Item 352
git_comments:

git_commits:

1. **summary:** [Feature] JVM parameter optimization , related issue #3370 (#3373)
   message: [Feature] JVM parameter optimization , related issue #3370 (#3373) * [Feature] JVM parameter optimization , related issue #3370 * [Feature] JVM parameter optimization , related issue #3370 Co-authored-by: qiaozhanwei <qiaozhanwei@analysys.com.cn>

github_issues:

1. **title:** [Feature] JVM parameter optimization
   body: **Describe the feature** JVM parameter optimization **Is your feature request related to a problem? Please describe.** 4G heap memory and default young generation size within 30s of starting, there will be one Minor GC and two Full GCs S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 **Describe the solution you'd like** A clear and concise description of what you want to happen. reasoning: If the old age is full, causing Full GC, then it is unlikely that the old age will be zero after Full GC, which is relatively small. Then there is the metadata area. Visually, MU uses a lot, so adjust the size of the metadata area. The metadata area mainly stores the description information and static variables of some classes. If there are more reflections, this area can be increased appropriately It is recommended to 256MB or 512M. Considering the user's machine memory, set 128M temporarily, which is smaller -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m After the optimization is started, there is no Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 Worker starts 20 processes, Linux memory takes up 1.530g, an average of about 78MB each, one process PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java Check the JVM memory at this time, and there is no change Example: -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running -- -- 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 14 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc -- -- 2G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 7 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc For the JDBC result set, there are 1000. According to the test of the task instance table, there are an average of 4 task instance result sets, each time in 30s. Each time young gc old age increases by 1M The new generation is about 30M per second Master JVM optimization 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running 3~4 process examples -- -- The new generation is about 40M per second Perform a minor gc once in 40s, and the old age will increase by about 1M after each minor gc Start a process every 1s, 20 tasks,

each task sleep 10s t_ds_process_instance 80 process instances The new generation has been around 500M per second Perform a minor gc once in 3s, and the old generation will increase by about 1M after each minor gc Api Server -- -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSize -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances Minor gc occurs once in about 30 minutes summary --- -- recommended that the heap memory of Master Server is 4g recommended that the heap memory of Worker Server is 2g recommended that the heap memory of Api Server is 1g recommended that the heap memory of Alert Server is 1g recommended that the heap memory of Logger Server is 1g --中文解释-- worker JVM优化 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 4g s0 108MB s1 108MB eden 865MB old 3014MB metaspace 41MB 启动完30s内，就会发生一次Minor GC和两次Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 推理： 如果要是老年代满了，引起Full GC，那老年代也不太可能Full GC之后，OU为零，可能性比较小 然后就是元数据区了，目测MU使用很多，所以调整一下元数据区域大小， 元数据区域主要存放的是一些类的描述信息和静态变量，如果反射比较多，这个区域可以适当的增大 建议256MB或者512M，考虑到用户机器内存，姑且设置128M，小一点 -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m 优化后启动之后，无Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 worker 启动20个进程，linux内存占用1.530g，平均每个78MB左右，一个进程 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java 此时查看JVM内存，并没有变化 示例：-- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 --- 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 14分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc --- 2G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 7分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc 对于JDBC结果集合，有1000条，根据任务实例表测试的，平均有4个任务实例结果集，30s一次yong gc 每次young gc老年代增加1M 新生代30M左右每秒 Master JVM优化 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -

Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 3~4个流程实例 --- 每秒新生代曾40M左右 40s进行一次minor gc，每次minor gc之后老年代增加1M左右 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 每秒新生代曾500M左右 3s进行一次minor gc，每次minor gc之后老年代增加1M左右 Api Server -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 30分钟左右发生一次minor gc 总结 -- 建议Master Server的堆内存是4g 建议Worker Server的堆内存是2g 建议Api Server的堆内存是1g 建议Alert Server的堆内存是1g 建议Logger Server的堆内存是1g
label: test

