Speedtables

A high-performance, memory-resident database for Tcl.

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.
  - Tcl_Obj structure overhead.
  - 24 bytes minimum for an integer.

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.
  - Arrays.
    - Even more overhead per object.
    - Multiple copies of key, or
    - Lists (see next slide)
    - Passed by name, have to use upvar.

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.
  - Lists - LISP was good model
  - But TCL isn't quite LISP
    - No references, lots of copying.
      - Even more memory.

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.
  - iTcl Objects.
  - Did we say memory overhead?

Problem

- Storing large amounts of data in Tcl takes a large amount of memory.
- Existing tricks to save memory are ad-hoc, manual, time-consuming, and error-prone.
- Complex data manipulation in Tcl is slow.

CTABLES

- Initial solution was for FlightAware
  - Thousands of flights
  - Millions of waypoints per day
  - Specialized C extension.
- Generalized for CableAware
  - Half a million cable modems
  - Automatically generated C code.

CTABLES

- Create a ‘C’ data structure to hold table rows.
  - Use Tcl hash tables for the primary key.
  - Efficiently convert between Tcl representation and C structure.

Defining CTABLES

```
package require ctable

CExtension aminfo 1.1 {
   CTable animation_characters {
       varstring name
       varstring home
       varstring show
       varstring dad
       boolean alive
       varstring gender
       int age
       int coolness
   }
   CTable another_table { ... }
}
```

Defining CTABLES

```
/* autogenerated by ctable table generator (date) */

#include "ctable.h"

struct animation_characters {
    ctable_HashEntry       hashEntry;
    ctable_LinkedListNode  _ll_nodes[1];
    char                  *name;
    int                    _nameLength;
    int                    _nameAllocatedLength;
[...]
    int                    age;
    int                    coolness;
    unsigned int           alive:1;
    unsigned int           _nameIsNull:1;
[...]
};
```

CTABLES

- 4 words overhead *per row*.
  - (32 bytes on x86, 64 bytes on AMD64)
- 1 bit overhead per field.
  - To allow for “null” values.
- 2 integers overhead for a string.
  - No overhead for integers, flags, etc.
- As fast to look up as a hash
  - Forked the hash code

Using CTABLES

```
package require Aminfo

animation_characters create t

t set shake \
   name "Master Shake" \
   show “Aqua Teen Hunger Force”
```

Using CTABLES

```
    t set shake age 4 coolness -5
    t incr shake age 1 coolness -1
5 -6
    t get shake age
5
    t get shake
{Master Shake} {} {Aqua teen Hunger Force} {} 1 {} 5 -6
    t array_get shake
name {master_shake} show {Aqua Teen Hunger Force} alive 1
age 5 coolness -6
```

Using CTABLES

```
     t fields
id name home show dad alive gender age coolness
     t type
animation_characters
     t needs_quoting name
1
     t exists frylock
0
     t count
1
```

Using CTABLES

```
    t delete shake
    t exists shake
0
    t count
0
    set fp [open anims.tsv r]
    t read_tabsep $fp
    close $fp
    t count
391
    t destroy
```

More COMMANDS

- write_tabsep
  - Writes whole ctable to a file
  - Replaced by search -write_tabsep
- foreach
  - Original method for walking a ctable
  - Replaced by search -glob
- batch
  - Avoid Tcl interpreter loop
  - Hide latency

More COMMANDS

- field
- needs_quoting
  - Get properties of fields
- key
- makekey
  - Data hiding and reflection
- array_get_with_nulls
  - Get array with {} for NULL
  - Compatibility with exiting code
  - Occasionally unbearably useful

More COMMANDS

- method
  - Creates a new command in Tcl
  - Plans for 'C' methods
    - It should be easy :)
    - No actual need yet...
    - So it's "Real Soon Now" for 2+ years
- methods
  - List of implemented methods
  - Including custom ones

PostgreSQL

- Read with import_postgres_result
  - Uses a pgsql result handle
- Write with write_tabsep
  - Execute a COPY FROM STDIN
  - write_tabsep to connection handle (!)

Problem

- Original scheme of walking the ctable with “foreach” was limited.
  - Brute force search only.
  - Simple key, simple search.
    - pattern match only.
  - Key not “in” row.
  - Still need to check other fields in Tcl.

SEARCH

- More complex searches.
- Replaces foreach and write_tabsep.
- More efficient, performs multiple comparisons in parallel

SEARCH

```
# Find the five coolest characters in ATHF...

t search -sort -coolness -limit 5 \
    -key key -array_get_with_nulls data \
    -compare {{= show “Aqua Teen Hunger Force”}} \
    -code {
        # Do something cool with them in here...
        [...]
    }
```

SEARCH

- First version an extended foreach.
  - Walked the whole hash table.
- Still much faster than Tcl arrays.
  - 60ns per row on AMD64
- Complex search (-compare).
- Sort, limit, like SQL "SELECT".

SEARCH

- Still too slow for multiple keys.
- Needed to add indexes.
- Used skiplists.
  - Simple implementation.
  - Potentially good for shared memory.
  - Decently behaved.

SEARCH

```
package require ctable

CExtension animinfo 1.2 {
   CTable animation_characters {
         varstring name indexed 1
         varstring home
         varstring show indexed 1
         varstring dad
         boolean alive
         varstring gender
         int age
         int coolness
   }
}
```

SEARCH

```
/* autogenerated by ctable table generator (date) */

#include "ctable.h"

struct animation_characters {
    ctable_HashEntry      hashEntry;
    ctable_LinkedListNode _ll_nodes[3];
    char                  *name;
    int                   _nameLength;
    int                   _nameAllocatedLength;
[...]
    int                   age;
    int                   coolness;
    unsigned int          alive:1;
    unsigned int          _nameIsNull:1;
[...]
};
```

SEARCH

- Extra overhead, 3 pointers per index.
- Plus the skiplist itself when it's used.
- Uses the row itself for the index.
  - Field not duplicated.
  - Field can be any data type.

SEARCH

- Using the indexed table

```
package require Animinfo 1.2

animation_characters create t
# create indexes on indexed fields
t index create name
t index create show

t set shake \
  name "Master Shake" \
  show “Aqua Teen Hunger Force”
```

SEARCH

```
# Find the five coolest characters in ATHF...

t search -sort -coolness -limit 5 \
    -key key -array_get_with_nulls data \
    -compare {{= show “Aqua Teen Hunger Force”}} \
    -code {
        # Do something cool with them in here...
        [...]
    }
```

SEARCH

- No change to search command
  - Originally index search with search+
  - Now hash and index search merged
- Search options

  -sort, -glob, -countOnly, -offset, -limit,
  -write_tabsep, -with_field_names,
  -compare, -key, -array, ...

Problems

- Single process, no shared data.
- Loading tables from SQL or TSV slow
- Multiple copies of tables, one per process
  - We were trying to avoid wasting memory

CTABLE SERVER

- Server keeps live data in memory
- Client object behaves like a ctable.
  - But it forwards calls to server over TCP
- Simple requests return Tcl lists
- Rows come back as TSV
- Client reads the rows and handles callbacks for code

CTABLE SERVER

- Client identifies server as a URL
  - ctable://host:port/tablename?...
- Server registers tables as URL
  - ctable://*:port/tablename
- Ctable Transfer Protocol

SPEEDTABLES

- Client identifies server as a URL
  - sttp://host:port/tablename?...
- Server registers tables as URL
  - sttp://*:port/tablename
- Speed Table Transfer Protocol
- Speed Table API

SPEEDTABLES

- STTP server code

```
package require ctable_server

# Set up a table somewhere in here...

::ctable_server::register \
    sttp://*/animinfo t

vwait Die
```

SPEEDTABLES

- STTP client code

```
package require ctable_client

remote_ctable sttp://127.0.0.1/animinfo t

t search -sort -coolness -limit 5 -key key \
    -array_get_with_nulls data
    -code {
        puts “$key -> $data”
    }
```

