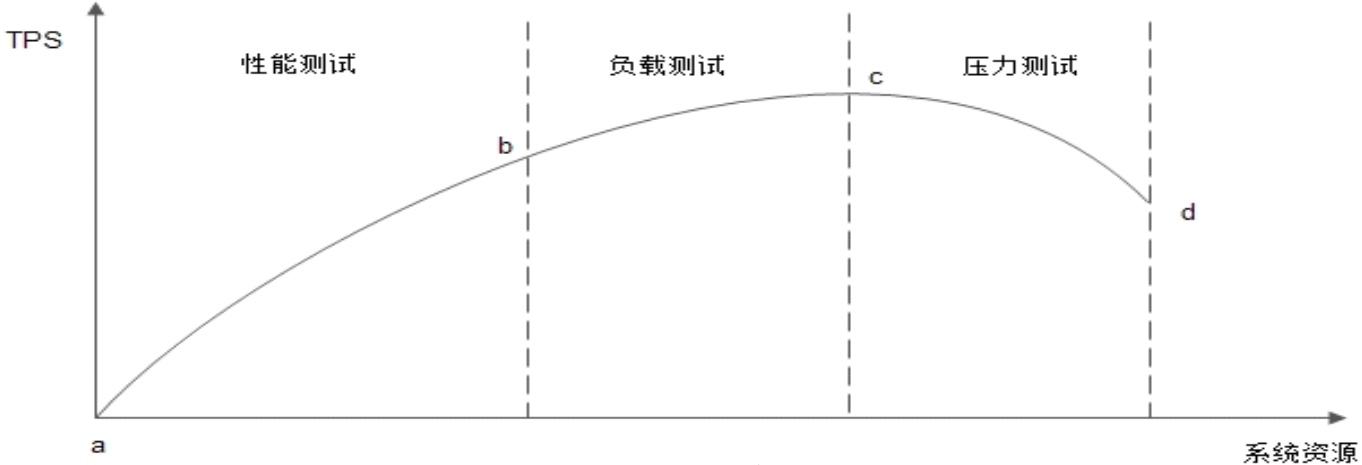
Spark源码性能优化案例分析

李智慧

性能调优的基础

- 你不能优化一个你不了解的系统
- 你不能优化一个你不能测试的系统

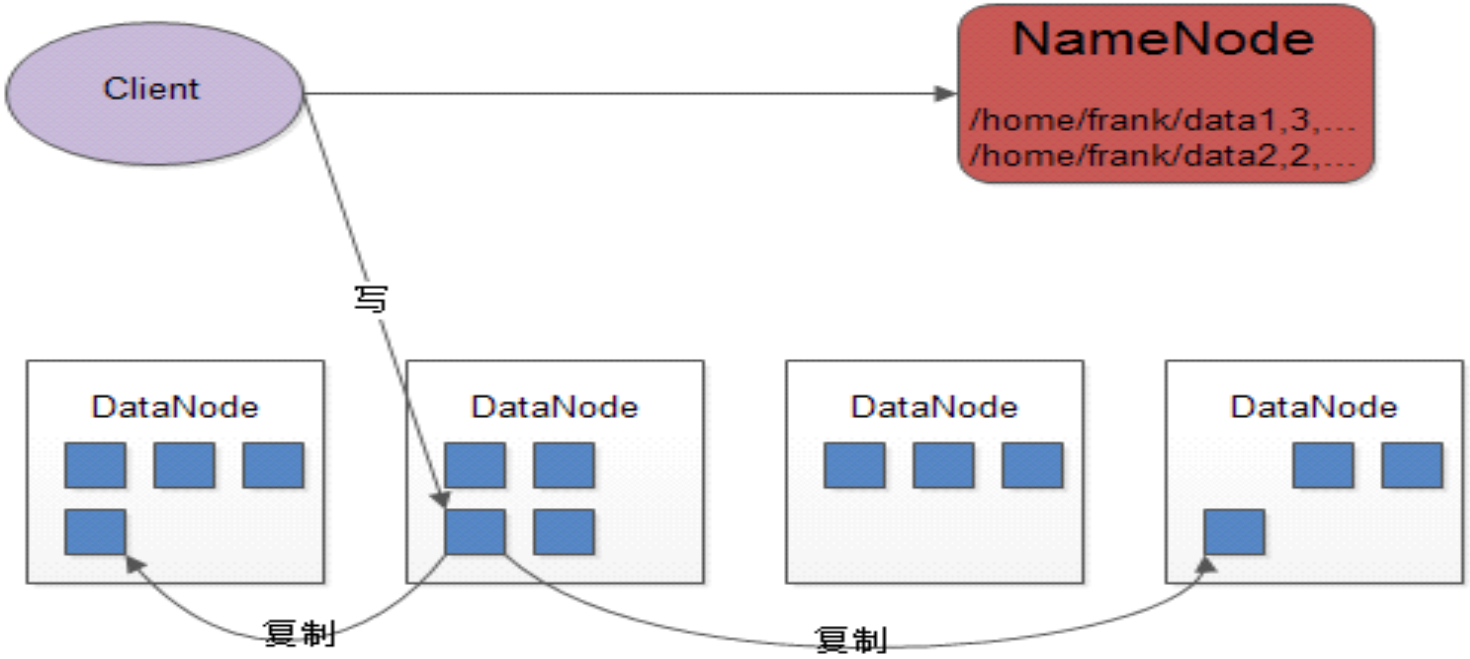


Web应用服务器性能测试曲线
来源《大型网站技术架构：核心原理与案例分析》

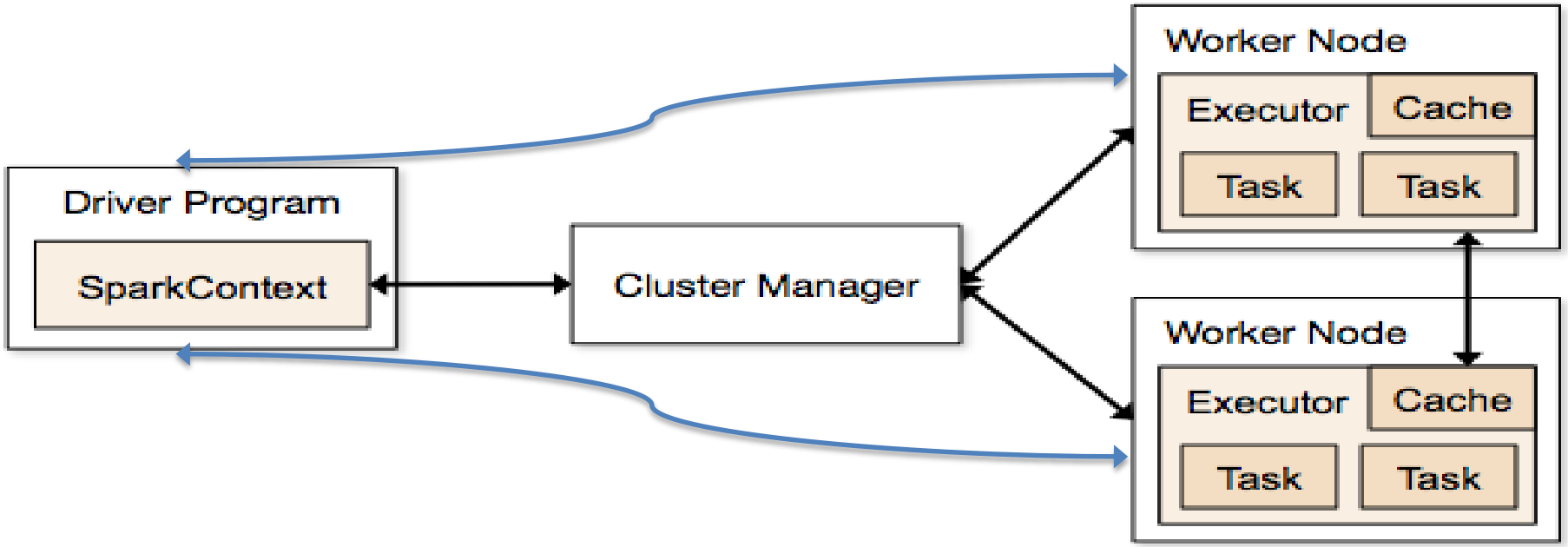
性能调优的步骤

1. 性能测试，观察系统性能特性
2. 资源（CPU、Memory、Disk、Net）利用分析，寻找资源瓶颈，提高资源利用