2. **title:** [Feature] JVM parameter optimization
body: **Describe the feature** JVM parameter optimization **Is your feature request related to a problem? Please describe.** 4G heap memory and default young generation size within 30s of starting, there will be one Minor GC and two Full GCs S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 **Describe the solution you'd like** A clear and concise description of what you want to happen. reasoning: If the old age is full, causing Full GC, then it is unlikely that the old age will be zero after Full GC, which is relatively small. Then there is the metadata area. Visually, MU uses a lot, so adjust the size of the metadata area. The metadata area mainly stores the description information and static variables of some classes. If there are more reflections, this area can be increased appropriately It is recommended to 256MB or 512M. Considering the user's machine memory, set 128M temporarily, which is smaller -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m After the optimization is started, there is no Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 Worker starts 20 processes, Linux memory takes up 1.530g, an average of about 78MB each, one process PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java Check the JVM memory at this time, and there is no change Example: -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running -- -- 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 14 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc -- -- 2G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new

generation is about 2M per second Perform minor gc once every 7 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc For the JDBC result set, there are 1000. According to the test of the task instance table, there are an average of 4 task instance result sets, each time in 30s. Each time young gc old age increases by 1M The new generation is about 30M per second Master JVM optimization 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" --- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running 3~4 process examples -- --- The new generation is about 40M per second Perform a minor gc once in 40s, and the old age will increase by about 1M after each minor gc Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances The new generation has been around 500M per second Perform a minor gc once in 3s, and the old generation will increase by about 1M after each minor gc Api Server --- -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSize -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances Minor gc occurs once in about 30 minutes summary --- --- recommended that the heap memory of Master Server is 4g recommended that the heap memory of Worker Server is 2g recommended that the heap memory of Api Server is 1g recommended that the heap memory of Alert Server is 1g recommended that the heap memory of Logger Server is 1g --中文解释-- worker JVM优化 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 4g s0 108MB s1 108MB eden 865MB old 3014MB metaspace 41MB 启动完30s内，就会发生一次Minor GC和两次Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 推理： 如果要是老年代满了，引起Full GC，那老年代也不太可能Full GC之后，OU为零，可能性比较小 然后就是元数据区了，目测MU使用很多，所以调整一下元数据区域大小， 元数据区域主要存放的是一些类的描述信息和静态变量，如果反射比较多，这个区域可以适当的增大 建议256MB或者512M，考虑到用户机器内存，姑且设置128M，小一点 -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m 优化后启动之后，无Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 worker 启动20个进程，linux内存占用1.530g，平均每个78MB左右，一个进程 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java 此时查看JVM内存，并没有变化 示例：--- 每5s启动一个流程，20个任务，每个任务 sleep 10s t_ds_task_instance 40个任务在运行 -- 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -

XX:HeapDumpPath=dump.hprof" 每秒新生代會2M左右 14分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc -- 2G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代會2M左右 7分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc 对于JDBC结果集合，有1000条，根据任务实例表测试的，平均有4个任务实例结果集，30s一次yong gc 每次young gc老年代增加1M 新生代30M左右每秒 Master JVM优化 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 3~4个流程实例 -- 每秒新生代會40M左右 40s进行一次minor gc，每次minor gc之后老年代增加1M左右 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 每秒新生代會500M左右 3s进行一次minor gc，每次minor gc之后老年代增加1M左右 Api Server -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 30分钟左右发生一次minor gc 总结 -- 建议Master Server的堆内存是4g 建议Worker Server的堆内存是2g 建议Api Server的堆内存是1g 建议Alert Server的堆内存是1g 建议Logger Server的堆内存是1g
label: code-design

