Optimising Cloud Computing with SBSE

David R. White & Jeremy Singer
{david.r.white, jeremy.singer}@glasgow.ac.uk
University of Glasgow

Monday 25 July 2011
OUTLINE

VIRTUAL MACHINES

OPPORTUNITIES FOR SBSE

TAKE-HOME
Outline

Virtual Machines

Opportunities for SBSE

Take-home
OPTIMISING VIRTUAL MACHINE MANAGEMENT
Optimising Virtual Machine Management
OPTIMISING VIRTUAL MACHINE MANAGEMENT
Optimising Virtual Machine Management
Optimising a Single Java Virtual Machine

Competition for resources must be managed.
JVM Configuration

What heap sizes should I use?

- `Xms<number><unit>` Initial size of heap
- `Xmx<number><unit>` Maximum size of heap
JVM Configuration - II

Which Garbage Collector should I use?
- SemiSpace
- MarkSweep
- GenCopy
- GenMS
- CopyMS
- RefCount
JVM Configuration - II

Which Garbage Collector should I use?

- SemiSpace
- MarkSweep
- GenCopy
- GenMS
- CopyMS
- RefCount

Many, many more decisions to be made…
## JVM Configuration: Detail - I

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedulingMultiplier</td>
<td>Should we eagerly finish sweeping at the start of a collection</td>
</tr>
<tr>
<td>eagerCompleteSweep</td>
<td>Should memory be protected on release?</td>
</tr>
<tr>
<td>protectOnRelease</td>
<td>Should finalization be disabled?</td>
</tr>
<tr>
<td>noFinalizer</td>
<td>Should reference type processing be disabled?</td>
</tr>
<tr>
<td>noReferenceTypes</td>
<td>Should a major GC be performed when a system GC is triggered?</td>
</tr>
<tr>
<td>fullHeapSystemGC</td>
<td>Should we ignore calls to java.lang.System.gc?</td>
</tr>
<tr>
<td>ignoreSystemGC</td>
<td>Should we shrink/grow the heap to adjust to application working set?</td>
</tr>
<tr>
<td>variableSizeHeap</td>
<td>Number of bits to use for the header cycle of mark sweep spaces</td>
</tr>
<tr>
<td>eagerMmapSpaces</td>
<td>Force a collection after this much allocation</td>
</tr>
<tr>
<td>markSweepMarkBits</td>
<td>Trigger a GC if the meta data volume grows to this limit</td>
</tr>
<tr>
<td>stressFactor</td>
<td>Bound the maximum size of the nursery to this value</td>
</tr>
<tr>
<td>metaDataLimit</td>
<td>Fix the minimum and maximum size of the nursery to this value</td>
</tr>
<tr>
<td>boundedNursery</td>
<td>Number of GC threads to use</td>
</tr>
<tr>
<td>fixedNursery</td>
<td>Should the adaptive system recompile hot methods?</td>
</tr>
<tr>
<td>enable_recompilation</td>
<td>Should the adaptive system precompile all methods given in the advice file before the user thread is started?</td>
</tr>
<tr>
<td>enable_precompile</td>
<td>Should we use adaptive feedback-directed inlining?</td>
</tr>
<tr>
<td>adaptive_inlining</td>
<td>Should AOS promote baseline-compiled methods to opt?</td>
</tr>
<tr>
<td>osr_promotion</td>
<td>Should recompilation be done on a background thread or on next invocation?</td>
</tr>
<tr>
<td>background_recompilation</td>
<td>How many timer ticks of method samples to take before reporting method hotness to controller</td>
</tr>
<tr>
<td>method_sample_size</td>
<td>After how many clock ticks should we decay</td>
</tr>
<tr>
<td>decay_frequency</td>
<td>What factor should we decay call graph edges hotness by</td>
</tr>
<tr>
<td>dgc_decay_rate</td>
<td>Initial edge weight of call graph is set to AL_SEED_MULTIPLIER * (1/A1_CONTROL_POINT)</td>
</tr>
<tr>
<td>dgc_sample_size</td>
<td>What percentage of the total weight of the dgc demarcates warm/hot edges</td>
</tr>
<tr>
<td>inline_ai_seed_multiplier</td>
<td>Name of offline inline plan to be read and used for inlining</td>
</tr>
<tr>
<td>inline_ai_hot_callsite_threshold</td>
<td>Value of controller clock at which AOS should exit if EARLY_EXIT is true</td>
</tr>
<tr>
<td>offlinePlan</td>
<td>Invocation count at which a baseline compiled method should be recompiled</td>
</tr>
<tr>
<td>early_exit_time</td>
<td>Opt level for recompilation in invocation count based system</td>
</tr>
<tr>
<td>invocation_count_threshold</td>
<td>What is the sample interval for counter-based sampling</td>
</tr>
<tr>
<td>invocation_count_opt_level</td>
<td>The maximum optimization level to enable.</td>
</tr>
<tr>
<td>counter_based_sample_interval</td>
<td>Focus compilation effort based on frequency profile data</td>
</tr>
<tr>
<td>max_opt_level</td>
<td>Should we constrain optimizations by enforcing reads-kill?</td>
</tr>
<tr>
<td>focus_effort</td>
<td>Inline statically resolvable calls</td>
</tr>
<tr>
<td>reads_kill</td>
<td>Guarded inlining of non-final virtual calls</td>
</tr>
<tr>
<td>inline</td>
<td>Speculatively inline non-final interface calls</td>
</tr>
<tr>
<td>inline.guarded</td>
<td></td>
</tr>
<tr>
<td>inline.guarded_interfaces</td>
<td></td>
</tr>
</tbody>
</table>
Virtual Machines

