alpha_2 W_1$; $A_i = (1-\alpha_2)A_{i-1} + \alpha_2 W_i (i \ge 2)$

$B_1 = \alpha_2$; $B_i = (1-\alpha_2)B_{i-1} + \alpha_2$ $(i \ge 2)$

[b] $\rho_{u,i} = U_i/T_i$, where

$T_1 = U_1 = U_2 = 0$

$U_i = (1-\alpha_3)U_{i-1} + \alpha_3(W_i - W_{i-1})(W_{i-1} - W_{i-2})$ $(i \ge 3)$

$T_i = (1-\alpha_3)T_{i-1} + \alpha_3(W_i - W_{i-1})^2$ $(i \ge 2)$

[c] $\rho_{w,i} = S_i/V_i$, where

$S_1 = V_i = 0$

$S_i = (1-\alpha_4)S_{i-1} + \alpha_4(W_i - \mu_i)(W_{i-1} - \mu_i$ $(i \ge 2)$

$V_i = (1-\alpha_4)V_{i-1} + \alpha_4(W_i - \mu_i)^2$

[d] $\mathrm{MSE}_1^{(k)} = 0$; $\mathrm{MSE}_i^{(k)} = (1/B_i)[(1-\alpha_5)\mathrm{MSE}_{i-1}^{(k)} + \alpha_5(\hat{W}_j^{(k)} - W_i)^2]$ $(i \ge 2)$, $k = 1, 2, 3, 4$.

Algorithm 5 is an "empirical Bayes" algorithm which was prompted by the observation that none of algorithms 1-4 was uniformly best over all programs tested. It consists of computing all four predictions, keeping track of the prediction errors of each one, and using the one which has been doing best for you lately.

The parameters μ, ρ_Δ and ρ_W were dynamically estimated, using an exponentially weighted average, as indicated in Table 1. The prediction errors in Algorithm 5 were also exponentially weighted. The results below were obtained using $\alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = 0.05$. (Results were comparable using any $\alpha_i < 0.10$; $\alpha_i > 0.10$ usually gave inferior results.)

The criterion of measurement was the root-mean-square prediction error. This method was chosen as convenient, and also because, in practice, a scheduler would be doing this estimation for many programs and adding the results to get an estimate for all programs combined. The bias in the algorithms tested was small or negligible, and thus the mean-square error in the sum is the sum of the mean-square-errors in the individual predictions. Thus, improvements in predicting for an individual program should be correspondingly reflected in the predictor for the sum. Similarly, the predictors were allowed to predict fractional values of W (though W is an integer) because the predictions for various programs would probably be summed.

• *Results*

The results of the tests are given in Tables 2 and 3. The entries in Table 2 are the root-mean-square prediction errors for each predictor. The entries in each row of Table 3 are the relative root-mean-square errors, i.e.,

RMS error (predictor i)

$\div$ min {RMS error (predictor i)}.

As Tables 2 and 3 indicate, Algorithms 2 and 4 are perhaps the best of the algorithms considered; Algorithm 4 being preferable for smaller window sizes (5K, 12.5K, sometimes 25K) and Algorithm 2 for the larger window sizes. The naive predictor, Algorithm 1, is almost never best; is typically 15 percent worse (in terms of RMS prediction error) than Algorithm 2 or 4; and can be as much as 49.8 percent worse. Algorithms 3 and 5, while often better than Algorithm 1, seem to offer no particular advantage over Algorithm 2 or 4.

As may be seen in Table 2, the RMS error in all algorithms is roughly proportional to the mean value of W, so the absolute size of W offers no way of choosing between Algorithms 2 and 4. The better performance of Algorithm 4 for small window sizes might be attributable to the fact that small window sizes mean more data. Algorithm 4 has more parameters to estimate than Algo-

rithm 2, and it may be that once it has enough data to estimate them well, it outperforms Algorithm 2. Examination of the prediction errors over time tended to confirm this, but other explanations are possible, too—perhaps large and small window sizes lead to fundamentally different kinds of behavior, requiring different predictors. The data considered here do not seem to support this idea, but it remains a possibility.

Figures 10 and 11 show the actual and predicted values of W for PERF92 for a 5K nominal window. Figure 10 is for the first part of the execution, where cyclical patterns are present, using Algorithm 4. At first, Algorithm 4 is "caught napping" by the sudden drops from the peaks, but by about the 80th prediction, it is compensating by not predicting such high values. Following this part of the execution, PERF92 enters a phase of nearly constant values of W (which is not illustrated). At the beginning of a subsequent, more variable phase, Algorithm 4 begins to track the fluctuations as they appear again.

By contrast, Algorithm 2 makes little effort to track fluctuations, as is shown for the first 100 predictions of PERF92 in Fig. 11. In terms of RMS prediction error, Algorithm 2 was about 12 percent worse than Algorithm 4. It also appears, though, that Algorithm 4 has a lower probability of larger errors.

Remarks and acknowledgment

As mentioned, the data considered here seem consistent with that reported by Rodriguez-Rosell [3]. My colleague Bard [12] has independently experimented with algorithms like Algorithm 4 for predicting resident set

Table 2 Root-mean-square prediction error for various algorithms.

Program	*Nominal window size* (K)	*Algorithm*[b] 1	2 •	3	4	5	*Number of predictions* (N)	*Mean working set size*
PERF92 (APL execution)	5	2.722[a]	2.815	2.744	2.517	2.724	683	7.416
	12.5	4.350	3.879	4.244	3.839	4.245	310	8.487
	25	5.612	4.775	5.133	4.799	5.138	156	10.269
	50	6.678	5.952	6.196	6.109	6.203	79	13.089
	125	8.790	7.098	7.967	7.618	8.269	31	17.161
	250	7.644	7.201	7.773	8.058	7.852	16	21.375
MAXMIN15 (PL/I compilation)	5	12.095	11.774	11.520	10.737	12.095	227	47.991
	12.5	14.323	13.337	14.389	13.266	14.328	95	60.526
	25	14.652	13.316	14.239	14.191	14.652	48	67.167
	50	19.661	16.268	20.254	17.891	19.661	24	72.917
	125	20.710	19.182	20.294	21.135	20.710	10	79.400
	250	28.245	28.855	30.048	31.517	28.245	5	80.200
MEDU (assemblies & FORTRAN)	5	6.031	5.293	5.499	4.936	5.328	871	16.983
	12.5	5.289	5.430	5.341	4.896	5.348	372	20.003
	25	6.659	6.010	6.213	5.821	6.659	189	22.175
	50	6.749	6.470	6.555	6.203	6.371	95	24.768
	125	6.132	7.050	5.956	6.363	5.962	38	28.316
	250	8.291	8.075	7.007	8.271	8.291	19	30.053
SWP15 (PL/I compilation)	5	14.878	14.285	14.440	13.186	14.878	336	42.673
	12.5	15.842	16.381	15.784	14.575	15.845	141	53.660
	25	16.270	16.920	16.199	15.850	16.270	72	61.597
	50	16.962	17.659	17.139	18.610	17.154	36	68.444
	125	16.678	15.476	16.736	18.349	16.678	14	77.929
	250	22.605	18.879	20.815	21.097	20.842	7	82.571
TRVU (edit session)	5	2.106	1.791	2.090	1.848	1.863	87	12.517
	12.5	2.818	1.891	2.478	1.881	2.028	35	13.686
	25	3.100	2.443	2.825	2.521	2.553	18	14.722
	50	3.145	2.591	3.173	2.849	3.128	9	16.222

[a] Entries are $\left[\frac{1}{N-1}\sum_{i=2}^{N}(W_i-\hat{W}_i)^2\right]^{1/2}$ in units of 4K pages.

[b] See Table 1 for descriptions of algorithms. Values here computed using $\alpha_2=\alpha_3=\alpha_4=\alpha_5=0.05$.

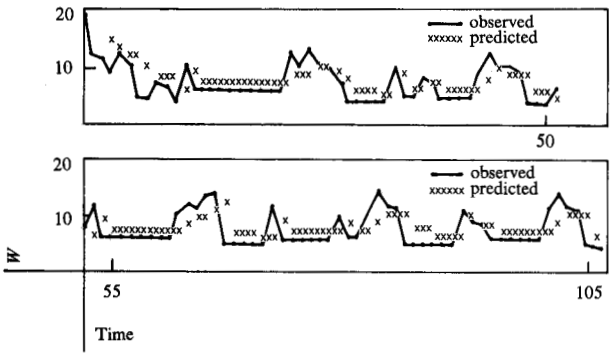


Figure 10 Working set sizes using Algorithm 4; program PERF92 (1st part); nominal window size, 5K.

sizes in VM/370, and has found that they give improved results over algorithms such as Algorithms 1 and 2. Ghanem and Kobayashi's model [7] of working set behavior, in particular their Eqs. (3.44) and (3.45), lead to predictors similar to Algorithm 4.

An interesting byproduct of algorithms like Algorithm 3 or 4 is the possibility of obtaining confidence bands for the predicted value as well as point estimates. The models used to derive those algorithms lead to estimates of the variance of the predictor in terms of σ^2 and ρ, for example. A scheduler might take into account the confidence to be placed in a given estimate and allow itself an appropriate margin of safety when it guesses whether the sum of the resource demands is likely to be within the capacity of the system.

For example, the model underlying Algorithm 4 is that (W_i, W_{i+1}) is normally distributed with mean vector (μ, μ) and covariance matrix

$$\sigma^2\begin{bmatrix}1 & \rho\\ \rho & 1\end{bmatrix}.$$

Under these conditions, it is easy to show that the conditional distribution of W_{i+1}, given W_i, is normal with mean

Table 3 Relative RMS prediction error for various algorithms.

Program	*Nominal window size* (K)	*Algorithm* 1	2	3	4	5	*Number of predictions* (N)
PERF92	5	1.082[a]	1.119	1.090	1.000	1.082	683
	12.5	1.133	1.010	1.105	1.000	1.106	310
	25	1.175	1.000	1.075	1.005	1.076	156
	50	1.122	1.000	1.041	1.026	1.042	79
	125	1.238	1.000	1.122	1.073	1.165	31
	250	1.062	1.000	1.080	1.119	1.090	16
MAXMIN15	5	1.127	1.097	1.073	1.000	1.127	227
	12.5	1.080	1.005	1.085	1.000	1.080	95
	25	1.100	1.000	1.069	1.066	1.100	48
	50	1.209	1.000	1.245	1.100	1.209	24
	125	1.080	1.000	1.058	1.102	1.080	10
	250	1.000	1.022	1.064	1.116	1.000	5
MEDU	5	1.222	1.072	1.114	1.000	1.079	871
	12.5	1.080	1.109	1.091	1.000	1.092	372
	25	1.144	1.032	1.067	1.000	1.144	189
	50	1.088	1.043	1.057	1.000	1.027	95
	125	1.030	1.184	1.000	1.068	1.001	38
	250	1.183	1.152	1.000	1.180	1.183	19
SWP15	5	1.128	1.083	1.095	1.000	1.128	336
	12.5	1.087	1.124	1.083	1.000	1.087	141
	25	1.027	1.068	1.022	1.000	1.027	72
	50	1.000	1.041	1.010	1.097	1.011	36
	125	1.078	1.000	1.081	1.186	1.078	14
	250	1.197	1.000	1.103	1.117	1.104	7
TRVU	5	1.176	1.000	1.167	1.032	1.040	87
	12.5	1.498	1.005	1.317	1.000	1.078	35
	25	1.269	1.000	1.156	1.032	1.045	18
	50	1.214	1.000	1.225	1.100	1.207	9

[a]Entries are ratios of entries in Table 2 to the minimum entries in the rows of Table 1. E.g., the entry in the first row, first column is $2.722 \div 2.517 = 1.082$.

$$\tilde{\mu} = \mu + \rho(W_i - \mu),$$

which is essentially the formula used in Algorithm 4, and variance

$$\tau^2 = \sigma^2(1 - \rho^2).$$

A $100(1-\alpha)$-percent confidence interval for W_{i+1}, given W_i, is then

$$[\tilde{\mu} - z_{\alpha/2}\tau, \tilde{\mu} + z_{\alpha/2}\tau],$$

where $z_{\alpha/2}$ is the upper $\alpha/2$-point of the standard normal distribution. By estimating $\tilde{\mu}$ and τ^2 from the data, we can thus obtain an approximate confidence interval. The confidence intervals for Algorithms 3 and 4 were evaluated for a few programs and the distribution of the prediction error was found to be generally more peaked than a normal distribution (e.g., 75 percent or more within ± one standard deviation), but a normal approximation would probably suffice.

I thank my colleagues Yonathan Bard and Martin Schatzoff for a number of very helpful conversations in connection with this work.

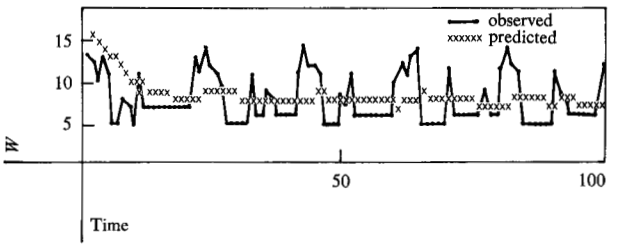


Figure 11 Working set sizes using Algorithm 2; program PERF 92 (1st 100 predictions); nominal window size, 5K.

References and notes

1. J. Rodriguez-Rosell and Jean-Pierre Dupuy, "The Design, Implementation and Evaluation of a Working Set Dispatcher," *Comm. ACM* **16,** 247 (1973).
2. G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden-Day Publishing Co., San Francisco, 1970.
3. J. Rodriguez-Rosell, "Empirical Working Set Behavior," *Comm. ACM* **16,** 556 (1973).
4. E. G. Coffman and T. A. Ryan, "A Study of Storage Partitioning Using a Mathematical Model of Locality," *Comm. ACM* **15,** 185 (1972).
5. P. J. Denning and S. C. Schwartz, "Properties of the Working Set Model," *Comm. ACM* **15,** 191 (1972).
6. P. J. Denning, "Virtual Memory," *Computing Surveys* **2,** 3 (1970).
7. M. Z. Ghanem and H. Kobayashi, "A Parametric Representation of Program Behavior in a Virtual Memory System" *Research Report RC-4560*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (October 1973).
8. *CP-67/CMS Manual*, Form No. GY20-0590, IBM Data Processing Division, White Plains, NY 10604 (1971).
9. C. Boksenbaum, S. Greenberg and C. Tillman, "Simulation of CP-67," *Report No. G320-2093*, IBM Cambridge Scientific Center, Cambridge MA 02139 (May 1973).
10. This is a simplification. A segment may be terminated by the occurrence of other events such as I/O, etc. See reference [9] for details. Thinking of them as they are described here should lead to no serious misunderstanding for the purposes of this investigation.
11. Two random variables x and y have a bivariate normal distribution with mean vector $\mu = (\mu_x, \mu_y)'$ and covariance matrix Σ if their joint density function is

$$f(x, y) = (2\pi)^{-1}\left(\det \sum\right)^{-1/2} \exp\left\{-(1/2) \times (x - \mu_x, y - \mu_y) \sum^{-1} (x - \mu_x, y - \mu_y)'\right\}.$$

12. Y. Bard, "Applications of the Page Survival Index (PSI) to Vitrual-memory System Performance," *IBM J. Res. Develop.* **19,** 212 (1975), this issue.

Received May 31, 1974; revised December 9, 1974

The author is located at the IBM Data Processing Division Cambridge Scientific Center, 545 Technology Square, Cambridge, MA 02139.