3. **title:** [Feature] JVM parameter optimization
body: **Describe the feature** JVM parameter optimization **Is your feature request related to a problem? Please describe.** 4G heap memory and default young generation size within 30s of starting, there will be one Minor GC and two Full GCs S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 **Describe the solution you'd like** A clear and concise description of what you want to happen. reasoning: If the old age is full, causing Full GC, then it is unlikely that the old age will be zero after Full GC, which is relatively small. Then there is the metadata area. Visually, MU uses a lot, so adjust the size of the metadata area. The metadata area mainly stores the description information and static variables of some classes. If there are more reflections, this area can be increased appropriately It is recommended to 256MB or 512M. Considering the user's machine memory, set 128M temporarily, which is smaller -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m After the optimization is started, there is no Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 Worker starts 20 processes, Linux memory takes up 1.530g, an average of about 78MB each, one process PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java Check the JVM memory at this time, and there is no change Example: -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running -- -- 4G

heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 14 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc -- -- 2G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 7 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc For the JDBC result set, there are 1000. According to the test of the task instance table, there are an average of 4 task instance result sets, each time in 30s. Each time young gc old age increases by 1M The new generation is about 30M per second Master JVM optimization 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running 3~4 process examples --- --- The new generation is about 40M per second Perform a minor gc once in 40s, and the old age will increase by about 1M after each minor gc Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances The new generation has been around 500M per second Perform a minor gc once in 3s, and the old generation will increase by about 1M after each minor gc Api Server --- -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSize -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances Minor gc occurs once in about 30 minutes summary --- -- recommended that the heap memory of Master Server is 4g recommended that the heap memory of Worker Server is 2g recommended that the heap memory of Api Server is 1g recommended that the heap memory of Alert Server is 1g recommended that the heap memory of Logger Server is 1g ---中文解释-- worker JVM优化 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 4g s0 108MB s1 108MB eden 865MB old 3014MB metaspace 41MB 启动完30s内，就会发生一次Minor GC和两次Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 推理： 如果要是老年代满了，引起Full GC，那老年代也不太可能Full GC之后，OU为零，可能性比较小 然后就是元数据区了，目测MU使用很多，所以调整一下元数据区域大小， 元数据区域主要存放的是一些类的描述信息和静态变量，如果反射比较多，这个区域可以适当的增大 建议256MB或者512M，考虑到用户机器内存，姑且设置128M，小一点 -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m 优化后启动之后，无Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT

GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 worker 启动20个进程，linux内存占用1.530g，平均每个78MB左右，一个进程 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java 此时查看JVM内存，并没有变化 示例：-- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 --- 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 14分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc --- 2G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 7分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc 对于JDBC结果集合，有1000条，根据任务实例表测试的，平均有4个任务实例结果集，30s一次yong gc 每次young gc老年代增加1M 新生代30M左右每秒 Master JVM优化 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" --- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 3~4个流程实例 --- 每秒新生代曾40M左右 40s进行一次minor gc，每次minor gc之后老年代增加1M左右 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 每秒新生代曾500M左右 3s进行一次minor gc，每次minor gc之后老年代增加1M左右 Api Server --- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 30分钟左右发生一次minor gc 总结 --- 建议Master Server的堆内存是4g 建议Worker Server的堆内存是2g 建议Api Server的堆内存是1g 建议Alert Server的堆内存是1g 建议Logger Server的堆内存是1g

4. **title:** [Feature] JVM parameter optimization
   body: **Describe the feature** JVM parameter optimization **Is your feature request related to a problem? Please describe.** 4G heap memory and default young generation size within 30s of starting, there will be one Minor GC and two Full GCs S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 **Describe the solution you'd like** A clear and concise description of what you want to happen. reasoning: If the old age is full, causing Full GC, then it is unlikely that the old age will be zero after Full GC, which is relatively small. Then there is the metadata

