Scalable Access to Large Tables using STIL

Mark Taylor, Bristol University
m.b.taylor@bristol.ac.uk

Initial work: Clive Page
More detail: VOTech DS6 Study Report sec. 5.5.1
(see DS6StudyReport on VOTech wiki: http://tinyurl.com/27cbk8)

Outline

Techniques

- Column-oriented access
- File mapping

Implementation

Data and results

Possible applications

Column-Oriented Access

Two obvious ways to store a table on disk:

row-oriented: store all of row 1, then all of row 2, . . .

column-oriented: store all of column 1, then all of column 2, . . .

Most existing table formats are row-oriented

- FITS, CSV, VOTable, nearly all RDBMS, . . .

The two are efficient for different things (especially with wide tables)

- row-oriented for all columns from a number of rows
  - ▷ selection of indexed rows, reorder rows in table
- column-oriented for all all rows from a number of columns, . . .
  - ▷ plots, selections from unindexed columns, full-table statistics on a column, . . .

File Mapping

File Mapping can be used as an alternative to seek/read type I/O

- Unix `mmap(2)`
- Java `java.nio.MappedByteBuffer`

Data accessed as if in in memory, read from disk on demand by OS

Several advantages:

- Instant "load"
- Somewhat faster data transfer (factor of 2?)
- Minimal penalty for random access to small items
- OS handles disk block caching

Potential OS-related issues

- Address space may become scarce for multi-Gb files on 32-bit OS

Implementation in STIL

Column-oriented formats defined, based on FITS

`colfits-basic`: 1-row BINTABLE, each cell holds *nrow*-element vector

`colfits-plus`: same, with metadata attached as VOTable in primary array HDU

- These are legal (if somewhat perverted) FITS files

STIL I/O handlers written for these formats

- STIL architecture means TOPCAT, STILTS etc can read/write these formats
- No changes to applications required
- Only difference is performance

Only really worth using for large datasets

- STIL already quite fast for $\sim$ 100 Mbyte tables

Data

Need a lot of data to test scalability

Used 2MASS supplied on 10×DVDs:

PSC: Full Point Source Catalogue

- 470,992,970 rows × 61 cols ≃ 111 Gbyte
- physical memory < single column

PSC_B: Northern-hemisphere Point Source Catalogue

- 177,756,896 rows × 61 cols ≃ 42 Gbyte
- physical memory > single column

XSC: Full Extended Source Catalogue

- 1,647,599 rows × 391 cols ≃ 2.2 Gbyte
- physical memory ≈ many columns

XSC_B: Northern-hemisphere Extended Source Catalogue

- 908,817 rows × 391 cols ≃ 1.2 Gbyte
- physical memory > entire table

Data Conversions

Use STILTS to convert gzipped ASCII → FITS, colfits, MySQL

- `stilts tcopy in=psc.txt.gz out=psc.fits`
- `stilts tcopy in=psc.txt.gz out=psc.colfits`
- `stilts tcopy in=psc.txt.gz omode=tosql protocol=mysql database=astro protocol=mysql newtable=psc`

Results

Ran three unindexable benchmarks for each dataset:

SEL2: Select on a function of two columns ($J - K > 8$)

STAT1: Calculate statistics on a single column (μ, σ, min, max)

STAT2: Calculate statistics on two columns (μ, σ, min, max)

Benchmark	MySQL		colfits		fits		fits-nomap	
X_SEL2	66	49	4.7	3.2	89	86	449	442
X_STAT1	65	49	2.0	1.8	51	39	208	223
X_STAT2	66	49	2.4	2.1	51	44	224	221
XB_SEL2	36	27	2.4	2.1	21	3.8	124	124
XB_STAT1	36	27	1.4	1.0	14	2.6	117	113
XB_STAT2	36	27	2.0	1.2	14	2.8	122	119
P_SEL2	3422		397		2417		10281	
P_STAT1	3390		105		2321		10256	
P_STAT2	3404		284		2351		10399	
PB_SEL2	1278		95	74	837		3840	
PB_STAT1	1330		37	28	811		3926	
PB_STAT2	1290		70	59	802		3926	

Colfits $\approx$ 10–40 times faster than MySQL *for suitable queries*

- XSC (2 Mrow) queries interactive (1 min $\longrightarrow$ 2 sec) — OK for TOPCAT/STILTS
- PSC (0.5 Grow) queries while you wait (1 hour $\longrightarrow$ 2 min) — OK for STILTS
- Should scale to the largest catalogues ($< 2^{63}$ bytes?)

Operating System Issues

Files/columns are *mapped* for efficient random access

Requires a 64-bit OS for very large tables

- Mapping $\gtrsim 4$ Gbyte exceeds 32-bit VM address space limit

Linux (kernel 2.6.9) had issues with swapping

- Scan through large mapped file swaps out process pages $\longrightarrow$ disk thrashing
- Virtual memory tuning issue?
- `echo 0 >/proc/sys/vm/swappiness` helps
- `swapoff -a` fixes it (not ideal)
- Other OSs (Solaris? other Linux kernels?) might be better. Or worse.

There are probably ways around this

- Unmap least recently used file blocks
- . . . or use normal file reads with cached blocks rather than mapping
- . . . or whatever RDBMS do for accessing large files
- Probably slightly slower, and more complex, but still get column-oriented speedups

Possible Applications

Hybrid (colfits/SQL) Large Table Data Servers

- Queries which can use an index are fastest with SQL
  - ▷ selections, extrema, joins *on indexed columns*
- Queries which require a full scan of few columns are fastest column-oriented
  - ▷ scatter plot/density map, statistics, operations using unindexed columns or expressions
- A data service could duplicate the data, one in RDBMS form, one column-oriented (disk is cheap)
- Incoming requests could be served by whichever backend was most appropriate (either selected by user or dynamically resolved)

Medium-sized Survey Archive

- Catalogues of $\lesssim$ 1 Gbyte (10^6 row $\times$ 10^2 col) are now suitable for interactive use (e.g. TOPCAT)
- Sometimes it is more convenient to download a whole catalogue than query it remotely
- An archive of $\lesssim$ 1 Gbyte full survey catalogues in pre-digested (colfits) form would be a useful resource

Conclusions

Column-oriented data access:

- Useful for full scans of a few columns
- Factor 10—40 improvements over MySQL for wide tables

File mapping

- Permits efficient random access easily
- Issues on Linux with large files?

Implemented in STIL $\rightarrow$ STILTS & TOPCAT