

Multiple-area collection, Regrouping strategies

Ideal strategy follows program execution behaviors

Multiple-area collection

- Problem:
 - CPU cost of scavenging depends in part on size of objects
 - Copying small objects no more expensive than marking with bitmap
 - Cost of copying large objects may be prohibitive
 - Typically contains bitmaps and strings (atomic)
- Solution:
 - Use large object space (separate memory region)
 - Assume objects have header and body
 - Keep header in semi-space
 - Keep body in large object space (use mark-sweep)

Multiple-area collection

- Problem:
 - Some objects may have some permanence
 - Repeatedly copying such objects is wasteful
- Solution:
 - Use separate static area
 - Do not garbage collect such region
 - Trace region for pointers to heap object outside static area
- Preview for generational garbage collection

Incrementally compacting collector

- Divide heap into multiple separately managed regions
 - Allows compacting of parts of the heap
 - Use mark-sweep or other approach on other regions
- Lang and Dupont:
 - Divide heap into $n + 1$ equally sized segments
 - At each GC cycle:
 - Choose 2 regions for copying GC
 - Mark-sweep other regions
 - Rotate regions used for copying GC
 - Collector chooses which transition to take next
 - Give preference to mark-sweep to limit growth of stack

Effects of incremental compactor

- Compact small fragments into single piece
- Compactor will pass through every segment of the heap in n collection cycle
- Small cost: extra segment used for a semi-space

How efficient is Cheney's alg.?

- Suppose:
 - $M \rightarrow$ size of each semi-space
 - $R \rightarrow$ number of reachable object
 - $s \rightarrow$ average size of each object
- Then:
 - # objects allocated between GC cycles: = $M/s - R$
 - If $R = k$, $M/s - R =$ # objects reclaimed in each GC cycle

How efficient is Cheney's alg.?

- Suppose:
 - g → CPU cost of GC per object reclaimed

- Then:

$$g = \frac{c}{\frac{M}{sR} - 1}$$

- g can be made arbitrary small by increasing M
 - Increasing heap size reduces GC time
 - See Jones and Lins, page 129

Garbage Collection locality issues

- **Spatial locality:** if a memory location is referenced at a particular time, then it is likely that its neighbors will be referenced in the near future
- Reasons for locality
 - **Predictability:**
 - one type of behavior in compute systems
 - **Program structure:**
 - related data stored in nearby locations.
 - Easy to access next item
 - **Linear data structure:**
 - code contains loops that tend to reference arrays or other data structures by indices

Garbage Collection locality issues

- On virtual memory systems:
 - Cost of page fault is expensive
 - Tens of thousands or
 - Millions of CPU cycles
 - Additional CPU cycles to minimize page faults are worthwhile

Garbage Collection locality issues

- Two spatial locality issues relevant here
 - MM system will touch every page in to-space
 - MM → allocator + garbage collector
 - Increasing heap size increases number of pages touched
 - Copying GC reorganizes the layout of objects in the heap
 - Will impact spatial locality of heap data structures
 - May compromise mutator's working set

Paging behavior: MSGC vs Copying

- Sophisticated MS
 - Use stack or bitmap for mark-phase
 - Mark phase does not touch/dirty heap pages
 - Lazy sweeping does not affect paging behavior
 - Linked into free list and will soon be reallocated
- Copying GC
 - Next page to be allocated is likely the one LRU
 - LRU is a virtual memory page replacement policy
 - If pages in memory too small to hold both semi-spaces
 - To-space pages will have been evicted before used for allocation

Paging behavior: MSGC vs Copying

- Zorn compared paging behavior of collectors
- Conclusions:
 - Virtual memory behavior of mark-sweep GC noticeably better than that of copying

Regrouping strategies

- Desire for relationship between data be reflected by their layout in heap
- More closely data are related the closer they should be placed in heap
- Relations may be
 - **Structural**: objects are part of same data structure
 - **Temporal**: objects accessed by mutator at similar times
- Placing related data on same page reduces page trafficking since bring data in memory also brings their neighbors

Regrouping strategies

- Objects typically created and destroyed in clusters
- Initial layout of objects in memory reflects future access patterns by user program
- Problem:
 - Copying objects may rearrange their order or layout in the heap
 - The way live objects are regrouped depends on the order that live graph is traversed.

Regrouping strategies

- Can use regrouping strategies to improve locality
 - **Static regrouping**
 - Analyze topology of heap data at collection time.
 - Move structurally related objects more closely
 - **Dynamic regrouping**
 - Cluster objects based on mutator access pattern
 - Objects regrouped on-the-fly by incremental copying collector
- Depth first copying generally yields better locality than breadth-first copying

Copying vs Mark-sweep

<i>Method/Cost</i>	<i>Mark-sweep</i>	<i>Copying</i>
Initialisation	clear mark-bits	flip semi-space
Cost	negligible	negligible
Tracing	mark objects	copy objects
Cost	$O(L)$	$O(L)$
Sweeping	lazily: transferred to allocation	none
Cost		
Allocation	lazily: dominated by init	done directly
Cost	$O(M - R)$	$O(M - R)$

L = volume live data in heap

R = residency user program

M = heap size

- Different constants of proportionality
- Object size is important, especially for copying
- Sophisticate copying collector easier to implement