ShawnHeisey

Shawn Heisey

Java 8 recommendation for Solr

Java 8 is *required* for Solr 6.x. For 4.x and 5.x releases, I strongly recommend it. Earlier releases only required Java 5 and have not been extensively tested with 8.

There have been some very significant memory management improvements in each release of Java 8, particularly with garbage collection, both in general and specifically the G1 collector. OpenJDK 8 should be almost as good as Oracle Java 8.

It's not a good idea to use a JVM from IBM. IBM aggressively enables a large number of performance optimizations by default, which helps a lot with performance, but some of those optimizations cause Solr/Lucene to encounter bugs.

OpenJDK should be fine, as long as it's 7 or later and meets the requirements of your Solr version. The latest release is recommended. OpenJDK 6 is known to have bugs. If you can get an Oracle JVM, you should.

Solr 7 is tested with Java 9 and should work. Earlier versions may encounter problems.

GC Tuning for Solr

⚠️ Starting in version 4.10, and greatly improved in 5.0, startup scripts were added that do GC tuning for you, using CMS (ConcurrentMarkSweep) settings very similar to the CMS settings that you can find further down on this page. If you are using the bin/solr or bin\solr script to start Solr, you already have GC tuning and probably won't need to worry about the recommendations here.

The secret to good Solr GC tuning: Eliminating full garbage collections. A full garbage collection is slow, no matter what collector you're using. If you can set up the options for GC such that both young and old generations are always kept below max size by their generation-specific collection algorithms, performance is almost guaranteed to be good.

Why is tuning necessary?

First you must understand how Java's memory model works, and what "garbage collection" actually means. I will not attempt to describe GC here ... click the link in the previous sentence or search the Internet if you want to know more.

The default garbage collector in Java 8 and below is designed to do a decent job when the major concern of your program is throughput – how fast it can transfer data. If a high-throughput program pauses for several seconds, that's no big deal, as long as it is pumping data quickly the rest of the time.

Good Solr performance requires optimizing for a different metric – latency. The jobs that Solr is doing must complete very quickly, because when a user is searching, they are expecting results very quickly ... usually less than a second or two. If your website search feature sometimes takes several seconds to respond, users will find one of your competitors instead of using your website.

The old default collector has all the wrong optimizations for Solr. In some ways that collector actually *prefers* full garbage collections. This preference helps throughput, because it means there are no collections at all for a very long time period, then the collector will spend a chunk of time (often tens of seconds) reclaiming a VERY large block of memory all at once. When that happens to a Solr install, any query that is underway will pause for many seconds. Java has other collector implementations (primarily CMS and G1) that are designed for latency rather than throughput, but even these collectors will experience long pauses with Solr if they are left at their default settings. Further tuning is required.

The default collector in Java 9 is G1, but as just mentioned, if its settings are left at default, there will still be long pauses, though they may be less frequent.

G1 (Garbage First) Collector

⚠️ It's important to point out a warning from the Lucene project about G1. They say "Do not, under any circumstances, run Lucene with the G1 garbage collector." Solr is a Lucene program. I personally have never had any problems at all with the G1 collector, but if you're going to use it, it's important that you know you're going against advice from smart people. **If this worries you at all, use the CMS settings instead.**

With the newest Oracle Java versions, G1GC is looking **extremely** good. Do not try these options unless you're willing to run the latest Java 7 or Java 8, preferably from Oracle. Experiments with G1 on early Java 7 releases were very disappointing. My most recent testing has been with Oracle 7u72, and I have been informed on multiple occasions that Oracle 8u40 and later has much better G1 performance.

I typically will use a Java heap of 6GB or 7GB. The following settings were created as a result of a discussion with Oracle employees on the hotspot-gc-use mailing list:

```
JVM_OPTS=" \
-XX:+UseG1GC \
-XX:+ParallelRefProcEnabled \
-XX:G1HeapRegionSize=8m \
-XX:MaxGCPauseMillis=200 \
-XX:+UseLargePages \
-XX:+AggressiveOpts \
"
```

Here's a graph of the GC times with the settings above. The heap size for this graph is -Xms4096M -Xmx6144M. All of the GC pauses are well under a second, and the vast majority of them are under a quarter second. The graph will be updated when more information is available:

gc-7u72-with-8m-region-200ms-45percentoccupancy.png]]

⚠️ Some notes about the !G1HeapRegionSize parameter: I was seeing a large number of "humongous allocations" in the GC log, which means that they are larger than 50 percent of a G1 heap region. Those allocations were about 2MB in size, and my 4GB minimum heap results in a default region size of 2MB, which is calculated by dividing the -Xms value by 2048. I suspect that those allocations were for filterCache entries, because the indexes on that instance are each about 16.5 million documents, and the bitset memory structure of a filter for an index that size is a little over 2MB. Setting the region size to 8MB ensures that those allocations are not categorized as humongous. See "Humongous Allocations" section on this blog entry for more information.

Garbage collection for allocations that are tagged as humongous is not fully handled in the concurrent and low-pause parts of the process. They can only be properly handled by a full garbage collection, which will always be slow. Depending on the size of your indexes and the size of your heap, the region size may require adjustment from the 8MB that I am using above. If you have an index with about 100 million documents in it, you'll want to use a region size of 32MB, which is the maximum possible size. Because of this limitation of the G1 collector, we recommend always keeping a Solr index below a maxDoc value of around 100 to 120 million.

Further down on this page, I have included GC logging options that will create a log you can search for "humongous" to determine if there are a lot of these allocations. I have been told that with Java 8u40, the !G1HeapRegionSize parameter is not required, because Java 8 manages large allocations a lot better than Java 7 does.

Current experiments

This is what I am currently trying out on my servers:

```
JVM_OPTS=" \
-XX:+UseG1GC \
-XX:+PerfDisableSharedMem \
-XX:+ParallelRefProcEnabled \
-XX:G1HeapRegionSize=8m \
-XX:MaxGCPauseMillis=250 \
-XX:InitiatingHeapOccupancyPercent=75 \
-XX:+UseLargePages \
-XX:+AggressiveOpts \
"
```

This is a graph of a session with the settings above on Java 7u72. The heap size for this graph is -Xms4096M -Xmx6144M. I am quite thrilled about this. You may notice that the graph shows a clearly different GC pattern starting at about 8 hours ... this is because a full index rebuild started at just before 8 hours and ran for about 8 hours. The bulk of the longer pauses occurred during the full reindex, but they are not numerous, and there are millions of documents being indexed at the time. Before indexing started, there were no long pauses, and after it stopped, there have only been a handful. I would like to fix all of the long pauses, but I can live with the results I am seeing, especially because the average GC pause time is only six hundredths of a second.

gc-7u72-2015-03-27-001.png]]

The PerfDisableSharedMem option is there because of something that is called the four month bug.

CMS (ConcurrentMarkSweep) Collector

These CMS (ConcurrentMarkSweep) settings produce very good performance with Java 7, but they have been replaced by G1 settings on my systems:

```
JVM_OPTS=" \
-XX:NewRatio=3 \
-XX:SurvivorRatio=4 \
-XX:TargetSurvivorRatio=90 \
-XX:MaxTenuringThreshold=8 \
-XX:+UseConcMarkSweepGC \
-XX:+CMSScavengeBeforeRemark \
-XX:PretenureSizeThreshold=64m \
-XX:CMSFullGCsBeforeCompaction=1 \
-XX:+UseCMSInitiatingOccupancyOnly \
-XX:CMSInitiatingOccupancyFraction=70 \
-XX:CMSTriggerPermRatio=80 \
-XX:CMSMaxAbortablePrecleanTime=6000 \
-XX:+CMSParallelRemarkEnabled
-XX:+ParallelRefProcEnabled
-XX:+UseLargePages \
-XX:+AggressiveOpts \
"
```

Note to self: The -XX:ParGCCardsPerStrideChunk option was mentioned to me on the #solr IRC channel. With values of 4096 and 32768, the IRC user was able to achieve 15% and 19% reductions in average pause time, respectively, with the maximum pause cut nearly in half. That option was added to the options you can find in the Solr 5.x start script, on a 12GB heap.

The UseLargePages Option

A note about the UseLargePages option: Currently I do not have huge pages allocated in my operating system. This option will not actually do anything unless you allocate memory to huge pages. If you do so, memory usage reporting with the "top" command will probably only show a few hundred MB of resident memory used by your Solr process, even if it is in fact using several gigabytes of heap. If you do enable huge pages in Linux, be aware that you might wish to turn off an operating system feature called transparent huge pages.

Garbage Collection Logging

Here are the GC logging options that I use. If you've decided to use the G1 options above and would like to know if you're having a lot of humongous allocations, the logfile created by these options will tell you. A few humongous allocations are perfectly normal ... but there should not be very many of them. You can also use the logfile to create a graph of your garbage collection activity to compare different GC tuning options. Be sure that you use a valid logfile path on the -Xloggc parameter:

```
GCLOG_OPTS=" \
-verbose:gc \
-Xloggc:logs/gc.log \
-XX:+PrintGCDateStamps \
-XX:+PrintGCDetails \
-XX:+PrintAdaptiveSizePolicy \
-XX:+PrintReferenceGC
"
```

Redhat/CentOS Init script

This init script is tailored to my custom installation described at the end of this page. It works flawlessly on Redhat/CentOS 6. On Redhat/CentOS 7, this script will work if it does not live in /etc/init.d, but once it is placed there (and called as part of init, either at boot or with the 'service' command), it no longer works. If anyone knows why and can fix it so it works on both versions, please do.

```
#!/bin/bash
#
# chkconfig: - 80 45
# description: Starts and stops Solr

# Source redhat init.d function library.
. /etc/rc.d/init.d/functions

# options that are commonly overridden
SOLRROOT=/opt/solr4
SOLRHOME=/index/solr4
LISTENPORT=8983
JMINMEM=256M
JMAXMEM=2048M
```

```
JBIN=/usr/bin/java
FUSER=/sbin/fuser
STOPKEY=mystopkey
STOPPORT=8078
JMXPORT=8686
STARTUSER=solr

# less commonly overridden options
PROC_CHECK_ARG="start\.jar"
LOG_OPTS="-Dlog4j.configuration=file:etc/log4j.properties"
JVM_OPTS="-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:NewRatio=3 -XX:
MaxTenuringThreshold=8 -XX:+CMSParallelRemarkEnabled -XX:+ParallelRefProcEnabled -XX:+UseLargePages -XX:
+AggressiveOpts"
GCLOG_OPTS="-verbose:gc -Xloggc:logs/gc.log -XX:+PrintGCDateStamps -XX:+PrintGCDetails"
JMX_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=${JMXPORT} -Dcom.sun.management.
jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"
SYSCONFIG=/etc/sysconfig/solr4

# Source sysconfig for this app. This is used to override options.
if [ -f ${SYSCONFIG} ];
then
  . ${SYSCONFIG}
fi

# Build required env variables.
STARTARGS="-Xms${JMINMEM} -Xmx${JMAXMEM} ${LOG_OPTS} ${JVM_OPTS} ${GCLOG_OPTS} ${JMX_OPTS} -Dsolr.solr.
home=${SOLRHOME} -Djetty.port=${LISTENPORT} -DSTOP.PORT=${STOPPORT} -DSTOP.KEY=${STOPKEY}"
STOPARGS="-DSTOP.PORT=${STOPPORT} -DSTOP.KEY=${STOPKEY}"
PIDFILE=${SOLRROOT}/run/jetty.pid

# Grab UID of start user.  Abort loudly if it doesn't exist.
STARTUID=`id -u ${STARTUSER} 2> /dev/null`
RET=$?
if [ ${RET} -ne 0 ];
then
        echo "User ${STARTUSER} does not exist."
        exit 1
fi

# If we're root, restart as unpriveleged user. Abort if anyone else.
if [ ${UID} -eq 0 ];
then
  exec su ${STARTUSER} -c "$0 $*"
elif [ ${UID} -eq ${STARTUID} ];
then
  echo "do nothing" > /dev/null 2> /dev/null
else
  echo "This needs to be run as root or ${STARTUSER}. "
  exit 1
fi

runexit() {
  echo "Already running!"
  exit 1
}

start() {
  echo -n "Starting Solr... "
  if [ "x${PID}" == "x" ];
  then
    PID=0
  fi
  # Let's check to see if the PID is up and is actually Solr
  checkproccmdline > /dev/null 2> /dev/null
  if [ ${RET} -eq 0 ];
  then
    # If PID is up and is Solr, fail.
    runexit
  fi
  # Pre-emptive extreme prejudice strike on anything using Solr's port
  killbyport
  sleep 1
  # Start 'er up.
  cd ${SOLRROOT}
  ${JBIN} ${STARTARGS} -jar ${SOLRROOT}/start.jar >logs/out 2>logs/err &
  PID=$!
  echo ${PID} > ${PIDFILE}
  disown ${PID}
  echo "done"
}

checkproccmdline() {
  grep ${PROC_CHECK_ARG} /proc/${PID}/cmdline > /dev/null 2> /dev/null
  RET=$?
}

termbypid() {
  checkpid ${PID}
  RET=$?
  if [ ${RET} -eq 0 ];
  then
    checkproccmdline
    if [ ${RET} -eq 0 ];
    then
      kill $PID
    else
      echo
      echo "pid ${PID} is not Solr!"
      echo > ${PIDFILE}
    fi
  fi
}

killbypid() {
  checkpid ${PID}
  RET=$?
  if [ ${RET} -eq 0 ];
  then
    checkproccmdline
    if [ ${RET} -eq 0 ];
    then
      kill -9 $PID
    else
      echo
      echo "pid ${PID} is not Solr!"
      echo > ${PIDFILE}
    fi
  fi
}

termbyport() {
  ${FUSER} -sk -n tcp ${LISTENPORT}
}

killbyport() {
  ${FUSER} -sk9 -n tcp ${LISTENPORT}
}

checkpidsleep() {
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
}

stop() {
  echo -n "Stopping Solr... "
  # check to see if PID is empty.
  if [ "x${PID}" == "x" ];
  then
    # term then kill anything using Solr's port, assume success.
    termbyport
    sleep 5
    killbyport
  else
    # Try Jetty's stop mechanism.
    ${JBIN} $STOPARGS -jar /${SOLRROOT}/start.jar --stop \
      > /dev/null 2> /dev/null
    checkpidsleep
    # Controlled death by PID.
    termbypid
    checkpidsleep
    # Controlled death by Solr port.
    termbyport
    checkpidsleep
    # Sudden death by PID and Solr port.
    killbyport
    killbypid
    checkpidsleep
    # report failure if none of the above worked.
    if checkpid $PID 2>&1;
    then
      echo "failed!"
      exit 1
    fi
  fi
  echo "done"
}

UP=`cat /proc/uptime | cut -f1 -d\ | cut -f1 -d.`
# If the machine has been up less than 3 minutes, wipe the pidfile.
if [ ${UP} -lt 180 ];
then
  echo > ${PIDFILE}
fi

PID=`cat ${PIDFILE}`

case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  restart)
    stop
    start
    ;;
  *)
    echo $"Usage: $0 {start|stop|restart}"
    exit 1
esac

exit $?
```

Ubuntu Init Script

This may work on Debian as well, but I have not verified that. This is not as battle-tested as the Redhat script. See the end of this page for info about the install that this script will handle.

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          solr
# Required-Start:    $remote_fs $time
# Required-Stop:     umountnfs $time
# X-Stop-After:      sendsigs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts and stops Solr
# Description:       Solr is a search engine.
### END INIT INFO

#
# Author: Shawn Heisey
#

# Source LSB init function library.
. /lib/lsb/init-functions

# options that are commonly overridden
SOLRROOT=/opt/solr4
SOLRHOME=/index/solr4
LISTENPORT=8983
JMINMEM=256M
JMAXMEM=2048M
JBIN=/usr/bin/java
FUSER=/bin/fuser
STOPKEY=solrjunkies
STOPPORT=8078
JMXPORT=8686
STARTUSER=solr

# less commonly overridden options
PROC_CHECK_ARG="start\.jar"
LOG_OPTS="-Dlog4j.configuration=file:etc/log4j.properties"
JVM_OPTS="-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:NewRatio=3 -XX:
MaxTenuringThreshold=8 -XX:+CMSParallelRemarkEnabled -XX:+ParallelRefProcEnabled -XX:+UseLargePages -XX:
+AggressiveOpts"
GCLOG_OPTS="-verbose:gc -Xloggc:logs/gc.log -XX:+PrintGCDateStamps -XX:+PrintGCDetails"
JMX_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=${JMXPORT} -Dcom.sun.management.
jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"

# Source defaults for this app.
DEFAULTS=/etc/default/solr4
if [ -f ${DEFAULTS} ];
then
  . ${DEFAULTS}
fi

# Build required env variables.
STARTARGS="-Xms${JMINMEM} -Xmx${JMAXMEM} ${LOG_OPTS} ${JVM_OPTS} ${GCLOG_OPTS} ${JMX_OPTS} -Dsolr.allow.unsafe.
resourceloading=true -Dsolr.solr.home=${SOLRHOME} -Djetty.port=${LISTENPORT} -DSTOP.PORT=${STOPPORT} -DSTOP.
KEY=${STOPKEY}"
STARTARGS="-Xms${JMINMEM} -Xmx${JMAXMEM} ${LOG_OPTS} ${JVM_OPTS} ${GCLOG_OPTS} ${JMX_OPTS} -Dsolr.solr.
home=${SOLRHOME} -Djetty.port=${LISTENPORT} -DSTOP.PORT=${STOPPORT} -DSTOP.KEY=${STOPKEY}"
STOPARGS="-DSTOP.PORT=${STOPPORT} -DSTOP.KEY=${STOPKEY}"
PIDFILE=${SOLRROOT}/run/jetty.pid

# Grab UID of start user.  Abort loudly if it doesn't exist.
STARTUID=`id -u ${STARTUSER} 2> /dev/null`
RET=$?
if [ ${RET} -ne 0 ];
then
        echo "User ${STARTUSER} does not exist."
        exit 1
fi

# If we're root, restart as unpriveleged user. Abort if anyone else.
if [ ${UID} -eq 0 ];
then
  exec sudo -u ${STARTUSER} $0 $*
elif [ ${UID} -eq ${STARTUID} ];
then
  echo "do nothing" > /dev/null 2> /dev/null
else
```

```
  echo "This needs to be run as root or ${STARTUSER}. "
  exit 1
fi

# Check if $pid (could be plural) are running
checkpid() {
 local i

 for i in $* ; do
  [ -d "/proc/$i" ] && return 0
 done
 return 1
}

runexit() {
  echo "Already running!"
  exit 1
}

start() {
  echo -n "Starting Solr... "
  if [ "x${PID}" == "x" ];
  then
    PID=0
  fi
  # Let's check to see if the PID is up and is actually Solr
  checkproccmdline > /dev/null 2> /dev/null
  if [ ${RET} -eq 0 ];
  then
    # If PID is up and is Solr, fail.
    runexit
  fi
  # Pre-emptive extreme prejudice strike on anything using Solr's port
  killbyport
  sleep 1
  # Start 'er up.
  cd ${SOLRROOT}
  mkdir -p tmp
  STARTARGS="-Djava.io.tmpdir=tmp ${STARTARGS}"
  ${JBIN} ${STARTARGS} -jar ${SOLRROOT}/start.jar >logs/out 2>logs/err &
  PID=$!
  echo ${PID} > ${PIDFILE}
  disown ${PID}
  echo "done"
}

checkproccmdline() {
  grep ${PROC_CHECK_ARG} /proc/${PID}/cmdline > /dev/null 2> /dev/null
  RET=$?
}

termbypid() {
  checkpid ${PID}
  RET=$?
  if [ ${RET} -eq 0 ];
  then
    checkproccmdline
    if [ ${RET} -eq 0 ];
    then
      kill $PID
    else
      echo
      echo "pid ${PID} is not Solr!"
      echo > ${PIDFILE}
    fi
  fi
}

killbypid() {
  checkpid ${PID}
  RET=$?
  if [ ${RET} -eq 0 ];
  then
    checkproccmdline
    if [ ${RET} -eq 0 ];
    then
      kill -9 $PID
    else
      echo
      echo "pid ${PID} is not Solr!"
      echo > ${PIDFILE}
    fi
  fi
}

termbyport() {
  ${FUSER} -sk -n tcp ${LISTENPORT}
}

killbyport() {
  ${FUSER} -sk -9 -n tcp ${LISTENPORT}
}

checkpidsleep() {
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
  checkpid ${PID} && sleep 1
}

stop() {
  echo -n "Stopping Solr... "
  # check to see if PID is empty.
  if [ "x${PID}" == "x" ];
  then
    # term then kill anything using Solr's port, assume success.
    termbyport
    sleep 5
    killbyport
  else
    # Try Jetty's stop mechanism.
    ${JBIN} $STOPARGS -jar /${SOLRROOT}/start.jar --stop \
      > /dev/null 2> /dev/null
    checkpidsleep
    # Controlled death by PID.
    termbypid
    checkpidsleep
    # Controlled death by Solr port.
    termbyport
    checkpidsleep
    # Sudden death by PID and Solr port.
    killbyport
    killbypid
    checkpidsleep
    # report failure if none of the above worked.
    if checkpid $PID 2>&1;
    then
      echo "failed!"
      exit 1
    fi
  fi
  echo "done"
}

UP=`cat /proc/uptime | cut -f1 -d\ | cut -f1 -d.`
# If the machine has been up less than 3 minutes, wipe the pidfile.
```

```
if [ ${UP} -lt 180 ];
then
  echo > ${PIDFILE}
fi

PID=`cat ${PIDFILE}`

case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  restart)
    stop
    start
    ;;
  *)
    echo $"Usage: $0 {start|stop|restart}"
    exit 1
esac

exit $?
```

Install info

For my Solr install on Linux, I use /opt/solr4 as my installation path, and /index/solr4 as my solr home. The /index directory is a dedicated filesystem, /opt is part of the root filesystem.

From the example directory, I copied cloud-scripts, contexts, etc, lib, webapps, and start.jar over to /opt/solr4. My stuff was created before 4.3.0, so the resources directory didn't exist. I was already using log4j with a custom Solr build, and I put my log4j.properties file in etc instead. I created a logs directory and a run directory in /opt/solr4.

My directory structure in the solr home, (/index/solr4) is more complex than most. What a new user really needs to know is that solr.xml goes here and dictates the rest of the structure. I have a symlink at /index/solr4/lib, pointing to /opt/solr4/solrlib - so that jars placed in `${solr.solr.home}/lib` are actually located in the program directory, not the data directory. That makes for a much cleaner version control scenario - both directories are git repositories cloned from our internal git server.

Unlike the example configs, my solrconfig.xml files do not have <lib> directives for loading jars. That gets automatically handled by the jars living in that symlinked lib directory. See SOLR-4852 for caveats regarding central lib directories.

If you want to run SolrCloud, you would need to install zookeeper separately and put your zkHost parameter in solr.xml. Due to a bug, putting zkHost in solr.xml doesn't work properly until 4.4.0.

CategoryHomepage