Opportunities for SBSE

Take-home

JVM Configuration: Detail - II

- **inline_preex**: Pre-existence based inlining
- **simplify_integer_ops**: Simplify operations on integers
- **simplify_long_ops**: Simplify operations on longs
- **simplify_float_ops**: Simplify operations on floats
- **simplify_double_ops**: Simplify operations on floats
- **simplify_ref_ops**: Simplify operations on references
- **simplify_tib_ops**: Simplify operations on TIBs
- **simplify_field_ops**: Simplify operations on fields
- **simplify_chase_final_fields**: Chase final fields avoiding loads at runtime
- **local_constant_prop**: Perform local constant propagation
- **local_copy_prop**: Perform local copy propagation
- **local_cse**: Perform local common subexpression elimination
- **control_static_splitting**: CFG splitting to create hot traces based on static heuristics
- **control_unwhile**: Turn whiles into untils
- **escape_simple_ipa**: Eagerly compute method summaries for simple escape analysis
- **escape_scalar_replace_aggregates**: If possible turn aggregates (objects) into variable definition/uses
- **escape_monitor_removal**: Try to remove unnecessary monitor operations
- **escape_invokee_thread_local**: Compile the method assuming the invokee is thread-local. Cannot be properly set on command line.
- **ssa**: Compile the method assuming the invokee is thread-local. Cannot be properly set on command line.
- **ssa_expression_folding**: Should we try to fold expressions with constants locally?
- **ssa_redundant_branch_elimination**: Eliminate redundant conditional branches
- **ssa_licm_ignore_pei**: Assume PEIs do not throw or state is not observable
- **ssa_load_elimination**: Should we perform redundant load elimination during SSA pass?
- **ssa_coalesce_after**: Should we coalesce move instructions after leaving SSA?
- **ssa_loop_versioning**: Create copies of loops where runtime exceptions are checked prior to entry
- **ssa_live_range_splitting**: Split live ranges using LIR SSA pass?
- **ssa_gcp**: Perform global code placement
- **ssa_gcse**: Perform global code placement
- **ssa_global_bounds**: Perform (incomplete/unsafe) global Array Bound Check elimination on Demand
- **ssa_splitblock_to_avoid_rename**: When leaving SSA create blocks to avoid renaming variables
- **ssa_splitblock_for_local_live**: When leaving SSA create blocks for local liveness
- **ssa_splitblock_into_infrequent**: When leaving SSA create blocks to avoid adding code to frequently executed blocks
- **reorder_code**: Reorder basic blocks for improved locality and branch prediction
- **reorder_code_ph**: Reorder basic blocks using Pettis and Hansen Algo2
- **h2i_inline_new**: Inline allocation of scalars and arrays
- **h2i_inline_write_barrier**: Inline write barriers for generational collectors
- **h2i_inline_primitive_write_barrier**: Inline primitive write barriers for certain collectors
- **h2i_no_callee_exceptions**: Assert that any callee of this compiled method will not throw exceptions. Cannot be properly set on command line
- **h2i_call_via_jtoc**: Plant virtual calls via the JTOC rather than from the tib of an object when possible
- **12m_handler_liveness**: Store liveness for handlers to improve dependence graph at PEIs
- **12m_schedule_prepass**: Perform prepass instruction scheduling
VIRTUAL MACHINES