area. Visually, MU uses a lot, so adjust the size of the metadata area. The metadata area mainly stores the description information and static variables of some classes. If there are more reflections, this area can be increased appropriately It is recommended to 256MB or 512M. Considering the user's machine memory, set 128M temporarily, which is smaller -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m After the optimization is started, there is no Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 Worker starts 20 processes, Linux memory takes up 1.530g, an average of about 78MB each, one process PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java Check the JVM memory at this time, and there is no change Example: -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running -- -- 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 14 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc -- -- 2G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" The new generation is about 2M per second Perform minor gc once every 7 minutes, survivor 40~50M, enter the old age without dynamic age judgment, without full gc For the JDBC result set, there are 1000. According to the test of the task instance table, there are an average of 4 task instance result sets, each time in 30s. Each time young gc old age increases by 1M The new generation is about 30M per second Master JVM optimization 4G heap memory export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:Bytes=128Enabled -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- -- Start a process every 5s, 20 tasks, each task sleep 10s t_ds_task_instance 40 tasks are running 3~4 process examples -- -- The new generation is about 40M per second Perform a minor gc once in 40s, and the old age will increase by about 1M after each minor gc Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances The new generation has been around 500M per second Perform a minor gc once in 3s, and the old generation will increase by about 1M after each minor gc Api Server -- -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSize -XX:LargePageSize +UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" Start a process every 1s, 20 tasks, each task sleep 10s t_ds_process_instance 80 process instances Minor gc occurs once in about 30 minutes summary -- -- recommended that the heap memory of Master Server is 4g recommended that the heap memory of Worker Server is 2g recommended that the heap memory of Api Server is 1g recommended that the heap memory of Alert Server is 1g recommended that the heap memory of Logger Server is 1g --中文解释-- worker JVM优化 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -

XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 4g s0 108MB s1 108MB eden 865MB old 3014MB metaspace 41MB 启动完30s内，就会发生一次Minor GC和两次Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 110720.0 110720.0 0.0 48623.7 886080.0 607260.9 3086784.0 0.0 42276.0 41076.2 5676.0 5418.5 1 0.066 2 0.305 0.370 推理： 如果要是老年代满了，引起Full GC，那老年代也不太可能Full GC之后，OU为零，可能性比较小 然后就是元数据区了，目测MU使用很多，所以调整一下元数据区域大小， 元数据区域主要存放的是一些类的描述信息和静态变量，如果反射比较多，这个区域可以适当的增大 建议256MB或者512M，考虑到用户机器内存，姑且设置128M，小一点 -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m 优化后启动之后，无Full GC S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT GCT 209664.0 209664.0 0.0 51195.2 1677824.0 1584070.1 2097152.0 0.0 43136.0 41916.0 5760.0 5425.1 1 0.071 0 0.000 0.071 worker 启动20个进程，linux内存占用1.530g，平均每个78MB左右，一个进程 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 7782 root 20 0 13.638g 1.530g 23908 S 0.0 1.2 0:56.66 java 此时查看JVM内存，并没有变化 示例： -- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 -- 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 14分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc -- 2G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" 每秒新生代曾2M左右 7分钟进行一次minor gc，survivor 40~50M，无动态年龄判断进入老年代，没有full gc 对于JDBC结果集合，有1000条，根据任务实例表测试的，平均有4个任务实例结果集，30s一次yong gc 每次young gc老年代增加1M 新生代30M左右每秒 Master JVM优化 4G堆内存 export DOLPHINSCHEDULER_OPTS="-server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=dump.hprof" -- 每5s启动一个流程，20个任务，每个任务sleep 10s t_ds_task_instance 40个任务在运行 3~4个流程实例 -- 每秒新生代曾40M左右 40s进行一次minor gc，每次minor gc之后老年代增加1M左右 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 每秒新生代曾500M左右 3s进行一次minor gc，每次minor gc之后老年代增加1M左右 Api Server -- export DOLPHINSCHEDULER_OPTS="-server -Xms1g -Xmx1g -Xmn500m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:LargePageSizeInBytes=128m -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -Xloggc:gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/opt/oom" 每1s启动一个流程，20个任务，每个任务sleep 10s t_ds_process_instance 80个流程实例 30分钟左右发生一次minor gc 总结 ----------------------------------

-- 建议Master Server的堆内存是4g 建议Worker Server的堆内存是2g 建议Api Server的堆内存是1g 建议Alert Server的堆内存是1g 建议Logger Server的堆内存是1g

github_issues_comments:

1. **body:** Hi, Have we tested G1 with CMS?
   label: test
2. **body:** > Hi, > Have we tested G1 with CMS? No，in my opinion . G1 suite for large JVM memory, such as 16G or more, currently dolphinscheduler simply JVM memory will not be set so large Generally small memory is now ParNew + CMS
   label: code-design

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

jira_issues_comments: