

Re: NT: son of VMS? (was Re: Portents of VMS death)

Source: <http://unix.derkeiler.com/Newsgroups/comp.os.vms/2003-05/2787.html>

From: Bill Todd (billtodd_at_metrocast.net)

Date: 05/26/03

Date: Sun, 25 May 2003 20:56:25 -0400

"Paul Sture" <p_sture@elias.decus.ch> wrote in message
news:H4lj5Ghx3MEB@elias.decus.ch...

> *In article <qMOdnfwqZcQmuU2jXTWcpA@metrocast.net>, "Bill Todd"*
<billtodd@metrocast.net> writes:

> <snip>

>

>

> > *NT probably does have a somewhat closer relationship to VMS than what*
I'd

> > *write for a record manager today would have to RMS,*

>

> *Out of curiosity, what features would you put into a new RMS, and is there*

> *anything in particular you would drop? As this is a theoretical question,*

> *I'm not especially concerned with backwards compatilby.*

Where to begin? At least in OSs where crossing the application/system interface is not unreasonably expensive, a record manager lends itself better to a Unix-style system where each record operation is handled by the OS using OS buffers, so VMS would not be the ideal platform on which to create it. Once you have that, you have protection for the internal structure of the files (users can't modify it directly; copy operations are done by system functions, which is safer than having applications do them anyway); you can also perform (and protect) updates using a transaction log, which allows record operations to be made persistent in a single log write (even multiple operations can be captured in a single log write if synchronous operation isn't requested and the applications use Flush-like mechanisms to establish consistency points) and things like index updating can safely be batched and performed lazily in the background (so can on-line reorganization) – you can even use pseudo-log-structured storage to batch multiple bucket updates into a single disk access (which makes the presence of a background on-line reorg mechanism more important, and also allows bucket compaction – or even full-fledged compression, though that's more expensive – such that partially-filled buckets occupy no more space on disk than their data requires), though this starts tying into underlying file-system changes that I'm working on. The presence of the transaction

log also facilitates user-level transaction support, though if *long* transactions are to be supported things can get considerably more complex.

I think I'd get rid of RFA mechanisms in indexed files: the permanent indirection and maintenance overheads don't seem justifiable. RMS-32 may even have implemented no-RFA single-key indexed files at one point - I know I discussed it with them. As a substitute, it's possible to define a flexible *unordered* container structure supporting permanent RFAs that never need get indirected, so multi-key files are still feasible, they just don't have a primary index but only secondary indexes. Sequential access in physical order is still fast, and reorganization into a preferred physical order is possible as long as the RFAs are only used internally (by the alternate indexes) rather than by applications. Another option is to use (unique, or if necessary 'uniquified') primary key values as permanent record identifiers (as I think NonStop SQL does) that alternate indexes can use for access.

Some kind of relatively simple support for 1:many parent:child relationships (supporting hierarchical structures) would be nice, likely implemented by physical sequencing within a primary data level to provide good access clustering. If you don't want the parent clustered with its children, then you can just implement the structure within the application using key-based pointers into separate data sets and suitable enclosing transactions for updates, though it might be nicer for the system to support that transparently within a single 'file' containing multiple record types in multiple data sets and managing the linkages among them. Record types should be able to evolve on line, with newer versions able to coexist with the old ones (which would be updated if/when their new fields got used). Some of this stuff starts encroaching on database territory, but only selectively: mostly, it offers a simple navigational alternative to the overheads and inefficiencies of a full-blown relational model upon which some higher-level constructs (like *simple* view mechanisms) could be layered.

There's a ton of complexity in RMS aimed at '70s-era speed and space optimizations that just aren't important any more, so out they go. 64 KB is too small a limit on bucket size, and bucket overflow mechanisms should remove all limits on record size (large records should be piece-wise-accessible like files are; limiting key size to 1 KB or so remains reasonable and useful, though).

That's what comes to mind off the top of my head, anyway. I'm sure there's a lot more beneath the surface. Feel free to offer suggestions or observations.

- bill