3-2018

Long Term Assessment of Object Strength in a Web Service as Managed by the Garbage Collection in Java Based Services

Patrick Jackson
St. Cloud State University, pjackson@stcloudstate.edu

Long Term Assessment of Object Strength in a Web Service as Managed by the Garbage Collection in Java Based Services

by

Patrick Jackson

A Thesis

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Information Assurance

March 2018

Committee members:
Dennis Guster, Chairperson
Erich Rice
Balasubramanian Kasi

Abstract

Garbage collection is proving to be an important feature that supports high-performance web services, especially those running data-intensive applications. Due to the use of the object-oriented paradigm, many applications have increasingly opted for the dynamic memory allocation method of assigning their objects in computer memory. During program execution, the application allocates its objects to a memory space called a heap and constantly references these objects within that memory space. With the passage of time, if the objects are not referenced, they become weak/dead to the extent that they can no longer be referenced by an application which allocated them. In such a scenario, the application is required to allocate new objects to a heap in order to continue performing its functions. And, there must be a garbage collection mechanism to remove the dead/weak (unreferenced) objects from the memory heap so that the memory space can be reclaimed and dynamically allocated to other application objects. Java as a Virtual machine, performs memory allocation and reclamation by itself thereby allowing the programmer to concentrate only on the functionality of the application. In other words, the developer is not concerned about how the memory will be managed during the program execution because that will be the duty of the Java language executing on Java Virtual machine.

Therefore, for Java to effectively manage the computer memory, it uses five garbage collection mechanisms which will be explained in detail in the introduction section.
Most of the garbage collections are triggered based on the objects' lifetime predictions set by the developer of the garbage collection algorithms. None or very few consider the strength of the objects that are no longer referenced in the heap. For example, some objects may still be strong enough that they can be referenced by the application but they are collected anyway because they have reached their predicted age threshold.

Garbage collection mechanisms also vary when used in a different framework other than the traditional (standalone) one. For example, garbage collection in distributed systems becomes more complicated as compared to the traditional garbage collection performed in standalone systems. Similarly, garbage collection in a web-service framework has slight differences as compared to the local/standalone systems due to the inclusion of web service technology elements.

In this paper, the goal is to strive to determine the strength of objects that are no longer referenced by an application in a web service as managed by Java-based services; in relation to the performance of a web application.

Keywords: Garbage collection, Apache Tomcat, Object strength, Web service, YourKit Java Profiler

Acknowledgements

It is amazing how some things are learned in life. Honestly I am not a fan of programming, hence I had no idea of what an “object” means in a programming context. In 2016, I enrolled in a data management class which consisted entirely of unix OS instruction and java programming. Day in and day out, I heard about objects and the word “object” kept on echoing in my mind. Eventually I developed an interest to know more about objects and how they are used in programming. In April 2016, I decided to work with Dr. Dennis Guster on a memory management project with the aim of learning more about object’s behavior and eventually develop the thesis paper you are holding right now.

I should admit that it was not easy to develop this thesis considering that the project I was involved in was a big one, and sometimes I would hit a wall due to its complexity. However, I thank Dr. Guster for the assistance he rendered in elaborating some of the critical and complicated processes. Let me also thank Dr. Rice and Dr. Kasi for persistently guiding me to produce this paper. Last but not least, I would like to thank my colleague Andrew Erickson for providing tremendous effort to setup the lab and help perform this complex experiment.

Finally I would like to thank my father Fernand, and my mom Elsie for encouraging me to undertake this project and develop this paper. My wife Mildred and my son Elkan also deserve a big recognition for the role they played in this project. They constantly gave me phone calls just to instill trust and encouragement in me. Glory be to God!

Table of Contents

Chapter Page

Chapter Page

List of Tables

List of Figures

Figure Page

Chapter I: Introduction

Garbage Collection technology has been widely used in managed runtime systems, such as Java virtual machine (JVM) and Common Language Runtime (CLR) systems (Shaoshan, Jie, Ligang, Xiao-Feng, & Jean-Luc, 2012). The principal aim of Garbage Collector design is to maximize the recycled space in memory as applications are running. Basically, a memory heap is segmented into Nursery Object Space (NOS) and Mature Object Space (MOS) and applications use these regions to allocate their objects during runtime.

Typically, during run-time, an application first sets an age threshold on a new object and allocates the object in Nursery space, sometimes called Eden space within the heap. When the nursery space fills up or the object's age is met, a minor garbage collection is triggered and the aged (referenced) objects are moved to the next survivor space and the dead (unreferenced) objects are collected. Eventually, the aged objects need to be collected through the process called major garbage collection but the strengths of these objects are not considered or known. This means that an object might be collected by a garbage collector while it is still strong to the extent that it can still be referenced by an application but it is collected anyway because it has reached its age threshold.

The purpose of this study is to identify the relationship between application performance and the strength of objects in a web-based service. The study involves discovering the strength of objects that have survived both minor garbage collection and general (major) garbage collection phases. In other words, determining the strength of objects in a mature object space (MOS) will help to discover if the objects are still in a state that they can be referenced by an application and how that factor affects the observed performance of the application.

Many algorithms have been implemented that predict the lifetime of an object, "predicting which allocations will result in long-lived objects and then allocate them to regions that are not frequently collected" (Hajime, Darko, & Forrest, 2006). However, no research has been conducted on finding out the strength of objects that have passed through nursery space, survivor space 0, survivor space 1 up to mature object space. This observation suggests the usefulness of thorough research to understand the strength of objects that have aged in a heap. In this paper, the assessment was based on the two main characteristics: responsiveness and throughput of an application. This was done by examining the application events implemented by Apache that supports high-performance web services. Upon examination of the application events, it became clear how strong the objects were with the passage of time. Responsiveness basically focuses on how quickly an application or system responds to a data request. For instance, a system's responsiveness can be measured by determining the time required for its user interface to respond to an event, the response time of a website to return a page, and time required for a database query to be returned.

On the other hand, throughput is a measure of the amount of work an application can complete in a given period of time. Examples of measures of throughput include the number of transactions completed in a specific time- period, the number of tasks that a batch program can complete in a specific time-period, and the number of database queries that can be completed within a given period of time. These two assessment criteria were used to determine the strength of objects in a web service as managed by the garbage collector.

In this study, Java virtual machine (JVM) was used because of its capabilities to perform automatic garbage collection, its platform independence, its object-oriented architecture, and its rich standard library.

Problem Statement

The ideal garbage collection algorithm should collect only those objects that can no longer be referenced by an application. Most of these algorithms are designed in such a way that they operate based on age threshold. Once the chosen age threshold is reached, the garbage collector is triggered to collect all the objects that are no longer referenced by an application. By doing so, the garbage collection process ensures that the computer memory is utilized efficiently by maximizing the throughput.

However, these algorithms are only triggered when a certain condition (the age threshold) is met not necessarily that most of the objects are dead/weak in a memory heap. In some circumstances, objects might be unreachable due to the idleness of the system or an application. In this case, the garbage collector might be triggered at a time when the application or system is idle, causing the garbage collector to still collect these unreferenced objects from the memory heap with an assumption that the objects are dead or weak. In fact, the objects are not necessarily dead or weak, but they simply cannot be referenced by the idle application which created them. Most often, an object dies (becomes unreferenced) when an assignment operation causes the object's last reference to be overwritten. Similarly, if an object's last reference is on a stack frame and suddenly the frame's method returns an exception, that causes an object to die.

The inefficiency of this type of garbage collection system represents a significant performance burden for the application. Consider the application that has just transitioned from the state of idle to active. In this scenario, an application is required to generate and allocate new objects to the memory heap since the previously allocated objects have been collected due to the idle state of the application. The generation of new objects requires computing power in the form of CPU utilization, hence, it affects the throughput of the system as a whole.

Nature and Significance of the Problem

Examining the strength of objects that are deemed to be dead or weak for not being referenced by the application, will help to design efficient and effective garbage collection algorithms that do not waste strong objects merely based on age threshold or idleness of the application that created them. Anything that the garbage collector cleans up is something that the application itself allocated, so in view of that, studying the strength of an object will not only help in designing effective garbage collection algorithms but also fine-tuning the application itself. Besides, the computing power wasted on repeatedly generating application objects can be used on other important computing functions hence maximizing throughput of the entire system.

Objective of the Study

The main objective of this study is to determine the strength of objects in a computer memory heap in relation to the performance of the application in a web service architecture as managed by Java services. The study will help to provide recommendations on the best way to develop a garbage collection algorithm that will not allow the wastage of objects that are still strong and at the same time maintain the performance of the application.

Hypotheses

If application objects in a memory are unreferenced for a certain period of time, they eventually die (lose pointers to the application) or become weak and the garbage collector removes them to create space for more object allocation. However, the fact that an object has lost its pointers to the application that created it does not necessarily mean that it is dead or weak. Sometimes, an object may lose its pointers or become weak due to the idleness of the application that created it. Most often, as already mentioned above, an object dies (becomes unreferenced) when an assignment operation causes the object's last reference to be overwritten. The study

conducted by Raqeeb, Guster, & Schmidt (2017) proved the point that object's last reference can be overwritten. In that study, it was observed that a number of objects became unreachable when the heap was filled with random characters (values). It suggests that the previously allocated objects references were overwritten by those new random characters causing them to lose their pointers to the application which created them.

Similarly, if an object's last reference is on a stack frame and suddenly the frame's method returns an exception, that causes an object to die. In that regard, it is believed that some application objects in the heap may still be strong enough even though they are not being referenced by the application. More precisely, the study wants to answer the following questions:

1. Can premature collection of objects be avoided
2. Are objects really collected at their weakest stage
3. Are application objects in a web architecture reusable

In an attempt to answer the three questions raised above, the author predicted the following:

1. Some application objects in the heap may still be strong enough even though they are not being referenced by the application
2. Some objects are prematurely collected, and thus, the scenario can be avoided
3. Objects in a web architecture might be reusable

Introducing Services

This section briefly describes the services that have been used in this study. The associated elements of each service are fully explained so as to give a clear understanding of how the elements interact with each other.

Web services. Simply put, a web service is an interface positioned between code and the user of that code. It provides a method of communication between two electronic devices over a

network that allows two software systems to exchange data over the internet regardless of infrastructure or language-specific details. Different software products may use different programming languages, so there is a need for a mechanism of data exchange that does not depend upon a particular programming language.

Most types of software have capabilities to interpret XML tags. Hence, web services often use XML files for data exchange (Wikipedia, 2017). In brief, XML is a data structure that contains both data and metadata within the same structure. A web service may also use the JSON structure to transfer data. WSDL is a metadata structure used for describing the services available and UDDI lists what services are available. These elements are described below.

JavaScript Object Notation (JSON). JavaScript Object Notation (JSON) language uses name/value pairs, similar to the tags used by XML. Despite using conventions of C-family of languages, JSON is completely language-independent whose data-interchange format can easily be read and written by humans. Primarily JSON is responsible for structuring data in a readable format and transmit that data between a server and web application. In fact, “XML and JSON are more flexible and dynamic as they capture the information and its metadata” (Helland, 2017). JSON is mainly built on two structures that can be nested, namely: a collection of name/value pairs and an ordered list of values (JSON, n.d.).

An example from the figure below shows, at the left, the XML tag of "<city>" with the value of "Lilongwe." The pairs for JSON are at the right. It similarly shows the name "city" is paired with the value "Lilongwe." Note that the name/value pairs do not need to be in a specific order.

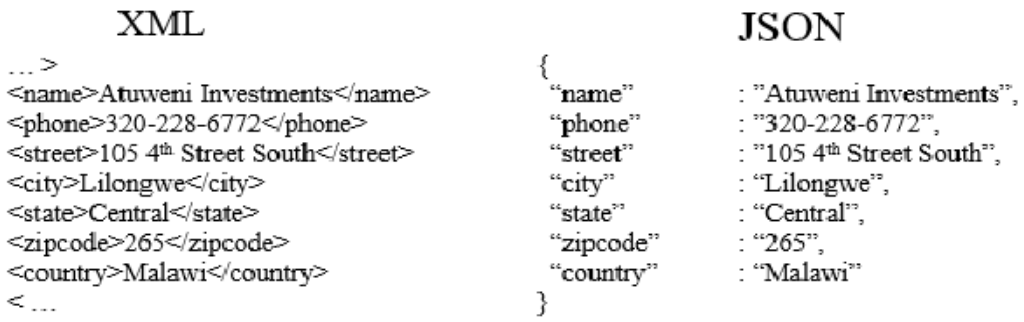


Figure 1. A Depiction of XML and JSON

Web Service Definition Language (WSDL). WSDL is an XML language that describes the functionality provided by web service. It includes defining "network services as a set of endpoints that function via exchanging messages containing either document-oriented or procedure-oriented information" (Guruge', 2004). Services are defined using seven major elements: data types, message, operation, port type, binding, port, and service which is used to combine a set of related ports. The figure below shows the WSDL definition of a simple service providing stock quotes and the associated elements mentioned above, used simultaneously with SOAP.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>
```

```
    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Figure 2. WSDL Document Example
Note: Reprinted from Christensen, Curbera, Meredith, & Weerawarana (2001).

Universal Description, Discovery and Integration (UDDI). This is a directory that defines the type of software system to contact for specific type of data. Every time an application, in this case a service requester, needs one particular report/data, it uses WSDL to contact the UDDI in order to locate any other system it can contact for receiving that report/data. The service broker would reply with the information about the system to contact in order to get the requested data/information. Once the application receives the reply and discovers which other system it should contact, it would then contact that system using the JSON protocol.

Prior to processing and sending data to the service requester under the JSON protocol, the service provider system would first validate the data request by referring to the WSDL file. The figure below demonstrates how UDDI interacts with other web elements.

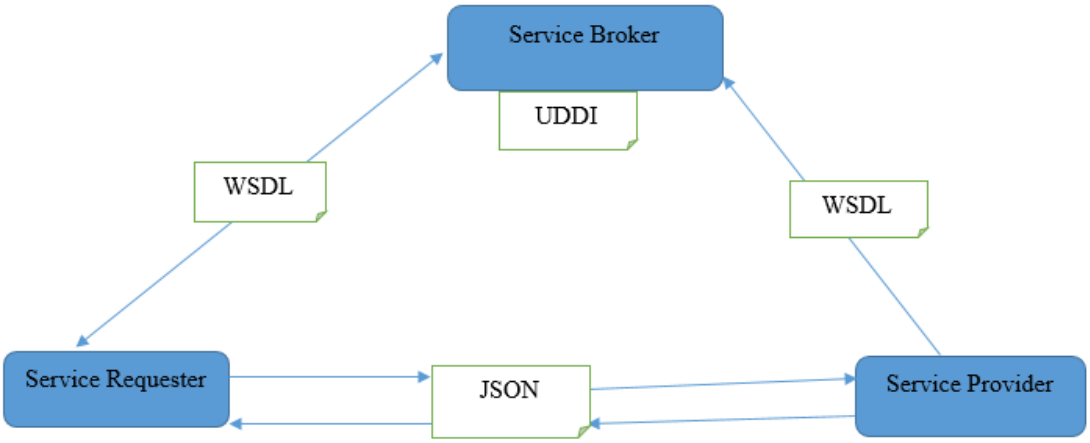


Figure 3. Web Service Architecture

The standards-based interfaces offered by web services enhance interoperability since any language that supports the web service can access the deployed application (Kulchenko, Tidwell, & Snell, December, 2001). For example, web services allow two separate infrastructures such as Windows and Unix systems to operate which would be difficult to integrate without the web.

Apache Tomcat. This application server is an open source implementation of the Java Servlet, Java Server Pages, Java Expression Language and Java Web Socket technologies. Apache Tomcat provides a "pure Java" HTTP web server environment to run Java code. It also includes configuration and management tools that allow direct configuration changes by editing XML configuration files. Apache Tomcat is a cross-platform software product, written in Java, which runs on a variety of operating systems.

Java Server Pages (JSPs). JSPs are basically web pages with embedded Java code. When the client requests data from the web server, the server executes the embedded Java code before returning the page to the browser. To illustrate JSPs, observe the following code:

```
<html>
    <body>
        <%
        out.println("<h1>Programming is fun!!!</h1>");
            %>
    </body>
</html>
```

The code above illustrates the standard HTML web page. The web page above is comprised of standard HTML and Java code embedded between the <% and %> character sequences. The <% and %> along with the embedded Java code is called a **scriptlet**. Executing the above code will produce the web page that shows plain text “Programming is fun!!!” Note that the text “Programming is fun!!!” could have directly been incorporated into the html code, but to illustrate dynamic web pages java script was used.

When the browser sends a request to the web server for a JSP, the web server transfers control to a JSP container. A **container** interacts with the web server to provide the runtime environment and other services a JSP needs. It knows how to interpret the special elements that are part of JSPs. If this is the first time this JSP has been invoked, the JSP container will convert it into an executable unit called a **servlet.** The entire page, including the parts that are in HTML, is translated into source code. Thereafter, when the code has been translated, the JSP container compiles the servlet, loads it automatically, and then executes it.

In addition, Mark, Allan, & Kunal (2003) elaborates that “typically, the JSP container checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP has not changed and that the generated servlet still matches the JSP’s contents”. Due to the length of time taken to generate and compile a servlet, the JSP container tries to minimize the number of compiles it has to perform by avoiding unnecessary compiles (Mark, Allan, & Kunal, 2003).

Servlets. Servlets form the underlying technology behind Java Server Pages. Servlets are composed entirely of Java code, unlike JSPs which are combination of HTML and scriptlets or other elements. In addition, JSPs focus more on presentation than servlets. Particularly, Mark et, al. (2003), explain that “when a request comes in for a particular servlet, the servlet container loads the servlet (if it has not yet been loaded) and invokes the servlet’s service method” (Mark, Allan, & Kunal, 2003). They further stated that “the method takes two arguments: an object containing information about the request from the browser and an object containing information about the response going back to the browser” (Mark, Allan, & Kunal, 2003). Thereafter, the servlet should inform the web browser the kind of content being returned. In several circumstances, HTML content is returned so it is better to set the content type to text/html before starting sending text back to the browser. However, content type can also be set to text/xml if you prefer to return data in XML format.

Many books have illustrated various procedures to run a servlet. Among them, the invoker servlet is the easiest. The administrator installing Tomcat must enable the invoker servlet by uncommenting the servlet declaration in web.xml file found in conf/ directory within Tomcat installation folder. Doing so informs Tomcat of the invoker servlet’s configuration.

Alternatively, a URL that includes the path “servlet/” can be used to invoke a servlet. The invoker matches whatever follows “servlet/” to a servlet class in the container’s classpath and executes it.

Note: Caution should be taken when using a /servlet/ pattern. It is not good practice to use a /servlet / pattern on production servers because it can randomly load Java classes despite not being servlets. The /servlet / pattern first loads the classes before it can determine whether they implement the servlet interface. Hence, a hacker could force the container to load a class and potentially harm or compromise the production server.

Despite the fact that servlets can be executed individually, it is recommended to package them into WAR (Web Archive) files. A WAR file may contain servlets, JSPs, HTML files, JAR files, and even other classes. This kind of component mix is well-managed by a file called web.xml. Web.xml is a deployment descriptor that contains all the necessary definitions for the application, including the list of servlets and their associated pathnames. The major difference between a JSP and a servlet is in the way responses are returned to the browser. While most JSP responses are embedded in the JSP in the form of static text in a template, the servlet response is usually in the form of code—usually calls to out.print and out.println. At minimum, a servlet needs to perform two tasks to send a response: (1) Set the content type and (2) write the response. The ServletRequest and ServletResponse objects assist with these tasks. Any references to these objects are given to the servlet as parameters of the service method. Every time a browser requests a file from the server, the server sends back a content type along with the file. The content type informs the web browser how it should display the file (Mark, Allan, & Kunal, 2003). Most web servers rely on filename extensions to determine the content type. With servlets, it is difficult (if at all possible) for the web server to precisely determine the content type the servlet is going to return. Instead, the servlet dictates the webserver what is being returned (Mark, Allan, & Kunal, 2003).

When the web server receives an initial request, a Java servlet handles the request. The Java code embedded on the servlet page is executed. Thereafter, the servlet calls a JSP to provide the output in order to send a response back to the browser (Mark, Allan, & Kunal, 2003).

