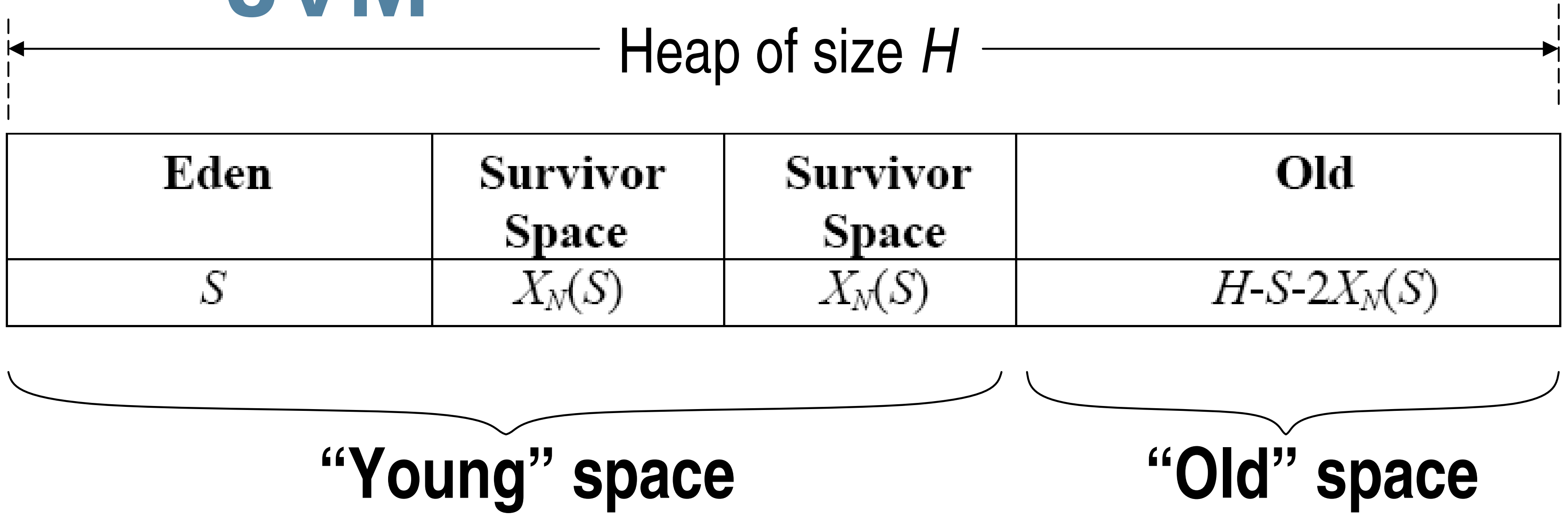
Modeling, Analysis and Throughput Optimization of a Generational Garbage Collector

David Vengerov
Sun Microsystems Laboratories
david.vengerov@sun.com

Outline

- Describe the heap structure of the HotSpot JVM
- Derive the throughput model for the garbage collector and explain its benefits
- Present ThruMax – an algorithm for iteratively adjusting parameters of the garbage collector in the direction of maximizing the GC throughput
- Present experimental evaluation using the SPECjbb2005 benchmark

Heap Structure of the HotSpot JVM



- Application allocates new objects in Eden.
- When Eden fills up, a *minor* GC occurs:
  - > Live data from the Young space is copied into one survivor space
- An object that survived *N* minor GCs is copied into Old
- When Old fills up, a *major* GC occurs:
  - > Live data in the whole heap is compacted into Old

Throughput Model

- Fraction of time when the application is working:

$$f_N(S) = \frac{A(S)K_N(S)}{t_N(S)K_N(S) + T + A(S)K_N(S)} \qquad (1)$$

  > $A(S)$: average time required to fill up Eden with new objects
  > $K_N(S)$: average number of minor GCs between two successive major GCs
  > $t_N(S)$: average time spent on a minor GC
  > T: average time spent on a major GC
  > $d_n(S)$: total size of objects from Eden that are expected to survive at least *n* minor GCs.
  > $X_N(S)$: steady-state amount of live data in the survivor space
  > J: amount of live data surviving a major GC

