

RMS Tuning in 60 minutes

"Become a star in just one hour"

thilo.lauer@compaq.com

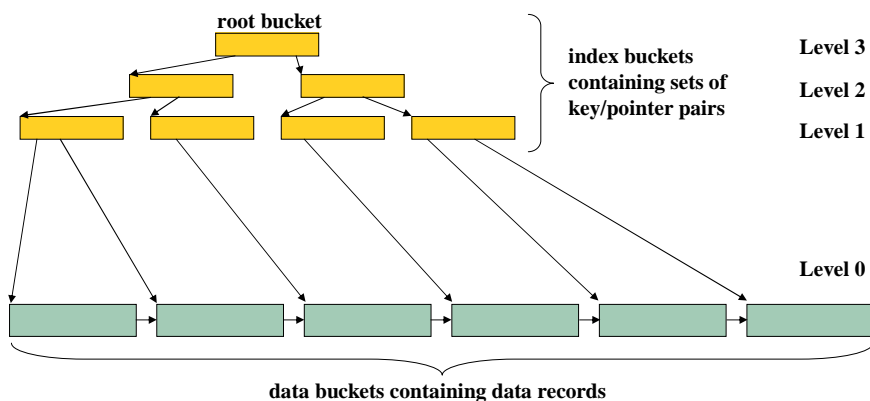
Topics

- Make up your mind!
- "What's in a name?" - some terminology
- Learn application behaviour
- Use the right tools
- The RMS Tuning Cookbook
 - things to try blindly
 - sophisticated techniques

Make up your mind

- RMS doesn't use magic (yet)
- RMS behaviour is predictable
- most changes are transparent to applications
- many misconceptions floating around
- you don't even need to understand it - just do it!
- it's open-ended: you want more? Give us a call!

Some Terminology



A bucket is the unit for any I/O performed by RMS!

Learn application behaviour

- hot files
- read/write (update, delete) ratio
- file growth rate
- locality of new records
- key patterns
- shared access, number of users
- when does performance suffer?
- gather metrics

Use the right tools

- OpenVMS-supplied
 - \$MONITOR (/RMS) to analyze I/O behaviour
 - \$ANALYZE/RMS (/FDL, /STATISTICS) to analyze a file's permanent attributes
 - \$EDIT/FDL to change internal file attributes
- Other tools (from OpenVMS freeware CD)
 - RMS_STATS: similar to \$MONITOR, but displays raw numbers, more details
 - SIDR: shows count and values of duplicate keys

The RMS Tuning Cookbook

- Apply CONVERT regularly
- Increase bucket size
- Enable Global Buffers for shared files
- Evaluate compression settings
- Watch out for duplicates
- Adjust fill factor for growing files
- Consider removing secondary keys

Apply CONVERT regularly

`$CONVERT input-file output-file`

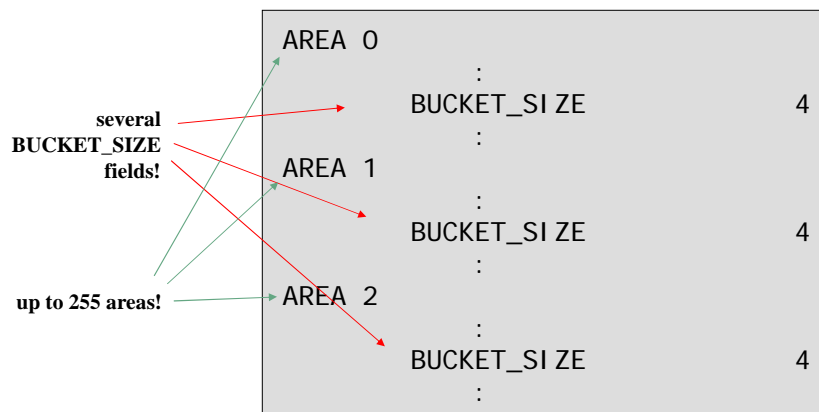
- reorganizes internal layout
- removes internal defragmentation
- (removes external defragmentation)
- removes levels of indirection (RRVs)
- removes deleted records

Increase bucket size - steps

- 1 change bucket size parameter in FDL file
 - let OpenVMS be smart:
`$EDIT/FDL/NOINTERACTIVE fdlfile.FDL`
 - be smart yourself:
calculate better values by hand and edit the FDL-file manually
- 2 convert the file with the new FDL file
 - `$CONVERT/FDL=fdlfile infile outfile`

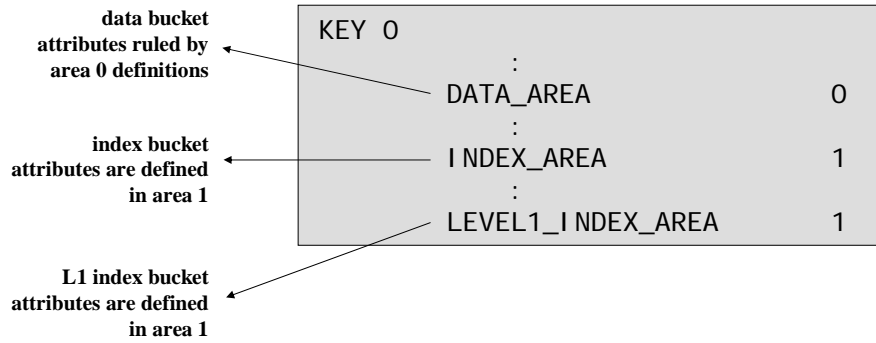
Increase bucket size - what to change

- which field should i change?



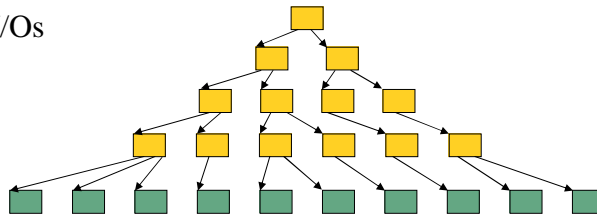
Increase bucket size - ...what to change

- get the area information from the key definition:

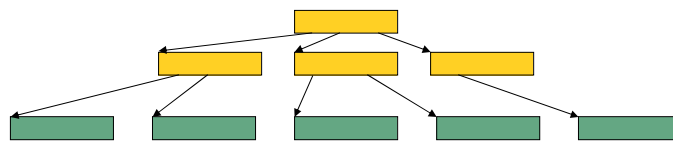


Increase bucket size - potential effect

old: 5 I/Os



new: 3 I/Os



Increase bucket size - real effect?

To judge the effectiveness of your changes, examine the index depths of the resulting file:

```
$ pipe analyze/rms/statistics newindexfile | -  
_ $ search sys$input/window=(0,2) "STATISTICS FOR KEY"  
  
STATISTICS FOR KEY #0  
  
Number of Index Levels: 4
```

This number should have decreased compared to the original file!

Enable Global Buffers for shared files

\$SET FILE/GLOBAL_BUFFER=n filename

- probably the easiest RMS tuning method with the highest effects!
- forget all rumours about global buffers
- forget former pitfalls
- adjust GBLPAGES, GBLPAGFIL (dynamic since V7.1)
- adjust GBLSECTIONS

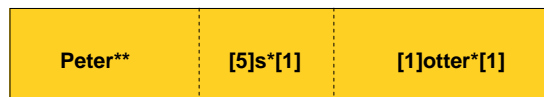
Evaluate compression settings

- concentrate on INDEX_COMPRESSION (FDL)
- default has been ON for historical reasons (disk space), changed in recent OpenVMS versions
- performance-wise, OFF would be better choice, but watch out for introducing additional index levels!
- easy as well:
 - change parameter in FDL file
 - perform CONVERT/FDL
 - enjoy!

Evaluate compression settings - effect

traversal of index bucket

with compression: sequential search, performance: $N/2$



500 entries:
~250 lookups

without compression: binary search, performance: $\log_2(N)$



500 entries:
~9 lookups!

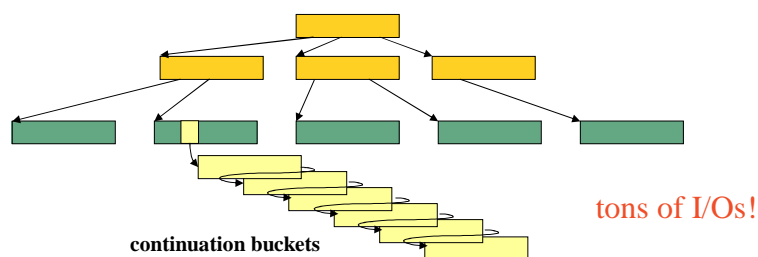
Result: less CPU overhead!

Evaluate compression settings

Recent performance tests have shown only marginal CPU-time improvements (<5%). Now, is this "improvement" worth thinking about index compression at all?