Java Expression Language. This is a library designed to support the implementation of dynamic and scripting features in Java-coded applications and frameworks. Using JEXL, a programmer is given the capability to enter his own expression into a program. For instance, a programmer is given the capability to plot user-defined functions, calculate integrals involving user-defined functions, and to fit the data by random user-defined functions. JEXL was designed for two main purposes: to create a simple expression compiler that will generate extremely fast executable code, and to make the language have direct access to all built-in Java data types and functions just like Java language. The Expression Language supports the dynamic access of data stored in the Java Bean component, and other objects like request, session, and application. JEXL's emphasis is mainly on code execution time and not on compilation time. It compiles expressions directly to Java bytecodes, making the evaluation process extremely fast. Moreover, no recompilation is required when JEXL is transferred from one platform to another. Generally, JEXL maximizes program performance because there is no need to write an interpreter for expressions.

Below is an example of JEXL code that is using an expression. As can be seen from the code, the test attribute of the conditional tag is provided with an EL expression that compares 0 with the number of items in the session-scoped bean named cart.

```
<c:if test="${sessionScope.cart.numberOfItems > 0}">

...

</c:if>
```

Note: Reprinted from Oracle (2010).

Java Web Socket technologies. WebSocket is a full-duplex protocol that uses a single TCP connection for communication between the client and the server/endpoint. Unlike HTTP, WebSocket allows for simultaneous two-way communication and has much smaller header. The small header allows for more efficient communication even over small packets of data.

The WebSocket protocol is divided into two parts: handshake and data transfer. A handshake is initiated when the client sends a request to a WebSocket endpoint using its URI. But, the handshake must be compatible with existing HTTP-based infrastructure. Web servers interpret a handshake as an HTTP connection upgrade request. Lissack (2013) briefly described the WebSocket lifecycle as follows:

1. Client sends the Server a handshake request in the form of a HTTP upgrade header with data about the WebSocket it's attempting to connect to (Lissack, 2013).
2. The Server responds to the request with another HTTP header, this is the last time a HTTP header gets used in the WebSocket connection. If the handshake was successful, then server sends a HTTP header telling the client it's switching to the WebSocket protocol (Lissack, 2013).
3. Now a constant connection is opened and the client and server can send any number of messages to each other until the connection is closed. These messages only have about 2 bytes of overhead (Lissack, 2013).

The code below demonstrates how a server socket is built:

```
import java.net.ServerSocket;
import java.net.Socket;
public class Server{
    public static void main(String[] args){
        ServerSocket server = new ServerSocket(8080);
        System.out.println("Server has started on 127.0.0.1:8080.\r\nWaiting for a
connection...");
        Socket client = server.accept();
        System.out.println("A client is connected.");
    }
}
```

When the ServerSocket class is instantiated, it is bound to the port number specified by the *port* argument. The WebSocket protocol is currently supported in most major browsers and minimizes latency due to the less bandwidth requirement.

Overall, it is worth noting that Tomcat can act as a stand-alone Web server and also as a servlet/JSP engine for other Web servers. Tomcat also includes tools for administering the server and applications. Furthermore, Tomcat comes in a version that can be embedded into other applications.

YourKit Java Profiler. YourKit Java Profiler is a full featured, easy to use, low overhead profiler for Java EE and Java SE platforms. YourKit solutions have facilitated both CPU and memory profiling on huge applications while maintaining maximum throughput and zero overhead. YourKit has brought many benefits to professional Java developers because of its capabilities to profile applications at both production and development phases. Its uniqueness in

terms of providing high level results and automated analysis surpasses a traditional profiler's capabilities. Specifically, Yourkit Java Profiler has the capability to profile memory leaks, usage, and garbage collection. According to the Yourkit official website, the profiler is able to produce memory usage graphs that show heap and non-heap memory pools, garbage collection activity and, if recorded, object creation rate per-second. The profiler also performs a comprehensive heap inspection and analysis (YourKit, 2017). In addition, YourKit Java Profiler records object allocation which is more useful in solving garbage collection and memory allocation issues. To some extent, it traces "paths from roots to analyze memory leaks and object retention, with the ability to immediately see what would happen if particular references were excluded (i.e. to test a proposed memory leak fix effect without re-running the application)" (YourKit, 2017). Sometimes, it identifies objects holding most memory with dominator tree and class list. And best of all, the profiler categorizes objects by class, class loader, web application, generation (time of creation), reachability, shallow size range and allocation point if recorded.

As already mentioned above, YourKit Java profiler has many advantages and some of the benefits are expanded below.

1. High-level results.

YourKit Java profiler achieves higher-level results through event recording and performance charts. The profiler can record higher level events, for instance, database queries, web requests and I/O calls besides low level profiling results like method calls. Generated performance charts display basic and higher-level telemetry graphs of web, database, and I/O activity.

2. Powerful analysis capabilities.

Yourkit Java Profiler produces real-time profiling results immediately. In some cases, the profiler is able to perform a sophisticated analysis by capturing a snapshot, saving the results for history records and eventually sharing the results. YourKit Java Profiler also has capabilities to compare performance or memory snapshots to determine if there are any changes. Most importantly, the profiler uses lightweight basic telemetry graphs which allow recent telemetry results to be remembered inside the profiler agent. This is a very useful and unique feature because it enables the user to see how the application behaved in the past by being able to connect to the profiled application on demand. Lastly, YourKit Java profiler has an automatic inspection feature that detects typical problems which would be very difficult to inspect manually.

Garbage Collection in Java. Garbage collection is very important in Java applications development to ensure memory is utilized efficiently. There are hundreds of Garbage collection algorithms but according to JDK 7, there are only five types of garbage collection in Java namely;

1. Serial garbage collection
2. Parallel garbage collection
3. Parallel Compacting garbage collection
4. Concurrent Mark and Sweep garbage collection
5. Garbage first (G1) collection.

The brief description of each of these methods is provided below.

Serial Garbage Collection. Before sweeping, the algorithm first marks the surviving objects in the old generation. Then the sweeping process involves checking the heap from the

forepart and leaving only the surviving objects behind. Lastly, the algorithm compacts the heap by filling up the heap from the forepart with the objects so that they are piled up consecutively. Thereafter, the algorithm divides the heap into two parts: one part with objects and another without objects (Lee, 2017). The figure below summarizes the process serial garbage collection follows.

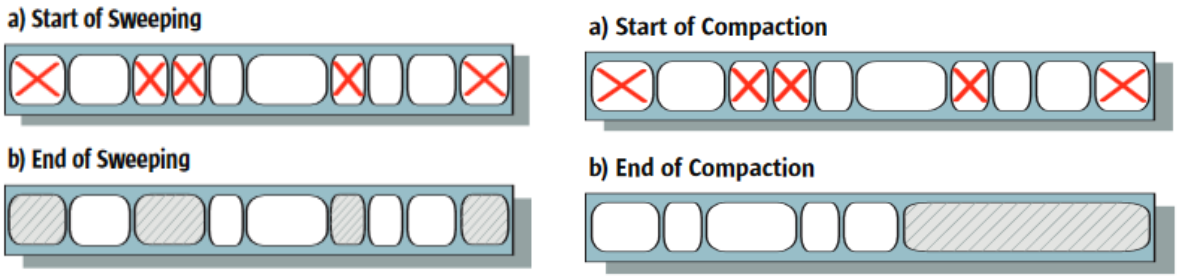


Figure 4: Illustration of Serial Garbage Collection
Note: Reprinted from Sun Microsystems (2006).

The serial Garbage Collection method works best for a small memory and a small number of CPU cores (Lee, 2017). It is not recommended to use Serial Garbage Collection on an operating server because it freezes all the application threads while performing garbage collection. Initially the algorithm was designed when there was only one CPU core on desktop computers. Hence, using this serial Garbage Collection will remarkably reduce application performance.

Parallel Garbage Collection. Unlike the serial Garbage Collection method which uses only single thread to process a garbage collection, the parallel Garbage Collection method uses multiple threads to perform a garbage collection. As a result, Parallel Garbage Collection is faster, compared to Serial Garbage Collection. However, it is worth noting that Parallel Garbage Collection is practical when there is sufficient memory and considerably a large number of cores

(Lee, 2017). In some books this type of algorithm is also called “throughput Garbage Collection”.

Parallel Compacting Garbage Collection. This type of algorithm goes through three steps: mark, summary, and compaction. In the summary phase, the algorithm identifies the surviving objects separately in order to re-allocate them to the areas that the Garbage Collection has previously processed. The algorithm is slightly different from the Parallel Garbage Collection in the sense that the Parallel Compacting Garbage Collection is usually used to clean old generation. When the algorithm marks the objects, it does not immediately perform the sweep process but rather it first summarizes the objects, making the process a bit more complex.

Concurrent Mark and Sweep Collection. This type of Garbage Collection algorithm is the most complex algorithm of those compared in this paper. It requires more memory and CPU as compared to other Garbage Collection types. And, by default the compaction step is not provided. However, the algorithm has the benefit of short stop-the-world time. With this algorithm, the surviving objects that are among the objects closest to the class loader are analyzed first (Lee, 2017). This is the early initial mark step and the pause time required is very brief. Then in the concurrent mark step, the algorithm tracks and checks the surviving objects that it has just confirmed. The concurrent mark step does not have a pause time; it progresses while other threads are being processed simultaneously. Next the algorithm goes into the remark step, whereby it checks the objects that were newly added or cease to be referenced in the concurrent mark step (Lee, 2017). Finally, the Garbage Collection procedure is triggered in the concurrent sweep step. So, other threads continue being processed while the garbage collection is performed (Lee, 2017). As already stated above, the algorithm has a very short pause time because of the manner it performs its procedures. Thus, response time from all applications is

paramount to the usefulness of this algorithm. Concurrent-Mark-Sweep algorithm is sometimes referred to as low latency Garbage Collection (Lee, 2017).

Garbage First (G1). Garbage first algorithm was created to replace the concurrent mark sweep garbage collection method which had several issues due to its complexity. The algorithm works as follows: First, it divides the heap into a set of equal-sized regions, each an adjacent range of virtual memory. Then it determines the live objects throughout the heap by marking them, the process called concurrent global marking. Upon completion of concurrent global marking phase, the algorithm knows specifically the regions which are mostly empty. These empty regions are collected first resulting in reclamation of large amount of space. Garbage first algorithm dedicates its collection and compaction activity on the areas containing many dead objects (garbage) attributing to its name "garbage first". Garbage first algorithm allows a user to specify pause time so that it can select the number of regions to collect. The algorithm achieves this by the use of a pause prediction model that works based on the user specified pause time target.

While live objects are being evacuated from the identified reclaimable regions, the algorithm simultaneously copies objects from one or more regions to a single region on the heap. The evacuation process is carefully performed in parallel on multi-processors, to reduce pause times and maximize throughput. Then the objects are compacted and in the process large memory space is released (Detlefs, Heller, Flood, & Printezis, 2004). Once that region is full, the live objects are allocated to another region and then a Garbage Collection is performed. Hence, every time Garbage collection is triggered, the algorithm tries to defragment the memory space but working within the user defined pause times. This is an added advantage to the Garbage first

algorithm as compared to parallel compacting garbage collection which compacts the whole heap and resulting in considerable pause times.

Garbage first algorithm is the fastest of all the algorithms discussed above resulting in the highest performance of all the above-mentioned algorithms. And it is worth noting that the Garbage First collector targets multi-processor machines with large memory space preferably servers. Below figure represents the summary of garbage collection in Java.

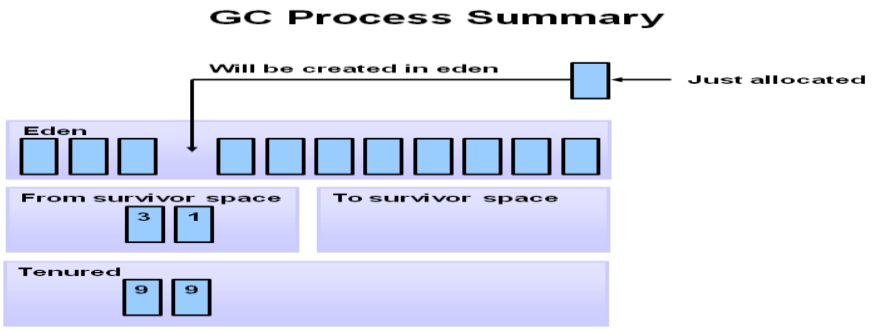


Figure 5: Garbage Collection Process in Java.
Note: Reprinted from Williams & Chitta (n.d.)

NB: In Java, JVM references GC roots objects which in turn keep every other object from being garbage-collected.

Definition of Terms

Table 1: Definition of Terms Used in this Study

Term	**Definition**
GC	Garbage collection; automatic process of freeing up space in a computer's memory by removing data that is no longer required or in use.
Algorithm	A process or set of rules to be followed in problem-solving operations by a computer.
Memory heap	Area of computer memory used for dynamic object allocations
CPU	Central Processing Unit of a computer.
RMI	Remote Method invocation

Chapter Summary

In this chapter, the process of garbage collection has been described and the garbage collection algorithms have been classified accordingly. It has been observed that most of these garbage collection algorithms remove unreferenced objects from the memory heap when objects' age threshold is met. The strength of the removed objects is generally not considered. And this study predicts that some application objects in the heap may still be strong enough to be referenced by an application even though they reach their age threshold. The study of the strength of these unreferenced objects will assist in the development of more effective garbage collection algorithms as well as web applications.

Finally, the chapter gave a description of some of the services as well as technical terms used in the paper. The next chapter will give insight into the background of the problem that has been explained above.

Chapter II: Background and Review of Literature

This chapter discusses efforts that have been put so far by various researchers to develop efficient garbage collection algorithms. In addition, the chapter gives a detailed explanation of various strategies employed by other researchers in an attempt to study the behavior of objects in a memory heap. Furthermore, the chapter distinguishes the garbage collection in a local systems from the distributed systems. An understanding of garbage collection in a distributed system will help to devise a good and clear approach to garbage collection in a web service since the two are closely related.

Background Related to the Problem

Previous research in garbage collection focused more on throughput maximization, than determining the strength of the object to be collected at a particular time. And, if research is restricted to object strength, there has been only a small amount of work done and the subject remains mostly unexplored. With an understanding of why the heap is separated into different generations, it is useful to look at the strength of objects found in those segments of the memory heap. In order to find the memory location of a heap, for example, a java program can be executed, and using the program's process ID a memory mapping command can be executed which will reveal the relative memory address range of the heap. The objects in this memory range can then be observed over time to analyze their strengths as the application continues to reference them.

Unfortunately, in most garbage collection algorithms developers did not include the strength of objects as a concern. But rather they were concerned with lifetime predictions of an object. For instance, Hajime, et.al (2006) constructed an object lifetime predictor which based its predictions on information available at allocation time. The information included the dynamic

sequence of method calls that caused the request and the actual type of the object being allocated. They classified this information as request allocation context. This predictor was designed in such a way that it would observe lifetimes of all objects with the same request allocation context. If there was uniformity in the lifetimes of all objects with the same request allocation context, then "the predictor would predict that value at runtime for all objects allocated at the site" (Hajime, Darko, & Forrest, 2006). It implied that the objects were being removed from the memory heap by the garbage collector when the value predicted at allocation time was met. The strengths of these objects at that particular time was not taken into account by the predictor.

Literature Related to the Problem

The issue of strength of object at the time of collection becomes more complicated when dealing with distributed systems. Sometimes it is possible for a premature collection of remote objects to occur. According to an Oracle article published in 2010, this is likely to occur in cases where a network partition exists between a client and a remote server object, causing the transport assume that the client crashed (Oracle, 2010). Due to the likelihood of premature collection, remote references cannot ensure referential integrity because not all objects exist for remote reference. Therefore, an application is required to handle "RemoteException" errors if a remote reference is made to a non-existent object (Oracle, 2010).

It is worth noting that it is preferable to automatically remove those objects that no longer hold references to the client in a distributed system just as in a local system. However, in a distributed system this is achieved by the use of a reference-counting garbage collection algorithm similar to Modula-3's Network Objects (Andrew, Greg, Susan, & Edward, 1995). The Oracle (2010) article states that a reference counting garbage collection algorithm helps ensure

that the object is not prematurely collected by making sure that the protocol maintains the ordering of referenced and unreferenced messages (Oracle, 2010).

The article further explains that within each Java virtual machine, all live references are tracked by the RMI runtime to achieve the reference-counting garbage collection. When the Java virtual machine receives a live reference, it increments the reference count of that particular live reference. The server receives a "referenced" message from the first reference to an object. Conversely, as live references become unreachable, the local virtual machine decrements the count. Finally, the server receives an "unreferenced message" when the last reference has been discarded (Oracle, 2010).

Then the RMI runtime refers to an unreferenced remote object using a weak reference. In the research 115 SRC report titled "Network Objects", Andrew, B. et al. (1995) observed that once the remote object is referenced using a weak reference, the Java virtual machine's garbage collector discards the object if no other local references to the object exist (Andrew, Greg, Susan, & Edward, 1995). This implies that any loss in connectivity will cause the remote object to be marked for collection due to a loss in reference even though the object might still be strong enough that it can be referenced by the client.

Literature Related to the Methodology

Similarly, in an attempt to find how secured a heap is in memory, Raqeeb, et.al (2017) managed to locate a heap in a very small segment of memory. They used the process ID of a java_class that was not yet loaded into Kernel space but rather in a user-space (low area of memory space) for easy viewing and modification of the heap. Thereafter, they attached a GNU debugger to the process ID of the java_class to dump the relative memory address associated with the heap and extract the contents for that process ID. Eventually, they were able to notice

the objects that the heap was mapping for that particular process ID (Raqeeb, Guster, & Schmidt, 2017).

The fact that they were able to see the mapping of objects and the heap, gives a very important clue to this research since it is possible to manipulate the heap and observe the behavior of mapped objects and eventually determine their strengths. This paper leverages the fact that the memory heap can be easily viewed and modified at a user space or lower memory level.

To some extent, Raqeeb, et al. (2017) tried to assess the strength of objects in a heap when they were analyzing denial of service and performance issues at memory level. Yourkit Java Profiler was installed on an Ubuntu server and executed a command to attach the profiler to a java application. After the attachment, the profiler and application were left running for a day. Immediately after execution, it was observed that most of the objects were being referenced as the client was making calls to the server. However, after a day, a higher percentage of objects became unreferenced but the objects were not yet collected by the garbage collector. The researchers attributed this loss of reference to objects as the result of not refreshing the application for a long period of time (Raqeeb, Guster, & Schmidt, 2017).

In addition, researchers modified the memory heap with random values which showed a rise in object allocation and a decrease in a number of objects reachable by the garbage collector (Raqeeb, Guster, & Schmidt, 2017). The study was not explicit as to whether the application objects were no longer reachable to the garbage collector due to a decline in strength or because other objects were allocated when random values were induced into the heap. This suggests that it would be useful to investigate more about the strength of these objects in a memory heap as managed by Java in a web-based service architecture.

Similarly, the study's approach to the problem resembles the one used by Raqeeb, et.al (2017) with some adjustments. Most importantly, this study was a prolonged study and the examination was based on the application running on a web-based architecture. Unlike the above-mentioned study, the manipulation of objects in the heap is simplified through the use of a user interface. Instead of using Linux commands to input or manipulate values, the user just have to input random values on the interface and observe changes in the memory heap using the Yourkit Java profiler program attached to the web application.

To complete the study, the Java Servlets framework had been chosen since it has a history of supporting the implementation of dynamic web pages. Specifically, the study used Apache Tomcat web server (a Java Servlet) so that the project would produce quality and desirable results.

Chapter Summary

This chapter has introduced the subject of garbage collection in distributed systems. Specifically, the chapter has pointed out that sometimes objects can be prematurely removed from the memory heap due to loss of network connectivity. Technically such objects might still be strong enough to be referenced by applications but loss of a network connection causes them to lose their pointers. This idea supports the hypotheses explained in the previous section that some objects might be removed by garbage collectors while they are still strong. The chapter has also covered a similar study previously conducted by a research team. The objective of the study was to determine how secured a memory heap is in order to survive a denial of service attack. However, in the course of performing the study, it was observed that some objects became unreachable in the heap as time elapsed. The unreachability of objects was attributed to lack of

application refreshing over a longer period of time. The following chapter provides the detailed methodology to be used in order to complete the study.

Chapter III: Methodology

Every study is unique and requires a different approach. Likewise, this study followed the descriptive survey approach with slight differences in terms of data gathering and analysis. In this chapter, the detailed design, tools, and techniques required for this study are presented.