Benefit of the Throughput Model

- Re-write GC throughput in (1) as $f_N(S) = 1/(1+1/u_N(S))$, where

$$u_N(S) = \frac{A(S)}{t_N(S) + T / K_N(S)} = \frac{A(S)}{t_N(S) + \frac{T \cdot d_{N+1}(S)}{H - J - 2X_N(S) - S}} \qquad (2)$$

- Impact of changing S or N can be evaluated after several minor GCs, which are needed to obtain estimates of $t_N(S)$ and $d_{N+1}(S)$
- Therefore, the adaptation process can be performed much quicker using the throughput model
  > Without it, one would need to wait for several major GCs to occur before evaluating the impact of any change in S or N
- The functions $t_N(S)$ and $d_{N+1}(S)$ are not known analytically => an iterative algorithm is required for maximizing $u_N(S)$

Mathematical Analysis

- Let $G(x)$ be the probability that the lifetime of a newly generated object is less than x. Then,

$$d_n(S) \approx r \int_{(n-1)S/r}^{nS/r} (1 - G(x))dx \qquad J = r\int_0^{\infty}(1 - G(x))dx$$

$$t_N(S) = c + aX_N(S) = c + a\sum_{n=1}^{N+1} d_n(S)$$

- Analysis shows that $t_N(S)$ is a concave increasing function of S and that $d_n(S)$ is most likely concave when it is increasing
- Therefore, $u_N(S)$ most likely increases with S until

  $S + 2X_N(S) > (H-J)/2$, which gives a good initial value of S for the iterative algorithm

ThruMax Algorithm for Iterative Throughput Maximization 1 of 2

1. After every change in S, adjust $X_N(S)$ so that the survivor space utilization will stay within some reasonable boundaries (say 75% to 90%).
2. Observe the new values of $t_N(S)$ and $d_{N+1}(S)$ and estimate numerically the derivatives $t'_N(S)$ and $d'_{N+1}(S)$.
3. Use the above derivatives to compute the predicted value of $u_N(S + \Delta S)$ and $u_N(S - \Delta S)$.
4. If $u_N(S + \Delta S) >$ max[$u_N(S - \Delta S)$, $u_N(S)$], then increase S.
5. If $u_N(S - \Delta S) >$ max[$u_N(S + \Delta S)$, $u_N(S)$], then decrease S.
6. If $u_N(S) >$ max[$u_N(S + \Delta S)$, $u_N(S - \Delta S)$], then stop changing S and switch to adjusting the tenuring threshold N.
7. Repeat steps 1-6 a maximum of 3 times and then switch to adjusting N.