3. 系统架构、代码分析，发现资源利用关键所在
4. 代码、架构、基础设施调优，优化、平衡资源利用
5. 性能测试，观察系统性能特性

HDFS集群部署架构



Spark集群部署架构



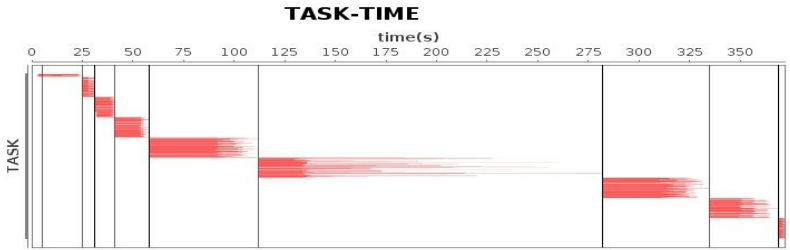
Spark性能测试工具

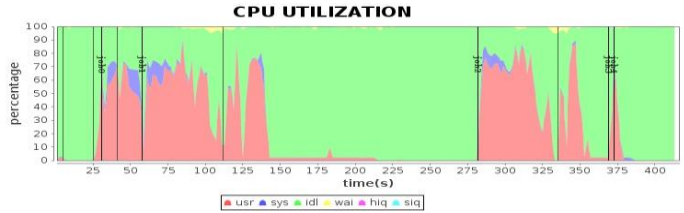
- Spark性能测试基准程序Benchmark
  - https://github.com/intel-hadoop/HiBench