Design of the Study

The study used a descriptive survey method to assess the strength of objects in a memory heap as managed by Java-based services executing in a web-based architecture. A descriptive survey falls in the category of quantitative research and it is crucial to this study because it provides an essential framework for data gathering, analyzing, classifying, and tabulation about trends or processes. Thus, the use of descriptive survey method clarified cause-effect relationships so that adequate and accurate interpretation of such data could be made and accurate conclusions be drawn. As the Java-coded web application was constantly running, the data was collected from the attached "YourKit Java Profiler" program over a specified lengthy period of time. The data collected includes, but is not limited to, memory heap allocation, CPU utilization, number of objects dead and collected, and object allocation rate. This data was analyzed by the researcher to determine if the claimed hypotheses hold true or if a further study on the subject is warranted with respect to the web application performance.

Data Collection

Data analyzed in this study was entirely collected from YourKit-Java Profiler toolkit and garbage collection logs. The profiler's agent was attached to the test web application used in this study. When the test web application was executing, the profiler consistently collected various information about the test web application. The process of starting and attaching the profiler to the web application is described in implementation section.

Tools and Techniques

The main tool that was used to collect data was the "YourKit Java Profiler" toolkit that was attached to the web application. As the web application was running, the profiler was able to collect data about the allocated objects as well as those objects that had been removed from the memory heap. However, it has to be noted that undertaking this type of study requires a moderate background in Java programming including experience with garbage collection. Besides, good analytical skills are also important. For analysis purposes, graphs and tables have been provided. At one point, an online analysis tool called "GCeasy" was used to analyze the 10-day period garbage collection logs.

Hardware and Software Environment

The project was implemented by applying a client-server model involving a web user interface for inducing objects into the memory heap. Mainly the project used Apache Tomcat, a free and open-source software, for easy access and processing of web pages. Apache Tomcat works on various operating systems and is an implementation of Java Servlet, Java Server Pages, Java Expression Language and Java Web Socket technologies. Thus, it provides an "absolute Java" HTTP web server environment in which Java code can be executed. Its configuration and management tools allow direct configuration by editing XML configuration files.

The web page(s) may be accessed using any browser, e.g. Internet Explorer, Firefox, or Google Chrome. However, the program that was used for experimental purposes was coded in Java and the program was deployed as a web-based service. Specifically, the program was implemented in Java programming language because Java is a powerful language that manages its own memory allocation including garbage collection. Once the program was deployed and running, a software program called YourKit Java Profiler was attached to the program and

relevant data was collected for analysis. Any input of data on the user interface triggered the creation of more objects in the memory heap and that was reflected in the YourKit Java Profiler. The user-interface below is the one that was used to input data.

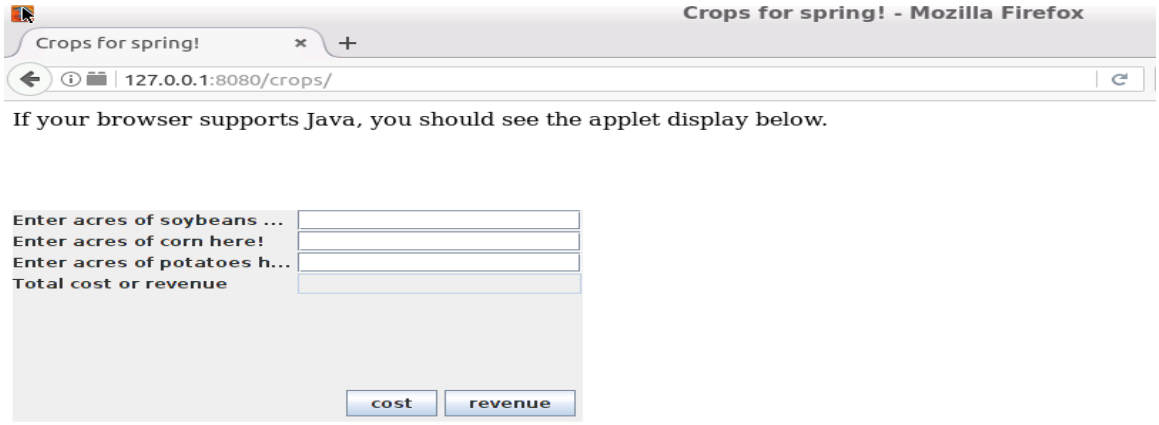


Figure 6: Web Application Interface

For the above-mentioned software to operate properly, the machine must be running on Ubuntu 16.04.2 LTS. For better performance, the machine should have a processor of 2x AMD Opteron(tm) 6174 2200.00 MHz and a 4GB memory.

Constraints

Initially the author developed and implemented a simple web application. But it was noticed that the web application was too small that it was not allocating many objects to the memory heap. That posed a threat of not being able to collect enough sample data for analysis. Lack of enough sample data would have led to biased or incorrect results of the study. As a remedy, another test web application capable of allocating several objects onto the memory heap was developed and it was the one used throughout this study.

Chapter Summary

In this chapter, the approach to the study has been presented. In particular, the study follows a descriptive survey whereby processes or trends are analyzed. The study used a data gathering tool called “YourKit Java Profiler” to collect necessary information and also used GCeasy tool to analyze the collected logs.

Furthermore, the study used a Linux machine with 4GB memory space. To accomplish the study, the author required good background in Java programming language as well as good analytical skills besides the automated analysis tools. The web application deployed on this Linux machine was accessed by users through the web browser (client). During the study, it was a challenge for the author to develop a fairly large web application capable of allocating moderate number of objects into the heap memory. The next chapter covers the processes followed to test the hypotheses outlined above, and ultimately accomplish the study.

Chapter IV: Implementation and Results

The implementation section uses a Java Program that calculates cost/revenue based on the number of acres for each type of crop input in the data entry boxes on the user interface. This section presents the system requirements, installation and setup procedures, and the outcomes.

Systems and Requirements

Server machine

Running on Ubuntu 16.04.2 LTS

Processor: 2x AMD Opteron(tm) processor 6174 2200.00 MHz

Memory: 4046MB

Client machine

Any browser such as Google Chrome, Firefox, Internet Explorer

The Crops Program

In an attempt to determine the strength of objects in a heap as managed by java in a web based service, the author deployed a simple java program on a Linux server. The client machine accesses the program through the web browser as dictated by client-server architecture. Basically, the program calculates the cost and revenue of crops based on the input values supplied in the data entry boxes. The figure below shows the interface of the web based java program hosted on a Linux server and accessed by the client through the web browser.
The source code of the program can be found in the appendix section.

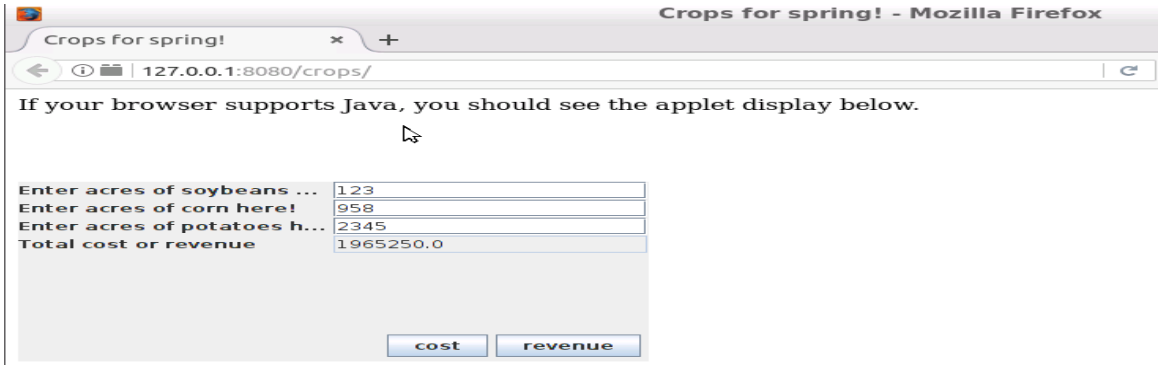


Figure 7: "Crops" Web Application Interface

Installation and Setup

Installing Apache Tomcat 8 on Ubuntu Server. First, install java8 and then create tomcat users. Thereafter, use the command below to install tomcat8 apache on the Ubuntu server:

sudo apt-get install tomcat8

For step by step details on how to install apache tomcat8 on Ubuntu server, see https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-ubuntu-16-04 or see the appendix.

Deploying the Crops Java Program. Web applications are deployed in apache tomcat 8 server in various ways. Some of the well-known deployment methods are: copying web application archive file (.war), copying unpacked web application directory, and using tomcat's manager application. In this study, the last method was used, the tomcat manager. But before that, the web archive (war) of the web application files to be used was created as shown below.

```
patrick@patrick:/var/lib/tomcat8/webapps/crops$ sudo jar -cvf crops.war *
[sudo] password for patrick:
added manifest
adding: Crop.class(in = 2779) (out= 1478)(deflated 46%)
```

adding: crop.html(in = 248) (out= 164)(deflated 33%)

adding: index.html(in = 248) (out= 164)(deflated 33%)

adding: WEB-INF/(in = 0) (out= 0)(stored 0%)

adding: WEB-INF/lib/(in = 0) (out= 0)(stored 0%)

adding: WEB-INF/web.xml(in = 429) (out= 235)(deflated 45%)

adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)

adding: WEB-INF/classes/Crop.java(in = 2140) (out= 693)(deflated 67%)

adding: WEB-INF/classes/Crop.class(in = 2779) (out= 1478)(deflated 46%)

patrick@patrick:/var/lib/tomcat8/webapps/crops$ ls

Crop.class crop.html crops.war index.html WEB-INF

patrick@patrick:/var/lib/tomcat8/webapps/crops$ cp crops.war ~

patrick@patrick:/var/lib/tomcat8/webapps/crops$

Deploying the program using Tomcat manager. Invoke the web browser, type localhost:8080/manager/html and press enter. Under tomcat8-admin, click on “webapp”. Then click on “Browse” and navigate to where the war file is located and select it; then click on “Deploy”.

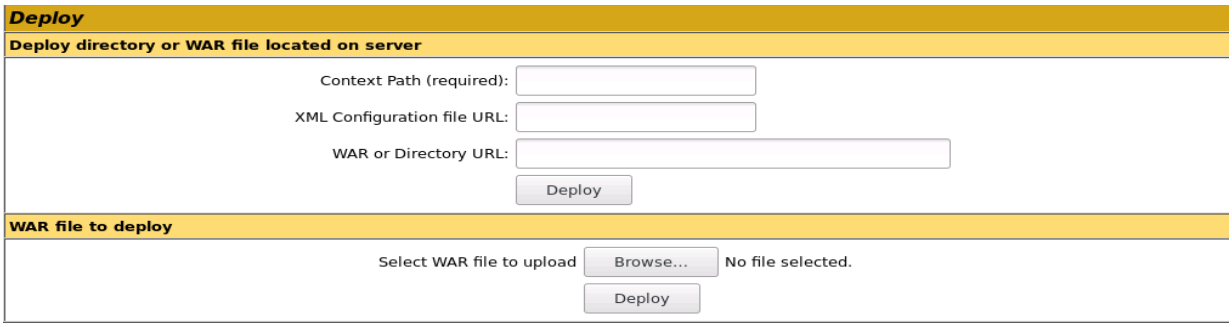
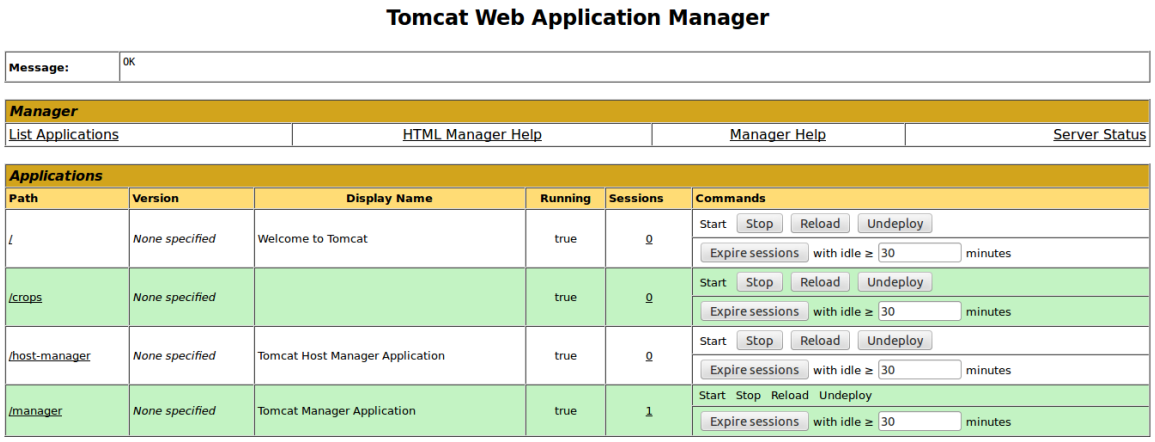


Figure 8: Tomcat Deployment Interface

After successful deployment, the program should be listed in the apache tomcat8 management

console as shown in the figure below. Just refresh the web page and the program will be listed as shown below.



Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/crops	*None specified*		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 9: Listing of Deployed Web Applications

After the program has been successfully deployed, it is possible to execute it and collect necessary information required for this study. Apache Tomcat 8 comes with examples of web applications. Notice that in the figure above those sample web applications were removed except manager applications to let only the web application of interest, in this case crops application to run in the tomcat container. The diagrammatic presentation of how the program was accessed is demonstrated in Figure 10 below.

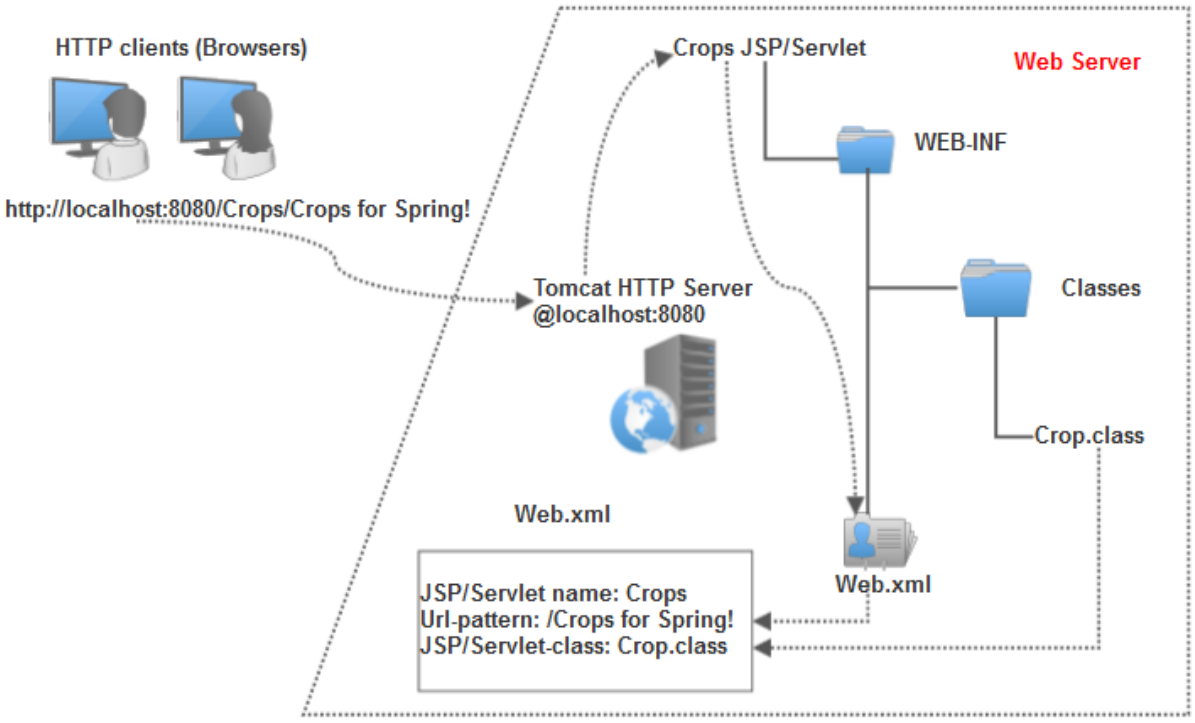


Figure 10: Architecture to Access Crops Program

Starting YourKit Java Profiler

Below are the steps that were followed in order to complete the data collection process. On the web server, execute the following steps:

1. Download and extract YourKit Java profiler for Ubuntu
2. Navigate to the desktop and open LXTerminal
3. To start the profiler, change from your home directory to the directory where the downloaded YourKit Java profiler program folder is stored. In this case, the profiler program was stored in the home directory, “Downloads” folder. Enter the command:
    a) cd /Downloads/yourkit/yjp-2017.02/bin and press enter.

b) Once you are in the profiler's directory, execute the YourKit Java Profiler program by typing: ./yjp.sh as shown below:

```
patrick@patrick:~$ cd Downloads/YourKit-JavaProfiler-2017.02/bin
patrick@patrick:~/Downloads/YourKit-JavaProfiler-2017.02/bin$ ./yjp.sh
[YourKit Java Profiler 2017.02-b71] Log file: /home/patrick/.yjp/log/yjp-3992.log
```

4. Once the command is executed, Yourkit Java Profiler screen will be displayed (Leave the window open.)
5. Open Firefox ESR on the desktop. Type in the name of the web application in the URL box. For this study, navigate in the browser to localhost:8080/Crops or go to the bookmarks and select Crops for Spring!.
6. Accept/allow all the notifications and eventually the web application should be displayed and usable. Once the web application is up and running you can navigate back to the profiler.
7. Within the profiler double-click on Tomcat and it should be able to attach to the profiler agent. Note that the PluginMain does not belong to the web application, it is a plugin that belongs to the web browser so do not attach it. Once Tomcat is successfully attached to the profiler, the below interface will be displayed.

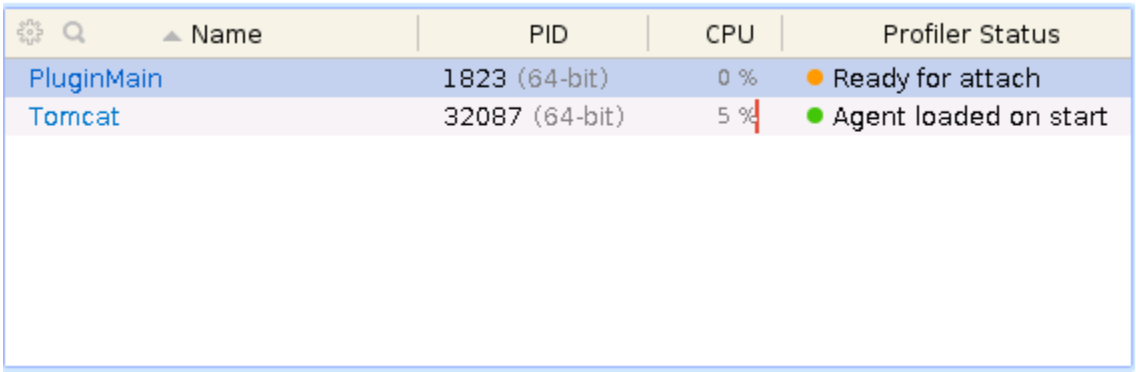


Figure 11: YourKit Java Profiler Attach Interface on Start with Tomcat

Note: The web application and java profiler should be run using the same user (who has the ownership on the two programs), otherwise, the application will not show in the profiler and you will not be able to attach the web application for profiling. Also to avoid time delays and be able to have full profiling capabilities, the profiler agent was made to execute upon startup of Tomcat. In this case, the profiler will show that it was started when Tomcat started as shown in Figure 11 above.

8. From this location navigate through the tabs and observe as well as collect the important data.

Results and Analysis: Input, Output, and Snapshots

The web application and YourKit Java Profiler were left running for 10 days. Every day, the necessary data was collected from the profiler and the following section presents the analysis of this 10-days data.

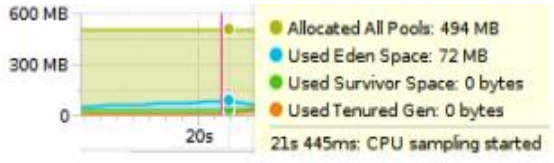
Locating memory heap for Tomcat. In an attempt to verify the amount of memory space Tomcat was occupying immediately after application execution, the below command was issued on the server on which the web application was deployed and the following results were obtained:

```
andrew@ubuntu2:~$ ps -aux | grep tomcat
andrew     447  0.0  0.0  14224   964 pts/1    S+   17:49   0:00 grep --color=auto tomcat
tomcat8  32087  7.7 10.5 4610120 426916 ?      Sl   15:33  10:34 /usr/lib/jvm/default-java/bin/java -Djava.util.logging.config.file=/var/lib/tomcat8/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.security.egd=file:/dev/./urandom -Djava.awt.headless=true -Xmx512m -XX:MaxPermSize=256m -XX:+UseConcMarkSweepGC -Xms512m -Xmx2048m -XX:MaxPermSize=256m -agentpath:/home/andrew/Downloads/YourKit-JavaProfiler-2017.02/bin/linux-x86-64/libyjpagent.so=disablestacktelemetry,exceptions=disable,delay=10000 -Djava.endorsed.dirs=/usr/share/tomcat8/endorsed -classpath /usr/share/tomcat8/bin/bootstrap.jar:/usr/share/tomcat8/bin/tomcat-juli.jar -Dcatalina.base=/var/lib/tomcat8 -Dcatalina.home=/usr/share/tomcat8 -Djava.io.tmpdir=/tmp/tomcat8-tomcat8-tmp org.apache.catalina.startup.Bootstrap start
andrew@ubuntu2:~$ cd /proc/32087
andrew@ubuntu2:/proc/32087$ sudo cat maps | grep heap
00c51000-00d93000 rw-p 00000000 00:00 0                                  [heap]
andrew@ubuntu2:/proc/32087$
```

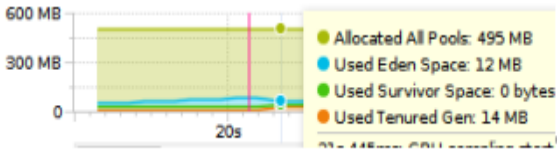
The results above show that the heap memory of Tomcat, which the profiler is inspecting and analyzing, resides on the memory range of ***00c51000-00d9300***. The hex value of this memory range is ***19E4000*** which is equivalent to ***25MB*** of memory used by the Tomcat and also the web application program is running within Tomcat in this memory range. But remember that the heap

memory size can increase or decrease when the application is executing up to the allocated limit of ***2GB***.

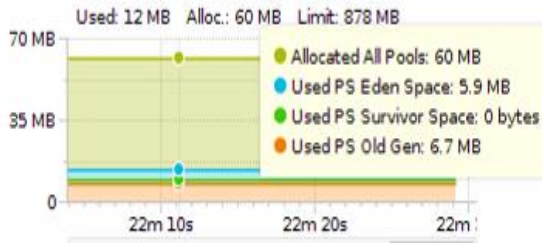
1st day of web application execution. When the application started, JVM created a heap in the memory to allocate the objects. It is worth noting that the heap may increase or decrease in size while the application executes but it has a virtual reservation limit of 2GB. In this study, upon execution of the web application, a 495MB memory heap was committed and segmented into Eden space, survivor space, and Old generation space as shown in Figure 12 below. As it can be seen from Figure 12a, the size of Eden space is greater than other spaces because that is the area where objects are allocated first before they are moved to the next spaces. After the garbage collection, the Eden size decreased and the old generation size increased because it received some objects promoted from Eden space as shown in Figure 12b.



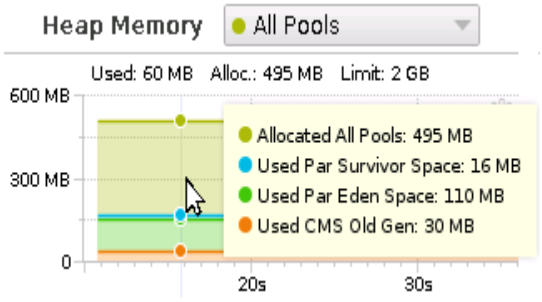
(a) Heap before GC



(b) Heap after GC



(c) Used Parallel Garbage Collector Only

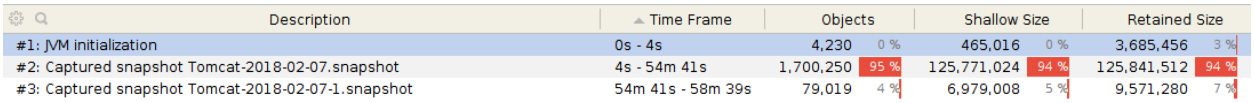


(d) Used Both Parallel and CMS Collectors

Figure 12: Heap Memory Segments

Once the JVM finished creating the memory heap, the web application started allocating its objects to the Eden space. The creation of memory heap took a few seconds as shown in

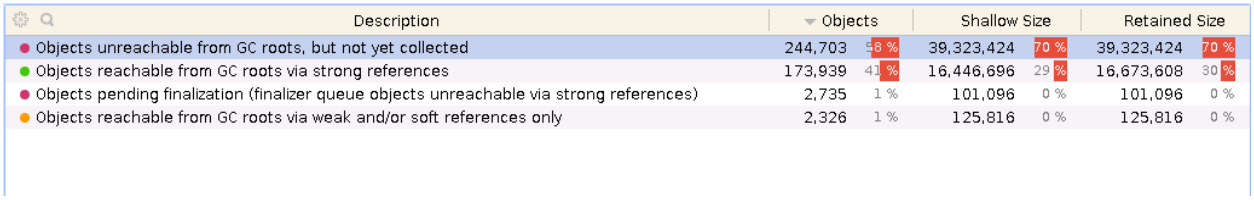
Figure 13 below. Out of 495MB allocated to heap memory, only 60MB was used by the application's objects at the start-up. Most often the process of creating memory heap is called the JVM initialization process. As shown in Figure 13 below, the initialization process took less than 4 seconds and thereafter the application started allocating the objects.



Description	Time Frame	Objects	Shallow Size	Retained Size
#1: JVM initialization	0s - 4s	4,230 0 %	465,016 0 %	3,685,456 3 %
#2: Captured snapshot Tomcat-2018-02-07.snapshot	4s - 54m 41s	1,700,250 95 %	125,771,024 94 %	125,841,512 94 %
#3: Captured snapshot Tomcat-2018-02-07-1.snapshot	54m 41s - 58m 39s	79,019 4 %	6,979,008 5 %	9,571,280 7 %

Figure 13: JVM Initialization Period

Immediately after the execution of the web application, 41% of its objects were reachable via strong references, and 1% of objects were referenced via weak references with 58% of unreachable objects as illustrated in Figure 14 below.



Description	Objects	Shallow Size	Retained Size
Objects unreachable from GC roots, but not yet collected	244,703 58 %	39,323,424 70 %	39,323,424 70 %
Objects reachable from GC roots via strong references	173,939 41 %	16,446,696 29 %	16,673,608 30 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	2,735 1 %	101,096 0 %	101,096 0 %
Objects reachable from GC roots via weak and/or soft references only	2,326 1 %	125,816 0 %	125,816 0 %

Figure 14: Objects Strength Immediately after Execution

In the first 40 seconds of execution, the application generated almost 7500 objects and within the time-frame of 9 minutes a total of 7 garbage collection pauses were observed. 4 out of the 7 garbage collection pauses were observed immediately after the web application executed as illustrated in Figure 16 below. It is possible for an application to have short-lived (temporary) objects that die immediately as they are being allocated to the heap and that would have caused the triggering of garbage collection 4 times immediately after the program executed. Figures 15 and 16 below show the recorded objects as well as the number of garbage collections.

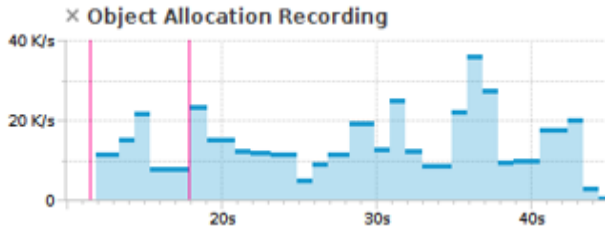


Figure 15: Object Allocation in First 40 Seconds

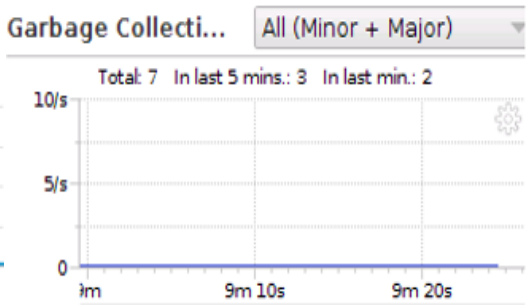


Figure 16: Garbage Collection Pauses

During the same period of start-up, the observed number of classes loaded into the memory also increased from approximately 2500 to 3355 before the number became constant. The figure below shows the number of loaded classes at the start-up of program execution.

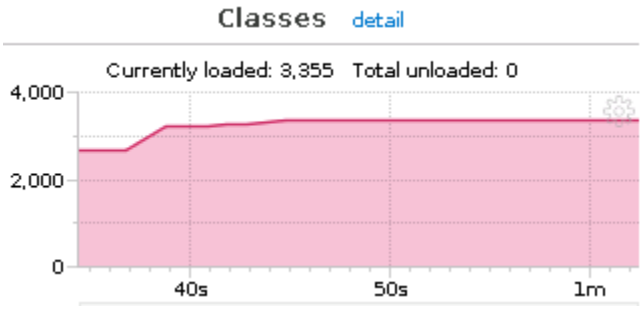


Figure 17: Loaded Classes

Figure 18 below shows the summary of allocated heap size, loaded classes, number of threads, and number of garbage collections immediately after the application was executed.

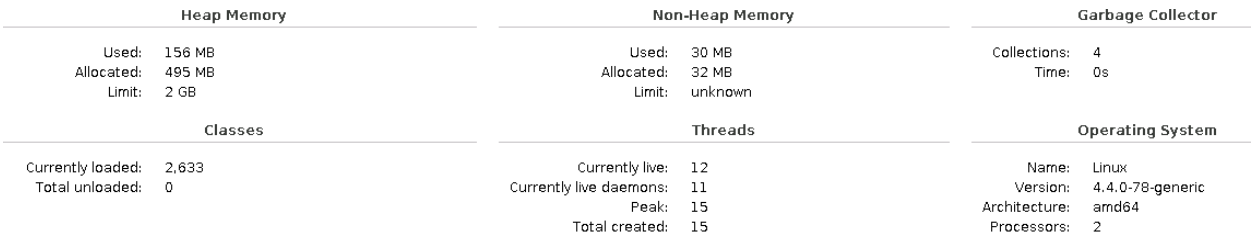


Figure 18: System State Immediately after Execution

Based on the following logs, let us analyze the first minor garbage collection that occurred immediately after program execution. Before garbage collection was triggered, out of 76672K (74MB) of Young generation, only 68160K (66MB) of Eden space was used and the

rest of the spaces (survivor spaces) were empty. Similarly, 439104K (428MB) of Old Generation space was never used immediately after execution. After 30 seconds, minor garbage collection was triggered which resulted in live objects being pushed to the next survivor space. Since the survivor space could only accommodate 8512K (8MB) of live objects, extra live objects were promoted to the old generation based on their ages. After the first minor garbage collection, out of 76672K (74MB) of Young generation only 8510K (8MB) of survivor space was used and the Eden space was empty. A large number of dead/unreferenced objects were collected. Again, 8949K (8MB) out of 439104K (428MB) of Old generation was used. Consequently, 50701K (50MB) of heap memory was reclaimed upon completion of the first minor garbage collection.

```
{Heap before GC invocations=0 (full 1):
 par new generation   total 76672K, used 68160K [0x00000000c0000000, 0x00000000c5330000,
0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 0K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 27.976: [GC (Allocation Failure) 27.979: [ParNew28.076: [CMS-concurrent-abortable-preclean:
 0.144/2.360 secs] [Times: user=1.67 sys=0.04, real=2.36 secs]
 : 68160K->8510K(76672K), 0.4608943 secs] 68160K->17460K(515776K), 0.4639007 secs] [Times:
 user=0.21 sys=0.01, real=0.46 secs]
Heap after GC invocations=1 (full 1):
 par new generation   total 76672K, used 8510K [0x00000000c0000000, 0x00000000c5330000,
0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,  99% used [0x00000000c4ae0000, 0x00000000c532fa50, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 8949K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
28.485: [GC (CMS Final Remark) [YG occupancy: 10826 K (76672 K)]28.485: [Rescan (parallel) ,
0.0632955 secs]28.548: [weak refs processing, 0.0012524 secs]28.549: [class unloading, 0.0356359
secs]28.585: [scrub symbol table, 0.0222441 secs]28.619: [scrub string table, 0.0014651 secs][1 CMS-
remark: 8949K(439104K)] 19776K(515776K), 0.1363331 secs] [Times: user=0.07 sys=0.00, real=0.13
secs]
28.645: [CMS-concurrent-sweep-start]
28.648: [CMS-concurrent-sweep: 0.003/0.003 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
28.660: [CMS-concurrent-reset-start]
29.015: [CMS-concurrent-reset: 0.346/0.355 secs] [Times: user=0.16 sys=0.02, real=0.36 secs]
```

The logs depicted above also explain one important aspect of why the garbage collector was triggered. Not only is the garbage collector triggered when the Eden space is 100% full, but

also when the data structure fails to fit in any region in Young generation. Thus, in the above scenario, the garbage collector was triggered when there was an allocation failure. Notice that the live objects were pushed to old generation right after the first minor garbage collection. Possibly the web application created a large number of objects which consumed Eden space. Most of these objects were still being referenced by live threads during the first minor garbage collection. The minor garbage collection was forced to push these objects to the next survivor space and eventually to old generation space in order to accommodate newly-generated objects. This is called premature object aging because the objects are being pushed to the old generation space not because they have reached their age limit but because the Eden space is filled up. Eventually, these prematurely aged objects will be collected by a major garbage collection in the old generation space. Even though some important objects will remain reachable in the old generation space until the program exits, some will still be prematurely collected during the major garbage collection. Therefore, it concurs with one of the hypotheses stated above that some objects are destroyed while in their active state in such a way that they can still be referenced by live threads. Since premature collection occurred due to the application generating more objects at startup, one could consider to reasonably increase the size of Eden and survivor spaces to maximize the full potential of the objects. For more detailed garbage collection activities happening in the JVM, see the 10-days logs included in Appendix C. The below graphs partially summarize the above-explained scenario.

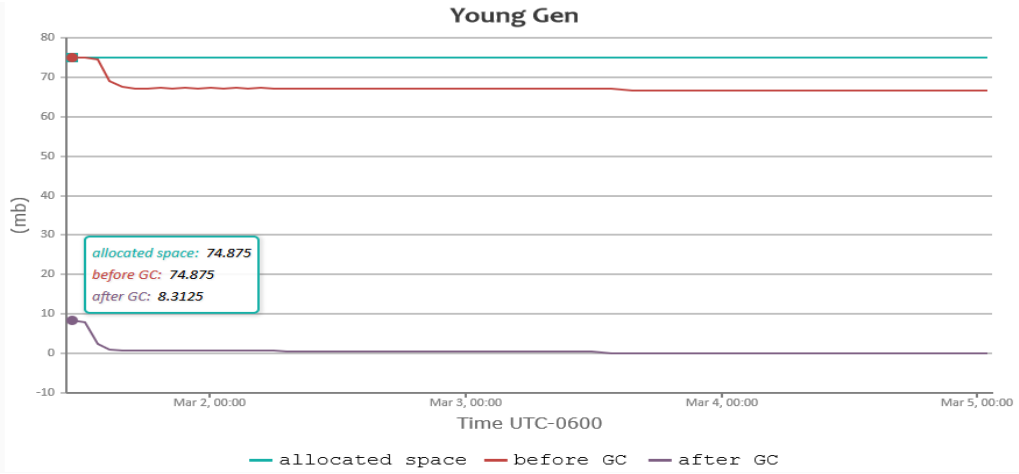


Figure 19: GC in Young Generation

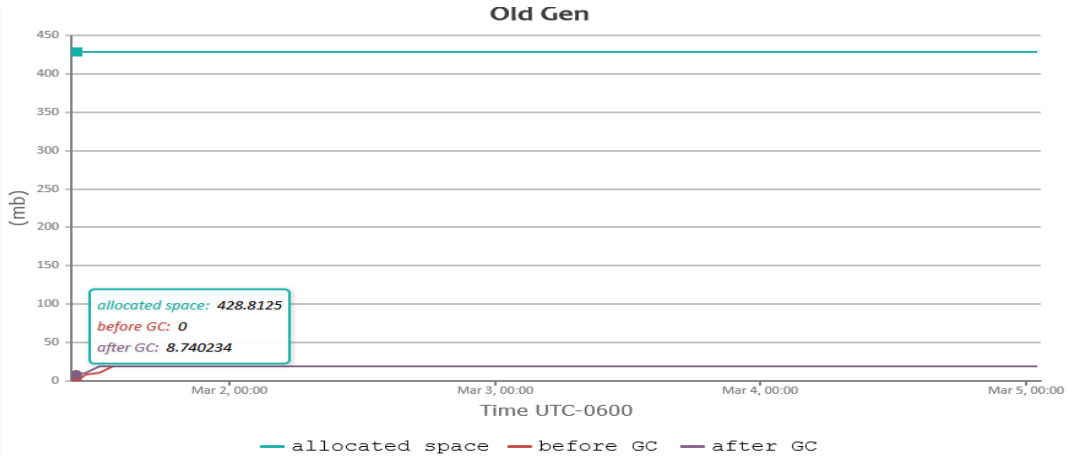


Figure 20: GC in Old Generation

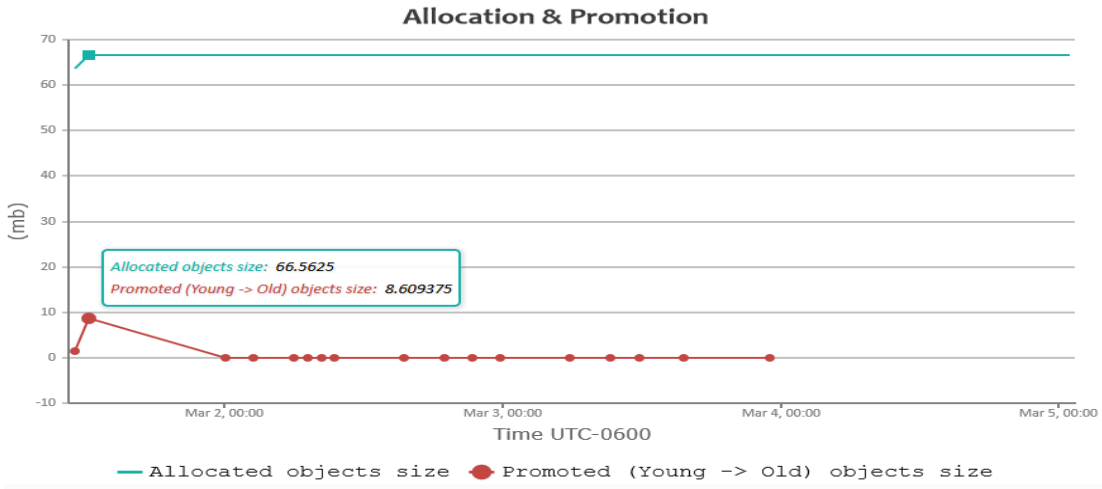


Figure 21: Object Allocation and Promotion

Now let us look at the choice of garbage collector used in this study. As already discussed in the garbage collection section above, Java language uses 5 types of garbage collectors namely: serial, parallel, parallel compacting, concurrent mark and sweep, and garbage first collectors. JVM chooses the garbage collector depending on the design of the garbage collector to satisfy the demands of both small and large (those with many threads and high transaction rates) applications. In this study, the garbage collector to be used was not specified in the configuration file to allow the JVM chose on its own. In J2SE 5.0 the choice of garbage collector is based on the type of the machine on which the application is run. When applications run on machines with two or more processors and large amount of memory, the parallel collector is selected by default. In that regard, in this study, JVM chose the parallel garbage collector for Eden space and Survivor space; compact mark and sweep garbage collector was chosen for the old generation space as depicted in Figure 12d above. This proves that the web application used for the study is large enough that serial garbage collector could not be selected by JVM. Most often serial garbage collectors are efficient for small applications requiring approximately 100 MB of heap on modern processors. Figure 12d shows that the JVM allocated 495 MB of heap which is greater than 100 MB hence the JVM could not select the serial garbage collection algorithm. It is important to note that the JVM selects heap size, garbage collector, and runtime compiler at startup within the initialization time-period.

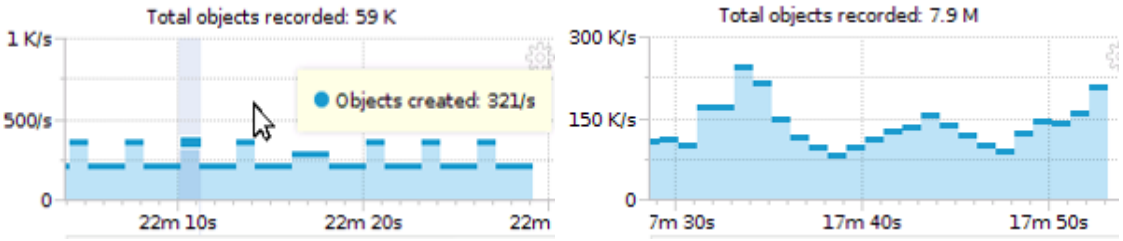
When the web application was being developed, it was tested on several stages of development. The first stage was when the application was small, and it was allocating fewer objects into the memory heap. Figure 12c shows that the JVM chose Parallel Scavenge collector for all the heap segments instead of selecting CMS for old generation as was the case in Figure 12d. As already explained above, this is because the JVM allocated 60 MB of memory heap

which is less than 100 MB, hence CMS could not be selected in that scenario. Parallel scavenge is also a “stop-the-world” copying collector that uses multiple GC threads but cannot be used with CMS. On the other hand, even though Garbage first (G1) collector works perfectly on both young and old generations, it could not be used in either of the above cases because it was designed for larger heap sizes of at least 10 GB.

Comparing the Figures 12c and 12d, it can be deduced that the web application in Figure 12c is allocating few objects as far as the heap size is concerned, unlike the web application in Figure 12d. It can also be seen that in Figure 12c, the survivor spaces could sometimes be empty (0bytes) meaning that the objects in Eden space are not yet aged to be pushed to the next (survivor) space. It is possible that the application is only generating short-lived objects that are being collected by minor garbage collection in such a way that they are not aging enough to be pushed to the survivor space. The other possibility is that the application is generating objects at a very slow rate so that it takes time for Eden space to be 100% full and calling for minor garbage collection, thus making the survivor space sometimes empty. Nevertheless, this is good because in ideal situations, one of the survivor spaces should be empty at any given time to serve as the destination of any live objects from Eden and/or the other survivor space during the next minor garbage collection.

In contrast, throughout the experiment it was observed that using the crops application, neither the survivor space 0 nor the old generation space was empty at any point in time after first minor garbage collection. This may imply that the application was allocating objects at a fast rate in such a way that the objects were aging rapidly to accommodate new objects. The figures below show the rate at which the web application is allocating objects. Figure 22a

represents the small application and Figure 22b is for the large application that has been used in the rest of the study.



(a) Small Application (b) Large Application

Figure 22: Object Allocation Variations

As the rate of object allocation suggests, the large web application is generating many objects per second causing the Eden space to fill up quickly and call for minor garbage collection; eventually major garbage collection would follow. That might be the reason why the survivor space 0 and old generation space have not been empty after the first minor garbage collection. Figure 23 below illustrates the rate of garbage collection being called by the JVM to clean up the dead objects in the first 22 minutes of program execution. Notice that the spikes are sparsely distributed and each spike takes a considerably small amount of time. This tells us that at this time most of the objects were reachable via strong references, thus, the garbage collector was not triggered so frequently. Even though the garbage collector was called from time to time, it would not take longer times because there were few unreferenced objects to clean up from the heap.

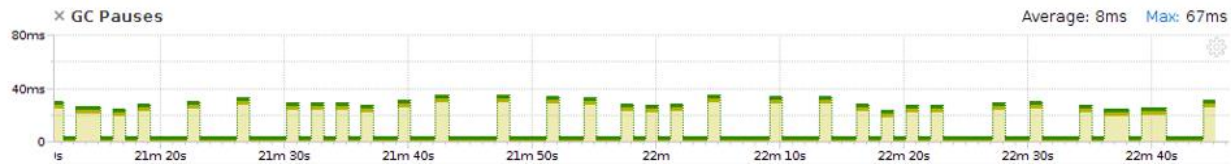


Figure 23. Minor+Major Garbage Collections on First Day of Execution

However, there is a danger in the sense that many of the objects might be prematurely collected in order to accommodate newly-generated objects. If the application is generating more objects than the Eden space can handle, the result is that minor garbage collection is triggered frequently. Most of the objects that are still active (being referenced from a live thread) will be pushed to the next survivor space and eventually to the old generation space where they will be cleaned up by major garbage collection. In this scenario it can be observed that the objects are being prematurely collected for the sake of accommodating newly-generated objects. Premature collection might compromise the study because it might be concluded that the objects generated by the web application are not strong enough to live longer, yet the JVM did not predict the object generation rate properly. Premature object collection mostly happens when the JVM did not appropriately estimate the number of active short-lived objects to be generated by the application in order to correctly size the Eden space during the start-up. In ideal situations, the old generation space is expected to be empty for the first 3-8 minor garbage collections depending on the size of the application. However, the generation rate does not matter that much for this study because with passage of time the number of objects retained within the 10-day period will matter.

On the first day of the experiment, the author was also interested in learning how the web application was handling request calls to the web server. It was observed that in the first 15 minutes the client made many requests to the server. But with time, the number of requests dropped and became constant as shown in the figure below.

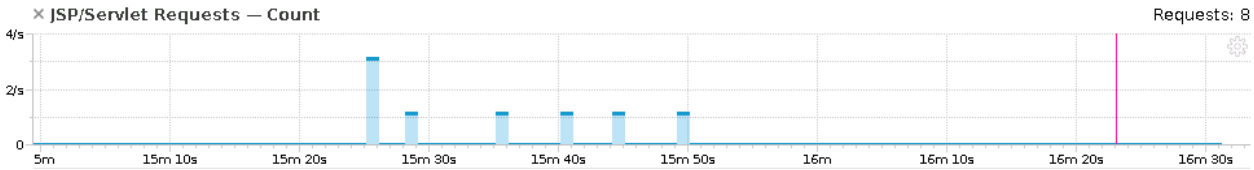


Figure 24: Client-Server Requests

These servlet requests are invoked in response to a URL request issued by the client (browser) for processing. When the values are inserted in the text box of the web application interface and click cost/revenue, the browser issues an HTTP request, and the server sends the request message to the "Crops" java program for processing. Among other objects, the HTTP request may contain two types of objects namely: HTTPRequest object and HTTPResponse object. The figure below shows the interaction between the client and the server. The application was fed with data and calculated cost/revenue three times, and those three times are reflected in the figure below when the client issued requests and got responses. The transmitted bytes are also captured in the same figure.

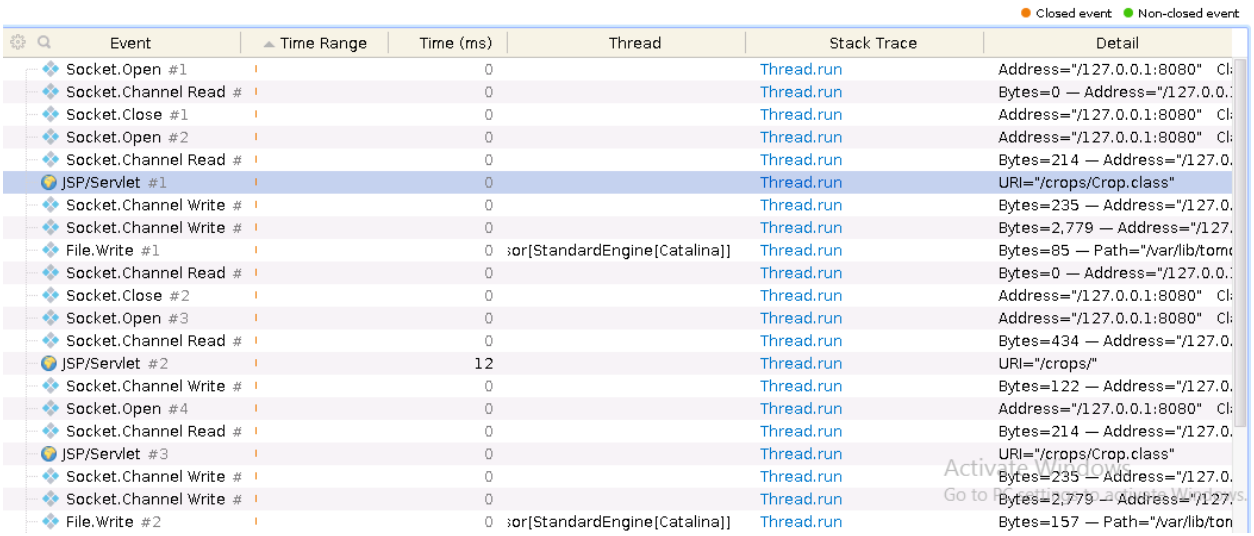
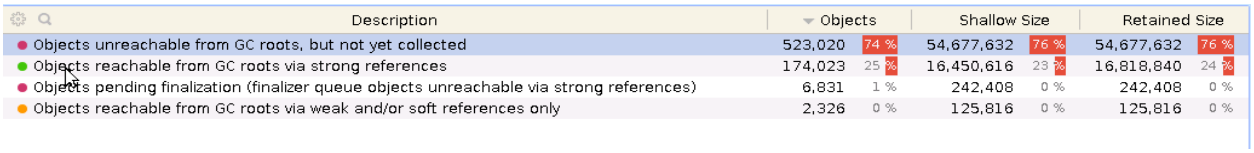


Figure 25: Calls to the Server

2nd day after web application execution. On second day of the trial, it was observed that the number of reachable objects via strong references declined by 18% as compared to the data obtained on the first day of execution. The figure below captures this change of object reachability.



Description	Objects		Shallow Size		Retained Size	
Objects unreachable from GC roots, but not yet collected	523,020	74 %	54,677,632	76 %	54,677,632	76 %
Objects reachable from GC roots via strong references	174,023	25 %	16,450,616	23 %	16,818,840	24 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	6,831	1 %	242,408	0 %	242,408	0 %
Objects reachable from GC roots via weak and/or soft references only	2,326	0 %	125,816	0 %	125,816	0 %

Figure 26: Object Strength after a Day of Execution

As shown in the Figure 26 above, 74% of floating garbage (objects unreachable from GC roots, but not yet collected) was recorded surpassing the objects that can be referenced by live threads. Only 1% of objects were pending finalization to be marked as unreachable. Usually the next concurrent collection cycle sweeps the floating garbage. Interestingly, even though Figure 26 shows that only 25% of objects were reachable via strong references, the number of objects reachable is almost the same as the number of objects that were 41% reachable via strong references immediately after execution as depicted in Figure 14. The assumption is that most of these 160 thousand plus objects are the important objects required for the application to execute and most of the unreferenced objects belong to web service elements. This assumption was proved by looking specifically at the objects allocated by the crops web application, excluding Tomcat objects and the results are shown in the following section.

As opposed to the first day of program execution, in the second day it was observed that garbage collection was triggered frequently and most of the time it took longer pauses than the first day of program execution. This might suggest that on the second day, most of the allocated objects had aged hence the frequent calling of the garbage collector to clean up the unreferenced objects. Figures 27 and 28 below illustrate the behavior of garbage collection for both minor and major collections during the second day of program execution.

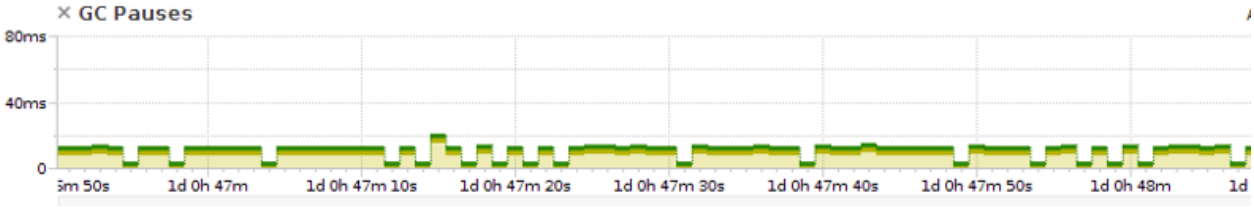


Figure 27: GC (Minor+Major) Pauses after a Day of Execution

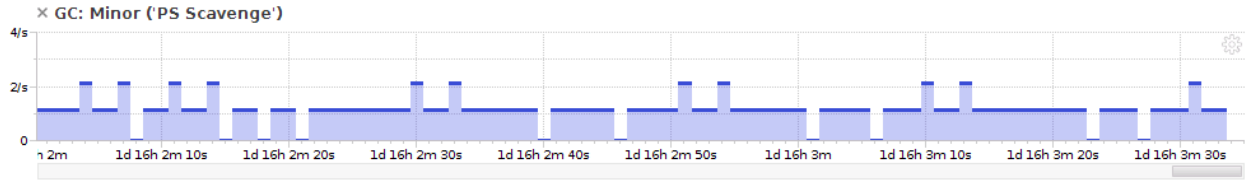


Figure 28: Minor Garbage Collection Pauses

The above two figures show that after a day of running the web application, there is an increase in number of GC pauses. Obviously garbage collection is called frequently because many objects could not be referenced at this time. As Figure 14 illustrates, 58% of the objects that were allocated immediately after application execution had decayed in such a way that they can no longer be referenced by the live threads. As time goes on, the application is becoming almost idle making some objects become unreachable by the live threads. Therefore, the calling of garbage collection now and then is to reclaim memory used by the dead (unreferenced) objects.

Heap manipulation. On the same second day of execution, the researcher also tried to induce extra objects into the heap by calculating cost and/or revenue several times and observing the results. It was discovered that CPU time, object allocation recording, and garbage collection increased drastically. The CPU usage increased from around 25% constant to about 80% as shown in Figure 29 below. At this time, the CPU was utilized to allocate the newly-induced objects into the heap and also trigger the garbage collector to sweep the aged objects for the purpose of creating memory space in the Eden space.

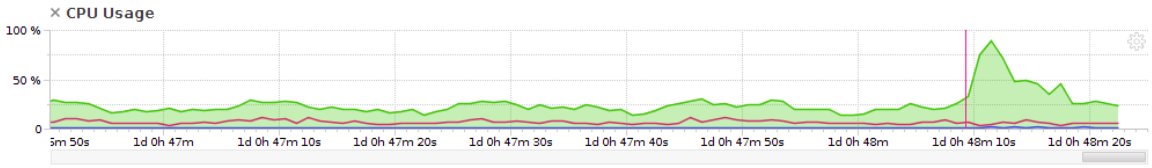


Figure 29: CPU Usage on Interaction with the Web Application

Also, the GC pauses increased dramatically after generating extra objects into the heap as can be seen in the Figure 30 below starting from the vertical red line and after sometime the spikes became sparsely distributed. This denotes that most of the objects are now reachable, thus, garbage collector can no longer be triggered frequently.

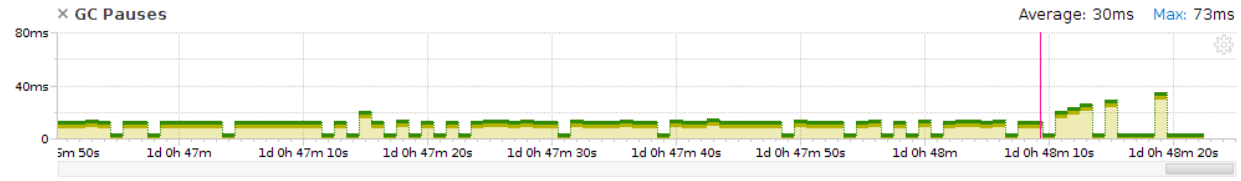


Figure 30: GC Pauses on Interaction with Web Application

Of course the newly-allocated objects were also depicted in the object allocation graph starting from the vertical red line in the Figure 31 below.

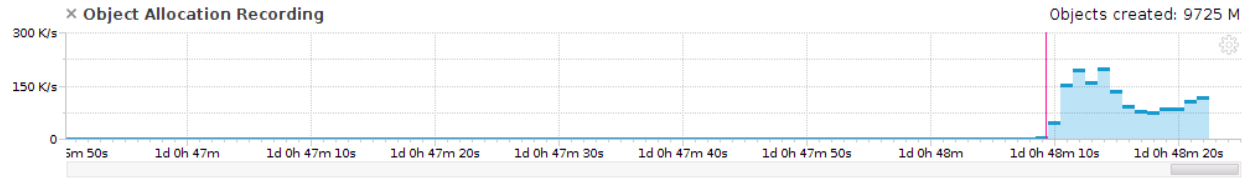
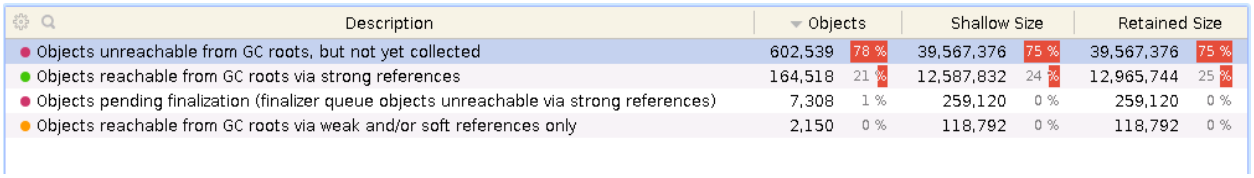


Figure 31: Object Allocation on Interaction with Web Application

3rd day after web application execution. On the third day, only 21% of the allocated objects were reachable via strong references and no objects were reachable via weak references. The decrease in number of reachable objects confirms the fact that some objects are not being referenced by the live threads due to inactivity of the web application, thus, the objects lose their pointers. These unreferenced objects eventually become garbage and are removed from the memory by the garbage collector. However, the strong reachability percentage decrease is slightly small (25%-21%) because some new objects were induced on the second day after web

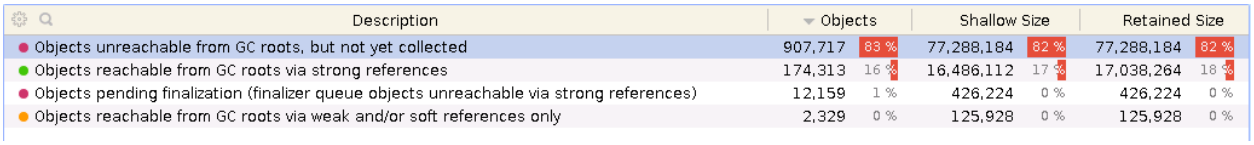
application execution. Again as mentioned above, the percentage of reachability declined but the number of those reachable objects remained almost constant.



Description	Objects		Shallow Size		Retained Size	
Objects unreachable from GC roots, but not yet collected	602,539	78 %	39,567,376	75 %	39,567,376	75 %
Objects reachable from GC roots via strong references	164,518	21 %	12,587,832	24 %	12,965,744	25 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	7,308	1 %	259,120	0 %	259,120	0 %
Objects reachable from GC roots via weak and/or soft references only	2,150	0 %	118,792	0 %	118,792	0 %

Figure 32: Object Strength 3 Days after Execution

4th day after web application execution. On the fourth day, the percentage of reachable objects via strong references declined further by 5% as illustrated in the figure below.



Description	Objects		Shallow Size		Retained Size	
Objects unreachable from GC roots, but not yet collected	907,717	83 %	77,288,184	82 %	77,288,184	82 %
Objects reachable from GC roots via strong references	174,313	16 %	16,486,112	17 %	17,038,264	18 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	12,159	1 %	426,224	0 %	426,224	0 %
Objects reachable from GC roots via weak and/or soft references only	2,329	0 %	125,928	0 %	125,928	0 %

Figure 33: Object Strength 4 Days after Execution

It was also observed that the rate of object allocation into the heap declined when the application stayed idle for four days. Remember, upon execution the application was allocating approximately 40K/s, but after four days the rate reduced to approximately 1.2K/s. The figure below shows that after four days the application allocated a total of 110 million objects although the allocation rate decreased when the application stayed idle for those four days.

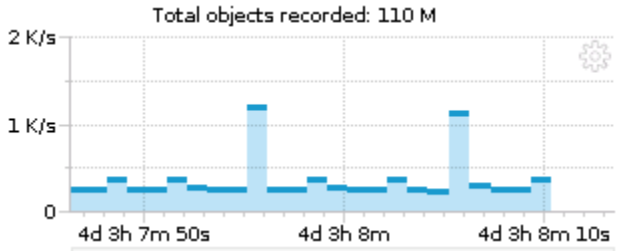


Figure 34: Object Allocation Rate on 4th Day

Furthermore, on this fourth day it was noticed that there was no much garbage collection activities taking place except one pause that was recorded as illustrated in the figure below. At this moment most of the objects that the application allocated on the first day of execution had

died (become unreachable) and swept by the garbage collector. Since there is no interaction with the web application, it is obvious that the application is no longer allocating new objects onto the heap let alone maintaining only those objects deemed to be important. This explains why the rate of object allocation and garbage collection decreased over the four-day period: the application was almost idle.