- Yes: for specific files with large index buckets (>30?), short keys (<10?), and a deep index (>2?).
- Having plain index entries might ease potential troubleshooting.
- Disabling index compression leads to increased space usage in the index, which might introduce another level!

Watch out for duplicates



- duplicate records are not maintained in the index
- sequential search through continuation buckets
- RMS preserves order of arrival: most recent record is at the end
- insertion of new duplicate is very expensive
- duplicates are often your most-frequent key value (default value?)

Watch out for duplicates - real-life example

Output of SIDR (file has ~46000 records):

Duplicate count	Buckets	Key value
17824	19	
2982	4	OHNE
1055	1	ohne
173	1	-
139	1	ENTF. LLT
132	1	ENTFAELLT
127	1	ENTF. LLT
82	1	ENTF.
66	1	noch ohne
65	1	OHNENR.

Watch out for duplicates - solution

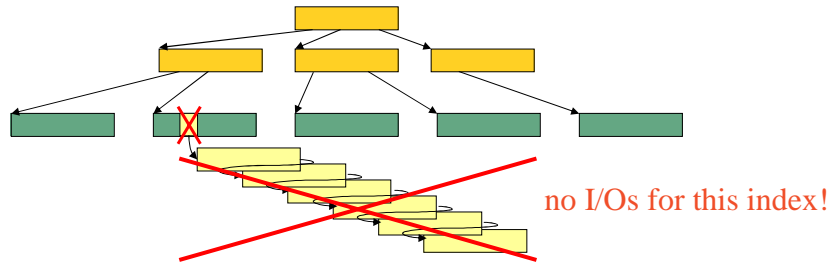
In FDL file: define SPACE character as NULL key value

```

KEY 3
  CHANGES                yes
  DATA_KEY_COMPRESSION  no
  DATA_AREA              2
  DATA_FILL              100
  DUPLICATES              yes
  :                       :
  LEVEL1_INDEX_AREA      2
  NULL_KEY                yes
  NULL_VALUE              ' '
  SEGO_LENGTH             13
  SEGO_POSITION           40
  TYPE                    string

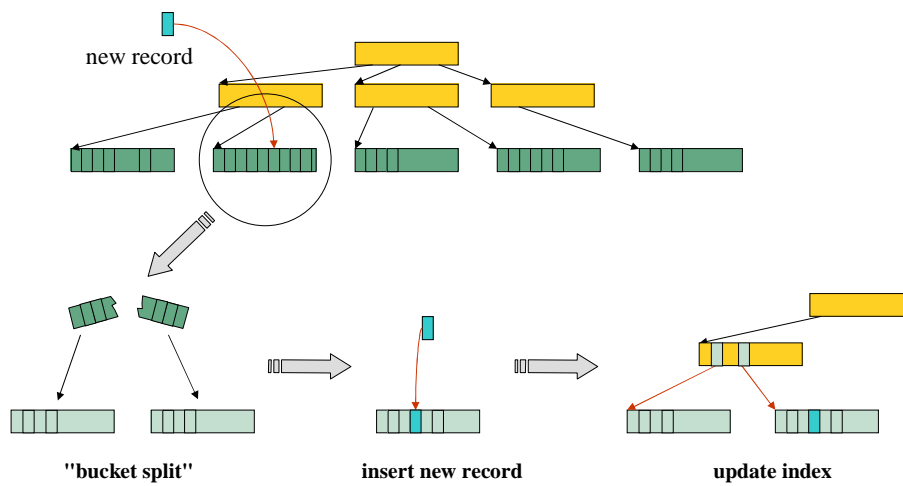
```

Watch out for duplicates - effect



- this particular key value is not maintained in the index
- records with this key value are not seen via indexed access
- you may have to implement alternate lookup method

Adjust fill factor for growing files - problem

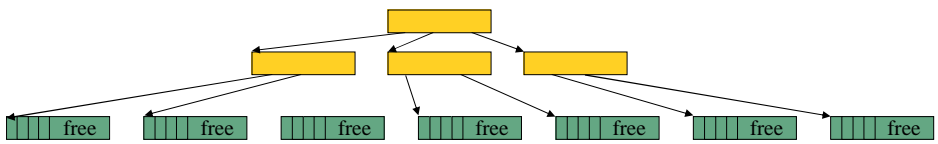


Adjust fill factor for growing files - solution

In FDL file: define Fill Factor for relevant areas

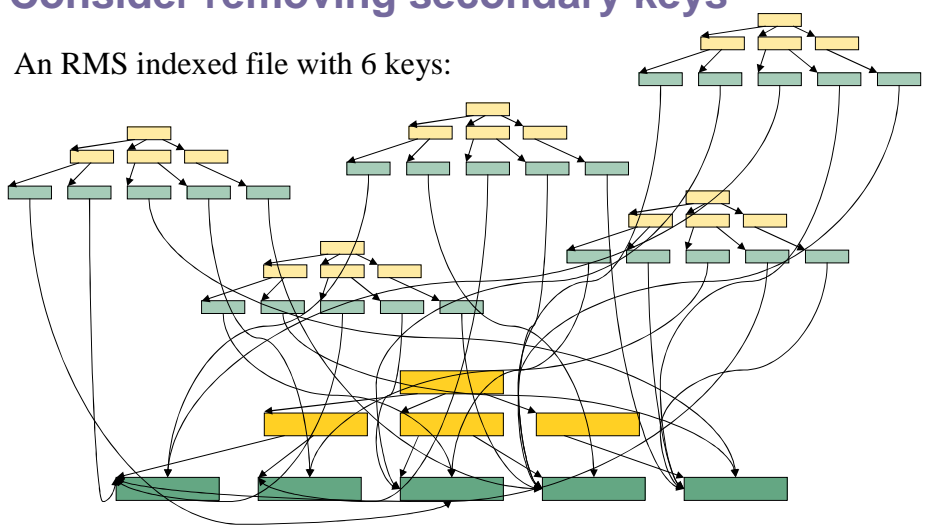
```

KEY 3      :
          DATA_FILL      50
          :
          LEVEL1_INDEX_AREA 2
          NULL_KEY        yes
          NULL_VALUE      ,
    
```



Consider removing secondary keys

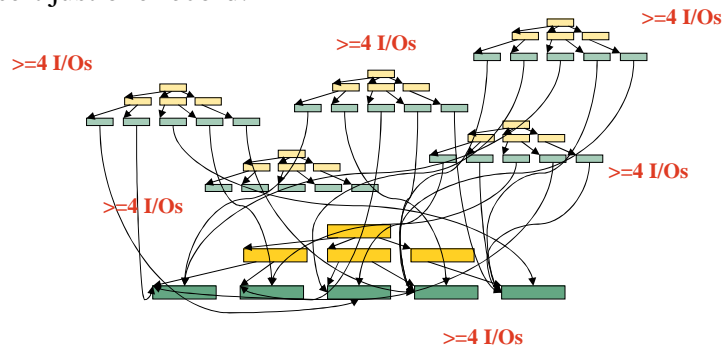
An RMS indexed file with 6 keys:



All this chaos works perfectly - but slowly!!

...Consider removing secondary keys

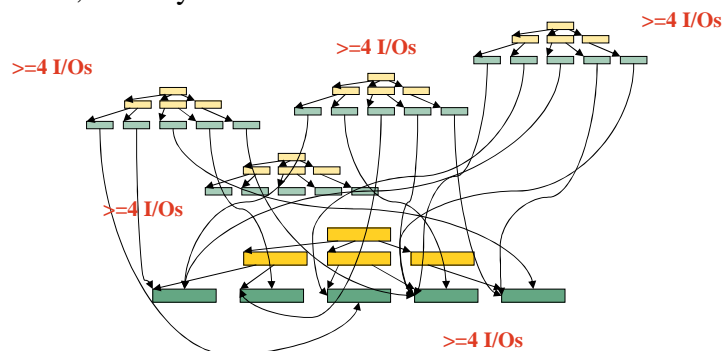
insert just one record:



Result: about at least 24 I/Os in optimum case, often much more!

...Consider removing secondary keys

same file, one key removed:



Removal of one index reduces administrative work (I/O+CPU)

...Consider removing secondary keys

Issues:

- application designers often implement keys 'just to have them handy'
- example: two identical key fields with different sorting order
- often secondary keys have poor 'duplicate behaviour'
- evaluate application impact!
- substitute keyed lookup by other methods
- You may need to change the application!

What if all that doesn't help?

- Visit T2G14 on friday:
"OpenVMS/RMS - Grundlagen, Internas, Tuning"
- Think about employing an RMS DB administrator
- Ask Compaq: we sell more than just PCs!
- (RMS) Engineering is open to all sorts of problem reports, enhancement requests, discussions...

The Compaq logo is centered within a red horizontal band. The word "COMPAQ" is written in a white, italicized, sans-serif font.

Questions...

thilo.lauer@compaq.com