TANDEMCOMPUTERS

The 5 Minute Rule for Trading Memory for Disc Accesses and the 5 Byte Rule for Trading Memory for CPU Time

Jim Gray
Franco Putzolu

Technical Report 86.1
May 1985, Revised February 1986
PN87615

THE 5 MINUTE RULE FOR TRADING MEMORY FOR DISC ACCESSES
and
THE 5 BYTE RULE FOR TRADING MEMORY FOR CPU TIME

Jim Gray
Franco Putzolu

Tandem Computers, Cupertino, CA, USA

Technical Report TR86.1

May 1985
Revised February 1986

ABSTRACT

Simple cost-benefit arguments allow one to compute the break even point for trading central-memory residence against disc accesses for data. If an item is accessed frequently enough, it should be main memory resident. In current technology, "frequently enough" means about every five minutes.

Along a similar vein, one can frequently trade memory space for cpu time. For example, bits can be packed in a byte at the expense of extra instructions to extract the bits. A simple arguement shows that one can spend five bytes of main memory to save one instruction per second.

THE FIVE MINUTE RULE

In Psychology, the answer is always 5±2. In Physics, the answer is always transcendental. In Digital computing the answer is always a multiple of 5 -- for example, how many fingers and toes do you have? In all fields, the problem is to find the question.

One interesting question is: When does it make economic sense to make a piece of data resident in main memory and when does it make sense to have it resident in secondary memory (disc) where it must be moved to main memory prior to reading or writing?

In some situations, response time dictates that the data be main-memory resident because disc accesses introduce too much delay. These situations are rare. More commonly keeping data main memory resident is purely an economic issue. When is it cheaper to keep a record in main memory rather than access it on disc? For high-end systems of the 1980's the answer is:

> THE FIVE MINUTE RULE
>
> Pages referenced every five minutes should be memory resident.

The argument goes as follows: A Tandem disc, and half a controller comfortably deliver 15 accesses per second and are priced at 15K$ for a small disc and 20K$ for a large disc (180Mb and 540Mb respectively). So the price per access per second is about 1K$. The extra CPU and channel cost for supporting a disc are 1K$/a/s. So one disc access per second costs about 2K$ on a Tandem system.

A megabyte of Tandem main memory costs 5K$, so a kilobyte costs 5$.

If making a 1Kb record resident saves 1a/s, then it saves about 2K$ worth of disc accesses at a cost of 5$, a good deal. If it saves .1a/s then it saves about 200$, still a good deal. Continuing this, the break even point is an access every 2000/5 ~ 400 seconds.

So, any 1KB record accessed more frequently than every 400 seconds should live in main memory. 400 seconds is "about" 5 minutes, hence the name: the Five Minute Rule.

For smaller records, the break even point is longer (1 hour for 100 byte records) and for larger records the break even point is shorter (2 minutes for 4K records).

At a certain point the record size exceeds the disc transfer size. For example, page-faulting a 100K program requires twenty five 4K disc reads. So above the transfer size (4K in Tandem's case) one must use the rule for the transfer size (2 minutes in Tandem's case).

A more formal derivation and statement is:

Let:

RI: expected interval in seconds between references to the page.

M$: be the cost of a byte of main memory ($/byte)

A$: be the cost of a disc access per second ($/a/s)

B: The size of the record/data to be stored in bytes.

Bmax: be the maximum transfer size of the disc in bytes.

Then, assuming B < Bmax, the savings in dollars of keeping the record B main memory resident is:

$$\frac{A\$}{RI} - M\$*B$$

At the break even point, this expression is zero. Solving for RI gives

$$RI = \frac{A\$}{M\$*B}$$

Substituting for the Tandem numbers:

$$RI = \frac{400,000}{B} \text{ seconds}$$

Plotting this:

```
              |       *
              |         *
        10000 +           *
              |             *
Break even    |               *
Reference     |                 *
Interval      |                   *
In       1000 +                     *
Seconds       |                       *
              |                         *
              |                           *
              |                             *
          100 +                               * * * * * *
              |
              |
              |
              |
           10 +-----+---------+---------+---------+------
                   10        100       1000     10000
                 Memory Resident Block Size in Bytes
```

A log-log graph of the break even reference interval (RI) vs the size (B) of the block being stored in main memory. This is computed for 1986 Tandem disc and memory prices. Notice that beyond Bmax (the maximum disc transfer size) the block size does not affect the reference interval.

As can be seen from this, the Five Minute Rule only approximates a particular region of the curve: B above 1K. Using the five minute rule anticipates the advent of cheaper memory. In the last year both disc prices and memory prices have declined 30%. The resulting ratio remained unchanged. In the near future 1Mbit memory chips will improve memory prices faster than the the price/performance improvements of next generation discs and processors. So, in the near future, the Five Minute Rule will apply for all block sizes.

The Five Minute Rule also seems to apply to IBM systems (prices are uniformly higher for IBM 30XX machines and about the same for IBM 43xx machines) and to mini-computers (where everything is uniformly less expensive).