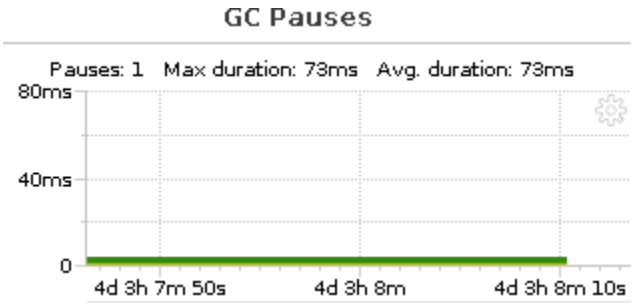


Figure 35: GC Pauses on 4th Day

5th-10th day after web application execution. Starting from day 5 up to day 10 the author observed a constant number of objects being referenced by live threads. Only 6% of the allocated objects was still accessible by the live threads. The figure below shows that about 92% of the allocated objects became unreachable from GC roots.

Description	Objects		Shallow Size		Retained Size	
Objects unreachable from GC roots, but not yet collected	2,497,055	92 %	148,938,768	89 %	148,938,768	89 %
Objects reachable from GC roots via strong references	175,294	6 %	16,572,240	10 %	17,792,896	11 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	31,739	1 %	1,101,856	1 %	1,101,856	1 %
Objects reachable from GC roots via weak and/or soft references only	2,153	0 %	118,800	0 %	118,800	0 %

Figure 36: Object Strength 5-10 Days after Execution

For five days the application maintained 6% of objects reachable from GC roots using strong references. The garbage collector could no longer sweep because most of the dead objects had been cleared. It is assumed that these maintained 6% of objects reachable via strong references are the most important and dominant GC roots objects and remains reachable until the program exits. These objects can still be referenced by the live threads in the old generation space throughout the program's execution cycle.

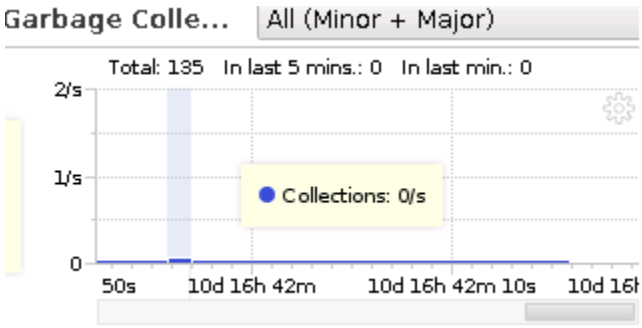


Figure 37: Garbage Collections for 10th Day

Figure 37 above shows that there was almost no garbage collection activity taking place on this day. As explained above, this might be because the application is no longer assigning new objects to the heap that would cause the garbage collector to be triggered.

Table 2: Summary of the Object Strength for 10 Days

Day	% of obj. reachable via strong references	% of obj. reachable via weak references	% of obj. unreachable by live threads
1	41	1	58
2	25	0	74
3	21	0	78
4	16	0	83
5	6	0	92
6	6	0	92
7	6	0	92
8	6	0	92
9	6	0	92
10	6	0	92

The results and analysis explained so far depict the decaying of objects as Tomcat is running. Since the crops java program is accessed through the web service, it was very important for this study to examine the strength of the objects of the web service as a whole, in this case the Apache Tomcat container. However, the profiler used provides an opportunity to isolate and examine the web application in particular to be able to determine the objects the application is

allocating and how many are retained by the end of the trial period. On the YourKit Java Profiler, the below processes were followed to find out the objects for specific web application:

1. Using the iconic menu, the researcher captured the snapshot and clicked on open
2. Navigated to web application→Right click on web application (crops)→Selected Objects→Reachability. The figure below depicts the process explained above.

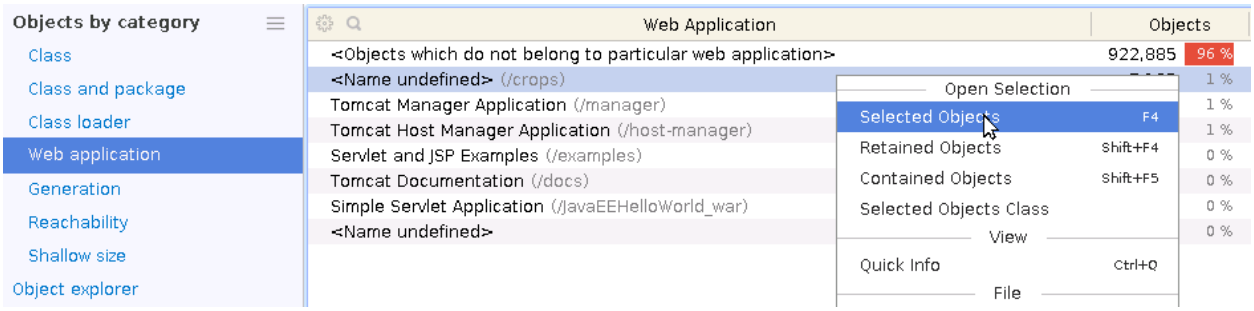


Figure 38: Process to See Web Application Objects

Ultimately you will land on a page that shows the web application's objects reachability and other information as shown in the figure below.

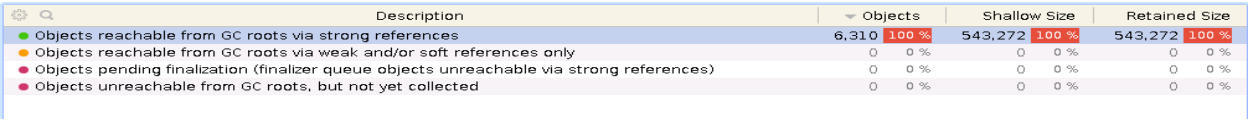
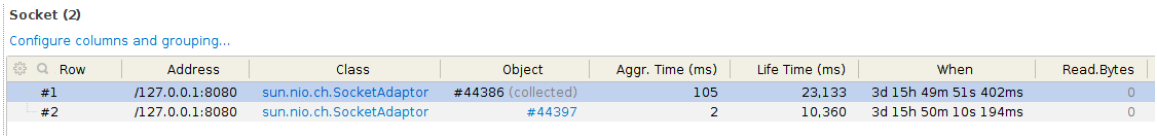


Figure 39: Objects Specific to Web Application Immediately after Execution

At the beginning of the trial, the author followed the process explained earlier to get the results depicted in Figure 39 above. As can be seen, 100% of the web application's objects were reachable via strong references even though the overall object reachability of the web service at start was 41% as depicted in Figure 14. One would argue that the 58% of objects that were unreachable at start-up belong to Tomcat (web service) elements and not the internal web application that it is managing. As the figure below shows, the web socket is also allocating its objects within the same heap and some of the dead objects are being collected. The figure shows

that object number 44386 belonging to web socket adaptor has already been collected and object number 44397 is still reachable to the live thread.



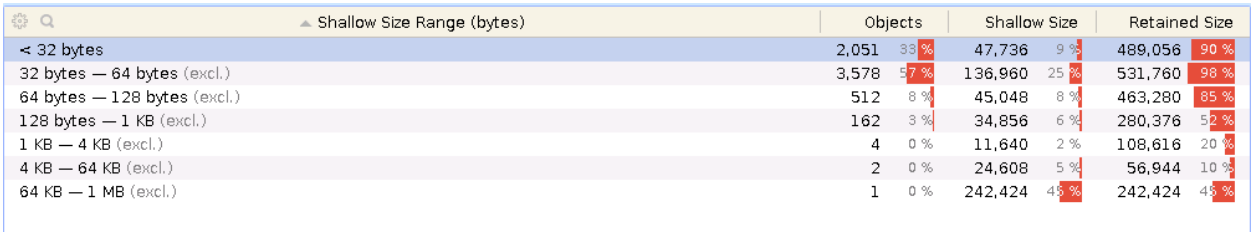
Socket (2)

Configure columns and grouping...

Row	Address	Class	Object	Aggr. Time (ms)	Life Time (ms)	When	Read.Bytes
#1	/127.0.0.1:8080	sun.nio.ch.SocketAdaptor	#44386 (collected)	105	23,133	3d 15h 49m 51s 402ms	0
#2	/127.0.0.1:8080	sun.nio.ch.SocketAdaptor	#44397	2	10,360	3d 15h 50m 10s 194ms	0

Figure 40: Web Element Objects

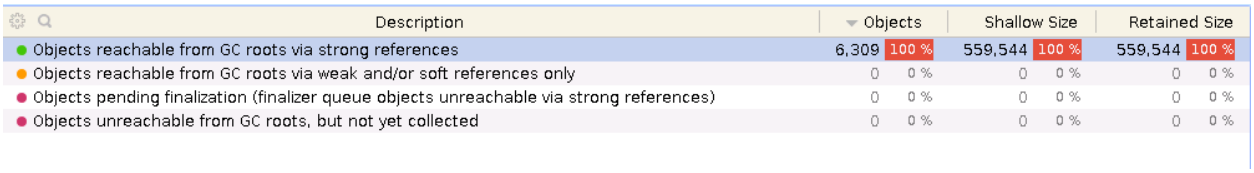
Again, going down with the menu of Figure 38 and selecting “Shallow size”, the author was able to see the sizes of the objects the application was allocating. The figure below shows the breakdown of the objects in terms of sizes and if the numbers are summed, a total number of 6,310 objects shown in the Figure 39 above is obtained. This type of object size breakdown might help the garbage collection developer to fine-tune the algorithm to suit the application simply by determining the sizes of objects being allocated by the application.



Shallow Size Range (bytes)	Objects		Shallow Size		Retained Size	
< 32 bytes	2,051	33 %	47,736	9 %	489,056	90 %
32 bytes — 64 bytes (excl.)	3,578	57 %	136,960	25 %	531,760	98 %
64 bytes — 128 bytes (excl.)	512	8 %	45,048	8 %	463,280	85 %
128 bytes — 1 KB (excl.)	162	3 %	34,856	6 %	280,376	52 %
1 KB — 4 KB (excl.)	4	0 %	11,640	2 %	108,616	20 %
4 KB — 64 KB (excl.)	2	0 %	24,608	5 %	56,944	10 %
64 KB — 1 MB (excl.)	1	0 %	242,424	45 %	242,424	45 %

Figure 41: Object Sizes Specific to Web Application

The information depicted in Figure 39 was collected on the first day of the trial immediately after starting up Tomcat. And at the end of the 10-day trial period, the author also wanted to know the number of objects specifically belonging to the web application that were retained. It was observed that only one object was deleted but still the reachability via strong references was 100% as shown in the figure below.



Description	Objects	Shallow Size	Retained Size
Objects reachable from GC roots via strong references	6,309 100 %	559,544 100 %	559,544 100 %
Objects reachable from GC roots via weak and/or soft references only	0 0 %	0 0 %	0 0 %
Objects pending finalization (finalizer queue objects unreachable via strong references)	0 0 %	0 0 %	0 0 %
Objects unreachable from GC roots, but not yet collected	0 0 %	0 0 %	0 0 %

Figure 42: Objects Specific to Web Application after 10 Days

At the end of trial period, the author also tried again to calculate the cost/revenue of the crops using the web application. Surprisingly the application was able to provide the correct results in a short period of time without even hanging. The researcher expected the application not to respond after staying idle for 10 days but the application responded normally. One would conclude that objects belonging to an application are long maintained in a web service and that most of the objects that become unreachable belong to the web elements. As it was observed throughout the 10-day trial period, the reachability percentage of objects kept on decreasing but the real number of objects that were reachable via strong references remained almost constant, about 160 thousand plus.

Chapter Summary

In this chapter, the author has demonstrated the aging of the web application's objects over a period of 10 days. It has been observed that most of the web application's objects become unreachable with the passage of time if there is no interaction with the web application. However, it was also noticed that there were some dominant objects that the web application maintained until it exited the execution cycle. Most importantly the author has shown that even though the percentage of objects reachable via strong references continued to decrease with the passage of time, the real number of objects referenced via strong GC roots references remained almost the same. It is shown that throughout the 10-day trial, the specific web application (crops) lost only one object out of the 6310 objects it allocated during the initial execution. This trend prompts to conclude that an application in a web service retains most of its objects for a long

time and that most of the objects that decay belong to the web elements and not the application program itself.

Chapter V: Conclusions, Discussion, and Future Work

This chapter concludes this thesis. It also suggests the future work which might be an extension to this work, but a necessary one.

Conclusion

The main aim of this thesis was to evaluate the strength of web application objects with the passage of time as the application keeps on executing. Due to idle state of an application, the application's allocated objects might be assumed dead/ weak (unreferenced) and swept by the garbage collectors. However, using YourKit-Java Profiler, the suggested hypotheses were tested and proved that the web application used, retained most of its objects for a long time until the program exited execution phase. The above-mentioned profiling tool was able to detect the various strengths of web application's objects at different levels of the trial. Most importantly, the profiler was able to separate the objects allocated by the web application and the web service elements so as to give a clear picture of the strength of objects belonging to the test web application in particular. Ultimately, it was proved that the allocated objects were still strong by observing the high-performance of the web application at the end of the 10-day trial period. This answered the question of object reusability in web architecture raised above. The web application reused the objects that were maintained in a pool for 10 days. Despite garbage collections being triggered for hundreds of times, the number of objects allocated by the web application remained almost constant. This suggests that most of the objects that became unreachable and collected by the garbage collectors belonged to web elements. With passage of time, the percentage of reachable objects via strong references declined but objects belonging to the test web application remained reachable throughout the study period. Furthermore, using the garbage collection logs, the study has shown that objects can be prematurely aged/collected and the situation can be

avoided by assigning a reasonably large amount of memory space in Eden space. In this case, the objects were not collected at their weakest stage but due to allocation failure which eventually caused premature collection.

Discussion

Object pooling. This study has shown that the test web application used mostly the technology called object pooling, a creational design pattern that places objects in a pool. From the study it can be seen that the web application is maintaining most of its objects until the program exits the execution phase. By using the object pooling method, the web application is placing its objects in a pool address instead of creating and destroying them on demand. This suggests that the web application's objects are considered to be expensive to be created and destroyed on demand. Only less expensive objects allocated by the web elements will be generated and destroyed on demand. And that explains why the author was able to use the application after 10-days of idleness without hanging, because object pooling allows object reusability and improves performance.

Future Work

This study used Apache Tomcat as a container for the test web application. Similarly, the same study can be retried on Microsoft IIS to determine if the web application behaves the same way on different infrastructure using the same java garbage collectors. Considering that Microsoft IIS and Apache Tomcat have many functionality differences despite offering the same web services, will the web application behave the same way as observed on the Apache Tomcat container?!

References

Andrew, B., Greg, N., Susan, O., & Edward, W. (1995). *Network Objects.* Palo Alto, Califonia: Digital Systems Research Center.

Chavhan, M. (2017, January 8). *(The Complete Guide) to Configure & Install Tomcat 8 on Ubuntu 14.04*. Retrieved from Poweruphosting.com: https://poweruphosting.com/blog/install-tomcat-8-ubuntu/

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001, March 15). *Web Services Description Language (WSDL) 1.1*. Retrieved from www.w3.org: https://www.w3.org/TR/2001/NOTE-wsdl-20010315

Detlefs, D., Heller, S., Flood, C., & Printezis, T. (2004). Garbage-First garbage collection. *4th International Symposium on Memory Management* (pp. 1-2). Vancouver: Sun Microsystems, Inc.

Guruge', A. (2004). *Web Services Theory and Practice.* Burlington: Elsevier Digital Press.

Hajime, I., Darko, S., & Forrest, S. (2006). On the Prediction of Java Object Lifetimes. *IEEE Transactions on Computers*, 880.

Helland, P. (2017, December). JSON and XML Are Like Cardboard. *Communications of the ACM*, p. 46.

JSON. (n.d.). *Introducing JSON*. Retrieved from Json.org: http://www.json.org/

Kulchenko, P., Tidwell, D., & Snell, J. (December, 2001). *Programming Web Services with SOAP*. CA: O'Really media, Inc.

Lee, S. (2017, May 31). Understanding Java Garbage Collection. *Become a Java GC Expert*.

Lissack, S. (2013, December 4). WebSockets – A Quick Introduction and a Sample Application. *IDR Solutions*. Retrieved from idrsolutions.com: www.idrsolutions.com

Mark, W., Allan, M., & Kunal, M. (2003). *Sams Teach Yourself JavaServer Pages 2.0 with Apache Tomcat in 24 Hours.* Indianapolis: Sams Publishing.

Oracle. (2010). *RMI System Overview; Garbage Collection of Remote Objects*. Retrieved from Java Remote Method Invocation: https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmi-arch4.html

Oracle Corporation. (2010). *The Java EE 6 Tutorial: Overview of the EL.* Retrieved from Oracle Corporation website: https://docs.oracle.com/cd/E19798-01/821-1841/bnahq/index.html

Raqeeb, A., Guster, D. C., & Schmidt, M. (2017). Application level memory management strategies via the "Garbage Collector": Performance and security ramifications. *mics_2017_proceedings* (pp. 7-12). St. Cloud: micsymposium.

Shaoshan, L., Jie, T., Ligang, W., Xiao-Feng, L., & Jean-Luc, G. (2012). Packer: Parallel Garbage Collection Based on Virtual Spaces. *IEEE Transactions on Computers*, 1611.

Sun Microsystems. (2006, April). *Memory Management in the Java Hotspot Virtual Machine.* Sun Microsystems, Inc.

Wikipedia. (2017, June 23). *Web service*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Web_service

Williams, M. J., & Chitta, K. (n.d.). *Java Garbage Collection Basics*. Retrieved from oracle.com: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

YourKit. (2017, June 2). *The Industry Leader Among Profiling Tools*. Retrieved from YourKit website: www.YourKit.com

Appendix A: Crops Program Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Crop extends JApplet implements ActionListener
{
private JTextField n1Text = new JTextField();
private JTextField n2Text = new JTextField();
private JTextField n3Text = new JTextField();
private JTextField ansText = new JTextField();
private JLabel Instructions = new JLabel("Enter number of acres for desired for each crop then
press cost or reveune");
private JLabel n1Label = new JLabel("Enter acres of soybeans here");
private JLabel n2Label = new JLabel("Enter acres of corn here!");
private JLabel n3Label = new JLabel("Enter acres of potatoes here");
private JLabel answerLabel = new JLabel("Total cost or revenue");
private JButton costButton = new JButton("cost");
private JButton revenueButton = new JButton("revenue");
double sum;
double n1;
double n2;
double n3;

public void init()
{
ansText.setEditable(false);
JPanel content = new JPanel(new GridLayout(4,2,1,1));
// content.add(Instructions);
content.add(n1Label);
content.add(n1Text);
content.add(n2Label);
content.add(n2Text);
content.add(n3Label);
content.add(n3Text);
content.add(answerLabel);
content.add(ansText);
JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
buttonPanel.add(costButton);
buttonPanel.add(revenueButton);

add(content, BorderLayout.NORTH);
```

```
add(buttonPanel, BorderLayout.SOUTH);
revenueButton.addActionListener(this);
costButton.addActionListener(this);

}
public void actionPerformed(ActionEvent e)
{
if (e.getActionCommand().equals("cost"))
{
n1 = stringToDouble(n1Text.getText());
n2 = stringToDouble(n2Text.getText());
n3 = stringToDouble(n3Text.getText());
sum = (n1 * 900) + (n2 * 100) + (n3 * 750);
ansText.setText(Double.toString(sum));
}
else if (e.getActionCommand().equals("revenue"))
{
n1 = stringToDouble(n1Text.getText());
n2 = stringToDouble(n2Text.getText());
n3 = stringToDouble(n2Text.getText());
sum = (n1 * 1300) + (n2 * 1650) + (n3 * 1200);
ansText.setText(Double.toString(sum));
}

else
{
ansText.setText("Error");
}}
private static double stringToDouble(String stringObject)
{
return Double.parseDouble(stringObject.trim());
}}
```

Appendix B: Apache Tomcat 8 Installation Procedure

1. Add Java repository into the server by typing the following command:

   sudo add-apt-repository ppa:webupd8team/java

   press enter;

   https://launchpad.net/~webupd8team/+archive/ubuntu/java

   Press [ENTER] to continue or ctrl-c to cancel adding it

   press enter;

2. If any repository package is missing, use the following command:

   sudo apt-get install software-properties-common -y

3. Update index repository to add Java using the command below:

   sudo apt-get update

4. Now you can install Java by typing:

   sudo apt-get install oracle-java8-installer -y

5. Create tomcat user and client by doing the following:

   sudo groupadd tomcat

   sudo useradd -s /bin/false -g tomcat -d /opt/apache-tomcat-8.0.49 tomcat

6. Download the tomcat from trusted source using the below command:

   sudo wget http://www-us.apache.org/dist/tomcat/tomcat-8/v8.0.49/bin/apache

7. Unzip the downloaded package and remove the archive file

   sudo tar -xvzf apache-tomcat-8.0.49.tar.gz && rm -rf apache-tomcat-8.0.49

8. Change the directory to tomcat folder

   cd apache-tomcat-8.0.49

9. Provide recursive permissions to conf folder

sudo chgrp -R tomcat conf/

10. Give tomcat client group access to the conf folder and read rights to the records in that folder

    sudo chmod g+rwx conf/

    sudo chmod g+r conf/*

11. Provide recursive permission to temp folder

    sudo chmod -R g+rwx work/ logs/ temp/

12. Check the location of installed Java

    sudo update-alternatives --config java

13. Open the tomcat configuration file by typing the command below:

    sudo nano /etc/init/tomcat.conf

14. Configure the tomcat configuration file by inserting the below script:

    start on runlevel [2345]

    stop on runlevel [!2345]

    respawn

    respawn limit 10 5

    setuid tomcat

    setgid tomcat

    env JAVA_HOME=/usr/lib/jvm/java-8-oracle/jre

    env CATALINA_HOME=/opt/apache-tomcat-8.0.49

    env JAVA_OPTS="-Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom"

env CATALINA_OPTS="-XX:+PrintGCDetails -XX:+PrintHeapAtGC -Xloggc:/opt/apache-tomcat-8.0.49/logs/gc.log

-agentpath:home/patrick/Desktop/YourKit-JavaProfiler-2017.02/bin/linux-x86-64/libyjpagent.so=disablestacktelemetry,exceptions=disable,delay=10000"

exec $CATALINA_HOME/bin/catalina.sh run

post-stop script

rm -rf $CATALINA_HOME/temp/*

end script

NB: Make sure the Java location is the one found on step 12 above.

15. Initialize the configurations made in step 14 by typing the command below:

    sudo initctl reload-configuration

16. Start tomcat by issuing the following command:

    sudo initctl start tomcat

17. Open the tomcat-users.xml file:

    sudo nano /opt/tomcat/conf/tomcat-users.xml

18. Add tomcat web-app users and roles by including the following script:

    <tomcat-users>

    <role rolename=”manager-gui”/>

    <role rolename=”admin-gui”/>

    <user username="Administrator" password="P@ssw0rd" roles="manager-gui, admin-gui”

    </tomcat-users>

19. Save and quit the tomcat users file

20. Restart the tomcat by issuing the following command:

    sudo initctl restart tomcat

21. Check the service if it is running:

    sudo netstat -antup | grep 8080

22. Access tomcat web application by typing the below URL

    http://"localhost":8080/manager/html

23. Finally create a log file called log.gc in /opt/apache-tomcat-8.0.49/logs by typing the command:

    sudo touch /opt/apache-tomcat-8.0.49/logs/log.gc

Note: Adapted from Chavhan (2017).

Appendix C: Garbage Collection Logs

Java HotSpot(TM) 64-Bit Server VM (25.161-b12) for linux-amd64 JRE (1.8.0_161-b12), built on Dec 19 2017 16:12:43 by "java_re" with gcc 4.3.0 20080428 (Red Hat 4.3.0-8)
Memory: 4k page, physical 2049908k(718868k free), swap 2096124k(2096124k free)
CommandLine flags: -XX:InitialHeapSize=536870912 -XX:MaxHeapSize=1073741824 -XX:MaxNewSize=87244800 -XX:MaxTenuringThreshold=6 -XX:NewSize=87244800 -XX:OldPLABSize=16 -XX:OldSize=174489600 -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintHeapAtGC -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:+UseParNewGC
2018-03-01T10:11:22.811-0600: 34.733: [GC (CMS Initial Mark) [1 CMS-initial-mark: 0K(439104K)] 61909K(515776K), 0.1870193 secs] [Times: user=0.09 sys=0.00, real=0.18 secs]
2018-03-01T10:11:22.998-0600: 34.920: [CMS-concurrent-mark-start]
2018-03-01T10:11:23.051-0600: 34.973: [CMS-concurrent-mark: 0.053/0.053 secs] [Times: user=0.01 sys=0.01, real=0.06 secs]
2018-03-01T10:11:23.065-0600: 34.987: [CMS-concurrent-preclean-start]
2018-03-01T10:11:23.067-0600: 34.989: [CMS-concurrent-preclean: 0.002/0.002 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]
2018-03-01T10:11:23.088-0600: 35.010: [CMS-concurrent-abortable-preclean-start]
{Heap before GC invocations=0 (full 1): par new generation   total 76672K, used 68160K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 0K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 21771K, capacity 22052K, committed 22292K, reserved 1069056K   class space   used 1896K, capacity 1978K, committed 2048K, reserved 1048576K
2018-03-01T10:11:27.901-0600: 39.823: [GC (Allocation Failure)
2018-03-01T10:11:27.902-0600: 39.824: [ParNew2018-03-01T10:11:28.003-0600: 39.925: [CMS-concurrent-abortable-preclean: 0.506/4.915 secs] [Times: user=2.31 sys=0.06, real=4.92 secs]
: 68160K->8512K(76672K), 0.6636648 secs] 68160K->17462K(515776K), 0.6640229 secs] [Times: user=0.29 sys=0.04, real=0.66 secs]
Heap after GC invocations=1 (full 1): par new generation   total 76672K, used 8512K [0x00000000c0000000,

0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K, 100% used [0x00000000c4ae0000, 0x00000000c5330000, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 8950K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 21771K, capacity 22052K, committed 22292K, reserved 1069056K   class space   used 1896K, capacity 1978K, committed 2048K, reserved 1048576K }
2018-03-01T10:11:28.580-0600: 40.502: [GC (CMS Final Remark) [YG occupancy: 11515 K (76672 K)]2018-03-01T10:11:28.587-0600: 40.509: [Rescan (parallel) , 0.0413391 secs]2018-03-01T10:11:28.636-0600: 40.558: [weak refs processing, 0.0139206 secs]2018-03-01T10:11:28.650-0600: 40.572: [class unloading, 0.0282938 secs]2018-03-01T10:11:28.684-0600: 40.605: [scrub symbol table, 0.0044955 secs]2018-03-01T10:11:28.688-0600: 40.610: [scrub string table, 0.0057054 secs][1
CMS-remark: 8950K(439104K)] 20465K(515776K), 0.1143235 secs] [Times: user=0.05 sys=0.00, real=0.11 secs]
2018-03-01T10:11:28.708-0600: 40.630: [CMS-concurrent-sweep-start]
2018-03-01T10:11:28.726-0600: 40.648: [CMS-concurrent-sweep: 0.018/0.018 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]
2018-03-01T10:11:28.726-0600: 40.648: [CMS-concurrent-reset-start]
2018-03-01T10:11:28.795-0600: 40.716: [CMS-concurrent-reset: 0.050/0.069 secs] [Times: user=0.03 sys=0.01, real=0.07 secs]
{Heap before GC invocations=1 (full 1): par new generation   total 76672K, used 76672K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K, 100% used [0x00000000c4ae0000, 0x00000000c5330000, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 8950K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23896K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2057K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T11:06:39.405-0600: 3352.178: [GC (Allocation Failure)
2018-03-01T11:06:39.445-0600: 3352.218: [ParNew: 76672K->8512K(76672K), 0.3291716 secs] 85622K->18983K(515776K), 0.3689226 secs] [Times: user=0.28 sys=0.04, real=0.37 secs]

```
Heap after GC invocations=2 (full 1):  par new generation   total 76672K, used 8512K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K, 100% used [0x00000000c4290000, 0x00000000c4ae0000,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 10471K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23896K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2057K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=2 (full 1):  par new generation   total 76672K, used 76672K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K, 100% used [0x00000000c4290000, 0x00000000c4ae0000,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 10471K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23904K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2057K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T12:15:58.783-0600: 7511.556: [GC (Allocation Failure)
2018-03-01T12:15:58.783-0600: 7511.557: [ParNew: 76672K->8122K(76672K), 0.2125507
secs] 87143K->27409K(515776K), 0.2136497 secs] [Times: user=0.15 sys=0.06, real=0.21 secs]
Heap after GC invocations=3 (full 1):  par new generation   total 76672K, used 8122K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,  95% used [0x00000000c4ae0000, 0x00000000c52ceae0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
```

```
 Metaspace       used 23904K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2057K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=3 (full 1):  par new generation   total 76672K, used 76282K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,  95% used [0x00000000c4ae0000, 0x00000000c52ceae0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23914K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2058K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T13:26:32.395-0600: 11745.168: [GC (Allocation Failure)
2018-03-01T13:26:32.395-0600:  11745.168:  [ParNew:  76282K->2436K(76672K),  0.1583342
secs] 95569K->21723K(515776K), 0.1587952 secs] [Times: user=0.11 sys=0.04, real=0.16 secs]
Heap after GC invocations=4 (full 1):  par new generation   total 76672K, used 2436K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,  28% used [0x00000000c4290000, 0x00000000c44f1358,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23914K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2058K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=4 (full 1):  par new generation   total 76672K, used 70596K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,  28% used [0x00000000c4290000, 0x00000000c44f1358,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
```

```
 Metaspace       used 23915K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2058K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T14:36:31.947-0600: 15944.721: [GC (Allocation Failure)
2018-03-01T14:36:31.948-0600: 15944.721: [ParNew: 70596K->1009K(76672K), 0.1302101
secs] 89883K->20296K(515776K), 0.1306909 secs] [Times: user=0.10 sys=0.03, real=0.13 secs]
Heap after GC invocations=5 (full 1):  par new generation   total 76672K, used 1009K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,  11% used [0x00000000c4ae0000, 0x00000000c4bdc7f8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23915K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2058K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=5 (full 1):
 par new generation   total 76672K, used 69169K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,  11% used [0x00000000c4ae0000, 0x00000000c4bdc7f8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23918K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T15:47:46.512-0600: 20219.285: [GC (Allocation Failure)
2018-03-01T15:47:46.512-0600: 20219.285: [ParNew: 69169K->658K(76672K), 0.1410045
secs] 88456K->19945K(515776K), 0.1414930 secs] [Times: user=0.12 sys=0.02, real=0.14 secs]
Heap after GC invocations=6 (full 1):  par new generation   total 76672K, used 658K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4334960,
0x00000000c4ae0000)
```

```
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23918K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=6 (full 1):  par new generation   total 76672K, used 68818K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4334960,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23919K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T16:58:33.273-0600: 24466.046: [GC (Allocation Failure)
2018-03-01T16:58:33.273-0600: 24466.046: [ParNew: 68818K->648K(76672K), 0.1441306
secs] 88105K->19935K(515776K), 0.1445697 secs] [Times: user=0.09 sys=0.05, real=0.14 secs]
Heap after GC invocations=7 (full 1):
 par new generation   total 76672K, used 648K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b821c0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23919K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=7 (full 1):  par new generation   total 76672K, used 68808K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b821c0,
```

0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23919K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T18:09:58.200-0600: 28750.973: [GC (Allocation Failure)
2018-03-01T18:09:58.201-0600: 28750.974: [ParNew: 68808K->703K(76672K), 0.1462758
secs] 88095K->19990K(515776K), 0.1466830 secs] [Times: user=0.10 sys=0.05, real=0.15 secs]
Heap after GC invocations=8 (full 1): par new generation   total 76672K, used 703K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c433ff00,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23919K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=8 (full 1): par new generation   total 76672K, used 68863K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c433ff00,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23933K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T19:20:25.744-0600: 32978.517: [GC (Allocation Failure)
2018-03-01T19:20:25.744-0600: 32978.517: [ParNew: 68863K->645K(76672K), 0.2038301
secs] 88150K->19932K(515776K), 0.2045051 secs] [Times: user=0.18 sys=0.02, real=0.20 secs]
Heap after GC invocations=9 (full 1): par new generation   total 76672K, used 645K
[0x00000000c0000000,

0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b81610,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23933K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=9 (full 1): par new generation   total 76672K, used 68805K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b81610,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23933K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T20:31:04.627-0600: 37217.400: [GC (Allocation Failure)
2018-03-01T20:31:04.627-0600: 37217.400: [ParNew: 68805K->725K(76672K), 0.1462418
secs] 88092K->20012K(515776K), 0.1472318 secs] [Times: user=0.12 sys=0.03, real=0.15 secs]
Heap after GC invocations=10 (full 1): par new generation   total 76672K, used 725K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4345770,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23933K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }

{Heap before GC invocations=10 (full 1): par new generation   total 76672K, used 68885K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4345770, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23935K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T21:41:42.762-0600: 41455.535: [GC (Allocation Failure)
2018-03-01T21:41:42.763-0600: 41455.536: [ParNew: 68885K->653K(76672K), 0.2693052 secs] 88172K->19940K(515776K), 0.2699700 secs] [Times: user=0.10 sys=0.04, real=0.27 secs]
Heap after GC invocations=11 (full 1): par new generation   total 76672K, used 653K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b83640, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23935K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=11 (full 1): par new generation   total 76672K, used 68813K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b83640, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)

```
 Metaspace       used 23935K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-01T22:52:41.151-0600: 45713.924: [GC (Allocation Failure)
2018-03-01T22:52:41.152-0600: 45713.925: [ParNew: 68813K->732K(76672K), 0.2653744
secs] 88100K->20018K(515776K), 0.2664209 secs] [Times: user=0.21 sys=0.05, real=0.26 secs]
Heap after GC invocations=12 (full 1):  par new generation   total 76672K, used 732K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4347040,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23935K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=12 (full 1):  par new generation   total 76672K, used 68892K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4347040,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23941K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T00:03:33.387-0600: 49966.160: [GC (Allocation Failure)
2018-03-02T00:03:33.387-0600: 49966.160: [ParNew: 68892K->666K(76672K), 0.1714158
secs] 88178K->19953K(515776K), 0.1719200 secs] [Times: user=0.14 sys=0.02, real=0.17 secs]
Heap after GC invocations=13 (full 1):  par new generation   total 76672K, used 666K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b868a0,
0x00000000c5330000)
```

```
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23941K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=13 (full 1):  par new generation   total 76672K, used 68826K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b868a0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23943K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T01:14:22.815-0600: 54215.588: [GC (Allocation Failure)
2018-03-02T01:14:22.815-0600: 54215.588: [ParNew: 68826K->731K(76672K), 0.1391793
secs] 88113K->20017K(515776K), 0.1400163 secs] [Times: user=0.11 sys=0.03, real=0.14 secs]
Heap after GC invocations=14 (full 1):  par new generation   total 76672K, used 731K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4346c60,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23943K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=14 (full 1):  par new generation   total 76672K, used 68891K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4346c60,
```

0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23943K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T02:26:51.522-0600: 58464.685: [GC (Allocation Failure)
2018-03-02T02:26:51.522-0600: 58464.686: [ParNew: 68891K->660K(76672K), 0.1424780
secs] 88177K->19947K(515776K), 0.1429617 secs] [Times: user=0.10 sys=0.03, real=0.14 secs]
Heap after GC invocations=15 (full 1): par new generation   total 76672K, used 660K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b852b0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23943K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=15 (full 1): par new generation   total 76672K, used 68820K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b852b0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23961K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T03:37:40.406-0600: 62713.569: [GC (Allocation Failure)
2018-03-02T03:37:40.406-0600: 62713.569: [ParNew: 68820K->719K(76672K), 0.2657879
secs] 88107K->20006K(515776K), 0.2661477 secs] [Times: user=0.09 sys=0.04, real=0.27 secs]
Heap after GC invocations=16 (full 1): par new generation   total 76672K, used 719K
[0x00000000c0000000,

0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4343e80,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23961K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=16 (full 1): par new generation   total 76672K, used 68879K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   8% used [0x00000000c4290000, 0x00000000c4343e80,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23961K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T04:48:30.252-0600: 66963.415: [GC (Allocation Failure)
2018-03-02T04:48:30.252-0600: 66963.415: [ParNew: 68879K->647K(76672K), 0.1385863
secs] 88166K->19933K(515776K), 0.1390805 secs] [Times: user=0.11 sys=0.02, real=0.14 secs]
Heap after GC invocations=17 (full 1): par new generation   total 76672K, used 647K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b81c08,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23961K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }

{Heap before GC invocations=17 (full 1): par new generation   total 76672K, used 68807K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b81c08, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23964K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K 2018-03-02T05:59:29.228-0600: 71222.392: [GC (Allocation Failure)
2018-03-02T05:59:29.228-0600: 71222.392: [ParNew: 68807K->640K(76672K), 0.1540331 secs] 88093K->19927K(515776K), 0.1551003 secs] [Times: user=0.12 sys=0.02, real=0.16 secs]
Heap after GC invocations=18 (full 1): par new generation   total 76672K, used 640K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4330260, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23964K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=18 (full 1): par new generation   total 76672K, used 68800K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4330260, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19286K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)

```
 Metaspace       used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T07:10:10.333-0600: 75463.497: [GC (Allocation Failure)
2018-03-02T07:10:10.334-0600: 75463.497: [ParNew: 68800K->636K(76672K), 0.1312685
secs] 88087K->19982K(515776K), 0.1317528 secs] [Times: user=0.11 sys=0.02, real=0.13 secs]
Heap after GC invocations=19 (full 1):  par new generation   total 76672K, used 636K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b7f360,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19345K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
}
{Heap before GC invocations=19 (full 1):  par new generation   total 76672K, used 68796K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b7f360,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19345K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T08:21:05.961-0600: 79719.124: [GC (Allocation Failure)
2018-03-02T08:21:05.961-0600: 79719.124: [ParNew: 68796K->583K(76672K), 0.1533227
secs] 88142K->19930K(515776K), 0.1537727 secs] [Times: user=0.12 sys=0.03, real=0.15 secs]
Heap after GC invocations=20 (full 1):  par new generation   total 76672K, used 583K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c4321d00,
```

0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=20 (full 1): par new generation   total 76672K, used 68743K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c4321d00,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T09:31:27.290-0600: 83940.453: [GC (Allocation Failure)
2018-03-02T09:31:27.290-0600: 83940.454: [ParNew: 68743K->574K(76672K), 0.1456788
secs] 88090K->19922K(515776K), 0.1463330 secs] [Times: user=0.10 sys=0.04, real=0.15 secs]
Heap after GC invocations=21 (full 1): par new generation   total 76672K, used 574K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6faa0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23965K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=21 (full 1): par new generation   total 76672K, used 68734K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)

```
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6faa0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23968K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T10:42:05.796-0600: 88178.959: [GC (Allocation Failure)
2018-03-02T10:42:05.796-0600: 88178.959: [ParNew: 68734K->617K(76672K), 0.1497064
secs] 88082K->19965K(515776K), 0.1502260 secs] [Times: user=0.13 sys=0.03, real=0.15 secs]
Heap after GC invocations=22 (full 1):  par new generation   total 76672K, used 617K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c432a790,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23968K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=22 (full 1):
 par new generation   total 76672K, used 68777K [0x00000000c0000000, 0x00000000c5330000,
0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c432a790,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23969K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T11:52:54.723-0600: 92427.886: [GC (Allocation Failure)
2018-03-02T11:52:54.723-0600: 92427.886: [ParNew: 68777K->587K(76672K), 0.1300942
secs] 88125K->19934K(515776K), 0.1304894 secs] [Times: user=0.11 sys=0.02, real=0.13 secs]
```

```
Heap after GC invocations=23 (full 1):  par new generation   total 76672K, used 587K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72d80,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23969K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=23 (full 1):  par new generation   total 76672K, used 68747K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72d80,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T13:03:55.612-0600: 96688.775: [GC (Allocation Failure)
2018-03-02T13:03:55.612-0600: 96688.775: [ParNew: 68747K->612K(76672K), 0.2581242
secs] 88094K->19959K(515776K), 0.2585423 secs] [Times: user=0.08 sys=0.04, real=0.26 secs]
Heap after GC invocations=24 (full 1):
 par new generation   total 76672K, used 612K [0x00000000c0000000, 0x00000000c5330000,
0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c43291b0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
```

