

Tweaking Jira's Garbage Collection flags for use with GCViewer

The takeaway from this article is that, if you run Jira Server, there is a config tweak you really ought to make, to allow the use of [GCViewer](#) to debug any future garbage collection problems:

For Jira, edit `/opt/atlassian/jira/bin/set-gc-params.sh` and add make this change:

```
diff --git a/bin/set-gc-params.sh b/bin/set-gc-params.sh
--- a/bin/set-gc-params.sh
+++ b/bin/set-gc-params.sh
@@ -6,7 +6,7 @@
 then
     # In Java 9, GC logging has been re-implemented using the Unified GC logging framework.
     # See http://openjdk.java.net/jeps/158 or https://docs.oracle.com/javase/10/jrockit-hotspot/logging.htm
-    GC_JVM_PARAMETERS="-Xlog:gc*:file=$LOGBASEABS/logs/atlassian-jira-gc-%t.log:time,uptime:filecount=5,
+    GC_JVM_PARAMETERS="-Xlog:gc*:file=$LOGBASEABS/logs/atlassian-jira-gc-%t.log:time,uptime,tags,level:
filecount=5,filesize=20M ${GC_JVM_PARAMETERS}"
+    GC_JVM_PARAMETERS="-Xlog:gc*:file=$LOGBASEABS/logs/atlassian-jira-gc-%t.log:time,uptime,tags,level:
filecount=5,filesize=20M ${GC_JVM_PARAMETERS}"
     GC_JVM_PARAMETERS="${GC_JVM_PARAMETERS} ${JVM_GC_ARGS}"
 else
     # Set the JVM arguments used to start Jira. For a description of the options, see
```

What's this all about?

Jira is written in Java, which is a "garbage-collected" language. This means that Java programmers don't need to worry about de-allocating memory; the Java virtual machine (JVM) periodically "garbage collects" any memory that is unused.

Garbage collection (GC) happens all the time in a running JVM, and typically takes only milliseconds. However if your JVM is under memory pressure, a full GC can't actually free up that much memory, and soon after another GC is needed, and another.. and you get a situation where the JVM is spending more time GC'ing than doing anything useful. This is called **GC thrashing**.

If you're lucky you'll see an `OutOfMemoryError` in your `/opt/atlassian/jira/logs/catalina.out` log file, making the diagnosis easy. But sometimes Jira will just be really slow, high CPU usage, and thread dumps showing a variety of threads doing normal CPU-bound things. but not quite enough to trigger an `OutOfMemoryError` on garbage collector threads trying to clear up memory.

The best way to diagnose GC thrashing, or any "what the hell is Jira so busy with?" situation, is to use `top` (or `htop` or `atop`) with threads enabled, so you can see which Java thread(s) are the CPU consumers. I like `atop` for its ability to rewind and view past states. Here is `atop` after pressing 'y' to enable threads:

PRC	sys	2m16s	user	1h51m	#proc	67367	#tslpi	1692	#tslpu	0	#zombie	0	#exit	>66873			
CPU	sys	25%	user	1122%	irq	1%	idle	390%	wait	61%	curf	3.60GHz	curscal	2%			
CPL	avg1	16.12	avg5	14.24	avg15	13.50	csw	15111234	intr	9917839			numcpu	16			
MEM	tot	30.6G	free	520.0M	cache	6.3G	buff	28.6M	slab	3.1G	vmbal	0.0M	hptot	0.0M			
SWP	tot	10.0G	free	7.9G							vmcom	27.2G	vmlim	25.3G			
PAG	scan	123088	steal	72739	stall	0					swin	142	swout	2734			
NET	transport		tcpi	911467	tcpo	950150	udpi	26776	udpo	26776	tcpao	1033	tcpo	5831			
NET	network		ipi	939502	ipo	930570	ipfrw	0	deliv	939502	icmpi	1258	icmpo	1			
NET	lo	----	pcki	786176	pcko	786176	sp	0 Mbps	si	7473 Kbps	so	7473 Kbps	erro	0			
NET	ens5	----	pcki	158716	pcko	193202	sp	0 Mbps	si	862 Kbps	so	2143 Kbps	erro	0			
Only 66873 exited processes handled -- 125664 skipped!																	
PID	TID	CID	SYSCPU	USRCPU	VGROW	RGROW	RDDSK	WRDSK	RUID	ST	EXC	THR	S	CPUNR	CPU	CMD	1/1677
23586	-	host-----	33.01s	1h48m	82476K	-55.4M	918.1M	2.4G	jira	--	-	432	S	12	1092%	java	
23586	23586	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	12	0%	java	
23586	23702	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	1	0%	java	
23586	23709	host-----	1.38s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	9	79%	ParGC Thread#0	
23586	23720	host-----	0.88s	93.58s	82476K	-55.4M	104K	322K	jira	--	-	1	S	3	16%	VM Thread	
23586	23724	host-----	0.02s	0.12s	82476K	-55.4M	0K	0K	jira	--	-	1	S	11	0%	Reference Hand	
23586	23725	host-----	0.01s	0.13s	82476K	-55.4M	0K	0K	jira	--	-	1	S	3	0%	Finalizer	
23586	23734	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	14	0%	Signal Dispatc	
23586	23735	host-----	0.03s	2.66s	82476K	-55.4M	4K	0K	jira	--	-	1	S	5	0%	C2 CompilerThr	
23586	23736	host-----	0.01s	0.13s	82476K	-55.4M	0K	4K	jira	--	-	1	S	5	0%	C1 CompilerThr	
23586	23737	host-----	0.03s	2.15s	82476K	-55.4M	0K	0K	jira	--	-	1	S	14	0%	Sweeper thread	
23586	23764	host-----	0.00s	0.01s	82476K	-55.4M	0K	0K	jira	--	-	1	S	0	0%	Common-Cleaner	
23586	23801	host-----	0.08s	0.46s	82476K	-55.4M	212K	40243K	jira	--	-	1	S	15	0%	JFR Recorder T	
23586	23881	host-----	0.10s	0.06s	82476K	-55.4M	0K	20K	jira	--	-	1	S	0	0%	JFR Periodic T	
23586	23886	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	12	0%	JFR Recording	
23586	23907	host-----	0.18s	0.54s	82476K	-55.4M	0K	0K	jira	--	-	1	S	9	0%	java	
23586	23908	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	1	0%	Service Thread	
23586	23909	host-----	0.14s	0.08s	82476K	-55.4M	0K	152K	jira	--	-	1	S	14	0%	VM Periodic Ta	
23586	23920	host-----	0.01s	0.01s	82476K	-55.4M	28K	124K	jira	--	-	1	S	9	0%	AsyncFileHandl	
23586	24107	host-----	0.01s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	7	0%	Abandoned conn	
23586	24147	host-----	0.35s	0.35s	82476K	-55.4M	0K	0K	jira	--	-	1	S	5	0%	NioBlockingSel	
23586	24149	host-----	1.20s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	12	79%	ParGC Thread#1	
23586	24150	host-----	1.14s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	15	79%	ParGC Thread#2	
23586	24151	host-----	1.20s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	1	79%	ParGC Thread#3	
23586	24152	host-----	1.24s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	10	79%	ParGC Thread#4	
23586	24153	host-----	1.03s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	8	79%	ParGC Thread#5	
23586	24154	host-----	1.11s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	2	79%	ParGC Thread#6	
23586	24155	host-----	1.14s	7m53s	82476K	-55.4M	4K	0K	jira	--	-	1	R	0	79%	ParGC Thread#7	
23586	24469	host-----	1.25s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	6	79%	ParGC Thread#8	
23586	24470	host-----	1.08s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	4	79%	ParGC Thread#9	
23586	24471	host-----	1.07s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	7	79%	ParGC Thread#1	
23586	24818	host-----	1.27s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	11	79%	ParGC Thread#1	
23586	24819	host-----	1.14s	7m53s	82476K	-55.4M	0K	0K	jira	--	-	1	R	3	79%	ParGC Thread#1	
23586	25829	host-----	0.05s	0.20s	82476K	-55.4M	332K	3174K	jira	--	-	1	S	5	0%	ContainerBackg	
23586	25831	host-----	0.00s	0.01s	82476K	-55.4M	0K	0K	jira	--	-	1	S	12	0%	NioBlockingSel	
23586	25833	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	6	0%	http-nio-127.0	
23586	25834	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	14	0%	http-nio-127.0	
23586	25835	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	4	0%	http-nio-127.0	
23586	25836	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	7	0%	http-nio-127.0	
23586	25837	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	0	0%	http-nio-127.0	
23586	25839	host-----	0.00s	0.00s	82476K	-55.4M	0K	0K	jira	--	-	1	S	4	0%	http-nio-127.0	

See in the CPU column: Jira is consuming all CPU (1092%), and specifically the only threads active (79% each) are **ParGC Thread** threads.

GC logs

Out the box, Jira (8.7.1+) is configured with these GC configuration parameters:

```
# in bin/setenv.sh
JVM_GC_ARGS="-XX:+ExplicitGCInvokesConcurrent"
# in bin/set-gc-params.sh
GC_JVM_PARAMETERS="-Xlog:gc*:file=$LOGBASEABS/logs/atlassian-jira-gc-%t.log:time,uptime:filecount=5,
filesize=20M $ {GC_JVM_PARAMETERS}"
GC_JVM_PARAMETERS="$ {GC_JVM_PARAMETERS} $ {JVM_GC_ARGS}"
```

If you are using Java 9 or above, this means the **G1GC** algorithm is in use (see [Disruptive Changes to GC Logging in Java 9](#)). Earlier versions of Jira used the older Parallel GC (see [JRASERVER-64331](#) - Ship Jira with G1GC enabled by default GATHERING INTEREST), and look like this:

```
GC_JVM_PARAMETERS="-Xlog:gc*:file=$LOGBASEABS/logs/atlassian-jira-gc-%t.log:time,uptime:filecount=5,
filesize=20M ${GC_JVM_PARAMETERS}"
GC_JVM_PARAMETERS="-XX:+UseParallelGC ${GC_JVM_PARAMETERS}"
```