ThruMax Algorithm for Iterative Throughput Maximization 2 of 2

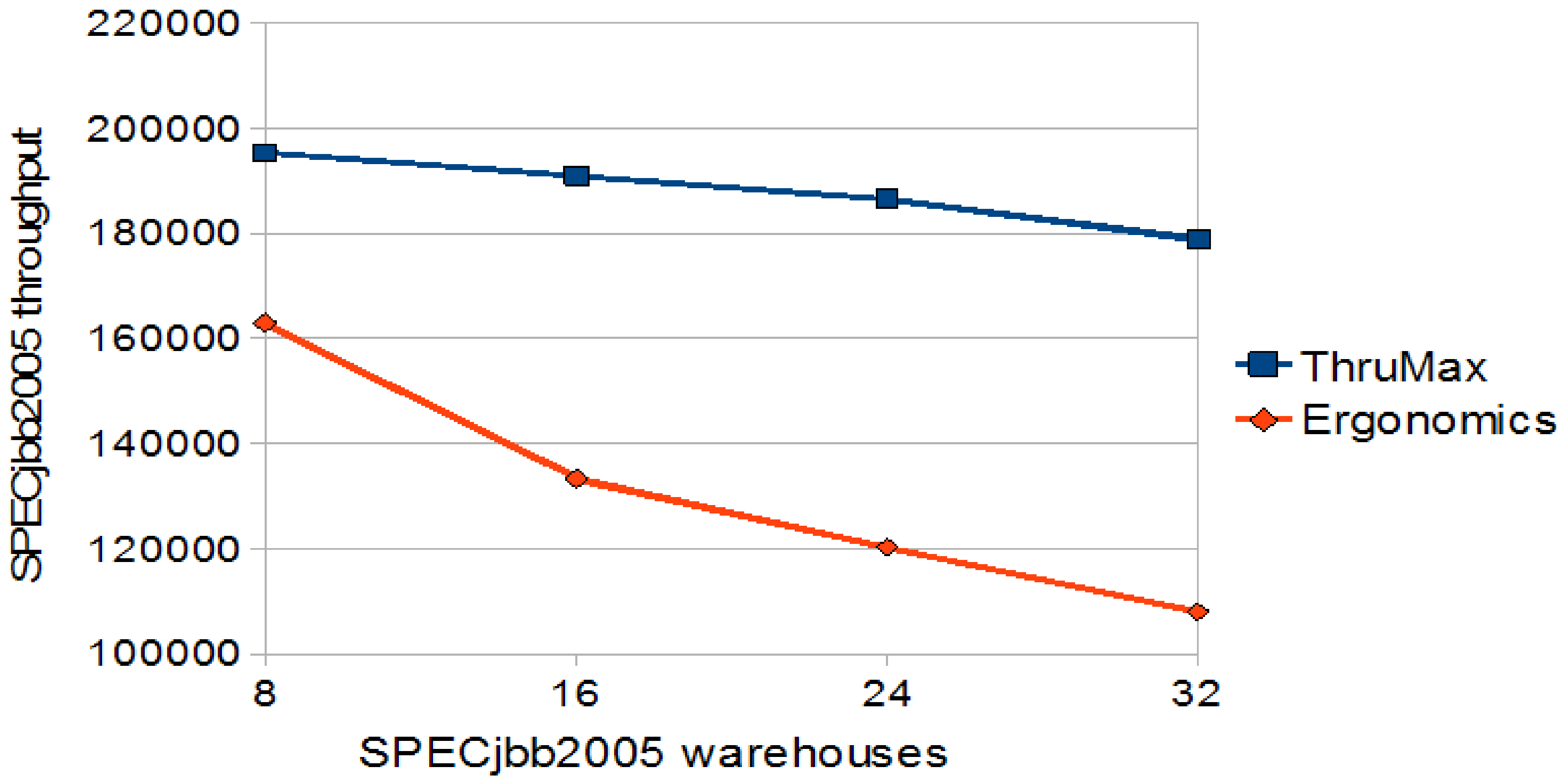
Adjusting the tenuring threshold *N*:

1. Increase N by 1 unless a decrease was suggested in step 4 below during the previous episode of adjusting N
2. Let several minor GCs pass, observe the new values of $t_N(S)$ and $d_{N+1}(S)$ and use them to compute $u_{N+1}(S)$
3. If $u_{N+1}(S) > u_N(S)$, then the increase was successful and another increase is made. Repeat steps 2 and 3 a maximum of 3 times, then switch to adjusting S.
4. If $u_{N+1}(S) < u_N(S)$, then undo the increase and switch to changing S. The next episode of adjusting N will start by decreasing N by 1.