The Five Minute Rule does not apply to personal computers for two reasons. First, one cannot add and subtract discs from PCs and workstations with the same freedom. Typically, one has the choice of zero or one hard disc. Second, memory and disc economics are different for PCs -- a hard disc costs the same as a megabyte of main memory.

The following case study illustrates an application of the Five Minute Rule. A customer wanted to keep his entire 500Mb database main memory resident. The following argument convinced him to adopt a hybrid disc-memory design.

The application transactions are all quite simple. Almost all the transactions accesses a single record and demand one second average response time. The transaction uses 80ms of cpu and 30ms of disc time. The application has a 600 transaction per second peak load.

In the all-in-main-memory design, the system needs about 60 TXP processors, each with 10MB of main memory. Two mirrored discs store the database, its indices and the programs. The discs are idle during normal operations since the system is memory resident. The average transaction has 150ms response time.

The all-on-disc design uses about 60 TXP processors, each with 2 MB of memory (a 380MB ~ 1.9M$ savings over the main memory system), but it uses 40 spindles (20 mirrored volumes) of disc (a .9M$ extra cost over the main memory design). At 80% cpu utilization and 50% disc utilization we estimate the average response time as 300ms, well within the 1 second limit. (This estimate is based on the 1/(1-u) multiplier familiar to queueing theory). This disc based solution does the job and is 1M$ cheaper than the main memory design.

The Five Minute Rule can be used to decide on an "optimal" disc-memory tradeoff. The 80-20 rule implies that about 80% of the accesses go to 20% of the data, and 80% of the 80% goes to 20% of that 20%. So 64% of the accesses go to just 4% of the database. Keeping that 4% of the database in the main memory disc cache saves 64% of the disc accesses over the all-on-disc design. The remaining 7 mirrored disc volumes each store 90MB deliver about 15 ios per second. This design saves 26 disc arms -- 390K$. The extra memory (24Mb) costs 120K$. This is a net 270K$ savings over the all-on-disc design and a 1.27M$ savings over the all-in-main-memory design.

The application's database reference string can be used with this logic to compute the optimal size of disc cache and optimal number of disc arms.

This Five Minute Rule applies equally well to virtual memory management. If a virtual memory page (typically 4K bytes) is referenced every 2 minutes, it should stay memory resident. Hence a

CLOCK virtual memory algorithm should be given enough memory to cycle once every minute at peak loads or one should try to detect such "hot" pages (using a 2 minute history string) and treat such "hot" pages specially.

THE FIVE BYTE RULE

Changing topics, another interesting question is: "When does it make economic sense to use more memory to save some cpu power?", or conversely save some memory at the expense of some cpu cycles? This issue comes up in code optimization where one can save some instructions by unwinding loops, and in data structure design where one can pack data at the expense of masking and shifting operations to extract the data.

The logic is quite similar to the Five Minute Rule. One picks a certain price for memory (say 5K$/MB) and a certain price per MIP (say 50K$/MIP). This means that 5 bytes cost about .005$. Similarly one instruction per second costs about .005$. So 5 bytes costs about as much as 1 instruction per second. This gives the rule:

> **THE FIVE BYTE RULE**
>
> Spend 5 bytes of main memory to save 1 instruction per second.

The Five Byte Rule is applied as follows:

1. I := How many instructions are saved by the new design.
   F := How frequently the instruction sequence is executed.
   The product of I and F is the instruction savings. It will be negative if the design adds instructions. This tells the MIP savings (cost) of the change.

2. S := How many bytes are saved by the new design.

3. Using the Five Byte Rule convert S from bytes to MIPS by dividing by 5. So the MIP savings of S is S/5.

4. Now compare the designs, the net savings is

   $$I * F - S / 5$$

   If it is a large positive number, then the new design provides a large savings.

As an example, suppose there is a sequence like:

```
LOAD         BYTE
MASK         FLAG
BRANCH ON    NONZERO
```

in the dispatcher. Suppose the dispatcher is invoked 1000 times each second.

* If flag were stored as a byte it would avoid the mask step and hence save 1000 instructions per second.

- This translates to about 5000 bytes of storage based on the Five Byte Rule.

- If flag were stored as a byte it would use eight times the storage. If there are 100 processes in the processor, this translates to about 90 extra bytes.

- Since we have a 5000 byte budget, this is a profit of 4910 bytes. A good trade -- a 50:1 return on investment.

On non-RISC machines, the MASK instruction may use 2 micro-clocks while the average instruction uses 6 micro-clocks. In this case, one needs to weight the saved instructions with their micro-clock cost. That is, in the example above, would save only 2/6 of an instruction each time we saved a MASK step. So the "real" savings on the hypothetical non-RISC machine would be only (50*(2/6)):1 ~ 18:1. Still a good deal.

Distributed by
TANDEMCOMPUTERS
Corporate Information Center
19333 Vallco Parkway MS3-07
Cupertino, CA 95014-2599