NG2C: N-Generational Garbage Collector for Big Data Memory Management

Rodrigo Bruno, Paulo Ferreira
{rodrigo.bruno,paulo.ferreira}@inesc-id.pt
INESC-ID - Instituto Superior Técnico, ULisboa
Motivation

- Applications use large amounts of memory to enable fast data access
  - Eg: memory caches used in Cassandra or RDDs on Spark
Motivation

- Applications use large amounts of memory to enable fast data access
  - Eg: memory caches used in Cassandra or RDDs on Spark

- Having lots of data in memory puts too much stress on current memory management technologies, in particular, the Garbage Collector (GC);
Motivation

- Applications use large amounts of memory to enable fast data access
  - Eg: memory caches used in Cassandra or RDDs on Spark

- Having lots of data in memory puts too much stress on current memory management technologies, in particular, the Garbage Collector (GC);

  - ! Leads to big application pauses which compromise performance and responsiveness!
Goals

1. Minimizing GC stop-the-world pauses. How?
Goals

1. Minimizing GC stop-the-world pauses. **How?**

2. Avoiding object copy within the heap. **Why?**
Goals

1. Minimizing GC stop-the-world pauses. **How?**

2. Avoiding object copy within the heap. **Why?**
   - We found that the cost of GC stop-the-world pauses is mostly dominated by the number (and size) of objects to copy:
Goals

1. Minimizing GC stop-the-world pauses. How?

2. Avoiding object copy within the heap. Why?
   - We found that the cost of GC stop-the-world pauses is mostly dominated by the number (and size) of objects to copy:
     - Promotion (moving objects from young to old generation)
     - Compaction (compacting objects within the old generation)
Solution

- Avoid copying objects in memory. How?
Solution

- Avoid copying objects in memory. **How?**

- Replacing the current heap layout of two generations (young and old) by an arbitrary number of generations. **How?**
Solution

- Avoid copying objects in memory. **How?**
- Replacing the current heap layout of two generations (young and old) by an arbitrary number of generations. **How?**
  - Giving the programmer the power to:
    - Create and collect specific generations (run time);
    - To allocate objects directly in a specific generation;
Solution

● Avoid copying objects in memory. **How?**

● Replacing the current heap layout of two generations (young and old) by an arbitrary number of generations. **How?**
  
  ○ Giving the programmer the power to:
    
    ■ Create and collect specific generations (run time);
    ■ To allocate objects directly in a specific generation;

● Each generation should contain objects with similar expected life-cycles
  
  ○ All objects in a generation are expected to die about the same time
  ○ Eg: all objects stored in a cache die when the cache is flushed;
  ○ Eg: all objects created to handle a specific computation task die when the task is finished.
Solution

- Objects allocated in the Eden are copied several times;

- Old generation is hard to collect, potentially leading to full GCs;

2-Generation (traditional) Heap Layout
Solution

✓ Objects allocated in specific generation, near other objects with the same life cycle.

✓ Heap is organized according to object life cycle. Easy to collect.
Code Sample

```java
public void runTask() {
    System.newGen();
    while (running) {
        DataChunk data = new @Gen DataChunk();
        initializeData(data);
        doComplexProcessing(data);
    }
    System.collectGen();
}
```

- Creates a new generation. Allocations with the `@Gen` annotation will go directly to this generation.
- Special annotation for allocating object in specific generation (other than Eden).
- Creates a new epoch in the current generation. Memory previously allocated is now ready to be collected.
Solution

- Is it a good idea to ask the programmer to give hints to the GC?
Solution

● Is it a good idea to ask the programmer to give hints to the GC?
  ○ For most applications: NO!
Solution

- Is it a good idea to ask the programmer to give hints to the GC?
  - For most applications: **NO!**
  - For applications with strict performance requirements: **YES!**
Solution

- Is it a good idea to ask the programmer to give hints to the GC?
  - For most applications: NO!
  - For applications with strict performance requirements: YES!
    - Most applications already resort to several tricks to circumvent the GC (eg: using offheap memory, keep memory objects bounded, etc...)
Solution

- Is it a good idea to ask the programmer to give hints to the GC?
  - For most applications: **NO**!
  - For applications with strict performance requirements: **YES**!
    - Most applications already resort to several tricks to circumvent the GC (eg: using offheap memory, keep memory objects bounded, etc…)
    - Places where generations are allocated and collected are usually well defined;
Preliminary Results

- Simple application (very similar to code sample):
  - 4 threads processing tasks;
  - Each task has 1 GB of data to process;
  - Total working set of 4 GBs.

- Three collectors used:
  - Concurrent Mark-and-Sweep (default GC for OpenJDK < 9)
  - Garbage First (default GC for OpenJDK >= 9)
  - N-Generational Garbage Collector (our collector)

- Both CMS and G1 with young generation sizes of 8 GBs (twice the working set). NG2C with 1 GB.

- Heap size fixed at 12 GBs for all collectors.
Preliminary Results
Preliminary Results

![Graph showing application pause time over time for CMS, G1, and NG2C with corresponding percentiles]

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>50</th>
<th>75</th>
<th>90</th>
<th>99</th>
<th>99.9</th>
<th>99.99</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS (ms)</td>
<td>461</td>
<td>670</td>
<td>853</td>
<td>1873</td>
<td>2048</td>
<td>2065</td>
</tr>
<tr>
<td>G1 (ms)</td>
<td>94</td>
<td>241</td>
<td>588</td>
<td>838</td>
<td>853</td>
<td>854</td>
</tr>
<tr>
<td>NG2C (ms)</td>
<td><strong>11</strong></td>
<td><strong>12</strong></td>
<td><strong>14</strong></td>
<td><strong>16</strong></td>
<td><strong>16</strong></td>
<td><strong>16</strong></td>
</tr>
</tbody>
</table>
Thank you for your time.
Questions?
Suggestions?

Rodrigo Bruno
email: rodrigo.bruno@tecnico.ulisboa.pt
webpage: www.gsd.inesc-id.pt/~rbruno