Working Set Analytics

Peter J. Denning

© 2022, Peter J. Denning

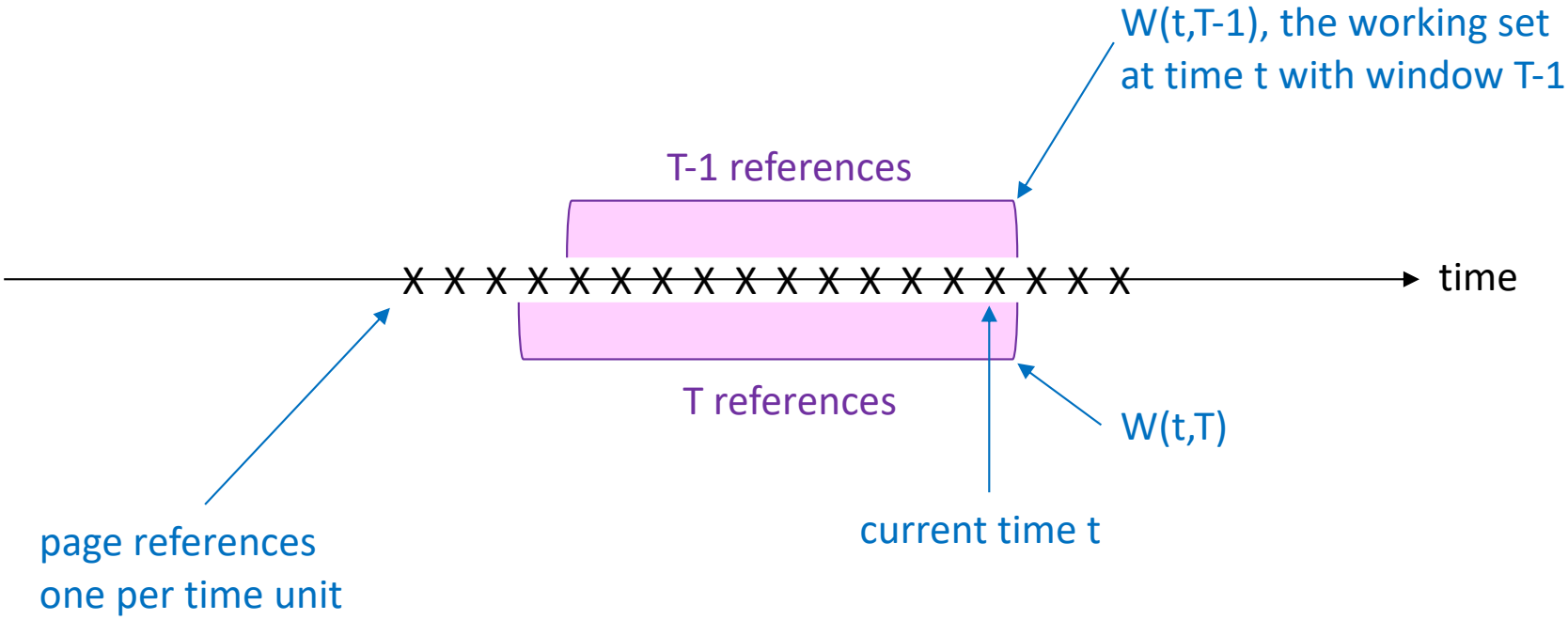
WS Definition

- Informal: smallest set of pages to be loaded in memory with low miss rate
- Formal: set of pages referenced in the interval [t-T-1,t], where window T is the number of references
  - Although the window contains T references, the number of distinct pages (size of the working set) is usually less than T
- From our previous discussion of Principle of Locality, we know that WS is close to optimal -- the formal definition satisfies the informal definition

© Copyright 2022, Peter J. Denning

WS Inclusion Property

Working set satisfies inclusion property on window size, which allows us to use similar methods as before to compute the miss rate for given window T.



© Copyright 2022, Peter J. Denning

Working Set Analytics

- Objective: Fast algorithms for computing working set statistics for a given address trace:
  - m(T), miss rate for window T
  - s(T), mean working set size for window T
- When done, we can plot the pairs (s(T), m(T)) to get a graph that shows the amount of paging for any given mean WS size.