Experimental Evaluation

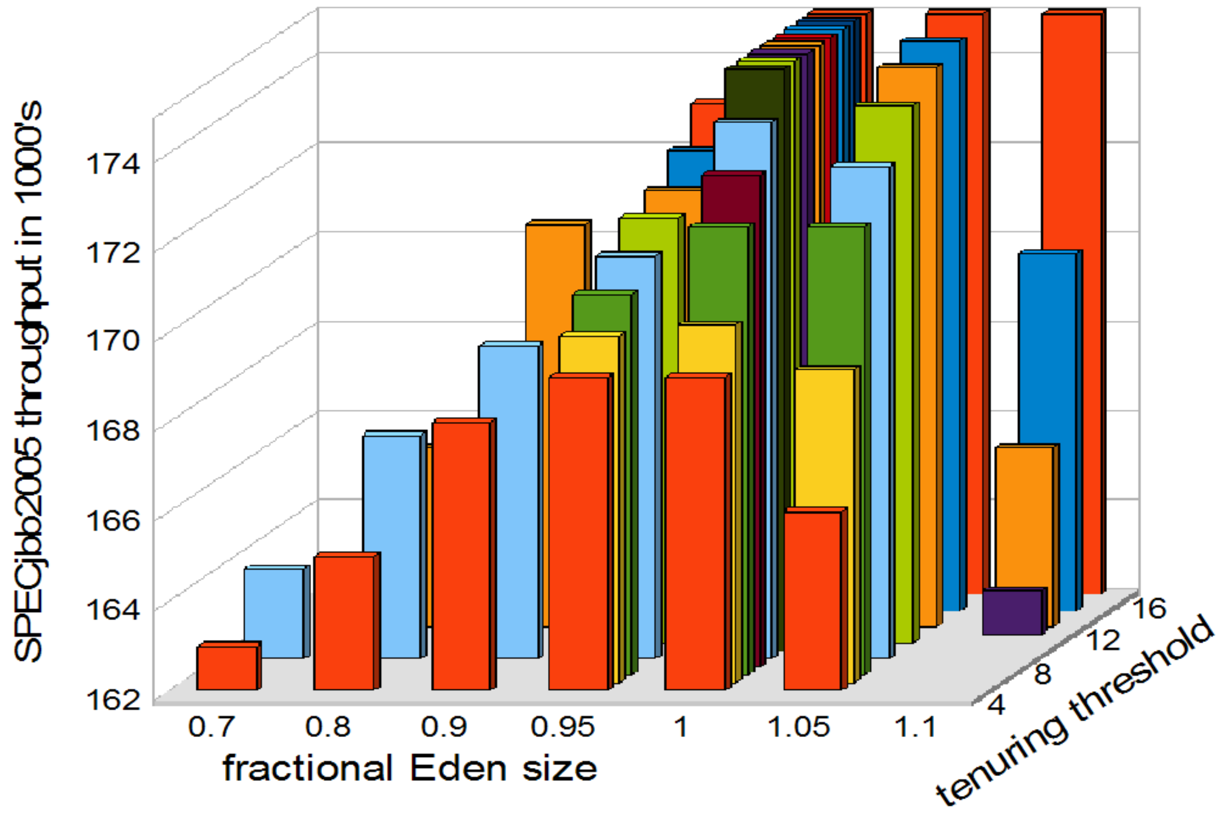
- ThruMax algorithm was implemented in a custom JVM based on the HotSpot sources currently shipped in JDK 6
- Its performance was evaluated using multiple benchmarks: SPECjbb2005, SPECjvm2008, SciMark, and Volano 2.5.
- SciMark and Volano: duration too small
- SPECjvm2008: sub-benchmarks rarely reach steady state
  - > Score on all sub-benchmarks was at least as good as that of the “ergonomics” policy and sometimes much better

SPECjbb2005 Experiments 1 of 3



- Server with 8 cores => maximum throughput at 8 warehouses

SPECjbb2005 Experiments 2 of 3



- Size of Eden given as a fraction of the size to which ThruMax converged for 32 warehouses (which was 861MB in a 2GB heap)
- ThruMax converged to the tenuring threshold of 9, starting from 1
- ThruMax converged to the **optimal** values of *S* and *N*!

SPECjbb2005 Experiments 3 of 3

- Increasing load modeled using the sequence of warehouses {1,2,3,…,32}
- ThruMax first increased the size of Eden until 1460 MB for 12 warehouses, and then started decreasing it until it reached 741 MB for 32 warehouses. The tenuring threshold was steadily increased from 1 until 13.
- The point (741MB,13) gives a near-optimal throughput for 32 warehouses, as can be seen on the previous slide!
- ThurMax algorithm is robust in its parameter adaptation with respect to different paths the workload can take as it reaches a particular level

Conclusions

- ThruMax algorithm continually adjusts the tunable parameters (*S* and *N*) in the gradient direction
  - > gradient of GC throughput with respect to each parameter is continually estimated using the developed throughput model
- Converges to the optimal parameter values for a steady workload
- Keeps re-optimizing parameters if the workload is changing over time