Either way, you should have a log file, /opt/atlassian/jira/logs/atlassian-jira-log-gc-*.log. GC logs looking something like this:

```
[2020-09-22T03:34:05.289-0700][2465737.513s] GC(355175) Pause Full (Ergonomics) 5020M->4521M(5587M) 3608.541ms
[2020-09-22T03:34:05.289-0700][2465737.513s] GC(355175) User=41.39s Sys=0.00s Real=3.60s
[2020-09-22T03:34:05.524-0700][2465737.748s] GC(355176) Pause Full (Ergonomics)
[2020-09-22T03:34:05.524-0700][2465737.748s] GC(355176) Marking Phase
[2020-09-22T03:34:06.432-0700][2465738.656s] GC(355176) Marking Phase 908.653ms
[2020-09-22T03:34:06.432-0700][2465738.656s] GC(355176) Summary Phase
[2020-09-22T03:34:06.432-0700][2465738.657s] GC(355176) Summary Phase 0.074ms
[2020-09-22T03:34:06.432-0700][2465738.657s] GC(355176) Adjust Roots
[2020-09-22T03:34:06.592-0700][2465738.816s] GC(355176) Adjust Roots 159.736ms
[2020-09-22T03:34:06.592-0700][2465738.816s] GC(355176) Compaction Phase
[2020-09-22T03:34:09.213-0700][2465741.437s] GC(355176) Compaction Phase 2620.355ms
[2020-09-22T03:34:09.213-0700][2465741.437s] GC(355176) Post Compact
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) Post Compact 25.488ms
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) PSYoungGen: 957440K->451409K(1527296K)
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) ParOldGen: 4194303K->4194162K(4194304K)
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) Metaspace: 759906K->759906K(1824768K)
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) Pause Full (Ergonomics) 5030M->4536M(5587M) 3714.746ms
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) User=42.65s Sys=0.10s Real=3.71s
[2020-09-22T03:34:09.387-0700][2465741.611s] GC(355177) Pause Full (Ergonomics)
[2020-09-22T03:34:09.387-0700][2465741.611s] GC(355177) Marking Phase
[2020-09-22T03:34:10.306-0700][2465742.531s] GC(355177) Marking Phase 919.544ms
[2020-09-22T03:34:10.306-0700][2465742.531s] GC(355177) Summary Phase
[2020-09-22T03:34:10.307-0700][2465742.531s] GC(355177) Summary Phase 0.078ms
[2020-09-22T03:34:10.307-0700][2465742.531s] GC(355177) Adjust Roots
[2020-09-22T03:34:10.470-0700][2465742.694s] GC(355177) Adjust Roots 163.227ms
[2020-09-22T03:34:10.470-0700][2465742.694s] GC(355177) Compaction Phase
[2020-09-22T03:34:13.473-0700][2465745.698s] GC(355177) Compaction Phase 3003.617ms
[2020-09-22T03:34:13.474-0700][2465745.698s] GC(355177) Post Compact
[2020-09-22T03:34:13.499-0700][2465745.723s] GC(355177) Post Compact 25.643ms
[2020-09-22T03:34:13.499-0700][2465745.723s] GC(355177) PSYoungGen: 957440K->512920K(1527296K)
[2020-09-22T03:34:13.499-0700][2465745.723s] GC(355177) ParOldGen: 4194162K->4194126K(4194304K)
[2020-09-22T03:34:13.499-0700][2465745.723s] GC(355177) Metaspace: 759906K->759906K(1824768K)
[2020-09-22T03:34:13.499-0700][2465745.724s] GC(355177) Pause Full (Ergonomics) 5030M->4596M(5587M) 4112.550ms
[2020-09-22T03:34:13.499-0700][2465745.724s] GC(355177) User=47.24s Sys=0.00s Real=4.11s
[2020-09-22T03:34:13.503-0700][2465745.727s] GC(355178) Pause Full (Ergonomics)
[2020-09-22T03:34:13.503-0700][2465745.727s] GC(355178) Marking Phase
[2020-09-22T03:34:14.417-0700][2465746.641s] GC(355178) Marking Phase 914.147ms
[2020-09-22T03:34:14.417-0700][2465746.641s] GC(355178) Summary Phase
[2020-09-22T03:34:14.417-0700][2465746.641s] GC(355178) Summary Phase 0.066ms
[2020-09-22T03:34:14.417-0700][2465746.641s] GC(355178) Adjust Roots
[2020-09-22T03:34:14.555-0700][2465746.779s] GC(355178) Adjust Roots 138.314ms
[2020-09-22T03:34:14.555-0700][2465746.779s] GC(355178) Compaction Phase
[2020-09-22T03:34:17.136-0700][2465749.361s] GC(355178) Compaction Phase 2581.276ms
[2020-09-22T03:34:17.137-0700][2465749.361s] GC(355178) Post Compact
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) Post Compact 24.319ms
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) PSYoungGen: 513664K->513419K(1527296K)
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) ParOldGen: 4194126K->4194120K(4194304K)
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) Metaspace: 759906K->759906K(1824768K)
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) Pause Full (Ergonomics) 4597M->4597M(5587M) 3658.596ms
[2020-09-22T03:34:17.161-0700][2465749.385s] GC(355178) User=42.50s Sys=0.01s Real=3.66s
[2020-09-22T03:34:17.318-0700][2465749.542s] GC(355179) Pause Full (Ergonomics)
[2020-09-22T03:34:17.318-0700][2465749.542s] GC(355179) Marking Phase
[2020-09-22T03:34:18.222-0700][2465750.446s] GC(355179) Marking Phase 903.470ms
[2020-09-22T03:34:18.222-0700][2465750.446s] GC(355179) Summary Phase
[2020-09-22T03:34:18.222-0700][2465750.446s] GC(355179) Summary Phase 0.074ms
[2020-09-22T03:34:18.222-0700][2465750.446s] GC(355179) Adjust Roots
```

The sample log above is from a thrashing server using the older ParallelGC algorithm. The lines to note are marked **Pause Full**. e.g.:

```
[2020-09-22T03:34:09.238-0700][2465741.462s] GC(355176) Pause Full (Ergonomics) 5030M->4536M(5587M) 3714.746ms
.....
[2020-09-22T03:34:13.499-0700][2465745.724s] GC(355177) Pause Full (Ergonomics) 5030M->4596M(5587M) 4112.550ms
[2020-09-22T03:34:13.499-0700][2465745.724s] GC(355177) User=47.24s Sys=0.00s Real=4.11s
```

By subtracting the timestamps we see a 4.26s interval between full GCs (2465745.724 - 2465741.462), of which the GC occupied 4.11s. This is what thrashing looks like.

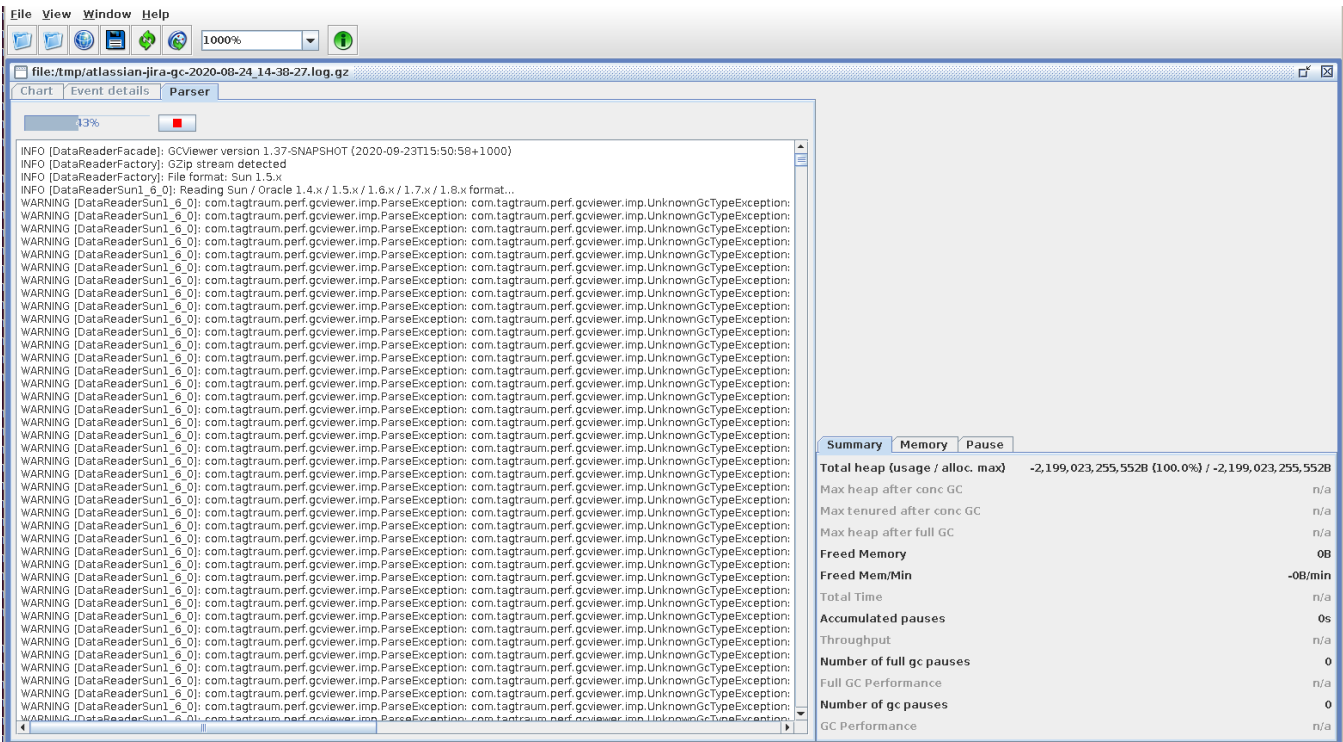
Interpreting GC log files is exceptionally tedious - finding relevant lines, doing timestamp math, etc. If you did want to further analyze this GC log file, what tools could you use?

GCViewer

My tool of choice is [GCViewer](#), because it has been around forever and does the job.

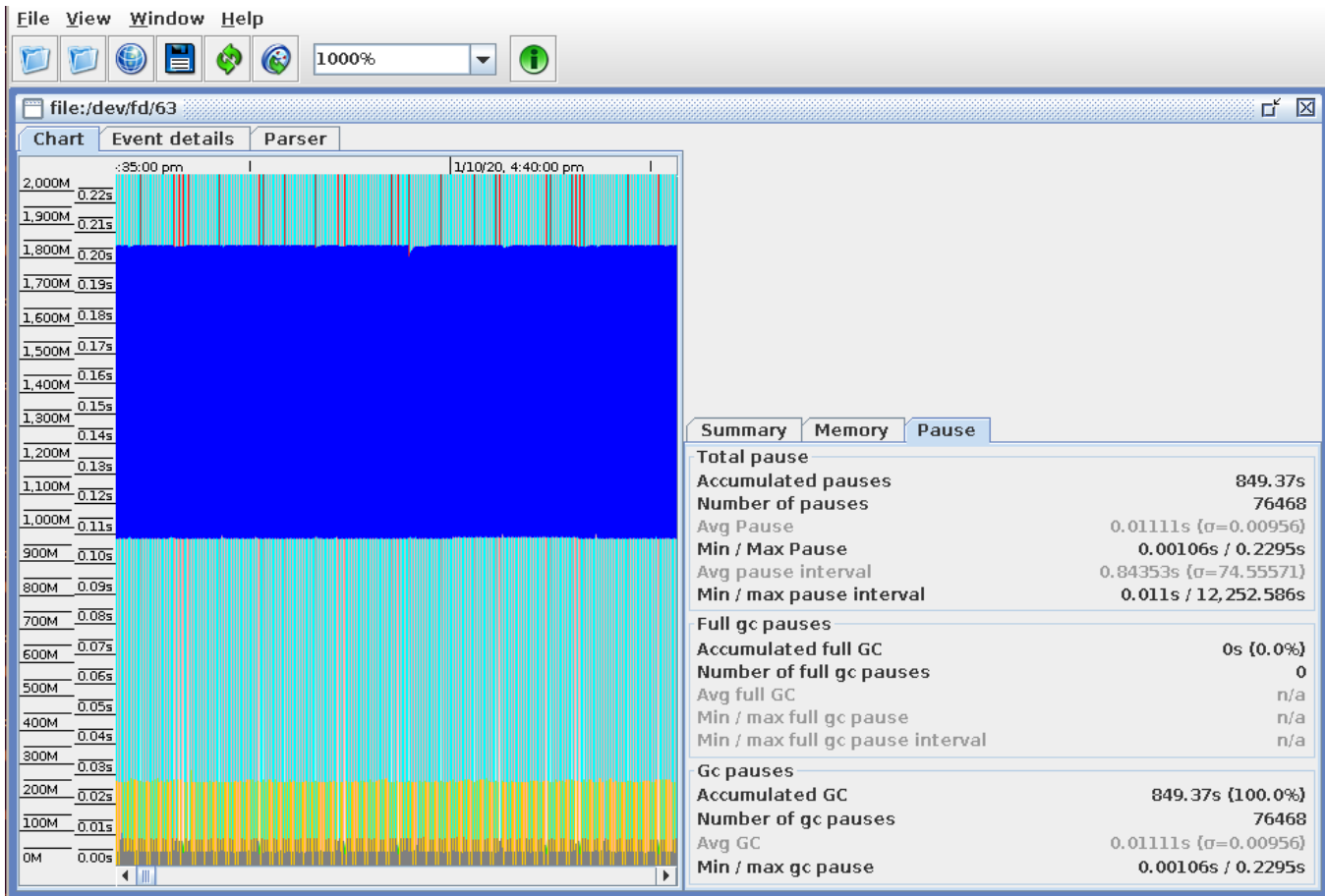
The problem alluded to in the introduction is that GCViewer cannot read the default format of GC logs generated by Jira and Confluence, when using Java 9+. You'll just get an error for each line parsed:

```
WARNING [DataReaderSun1_6_0]: com.tagtraum.perf.gcviewer.imp.ParseException: com.tagtraum.perf.gcviewer.imp.
UnknownGcTypeException: Unknown gc type: '-' Line 1: [2020-09-19T22:16:23.955-0700][2273876.179s] GC(326262)
ParOldGen: 2230694K->2232246K(2380800K)
```



After making the `-Xlog:gc` tweak suggested in the intro, and restarting Jira, GCViewer should be able to load the `/opt/atlassian/jira/logs/atlassian-jira-gc-*.log` files:

(example from a perfectly healthy server, sorry)



GCEasy.io

Discovered after reading these [GC tuning tips](#); <https://gceasy.io> is likely worth the \$10/mo. Unlike GCViewer, no `-Xlog:gc` tweaks are needed. The free tier is enough to give a general idea of GC activity:

