TAKING OFF WITH SOLARIS

Dennis Chernoivanov, Senior Software Engineer at Tbricks
COMPANY OVERVIEW

- Founded 2006, privately held
- Offices in Stockholm and St. Petersburg, ~25 employees
- Objective: Algorithmic Trading Solution
  - Ultra-low latency (under 1ms)
  - Highly scalable
  - Easy to setup, deploy and manage
  - Reliable, through fail-over capabilities
  - Zero configuration
STRATEGY ENGINE OVERVIEW
PART I: OS OF CHOICE
MAKING THE CHOICE

- Need highly scalable, tuned kernel
  - Solaris: > 20 years SMP development, known to perform well
  - Linux: Quickly gets up to speed
- Stable API, manageable upgrades
  - Solaris: Already there
  - Linux: OS version + GLIBC version + ...
MAKING THE CHOICE

- Good 64-bit development tools
  - Solaris: SUNWspro, DTrace, ProcUtils, ...
  - Linux: GCC/GDB (SystemTap to come)
- Reliable and fast file storage, preferably with backup solution
  - Solaris: ZFS
    - Fast
    - Easy built-in backup through snapshots
    - Easy to use, no more RAID nightmares
  - Linux: Traditional filesystems and tools
- Predictable scheduler performance and latency
MAKING THE CHOICE

Count of operations per second for last 13 days
Count of operations per second for <tst_51> last 13 days

ops/sec

Unexpected degradation
Peak1 restart
Peak2 restart
REFERENCE PLATFORM

- Solaris 10/x86_64
- Sun Fire x4150 Server with 2 Quad-core Intel Xeons
- ZFS as an ultimate storage and backup
ALSO IN THE GAME

- Solaris/SPARC (T1000)
- Windows/x86
- MacOSX/x86_64 (Leopard 10.5.1)
- Linux/x86_64 (RHEL4 ES)
PART II: BUILD FACTORY
CODE BASE

- ~500,000 lines of code backed by Subversion
  - Servers are in C++
  - Client is in C#
  - ~150,000 auto-generated C++ code
  - Some perl scripts
- Much larger code base is free software sources
  - ACE/QuickFIX/... (C++)
  - LibXML2/LibXSLT (C/C++)
COMPILER SUITES

- SUN Studio 12
  - Optimizes well for SPARC and x86_64
  - Sticks to the Standard
  - But provokes to use non-standard STL (libCstd.so.1)
  - And comes with STLport4 not optimized for x86_64

```cpp
class Any
{
  ...
private:
  union {
    bool m_boolean;
    int64_t m_integer;
    double m_double;
  } m_value;
  std::string m_string; // takes an allocator-wide lock bringing the system down
  // Apache's stdcxx library optimizes this away using atomic ops
};
```
COMPILER SUITES

- GCC 3.x and 4.x
  - Most portable
  - Fairly strict
- MS Visual Studio 8
  - Fast
  - But has issues with portability (how about stdint.h?)
BUILD SYSTEM

- GNU Make 3.81
  - Feature rich
  - Supports parallel builds
  - Natively builds most of the platforms

- On Windows:
  - GNU Make 3.81
  - Custom build helper programs
  - MPC
  - MS VS msbuild and vcbuild as the engine
BUILD TIMES (NFS)

- **Linux: 2xQuad-core Intel Xeon Harpertown 2.83 GHz 8Gb RAM**
  
  ```
  $ time tbmake -j
  real 8m48.656s
  user 20m38.300s
  sys 17m53.620s
  ```

- **Solaris: SunFire x4150 16Gb RAM**
  
  ```
  $ time tbmake -j
  real 15m13.185s
  user 38m45.669s
  sys 9m59.379s
  ```

- **Mac OS X: Apple Xserve Quad-core Xeon 2.66 GHz 4Gb RAM**
  
  ```
  $ time tbmake -j
  real 54m36.624s
  user 18m46.978s
  sys 14m59.874s
  ```

- **Windows: VmWare i386 virtualization box**
  
  ```
  $ time tbmake -j
  real 94m4.187s
  user 0m0.015s
  sys 0m0.093s
  ```
DEVELOPMENT WORKFLOW

- Design/Code/Build/Debug
- Commit reviews (untested commits prohibited)
- Automated per-commit, per-component tests
- Full system tests few times a day
  - Leaks detection
  - Performance inspection
  - Memory consumption analysis
  - Run-time checks (with Rational tools)
  - Test coverage computation
- Static code analysis
viid_set_id [id=185] [revision from 9018 to 12458]
DEVELOPER’S DESKTOP

- 15% Native Windows development
- 55% Windows as a terminal to Solaris/Linux
- 25% MacOSX
- 5% Linux
DEVELOPER’S DESKTOP

- 50% : ViM
- 30% : MS Visual Studio
- 20% : Other (Emacs/Xcode/Joe)
NO SOLARIS???

- Not promoted as a desktop system
- Lacks support from software vendors
  - VmWare is of high importance to us
  - I want my Firefox 3 RC with Flash plug-in!
  - Even configure is not always adopted for Solaris/x86
- Windows is de-facto standard
  - Means: You must be better by far to make people move
PART III: TOOLS
DTRACE

- Non-intrusive
- Very limited performance impact to the traced processes
- Does not require change/build/run cycle
- Provides for process sampling
- Incredible for troubleshooting: Even customers can use it!
#include "make_uuid.d"
string strategy_id;
string parent_id;
BEGIN {
  printf( "\nTracing SE for 'order latency' strategy checking.\n\n");
  start_time = timestamp;
}
#define LATENCY_ENTRY(name, display_name, direction)                                                                                                                             
se_latency$target:::name                                                                                                                                                                          
{                                                                                                                                                                                                                 
  MAKE_UUID( strategy_id, arg1, arg2);
  MAKE_UUID( parent_id, arg3, arg4);
  nsec = timestamp - start_time;
  sec = nsec / 1000000000;
  nsec -= sec * 1000000000;
  msec = nsec / 1000000;
  nsec -= msec * 1000000;
  usec = nsec / 1000;
  nsec -= usec * 1000;
  printf( "[%03d.%03d.%03d.%03d] %04d display_name %s direction %s\n", sec, msec, usec, nsec, arg0, strategy_id, parent_id);
}
LATENCY_ENTRY(Engine_create_strategy,create_strategy,<-)
LATENCY_ENTRY(Engine_handle_strategy,handle_strategy,->)
LATENCY_ENTRY(SH_HSI_execute,call_h_strategy,->)
BBO LATENCY

[cdir@k2(pts/3) ~/src/tb/trunk/src/scripts/dtrace]$ dtrace -CqI /home/german/trunk/src/scripts/dtrace -s /home/german/trunk/src/scripts/dtrace/se_order_latency.d -P se_latency/* -p 3289

Tracing SE for 'order latency' strategy checking.

[052.569.044.552] 0000 handle_strategy 2950c08a-36f8-11dd-97a5-09d0a6796c37 -> 8ddd9ac6-5003-4e42-bff9-0c499a4fadc5
[052.511.740.421] 0000 create_strategy 2950c08a-36f8-11dd-97a5-09d0a6796c37 <- 8ddd9ac6-5003-4e42-bff9-0c499a4fadc5
[052.569.270.343] 0000 call_h_strategy 2950c08a-36f8-11dd-97a5-09d0a6796c37 -> 8ddd9ac6-5003-4e42-bff9-0c499a4fadc5
DTRACE: I/O SAMPLING

[cdd@k2(pts/3) ~/src/tb/trunk/src/scripts/dtrace]$ dtrace -Cqs ./storage.d 3289
Tracing BDB database calls every 10 seconds.

Call count during last 10 seconds:

<table>
<thead>
<tr>
<th>Call</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>StorageBDBBackend::put()</td>
<td>1</td>
</tr>
<tr>
<td>StorageBDBBackend::BDBCursor::get()</td>
<td>3</td>
</tr>
<tr>
<td>StorageBDBBackend::BDBCursor::BDBCursor()</td>
<td>6</td>
</tr>
<tr>
<td>StorageBDBBackend::BDBCursor::~BDBCursor()</td>
<td>6</td>
</tr>
</tbody>
</table>

<...>

Total summary:

Total number of calls (count):

<table>
<thead>
<tr>
<th>Call</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td></td>
</tr>
<tr>
<td>total StorageBDBBackend::modify()</td>
<td>2</td>
</tr>
<tr>
<td>total StorageBDBBackend::BDBCursor::get()</td>
<td>3</td>
</tr>
<tr>
<td>total StorageBDBBackend::put()</td>
<td>5</td>
</tr>
<tr>
<td>total StorageBDBBackend::BDBCursor::BDBCursor()</td>
<td>6</td>
</tr>
<tr>
<td>total StorageBDBBackend::BDBCursor::~BDBCursor()</td>
<td>6</td>
</tr>
</tbody>
</table>

Average time spent in each call (microseconds):

<table>
<thead>
<tr>
<th>Call</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>StorageBDBBackend::BDBCursor::get()</td>
<td>4</td>
</tr>
<tr>
<td>BDBCursor open &lt;-&gt; close</td>
<td>69</td>
</tr>
<tr>
<td>StorageBDBBackend::put()</td>
<td>319</td>
</tr>
<tr>
<td>StorageBDBBackend::modify()</td>
<td>351</td>
</tr>
</tbody>
</table>
DTRACE: I/O SAMPLING

xcelerg call-count (handle_output) in last 10 seconds [id=124]
WHAT’S MISSING?

- Documentation
  - “No inter-process tracing (or at least gettimeofday() call)”
- TCP/IP checkpoints [tcpsnoop not integrated for ~2 years]
- Static checkpoint signature length limited to 128 characters
- Limited D syntax (only integral types for local variables)
- No distributed measurements
TBSNOOPER SAMPLING
RIVALS: SYSTEMTAP (LINUX)

- Script syntax close to C (ifs, cycles, and more)
- TCP/IP checkpoints built-in
- No user-level checkpoints yet
LIBUMEM

- Provides object-caching memory allocator
- Fast (SE developer has no more fear for operator new)
- Designed for use in multi-threaded applications
- Built-in leaks detection
- Built-in memory corruption detection
ANYTHING ELSE OUT THERE?

- MacOSX Leaks: Much easier to use
- Valgrind
  - Memory error detection
  - Memory consumption analysis (even if reference is kept)
  - Heap profiler
  - Very good documentation
  - Easy to use
WHY LEAKS AND VALGRIND?

```bash
$ export UMEM_DEBUG=default
$ export UMEM_LOGGING=transaction
$ export LD_PRELOAD=libumem.so.1
<start program>
$ echo "::findleaks" | mdb -p <pid>

<table>
<thead>
<tr>
<th>BYTES</th>
<th>LEAKED</th>
<th>VMEM_SEG CALLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>1</td>
<td>fffffd7ffbca9000 MMAP</td>
</tr>
</tbody>
</table>

---

Total 1 oversized leak, 4096 bytes

<table>
<thead>
<tr>
<th>CACHE</th>
<th>LEAKED</th>
<th>BUFCTL CALLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000008e6668 12</td>
<td>0000000003961d20 libCrun.so.1`_<em>1c2n6FL_pv</em>+0x29</td>
<td></td>
</tr>
<tr>
<td>00000000008f1028 2</td>
<td>000000000273b7e0 libdb_cxx-4.6.so`__os_malloc+0x9a</td>
<td></td>
</tr>
</tbody>
</table>

$ echo "0000000003961d20" | mdb -p <pid>

<table>
<thead>
<tr>
<th>ADDR</th>
<th>BUFADDR</th>
<th>TIMESTAMP</th>
<th>THREAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>3961d20</td>
<td>395b740</td>
<td>5810d3912710d</td>
<td>18</td>
</tr>
</tbody>
</table>

libCrun.so.1`umem_cache_alloc_debug+0x12b
libCrun.so.1`umem_cache_alloc+0x8c
libCrun.so.1`umem_alloc+0xf
libCrun.so.1`malloc+0x2e
libCrun.so.1`__1c2n6FL_pv_+0x29
libstrategy.so`__1cHtbricksGEngine<...>SrategyContext_rkn0AUInstrumentIdentifier__n0AKInstrument__ +0x111
strategy.bin`__1cFOrderVHandleValidateRequest6MrknHtbricksSStrategyParameters_r0BLEDitContext_v_+0x89c
libstrategy.so`__1cHtbricksIStrategyVHandleValidateRequest6Mrn0ARValidationContext_v_+0xa6
__1cQValidateStrategyHexecute6M_v_+0x77
```

WHY LEAKS AND VALGRIND?
$ export MallocStackLogging=1
<start program>
$ leaks <pid>

Leak: 0x1030b9600  size=4608
  0x00001050 0x00000000 0x00000000 0x00000000  P............
  0x00000002d 0x00000000 0xffffffff 0xffffffff  ~............
  0x01682048 0x00000000 0x00000001 0x00000000  H h............
  0x0498db78 0x00000000 0x00000000 0x00000000  x............
  0xffffffff 0xffffffff 0xffffffff 0xffffffff  ................
  0x00000000 0x00000000 0x00000000 0x00000000  ................
  0x00000000 0x10000000 0x00000501 0x00000000  ................
  0x00000001 0x00000000 0x00000000 0x00000000  ................
...
  Call stack: [thread 0x1050a5000]: | thread_start | _pthread_start | ace_thread_adapter | ACE_Thread_Adapter::invoke() |
  ACE_Thread_Adapter::invoke_i() | ACE_Task_Base::svc_run(void*) | Task::svc() | strategy_storage::GetSnapshot::execute() |
  strategy_storage::StorageTask::get_snapshot(tbricks::types::UUID const&, tbricks::filter::Filter const&) |
  tbricks::storage::MessageStorageT<tbricks::types::UUID, tbricks::protocol::StrategyInstance, |
  <...>

Leak: 0x1049c2550  size=400  instance of 'tbricks::types::Instrument', type C++, implemented in libprotocol.dylib
  0x0089f338 0x00000001 0x00000000 0x00000000  8............
  0x00000000 0xc0000000 0x00000001 0xc0000000  ................
...
  Call stack: [thread 0x104f1c000]: | thread_start | _pthread_start | ace_thread_adapter | ACE_Thread_Adapter::invoke() |
  ACE_Thread_Adapter::invoke_i() | ThreadPool::svc_run(void*) | ThreadPool::svc() | ValidationPool::execute_event(SH*) |
  SH::execute_validation_event() | SH::execute_n_check_delete(StrategyEvent*) | ValidateStrategy::execute() |
  tbricks::Strategy::HandleValidateRequest(tbricks::ValidationContext&) | Order::HandleValidateRequest(tbricks::StrategyParameters const&, tbricks::EditContext&) |
  tbricks::Engine::ResolveInstrument(tbricks::StrategyContext const&, tbricks::InstrumentIdentifier const&) const |
  operator new(unsigned long) | malloc | malloc_zone_malloc

WHY LEAKS AND VALGRIND?
SO, WE DELIVER ON SOLARIS

- For its scalable and tuned kernel
- Stable API and manageable upgrades
- Reliable scheduler
- Compilers that are so standard compliant
- ZFS
- Runtime analysis tools
BUT WE RUN LINUX

- For its faster builds
- And it’s just faster (at least when not overused)
- Because it has better support from software vendors
AND MAC OS X

- As it has extremely easy to use analysis tools
- Quite easy to setup an use
- Of course, religion issues
AND WINDOWS

- Because C# makes GUI development blazingly fast
- De-facto standard for desktop computing
WHAT CAN WE ASK FOR?

- Documentation
- Performance
- Faster feature turnaround times
- User friendliness
- More interaction with software vendors (e.g. market API)
- More hypes about the platform
Dennis Chernoivanov
Senior Software Engineer at Tbricks

cdi@tbricks.com
www.tbricks.com