© Copyright 2022, Peter J. Denning

- All statistics derived from Reuse Distribution
  - c(k), number of reuse intervals of length k
- Fast because:
  - c(k) measured in single pass of trace
  - m(T) linear recursion from c(k)
  - s(T) linear recursion from m(T)
- All are order O(N), N = length of address trace
- Faster than algorithms for stack analytics, $O(N^2)$

© Copyright 2022, Peter J. Denning

- The computational algorithms for these statistics measure counts, get rates from dividing by N, the length of the address trace
  - F(T) = miss count, m(T) = F(T)/N
  - st(T) = memory space-time, s(T) = st(T)/N

© Copyright 2022, Peter J. Denning

- Memory space-time is the number of pages in WS times the time they are loaded
  - Let accumulating space-time be ast(t,T) at time t
  - ast(0,T) = 0
  - ast(t,T) = ast(t-1,T)+size(W(t,T))
- Then st(T) = ast(N,T)
- and s(T) = ast(N,T)/N

© Copyright 2022, Peter J. Denning

Reuse interval: time between successive references to same page

First reference: reuse interval is infinite (we write it as “x” rather than “∞”)

Miss (page fault) occurs at time t if and only if the backward reuse interval >T

Total number of faults F(T) = {number of reuse intervals >T}

Miss rate m(T) = F(T)/N, N = length of address trace

© Copyright 2022, Peter J. Denning

r(t):	1	2	3	4	1	2	5	1	2	3	4	5
ru(t):	x	x	x	x	4	4	x	3	3	7	7	5

r(t) is the address trace used previously

ru(t) is the sequence of reuse intervals; “x” marks first references

The histogram of reuse intervals is tabulated below. The fault counts are simply the sum of the c( ) counters for >k

This is the **reuse distribution** c(k) mentioned earlier

Same idea as stack distances in the fixed-space model.

k	c(k)	F(k)
1	0	12
2	0	12
3	2	10
4	2	8
5	1	7
6	0	7
7	2	5
x	5	

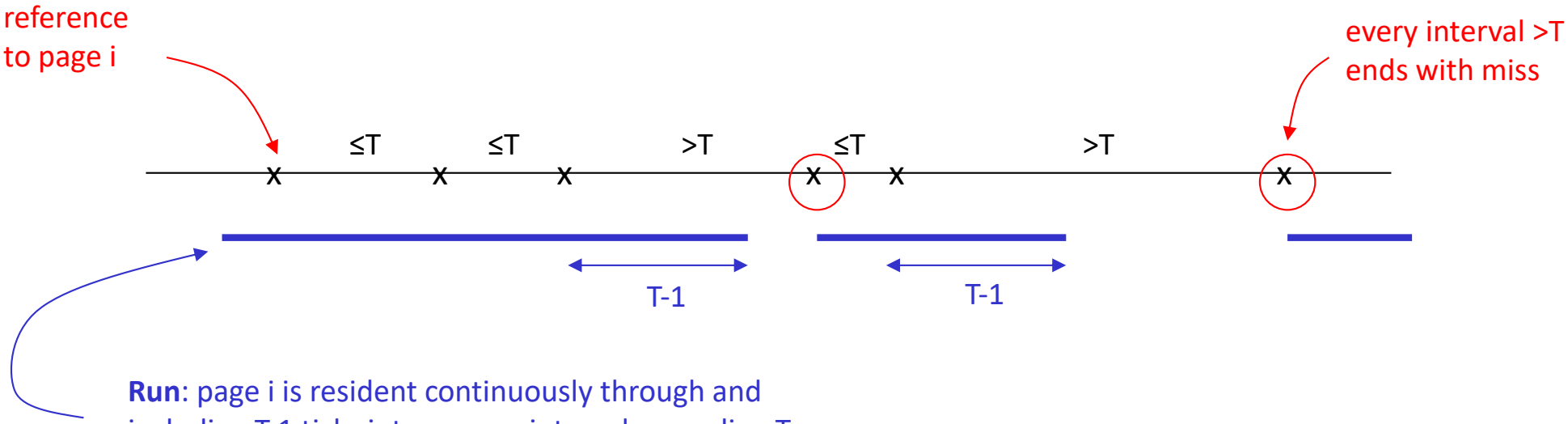
© Copyright 2022, Peter J. Denning

- Computing F(T) is fast. What about s(T)?
- Easiest way to s(T) is measuring space-time st(T)
- This can be done by a series of simulations for each T, each yielding st(T)
- Unfortunately, this is computationally intensive

© Copyright 2022, Peter J. Denning

Run: continuous series of time units where a given page is in WS

Display runs in the simulations



© Copyright 2022, Peter J. Denning

runs: (T=2)

1	2	3	4	1	2	5	1	2	3	4	5
1	1			1	1		1	1			
	2	2			2	2		2	2		
		3	3						3	3	
			4	4						4	4
						5	5				5

s(2)=23/12=1.92

runs: (T=3)

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1		1	1	1	1	1	1		
	2	2	2		2	2	2	2	2	2	
		3	3	3					3	3	3
			4	4	4					4	4
						5	5	5			5

s(3)=33/12=2.75

runs: (T=4)

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3				3	3	3
			4	4	4	4				4	4
						5	5	5	5		5

s(4)=40/12=3.33

runs: (T=5)

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3			3	3	3
			4	4	4	4	4			4	4
						5	5	5	5	5	5

s(5)=44/12=3.67

runs: (T=6)

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3		3	3	3
			4	4	4	4	4	4		4	4
						5	5	5	5	5	5

s(6)=46/12=3.83

These simulations show the mean working set sizes for different window sizes.

Each diagram shows the runs for each page.

Working sets are the vertical columns.

The working set size at time t is the number of entries in the column at time t.

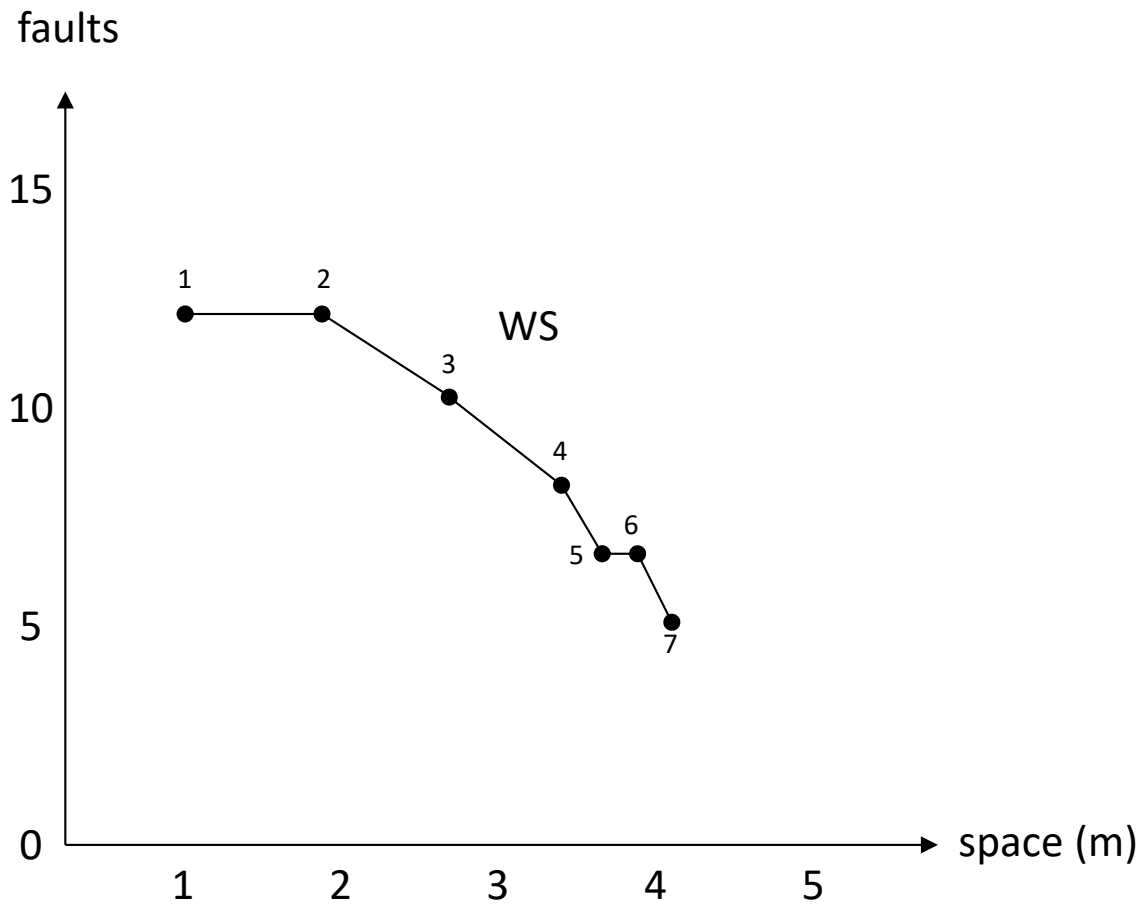
Notice WS size varies over time.

Notice the inclusion property: W(t,T-1) contained in W(t,T)

The mean working set size is total number of non-blank entries divided by N=12.

For T≥7, there are no blank spots in the array (because the longest reuse time is 7 between the two ”3” references)

© Copyright 2022, Peter J. Denning

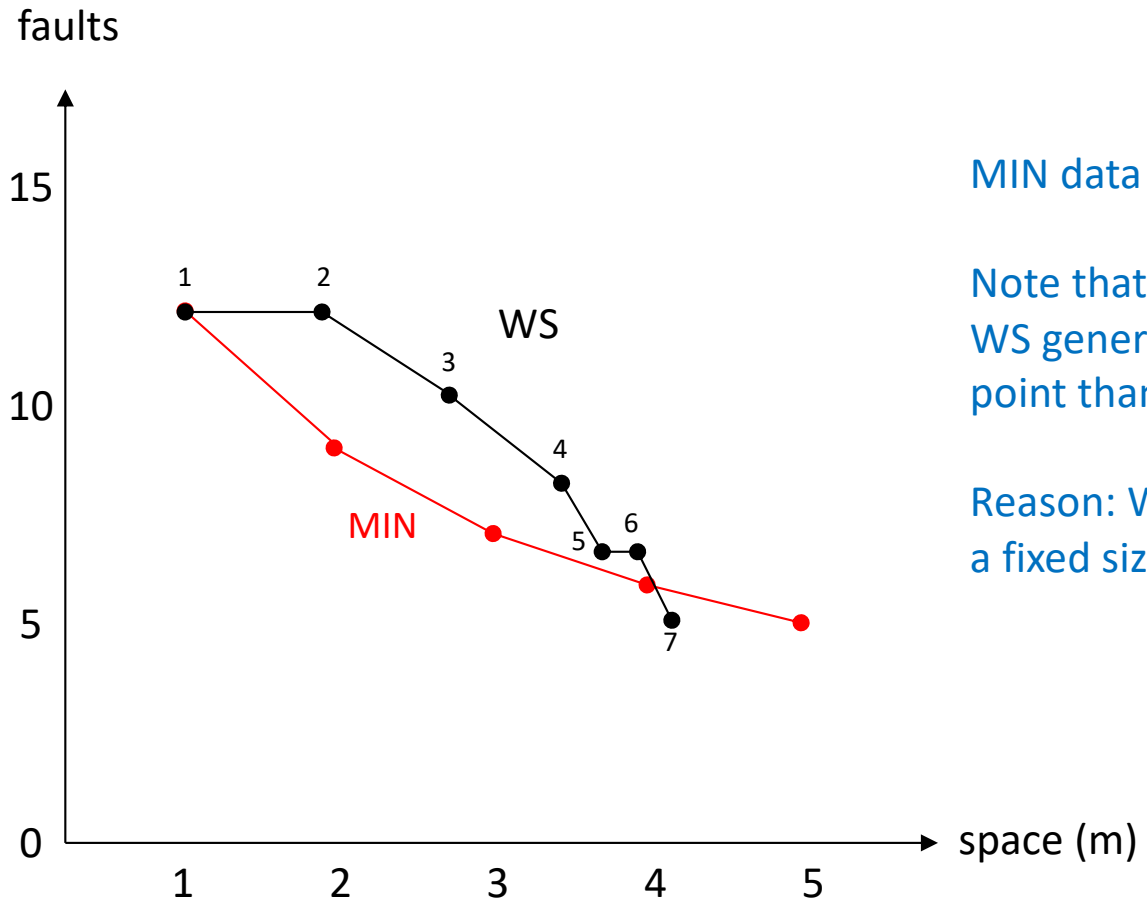


This graph plots the data. The mean working set sizes are points on the space-axis. The corresponding miss counts are points on the faults-axis. The numbers are the window sizes T.

For each T the pair (s(T), F(T)) is plotted. For example, at T=4 the coordinates are (3.33, 8)

The 7 data points are connected with straight lines.

© Copyright 2022, Peter J. Denning



MIN data added to the graph.

Note that at one point (T=7) WS generates a better operating point than (fixed-space) MIN.

Reason: WS is not constrained to a fixed size.

© Copyright 2022, Peter J. Denning

Is there a way to compute the sizes in one pass?

© Copyright 2022, Peter J. Denning

Working set recursion

Objective: use inclusion property to find a fast way to compute the working set sizes.

© Copyright 2022, Peter J. Denning

- Recall definition of **run**: series of reuse references to a page that are ≤T apart terminating with T-1 overhang into a reuse interval >T
- Every run begins with a miss
- Therefore, number of runs is F(T)

© Copyright 2022, Peter J. Denning

- What happens when window size increased by 1?
- We answer for very long address traces only
  - Shorter traces need a few end corrections (see below)
  - End corrections insignificant for long address traces
- When window changes T to T+1:
  - Every run extended by 1 time unit
  - There are F(T) runs
  - Therefore

$$st(T+1) = st(T)+F(T)$$

© Copyright 2022, Peter J. Denning

- This recursion starts with st(1)=N
- Because runs ending with reuse interval T+1 coalese with next run, F(T+1) is smaller than F(T)
- Because our example trace is very short, the recursion overestimates the space-times

actuals

k	c(k)	F(k)	st(k)	
1	0	12	12	12
2	0	12	24	23
3	2	10	36	33
4	2	8	46	40
5	1	7	54	44
6	0	7	61	46
7	2	5	68	48
x	5			

© Copyright 2022, Peter J. Denning

- How close is WS to optimal?

- Intuitively, tracking locality sets is optimal.

© Copyright 2022, Peter J. Denning

Optimal Variable Space Policy (VMIN)

- What is the best we can do if we allow the policy to vary the space?
- Better than MIN because the fixed-space constraint is removed.

© Copyright 2022, Peter J. Denning

- VMIN = variable space MIN (optimal)
- Parameter T, window size as in WS
- Now the window looks **forward**
- At **each reference** to a page:
  - **retain** page till next reference if forward time to next reuse is ≤ T.
  - otherwise **immediately remove** the page.
- Since criterion for retaining is identical to WS,
  - WS and VMIN have identical miss sequences and miss counts F(T) for given window size T

© Copyright 2022, Peter J. Denning

Here's why VMIN is optimal:

- Let A = cost of page fault, expressed in space-time units
  - Example: disk service time is 10 ms and overall mean working set size is 100 pages, so A = 10^{-2} ms x 10^{2} pages = 1 page-second
  - And if one memory tick is 10^{-9} sec, A = 10^{9} page-ticks
- Let B = cost of keeping referenced page resident until next reuse
  - Example: current page used at time t and next at t+10, so B=10 page-ticks at time t
- For any policy P, the total cost is the sum of the costs for each reference. VMIN is the P that minimizes every reference cost.
- VMIN = Chose lower cost: retain till reuse, or replace immediately. Retaining for part of reuse interval and deleting before reuse increases cost. Thus the VMIN rule is
  - If A > B, retain in memory until next reuse
  - If A ≤ B, remove immediately

© Copyright 2022, Peter J. Denning

r(t):	1	2	3	4	1	2	5	1	2	3	4	5
	1	1	1	1	1	1	1	1	1	1	1	
runs:		2	2	2	2	2	2	2	2	2	2	2
(T=4)			3	3	3	3				3	3	3
				4	4	4	4				4	4
							5	5	5	5		5
ru(t):	x	x	x	x	4	4	x	3	3	7	7	5

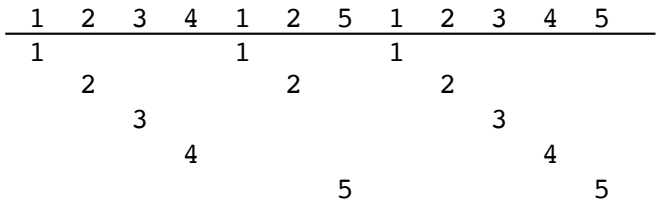
VMIN mean size:
v(4)=22/12=1.83

The runs chart for VMIN illustrates for T=4. It is similar to that for WS, except that VMIN does not hold a page into a long reuse interval.

VMIN has the same page fault sequence but a lower mean memory size than WS. The red letters show the contributions to WS space-time that VMIN eliminates.

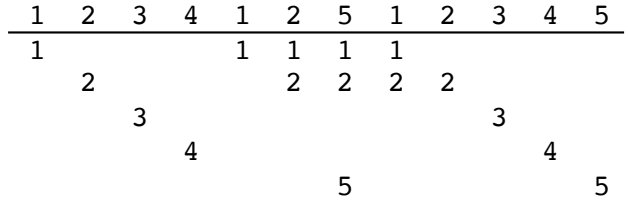
© Copyright 2022, Peter J. Denning

runs: (T=2)



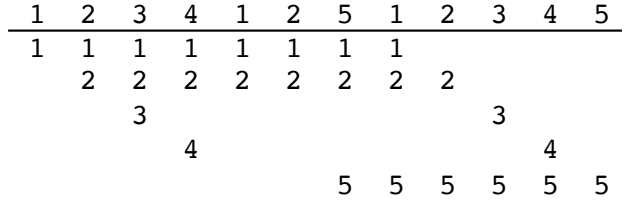
v(2)=12/12=1.00

runs: (T=3)



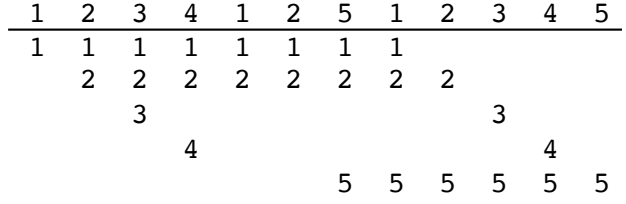
v(3)=16/12=1.33

runs: (T=5)



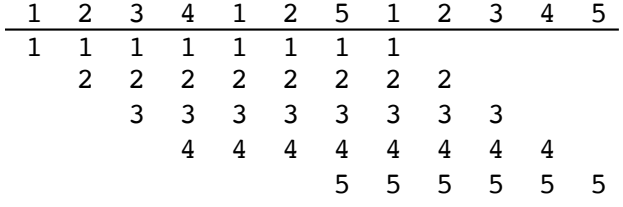
v(5)=26/12=2.17

runs: (T=6)



v(6)=26/12=2.17

runs: (T=6)

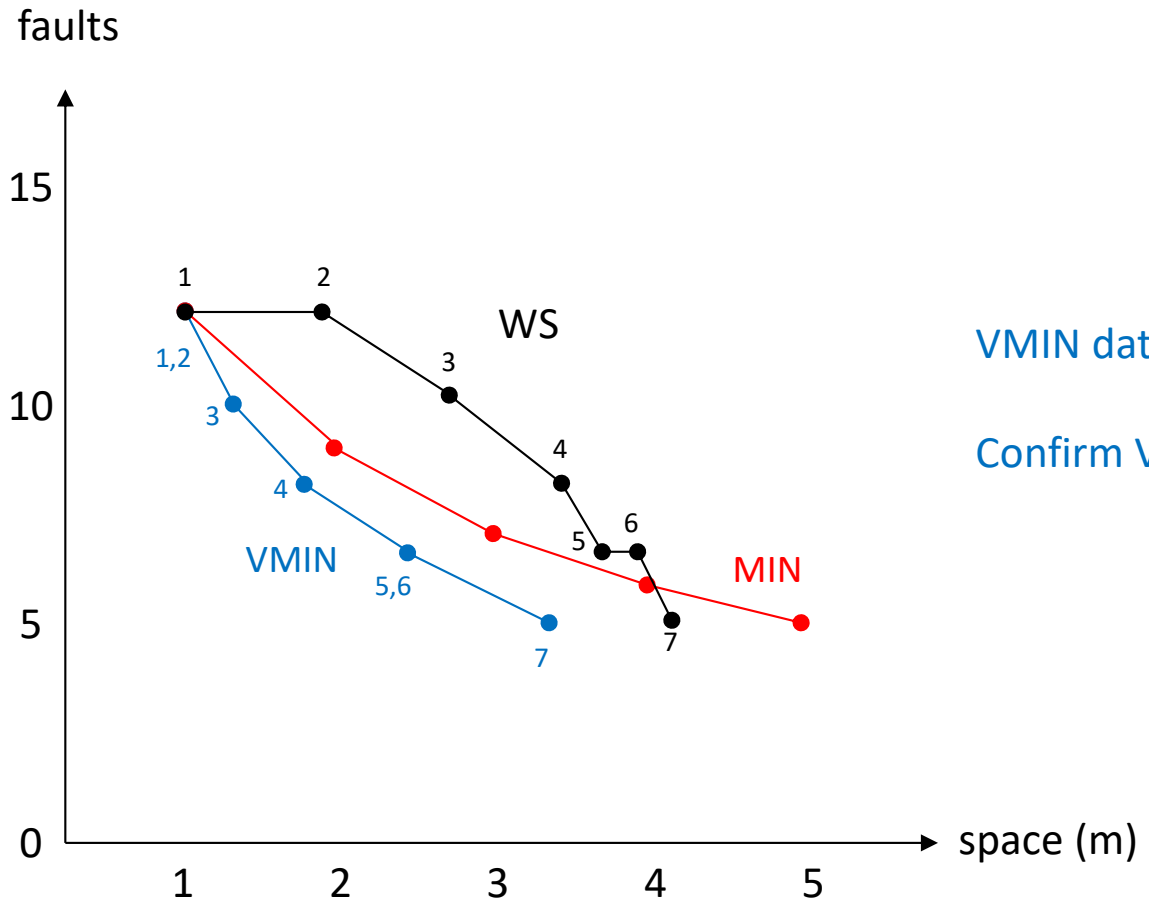


v(7)=38/12=3.17

If we augment the T=4 simulation with other simulations for the other T values, we get this series of diagrams.

The mean space used by VMIN is vt(T).

© Copyright 2022, Peter J. Denning



VMIN data added to the graph.

Confirm VMIN is better than MIN.

© Copyright 2022, Peter J. Denning

Calculating VMIN without simulation:

From the memory charts of WS and VMIN we see that the space-time difference is due to **overhang** of WS in each reuse interval >T.

The number of such intervals is F(T).

The overhang in each such interval is (T-1).

The total overhang is (T-1)F(T)

The difference between VMIN and WS space time is the total overhang:

$$vt(T) = st(T) - (T-1)F(T)$$

This works for long address traces; short traces need end corrections.

© Copyright 2022, Peter J. Denning

If the program has long phases and T is set less than the phase lengths, then VMIN = WS during most of each phase.

VMIN's advantage: VMIN unloads old pages before entering the new phase, while WS retains old pages for a while after entering the new phase (overhang). See diagram on next slide.

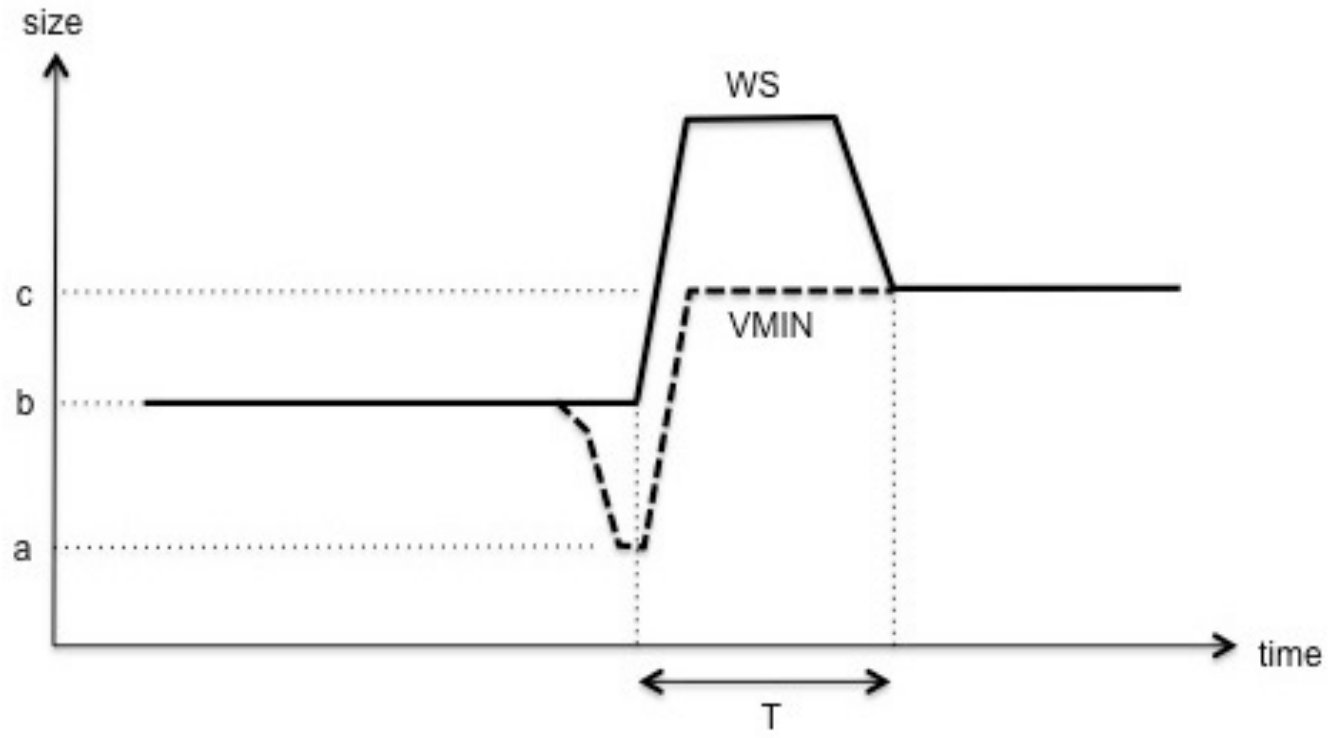
Each pixel of page reference map represents a packet of space-time (= one page times length of sample window)

For a phase of L pixels, VMIN=WS for L-2 of them. In the most extreme case, VMIN has no space-time packet in the first and last windows and the WS has an extra packet of overhang in its first window. And so the ratio of WS/VMIN space-time is < (L+1)/(L-2).

For the page reference map shown earlier L is around 200 for most phases and the ratio is less than 1% difference.

Therefore, because of principle of locality, WS is nearly the same as VMIN.

© Copyright 2022, Peter J. Denning



WS and VMIN are identical except at transitions between phases

What is the difference in space-time between them?

The difference is due partly to WS "overhang" at phrase begin and to VMIN unloading old locality pages at phase end

© Copyright 2022, Peter J. Denning

When it's all put together …

Load controlled WS job scheduling prevents thrashing and generates system throughput within a few percent of optimal VMIN

© Copyright 2022, Peter J. Denning

finis

© Copyright 2022, Peter J. Denning