JVM Configuration: Detail - III

regalloc_coalesce_moves: Attempt to coalesce to eliminate register moves?
regalloc_coalesce_spills: Attempt to coalesce stack locations?
adaptive_instrumentation_sampling: Perform code transformation to sample instrumentation code.
adaptive_no_duplication: When performing inst. sampling, should it be done without duplicating code?
adaptive_processor_specific_counter: Should there be one CBS counter per processor for SMP performance?
adaptive_remove_yp_from_checking: Should yieldpoints be removed from the checking code (requires finite sample interval)?

osr_guarded_inlining: Insert OSR point at off branch of guarded inlining?

osr_inline_policy: Use OSR knowledge to drive more aggressive inlining?

profile_edge_count_input_file: Input file of edge counter profile data
profile_infrequent_threshold: Cumulative threshold which defines the set of infrequent basic blocks
profile_cbs_hotness: Threshold at which a conditional branch is considered to be skewed
escape_max_array_size: Maximum size of array to replaced with registers by simple escape analysis
ssa_load_elimination_rounds: How many rounds of redundant load elimination will we attempt?
l2m_max_block_size: Maximum size of block for BURS, larger blocks are split
regalloc_simple_spill_cost_move_factor: Spill penalty for move instructions
regalloc_simple_spill_cost_memory_operand_factor: Spill penalty for registers used in memory operands

control_tableswitch_cutoff: If a tableswitch comprises this many or fewer comparisons convert it into multiple if-then-else style branches
control_cond_move_cutoff: How many extra instructions will we insert in order to remove a conditional branch?
control_unroll_log: Unroll loops. Duplicates the loop body 2^n times.
control_static_splitting_max_cost: Upper bound on the number of instructions duplicated per block when trying to create hot traces with static splitting
control_well_predicted_cutoff: Don’t replace branches with conditional moves if they are outside of the range of 0.5 ± this value
inline_max_target_size: Static inlining heuristic: Upper bound on callee size
inline_max_inline_depth: Static inlining heuristic: Upper bound on depth of inlining
inline_max_always_inline_target_size: Static inlining heuristic: Always inline callees of this size or smaller
inline_max_arg_bonus: Static inlining heuristic: If root method is already this big, then only inline trivial methods
inline_precise_reg_array_arg_bonus: Maximum bonus for reducing the perceived size of a method during inlining
inline_declared_aastored_array_arg_bonus: Bonus given to inlining methods that are passed a register of a known precise type.
inline_precise_reg_class_arg_bonus: Bonus given to inlining methods that are passed a register of a known precise type.
inline_extant_reg_class_arg_bonus: Bonus given to inlining methods that are passed a register that’s known not to be null.
inline_int_const_arg_bonus: Bonus given to inlining methods that are passed an int constant argument
inline_null_const_arg_bonus: Bonus given to inlining methods that are passed a null constant argument
inline_object_const_arg_bonus: Bonus given to inlining methods that are passed an object constant argument
inline_call_depth_cost: As we inline deeper nested methods what cost (or bonus) do we wish to give to deter (or encourage) nesting?
inline_ai_max_target_size: Adaptive inlining heuristic: Upper bound on callee size
inline_ai_min_callsite_fraction: Adaptive inlining heuristic: Minimum fraction of callsite distribution for guarded inlining of a callee
OUTLINE

VIRTUAL MACHINES

OPPORTUNITIES FOR SBSE

TAKE-HOME
Why SBSE?

Manual optimisation is difficult: we have a complex system

Variables include software’s behaviour, phase, and interactions

Solutions are non-obvious and require creativity.
A Commercially Relevant Problem for GP

“I think GP has a toy problem problem.”


This is most certainly not a toy problem!
How are decisions made at the moment?
**Existing Optimisations**

How are decisions made at the moment?

Answer: they’re often not - just deferred!
EXISTING OPTIMISATIONS

How are decisions made at the moment?

Answer: they’re often not - just deferred!

Most recent version of Hotspot JVM has some adaptation.
Existing Optimisations

How are decisions made at the moment?

Answer: they’re often not - just deferred!

Most recent version of Hotspot JVM has some adaptation.

This is a case of “Best effort” - so why not GP?
**A Concrete Example: Heap Size Control**

Execution Time vs. Heap Size:
- Execution Time increases as Heap Size increases.
- Execution Time decreases as Heap Size decreases.

- **High GC**: Execution is faster, but high garbage collection overhead.
- **Unable to complete**: Execution fails due to resource constraints.
- **Sweet Spot**: Optimal Heap Size where execution is efficient and resource usage is balanced.
- **Excessive Paging**: Execution is slow due to frequent paging.

Diagram:
- X-axis: Heap Size
- Y-axis: Execution Time
- "Sweet Spot" indicates the optimal Heap Size.
## Jikes RVM Heap Resizing

<table>
<thead>
<tr>
<th>GC Overhead</th>
<th>Heap Occupancy</th>
<th>0.00</th>
<th>0.10</th>
<th>0.30</th>
<th>0.60</th>
<th>0.80</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.90</td>
<td>0.90</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.01</td>
<td>0.90</td>
<td>0.90</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.02</td>
<td>0.95</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.07</td>
<td>1.00</td>
<td>1.00</td>
<td>1.10</td>
<td>1.15</td>
<td>1.20</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>0.15</td>
<td>1.00</td>
<td>1.00</td>
<td>1.20</td>
<td>1.25</td>
<td>1.35</td>
<td>1.30</td>
<td>1.30</td>
</tr>
<tr>
<td>0.40</td>
<td>1.00</td>
<td>1.00</td>
<td>1.25</td>
<td>1.30</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.25</td>
<td>1.30</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
</tr>
</tbody>
</table>

Look-up table for heap-resize coefficient.
Design Decisions

(Private Communication)

“...back in 2003 [anon] and I did some experimental tuning and came up with the numbers by eyeballing things. At the time, it seemed to be somewhat stable and making reasonable decisions but that was also about 4 major versions ago and I don’t think anyone has really looked at it seriously since then. I think there was some amount of sensitivity to the values...”
OUTLINE

VIRTUAL MACHINES

OPPORTUNITIES FOR SBSE

TAKE-HOME
SUMMARY

- Cloud infrastructure design presents new software engineering challenges.
- Scheduling and memory management are open to optimisation.
- Many problems are amenable to SBSE.
- We are looking for collaborators!

...and relevant existing work!

{david.r.white, jeremy.singer}@glasgow.ac.uk


Previous work on service (application) level by Wada et al. and Nallur et al.

{david.r.white, jeremy.singer}@glasgow.ac.uk