SPEEDTABLES

- Restrictions
  - Single connection, not reentrant.
    - Can't make a request from code block.
    - *Could* set up multiple connections.
    - In practice not been a problem.
- Not all search terms implemented.
  - In particular, null values don't work.
    - But we have array...with_nulls

SPEEDTABLES API

- Further generalization of ST API
- stapi_server
  - Generates speed tables from PostgreSQL
  - Acts like a read cache for STTP clients
    - Periodic refresh from SQL
    - Partial update, pick up changed rows only

SPEEDTABLES API

```
package require st_server

# Set up pgtcl connection...
::stapi::init -conn $pgtcl_connection

# Define a table
::stapi::init_ctable myanims anims \
    "show = 'Aqua Teen Hunger Force'" [
    ::stapi::from_table anims key \
        -index name -index show
]

# Load the table
::stapi::open_cached myanims \
    -index name -index show

::ctable_server::register myanims \
    sttp://*:1234/anims

vwait Die
```

SPEEDTABLES API

- I admit it's not very clean
- It's doing a big job, though.
  - Can derive a speedtable from a single table
  - Or multiple tables
  - Handles duplicate row names
  - Generates speed table + complex SQL
- But mostly, it hasn't been touched in a long time.

SPEEDTABLES API

- The client side is very very cool.
- stapi_client
  - Generalizes the CLIENT side of STTP
  - Looks like a speed table to Tcl (ST API)
  - Specified using URLs, eg:
    - sttp://host:port/table
    - sql://database/table
    - Etc...

SPEEDTABLES API

```
# Connect to that ctable server...

package require st_client

set t [
    ::stapi::connect sttp://localhost:1234/anims
]

$t search -sort -coolness -limit 5 -key key \
    -array_get_with_nulls data
    -code {
        puts “$key -> $data”
    }
```

SPEEDTABLES API

```
# Connect to PostgreSQL database directly...
package require st_client_pgtcl
::stapi::init_conn $pgtcl_connection

set t [
    ::stapi::connect sql:///anims?_key=key
]

$t search -sort -coolness -limit 5 -key key \
    -array_get_with_nulls data
    -code {
        puts “$key -> $data”
    }
```

SPEEDTABLES API

- The good
  - Automatically generates SQL
    - Does not create a ctable, just acts like one.
      - Only need to define key column
      - Can refine query by adding ?col=type
      - Can fake columns with ?col=sql
      - Can define a complex key with ?_key=sql
      - Can limit to specific columns
        - sql:///anims/name/show/coolness

SPEEDTABLES API

- The bad
  - Only handles a single table
    - But you can use views instead.
  - Not a cache, uses SQL for every request
    - Also an advantage...
  - stapi_server doesn't use it (yet)
  - Should be able to specify db in URL
    - sql://user:pass@host:dbname/anims
    - Future work.

SPEEDTABLES API

- st_display
  - Based on DIOdisplay
    - http://tcl.apache.org/rivet/
- Uses STAPI calls instead of SQL calls
- Implements most of DIOdisplay
  - Everything we needed, anyway :)
- Significantly enhanced

SPEEDTABLES API

- Supports complex searches
  - iTunes style 'smart playlist' UI
- Filter columns, pseudo-columns.
- Rewrite column using Tcl proc
  - eg: add hyperlinks.

Show All | Select: Serial | = | 000039267A73 | + | Search

Time	Account	Serial	Biz Unit	Type	Status	Location	Address	Code	Last update	Operator	Transaction	Explanation
2006-11-21 22:43:12		000039267A73	81110000	2 Rr Modem	Active							
2006-11-17 21:35:20		000039267A73	81110000	2 Rr Modem	Active							
2006-11-17 21:33:11		000039267A73	81110000	2 Rr Modem	Disabled							
2006-11-17 21:33:11		000039267A73	81110000	2 Rr Modem	Active							
2006-11-17 21:32:10		000039267A73	81110000	2 Rr Modem	Active							

Download CSV file | Previous: equipment_history.csv: 192 bytes, 05-Dec-2006 13:23:14

Problems

- STTP still not fast enough
- Single-threaded server a bottleneck
- Needs real shared memory access
- Skiplists designed for shared memory

SPEEDTABLES

- Shared memory speedtables
  - Implemented over the past few months.
  - Tried various approaches.
  - Settled on lockless shared memory
    - skiplists support lockless readers
    - STTP model has only one writer
    - Writes are dominated by one source
    - Can use STTP for writing back when necessary

SPEEDTABLES

- Lockless malloc
  - Master process increments a write counter
  - Reader processes get one writable word
    - Copy counter to it when they start a search
    - Clear counter when finished
- Master updates links in "safe" order
- Freed memory goes into garbage list
  - Tagged by current cycle
  - Garbage collected when older than any reader

SPEEDTABLES

- Lockless skiplists
  - Implementation we started with wasn't safe
    - Sharable data was mixed with private data
    - Link updates were not always in safe order
- Search in two passes
  - Accumulates search into row array
  - Restarts search if skiplist changed
  - Runs code blocks after search complete

SPEEDTABLES

- Performance
  - Writes sometimes 10% slower
  - *Search 6000 times faster!*

"Master"

```
# Low level API
package require ctable_server
package require Shared_anims

# Create shared ctable in a 24M segment
animation_characters create m \
     master size 24M \
     file /var/shares/anims.dat

[...]

::ctable_server::register m sttp://*:1234/anims
vwait Die
```

"Reader"

```
# Low level API
package require st_client
package require Shared_anims

set m [
    ::stapi::connect sttp://localhost:1234/anims
]

# Let master know my process ID and
# get connection list
set list [$m attach [pid]]

set r [
     animation_characters create reader $list
]
```

SPEEDTABLES

- Master must be on localhost.
- Connection list contains...
  - file name
  - name of shared object
  - possibly other things
  - treat it as an opaque token
    - You CAN fake it up for read-only access to an orphan shared memory image.

SPEEDTABLES

- High level API using STAPI

```
package require st_shared

set r [
    ::stapi::connect shared://1234/anims
]
```

- Much simpler!
- Sends search through shared memory, everything else through STTP.
  - This needs a little tweaking... needs to use master for things like -delete.

Problem

- Tcl ckalloc hates to run out of memory
  - But VM is 'infinite'.
- Speedtables written using ckalloc
  - Shared memory definitely finite!
- Still a fatal error by default
  - Some allocations happening in hash and skiplist code
  - Not always a place to throw a Tcl exception when allocation fails
- Working on fixing this
  - Hoisting memory allocations to top level
  - Pre-allocate where that can't be done

SPEEDTABLES

- Spinoffs from shared memory work...
  - Search optimizations
  - Any field can be a key
    - Pseudo-field "_key" used if not specified.
- Pseudo-field "_dirty" set if row modified.
  - Only cleared manually
- Search becoming transaction-like
  - -update
  - -delete

SPEEDTABLES

- Where is it?
  - The Speed Tables package is distributed under the same version of the BSD license as Tcl.
  - http://sourceforge.net/projects/speedtables
  - Included with speed tables is a 65 page developer's manual. A test suite includes dozens of tests.

Future Work

- General clean up
  - Still using "ctable" in code in lots of places.
  - Compiler flags hardcoded for FreeBSD & Darwin
    - Should use tclConfig.sh for this
  - Maybe use Miguel's new hash code. :)
- Shared memory allocation failures
  - Callback to let you release memory and try again.
  - Generates TCL_ERROR otherwise

Future Work

- Databases...
  - Make st_server simpler, using st_client.
    - maybe even just eliminate it and use sql://
  - Make sql://database/table work.
  - Support other databases.
    - MySQL, sqlite, Oracle? SQL Server?

Future Work

- Generalize further...
  - Make it accessible to clients not using Tcl
    - Modify STTP?
    - Maybe even something like WebDAV?
- Porting to non-UNIX systems?
  - It's kind of dependent on a decent C compiler.

Speedtables

A high-performance, memory-resident database for Tcl.

http://sourceforge.net/projects/speedtables