```
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=24 (full 1):  par new generation   total 76672K, used 68772K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c43291b0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T14:14:42.924-0600: 100936.087: [GC (Allocation Failure)
2018-03-02T14:14:42.924-0600: 100936.088: [ParNew: 68772K->588K(76672K), 0.1500159
secs] 88119K->19935K(515776K), 0.1508630 secs] [Times: user=0.10 sys=0.04, real=0.15 secs]
Heap after GC invocations=25 (full 1):  par new generation   total 76672K, used 588K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b73058,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=25 (full 1):  par new generation   total 76672K, used 68748K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b73058,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
```

0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T15:26:31.030-0600: 105244.194: [GC (Allocation Failure)
2018-03-02T15:26:31.031-0600: 105244.194: [ParNew: 68748K->570K(76672K), 0.1865460 secs] 88095K->19918K(515776K), 0.1868992 secs] [Times: user=0.13 sys=0.05, real=0.18 secs]
Heap after GC invocations=26 (full 1): par new generation   total 76672K, used 570K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431ebc8, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=26 (full 1): par new generation   total 76672K, used 68730K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431ebc8, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T16:38:00.253-0600: 109533.416: [GC (Allocation Failure)
2018-03-02T16:38:00.253-0600: 109533.416: [ParNew: 68730K->560K(76672K), 0.1800214 secs] 88078K->19907K(515776K), 0.1804499 secs] [Times: user=0.13 sys=0.05, real=0.18 secs]
Heap after GC invocations=27 (full 1): par new generation   total 76672K, used 560K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6c078,

0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=27 (full 1): par new generation   total 76672K, used 68720K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6c078,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T17:49:28.057-0600: 113821.221: [GC (Allocation Failure)
2018-03-02T17:49:28.057-0600: 113821.221: [ParNew: 68720K->604K(76672K), 0.2932165
secs] 88067K->19951K(515776K), 0.2935994 secs] [Times: user=0.11 sys=0.04, real=0.30 secs]
Heap after GC invocations=28 (full 1): par new generation   total 76672K, used 604K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4327250,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=28 (full 1): par new generation   total 76672K, used 68764K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)

```
  from space 8512K,   7% used [0x00000000c4290000, 0x00000000c4327250,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T19:00:58.549-0600: 118111.713: [GC (Allocation Failure)
2018-03-02T19:00:58.549-0600: 118111.713: [ParNew: 68764K->582K(76672K), 0.2948191
secs] 88111K->19930K(515776K), 0.2951752 secs] [Times: user=0.11 sys=0.04, real=0.30 secs]
Heap after GC invocations=29 (full 1):  par new generation   total 76672K, used 582K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b71af0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=29 (full 1):  par new generation   total 76672K, used 68742K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b71af0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T20:12:53.676-0600: 122426.839: [GC (Allocation Failure)
2018-03-02T20:12:53.676-0600: 122426.839: [ParNew: 68742K->570K(76672K), 0.1330805
secs] 88090K->19917K(515776K), 0.1334585 secs] [Times: user=0.08 sys=0.04, real=0.13 secs]
```

Heap after GC invocations=30 (full 1): par new generation   total 76672K, used 570K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431e938, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=30 (full 1): par new generation   total 76672K, used 68730K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431e938, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T21:25:59.035-0600: 126808.776: [GC (Allocation Failure)
2018-03-02T21:25:59.035-0600: 126808.777: [ParNew: 68730K->587K(76672K), 0.4216312 secs] 88077K->19935K(515776K), 0.4219700 secs] [Times: user=0.18 sys=0.02, real=0.42 secs]
Heap after GC invocations=31 (full 1): par new generation   total 76672K, used 587K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72ee0, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)

```
 Metaspace       used 23971K, capacity 24464K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=31 (full 1):  par new generation   total 76672K, used 68747K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72ee0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T22:38:04.978-0600: 131134.719: [GC (Allocation Failure)
2018-03-02T22:38:04.978-0600: 131134.719: [ParNew: 68747K->559K(76672K), 0.1489634
secs] 88095K->19906K(515776K), 0.1493338 secs] [Times: user=0.11 sys=0.03, real=0.15 secs]
Heap after GC invocations=32 (full 1):  par new generation   total 76672K, used 559K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431bc08,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=32 (full 1):  par new generation   total 76672K, used 68719K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431bc08,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19347K [0x00000000c5330000,
```

0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-02T23:50:21.645-0600: 135471.386: [GC (Allocation Failure)
2018-03-02T23:50:21.645-0600: 135471.386: [ParNew: 68719K->598K(76672K), 0.3593216 secs] 88066K->19946K(515776K), 0.3596166 secs] [Times: user=0.15 sys=0.03, real=0.36 secs]
Heap after GC invocations=33 (full 1): par new generation   total 76672K, used 598K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b75990, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=33 (full 1): par new generation   total 76672K, used 68758K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b75990, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T01:02:39.007-0600: 139808.748: [GC (Allocation Failure)
2018-03-03T01:02:39.008-0600: 139808.749: [ParNew: 68758K->567K(76672K), 0.4041199 secs] 88106K->19915K(515776K), 0.4057014 secs] [Times: user=0.16 sys=0.06, real=0.41 secs]
Heap after GC invocations=34 (full 1): par new generation   total 76672K, used 567K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431dee0,

0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=34 (full 1): par new generation   total 76672K, used 68727K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431dee0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T02:14:55.707-0600: 144145.448: [GC (Allocation Failure) 2018-03-
03T02:14:55.707-0600: 144145.448: [ParNew: 68727K->552K(76672K), 0.1339622 secs]
88075K->19900K(515776K), 0.1343080 secs] [Times: user=0.09 sys=0.04, real=0.14 secs]
Heap after GC invocations=35 (full 1): par new generation   total 76672K, used 552K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6a1a8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=35 (full 1): par new generation   total 76672K, used 68712K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)

```
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b6a1a8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T03:25:41.906-0600: 148391.648: [GC (Allocation Failure)
2018-03-03T03:25:41.907-0600: 148391.648: [ParNew: 68712K->572K(76672K), 0.1342712
secs] 88060K->19920K(515776K), 0.1347573 secs] [Times: user=0.10 sys=0.02, real=0.14 secs]
Heap after GC invocations=36 (full 1):  par new generation   total 76672K, used 572K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431f0f0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
}
{Heap before GC invocations=36 (full 1):  par new generation   total 76672K, used 68732K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431f0f0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T04:36:56.148-0600: 152665.889: [GC (Allocation Failure)
2018-03-03T04:36:56.148-0600: 152665.889: [ParNew: 68732K->618K(76672K), 0.1354810
secs] 88080K->19966K(515776K), 0.1358861 secs] [Times: user=0.12 sys=0.02, real=0.14 secs]
```

```
Heap after GC invocations=37 (full 1):  par new generation   total 76672K, used 618K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b7a970,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=37 (full 1):  par new generation   total 76672K, used 68778K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b7a970,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T05:47:56.785-0600: 156926.790: [GC (Allocation Failure)
2018-03-03T05:47:56.785-0600: 156926.790: [ParNew: 68778K->592K(76672K), 0.2598813
secs] 88126K->19941K(515776K), 0.2657309 secs] [Times: user=0.11 sys=0.02, real=0.27 secs]
Heap after GC invocations=38 (full 1):  par new generation   total 76672K, used 592K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c43242f0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
```

```
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=38 (full 1):  par new generation   total 76672K, used 68752K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c43242f0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T06:58:12.838-0600: 161142.843: [GC (Allocation Failure)
2018-03-03T06:58:12.838-0600: 161142.843: [ParNew: 68752K->587K(76672K), 0.2620563
secs] 88101K->19935K(515776K), 0.2623916 secs] [Times: user=0.11 sys=0.02, real=0.26 secs]
Heap after GC invocations=39 (full 1):  par new generation   total 76672K, used 587K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72e10,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23980K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=39 (full 1):  par new generation   total 76672K, used 68747K
[0x00000000c0000000, 0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4ae0000, 0x00000000c4b72e10,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
```

```
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T08:09:05.584-0600: 165395.589: [GC (Allocation Failure)
2018-03-03T08:09:05.584-0600: 165395.589: [ParNew: 68747K->575K(76672K), 0.1306504
secs] 88095K->19923K(515776K), 0.1310088 secs] [Times: user=0.09 sys=0.04, real=0.13 secs]
Heap after GC invocations=40 (full 1):  par new generation   total 76672K, used 575K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431fc70,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=40 (full 1):  par new generation   total 76672K, used 68735K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c431fc70,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T09:19:58.161-0600: 169648.166: [GC (Allocation Failure)
2018-03-03T09:19:58.161-0600: 169648.167: [ParNew: 68735K->614K(76672K), 0.1345798
secs] 88083K->19963K(515776K), 0.1349286 secs] [Times: user=0.09 sys=0.04, real=0.14 secs]
Heap after GC invocations=41 (full 1):  par new generation   total 76672K, used 614K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b79bb0,
0x00000000c5330000)
```

```
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=41 (full 1):  par new generation   total 76672K, used 68774K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   7% used [0x00000000c4ae0000, 0x00000000c4b79bb0,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T10:30:55.296-0600: 173905.301: [GC (Allocation Failure)
2018-03-03T10:30:55.296-0600: 173905.302: [ParNew: 68774K->591K(76672K), 0.1860584
secs] 88123K->19939K(515776K), 0.1863945 secs] [Times: user=0.10 sys=0.03, real=0.18 secs]
Heap after GC invocations=42 (full 1):  par new generation   total 76672K, used 591K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c4323da0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23982K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2059K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=42 (full 1):  par new generation   total 76672K, used 68751K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
```

```
  from space 8512K,   6% used [0x00000000c4290000, 0x00000000c4323da0,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23988K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T11:49:53.543-0600: 178643.591: [GC (Allocation Failure)
2018-03-03T11:49:54.500-0600: 178644.505: [ParNew: 68751K->467K(76672K), 177.0854452
secs] 88099K->19816K(515776K), 178.1548781 secs] [Times: user=0.71 sys=0.23, real=178.16
secs]
Heap after GC invocations=43 (full 1):  par new generation   total 76672K, used 467K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   5% used [0x00000000c4ae0000, 0x00000000c4b54f38,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23988K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=43 (full 1):  par new generation   total 76672K, used 68627K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   5% used [0x00000000c4ae0000, 0x00000000c4b54f38,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T13:42:53.213-0600: 185424.233: [GC (Allocation Failure)
```

```
2018-03-03T13:42:53.426-0600: 185424.415: [ParNew: 68627K->52K(76672K), 37.5398080
secs] 87976K->19400K(515776K), 37.9694845 secs] [Times: user=0.68 sys=0.44, real=37.98
secs]
Heap after GC invocations=44 (full 1):  par new generation   total 76672K, used 52K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c429d168,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=44 (full 1):  par new generation   total 76672K, used 68212K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c429d168,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T15:37:15.317-0600: 192288.337: [GC (Allocation Failure)
2018-03-03T15:37:15.652-0600: 192288.659: [ParNew: 68212K->19K(76672K), 63.6918074
secs] 87560K->19368K(515776K), 64.1657388 secs] [Times: user=0.80 sys=0.53, real=64.19
secs]
Heap after GC invocations=45 (full 1):  par new generation   total 76672K, used 19K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae4fa8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
```

```
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=45 (full 1):  par new generation   total 76672K, used 68179K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae4fa8,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T17:32:45.820-0600: 199221.935: [GC (Allocation Failure)
2018-03-03T17:32:46.094-0600: 199222.167: [ParNew: 68179K->15K(76672K), 73.0527890
secs] 87528K->19363K(515776K), 73.4664910 secs] [Times: user=0.83 sys=0.51, real=73.47
secs]
Heap after GC invocations=46 (full 1):  par new generation   total 76672K, used 15K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4293d60,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=46 (full 1):  par new generation   total 76672K, used 68175K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4293d60,
```

0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T19:25:21.304-0600: 205979.623: [GC (Allocation Failure)
2018-03-03T19:25:21.459-0600: 205979.778: [ParNew: 68175K->7K(76672K), 3.7110027
secs] 87523K->19355K(515776K), 3.9033487 secs] [Times: user=0.33 sys=0.09, real=3.91 secs]
Heap after GC invocations=47 (full 1): par new generation   total 76672K, used 7K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae1c10,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=47 (full 1): par new generation   total 76672K, used 68167K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae1c10,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T21:16:21.072-0600: 212641.903: [GC (Allocation Failure)
2018-03-03T21:16:21.073-0600: 212641.904: [ParNew: 68167K->9K(76672K), 0.1999384
secs] 87515K->19357K(515776K), 0.2007248 secs] [Times: user=0.16 sys=0.04, real=0.20 secs]
Heap after GC invocations=48 (full 1): par new generation   total 76672K, used 9K
[0x00000000c0000000,

0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4292550,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=48 (full 1): par new generation   total 76672K, used 68169K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4292550,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19348K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-03T23:06:04.364-0600: 219227.740: [GC (Allocation Failure)
2018-03-03T23:06:04.364-0600: 219227.740: [ParNew: 68169K->3K(76672K), 0.2805121
secs] 87517K->19355K(515776K), 0.2811219 secs] [Times: user=0.15 sys=0.06, real=0.29 secs]
Heap after GC invocations=49 (full 1): par new generation   total 76672K, used 3K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0c50,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19351K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }

{Heap before GC invocations=49 (full 1): par new generation   total 76672K, used 68163K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0c50, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19351K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T00:56:59.813-0600: 225885.730: [GC (Allocation Failure)
2018-03-04T00:56:59.813-0600: 225885.730: [ParNew: 68163K->2K(76672K), 0.1996204 secs] 87515K->19354K(515776K), 0.2000248 secs] [Times: user=0.13 sys=0.06, real=0.20 secs]
Heap after GC invocations=50 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=50 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K

2018-03-04T02:47:30.120-0600: 232518.613: [GC (Allocation Failure)
2018-03-04T02:47:30.121-0600: 232518.614: [ParNew: 68162K->2K(76672K), 0.2040379 secs] 87514K->19354K(515776K), 0.2046078 secs] [Times: user=0.16 sys=0.04, real=0.20 secs]
Heap after GC invocations=51 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=51 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T04:38:29.211-0600: 239178.992: [GC (Allocation Failure)
2018-03-04T04:38:29.211-0600: 239178.992: [ParNew: 68162K->2K(76672K), 0.1921370 secs] 87514K->19354K(515776K), 0.1927518 secs] [Times: user=0.14 sys=0.05, real=0.20 secs]
Heap after GC invocations=52 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)

```
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=52 (full 1):  par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T06:29:23.254-0600: 245835.661: [GC (Allocation Failure)
2018-03-04T06:29:23.254-0600: 245835.661: [ParNew: 68162K->2K(76672K), 0.2545379
secs] 87514K->19354K(515776K), 0.2550031 secs] [Times: user=0.13 sys=0.08, real=0.26 secs]
Heap after GC invocations=53 (full 1):  par new generation   total 76672K, used 2K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=53 (full 1):
 par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
```

0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T08:20:05.350-0600: 252480.382: [GC (Allocation Failure)
2018-03-04T08:20:05.350-0600: 252480.383: [ParNew: 68162K->2K(76672K), 0.1996202 secs] 87514K->19354K(515776K), 0.2001894 secs] [Times: user=0.15 sys=0.05, real=0.20 secs]
Heap after GC invocations=54 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=54 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T10:10:35.548-0600: 259113.263: [GC (Allocation Failure)
2018-03-04T10:10:35.549-0600: 259113.263: [ParNew: 68162K->2K(76672K), 0.2012052 secs] 87514K->19354K(515776K), 0.2016894 secs] [Times: user=0.16 sys=0.04, real=0.20 secs]
Heap after GC invocations=55 (full 1):
 par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,

0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=55 (full 1): par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T12:01:20.997-0600: 265761.391: [GC (Allocation Failure)
2018-03-04T12:01:20.998-0600: 265761.392: [ParNew: 68162K->2K(76672K), 0.2018385
secs] 87514K->19354K(515776K), 0.2023533 secs] [Times: user=0.14 sys=0.06, real=0.21 secs]
Heap after GC invocations=56 (full 1): par new generation   total 76672K, used 2K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=56 (full 1): par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)

```
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T13:52:03.121-0600: 272406.220: [GC (Allocation Failure)
2018-03-04T13:52:03.122-0600: 272406.220: [ParNew: 68162K->2K(76672K), 0.1906724
secs] 87514K->19354K(515776K), 0.1910330 secs] [Times: user=0.15 sys=0.04, real=0.19 secs]
Heap after GC invocations=57 (full 1):  par new generation   total 76672K, used 2K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23990K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=57 (full 1):  par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T15:42:56.621-0600: 279061.076: [GC (Allocation Failure)
```

2018-03-04T15:42:56.621-0600: 279061.076: [ParNew: 68162K->2K(76672K), 0.1916484 secs] 87514K->19354K(515776K), 0.1922134 secs] [Times: user=0.15 sys=0.04, real=0.19 secs]
Heap after GC invocations=58 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=58 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T17:33:38.764-0600: 285705.969: [GC (Allocation Failure)
2018-03-04T17:33:38.764-0600: 285705.969: [ParNew: 68162K->2K(76672K), 0.1980523 secs] 87514K->19354K(515776K), 0.1990939 secs] [Times: user=0.14 sys=0.05, real=0.20 secs]
Heap after GC invocations=59 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,

0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=59 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810, 0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000, 0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T19:24:28.756-0600: 292358.745: [GC (Allocation Failure)
2018-03-04T19:24:28.756-0600: 292358.745: [ParNew: 68162K->2K(76672K), 0.1970454 secs] 87514K->19354K(515776K), 0.1977691 secs] [Times: user=0.15 sys=0.04, real=0.20 secs]
Heap after GC invocations=60 (full 1): par new generation   total 76672K, used 2K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000, 0x00000000e0000000, 0x0000000100000000)
 Metaspace   used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class space   used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=60 (full 1): par new generation   total 76672K, used 68162K [0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000, 0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810, 0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000, 0x00000000c5330000)

```
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T21:14:59.321-0600: 298992.122: [GC (Allocation Failure)
2018-03-04T21:14:59.321-0600: 298992.122: [ParNew: 68162K->2K(76672K), 0.2248872
secs] 87514K->19354K(515776K), 0.2255714 secs] [Times: user=0.16 sys=0.06, real=0.22 secs]
Heap after GC invocations=61 (full 1):  par new generation   total 76672K, used 2K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=61 (full 1):  par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0810,
0x00000000c5330000)
  to   space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290000,
0x00000000c4ae0000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-04T23:05:30.551-0600: 305625.786: [GC (Allocation Failure)
2018-03-04T23:05:30.551-0600: 305625.786: [ParNew: 68162K->2K(76672K), 0.2020811
secs] 87514K->19354K(515776K), 0.2024862 secs] [Times: user=0.16 sys=0.05, real=0.20 secs]
Heap after GC invocations=62 (full 1):  par new generation   total 76672K, used 2K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K,   0% used [0x00000000c0000000, 0x00000000c0000000,
0x00000000c4290000)
```

```
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K }
{Heap before GC invocations=62 (full 1):  par new generation   total 76672K, used 68162K
[0x00000000c0000000,
0x00000000c5330000, 0x00000000c5330000)
  eden space 68160K, 100% used [0x00000000c0000000, 0x00000000c4290000,
0x00000000c4290000)
  from space 8512K,   0% used [0x00000000c4290000, 0x00000000c4290810,
0x00000000c4ae0000)
  to   space 8512K,   0% used [0x00000000c4ae0000, 0x00000000c4ae0000,
0x00000000c5330000)
 concurrent mark-sweep generation total 439104K, used 19352K [0x00000000c5330000,
0x00000000e0000000, 0x0000000100000000)
 Metaspace       used 23991K, capacity 24528K, committed 24704K, reserved 1071104K   class
space    used 2060K, capacity 2154K, committed 2176K, reserved 1048576K
2018-03-05T00:56:03.553-0600: 312260.512: [GC (Allocation Failure)
2018-03-05T00:56:03.553-0600: 312260.513: [ParNew: 68162K->2K(76672K), 0.3505282
secs] 87514K->19354K(515776K), 0.3509026 secs] [Times: user=0.29 sys=0.06, real=0.36 secs]
```