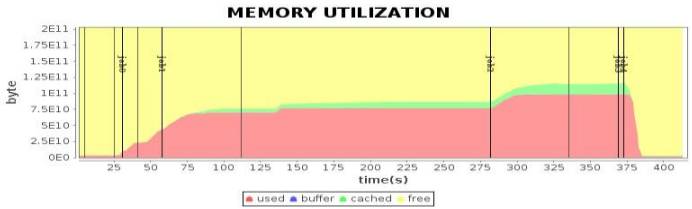
- Spark性能测试与分析可视化工具
  - https://github.com/zhihuili/Dew

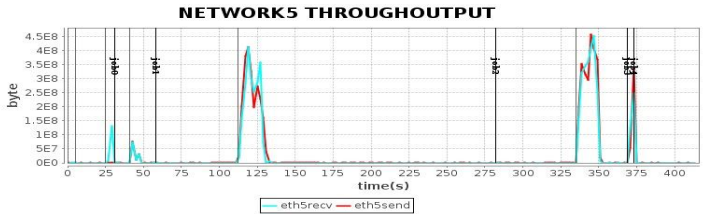
Spark性能测试

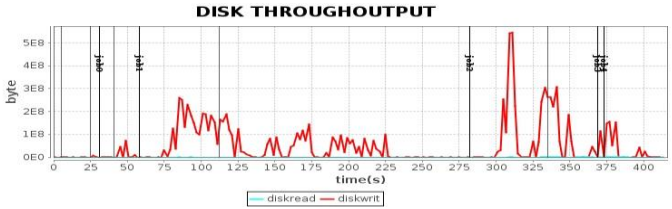
Spark作业调度的几个概念：
Job，Stage，Task





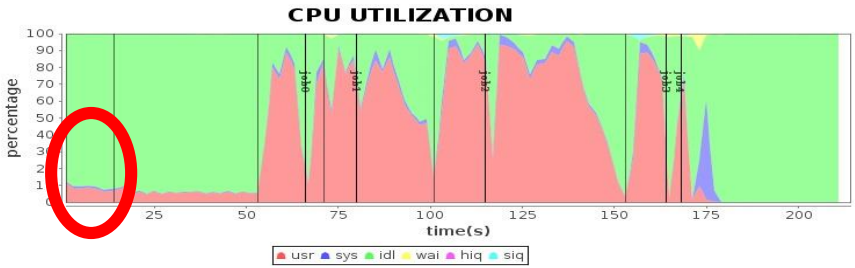


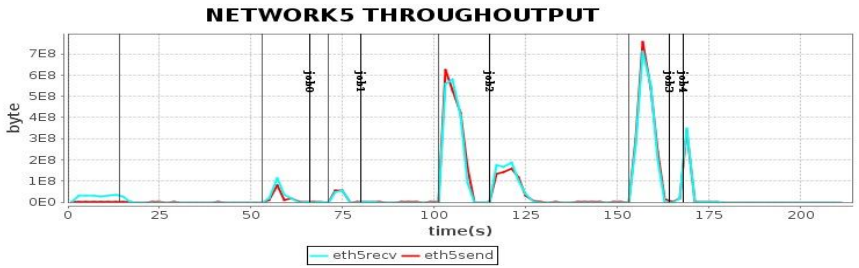




1 Spark任务文件初始化调优

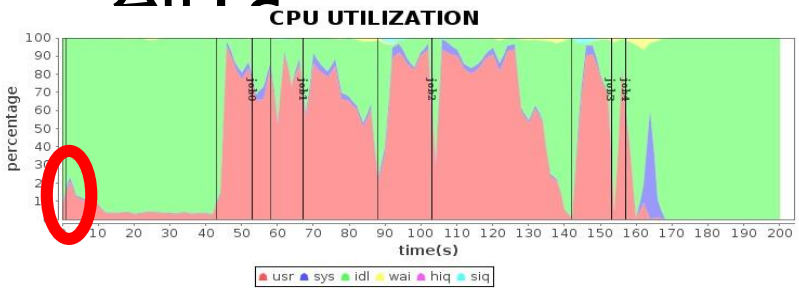
资源分析，发现第一个stage时间特别长，耗时长达14s，CPU和网络通信都有一定开销，不符合应用代码逻辑。





- 打开Spark作业log，分析这段时间的Spark运行状况。
- 根据log分析结果，阅读Spark相关源码。
- 发现Spark在任务初始化加载应用代码的时候，每个Executor都加载一次应用代码，当时每台服务器最多可启动48个Executor，每个应用代码包17M大小，导致加载开销巨大。

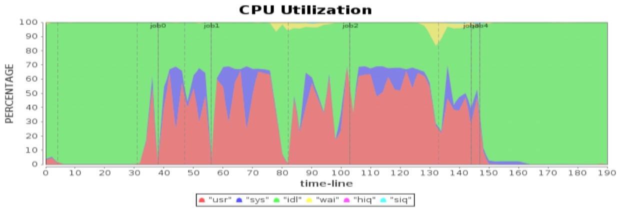
- 优化方案：Executor加载应用程序包启用本地文件缓存模式。[SPARK-2713]
- 优化效果：Stage1运行时间从14s下降到不到1s

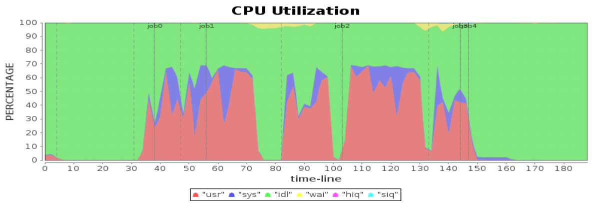


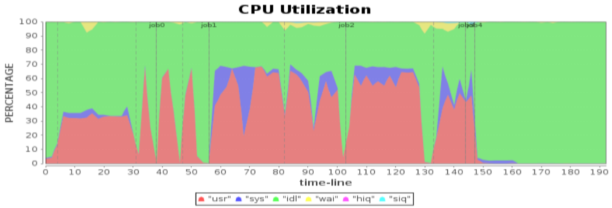

```
 * Copy cached file to targetDir, if not exists, download it from url.
 */
def fetchCachedFile(url: String, targetDir: File, conf: SparkConf, securityMgr: SecurityManager,
  timestamp: Long) {
  val fileName = url.split("/").last
  val cachedFileName = fileName + timestamp
  val targetFile = new File(targetDir, fileName)
  val lockFileName = fileName + timestamp + "_lock"
  val localDir = new File(getLocalDir(conf))
  val lockFile = new File(localDir, lockFileName)
  val raf = new RandomAccessFile(lockFile, "rw")
  val lock = raf.getChannel().lock() // only one executor entry
  val cachedFile = new File(localDir, cachedFileName)
  if (!cachedFile.exists()) {
    fetchFile(url, localDir, conf, securityMgr)
    Files.move(new File(localDir, fileName), cachedFile)
  }
  Files.copy(cachedFile, targetFile)
  lock.release()
}
```

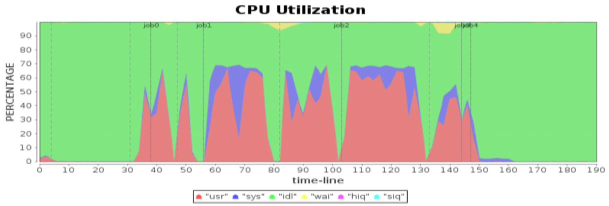
2 Spark任务调度优化

- 资源分析，发现stage2只有一台服务器上的CPU被使用，其他服务器CPU完全空闲。



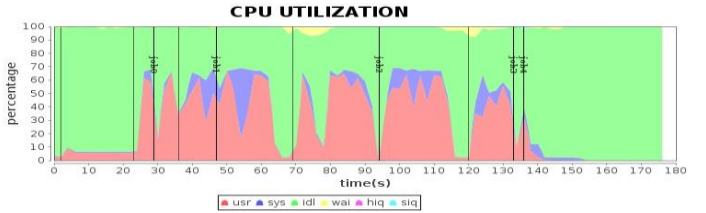


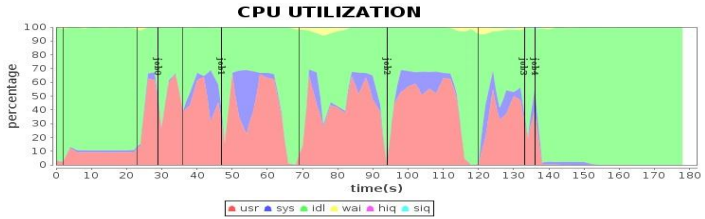


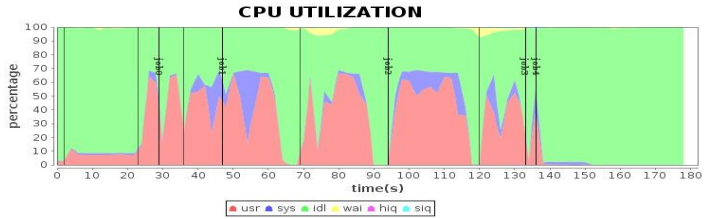


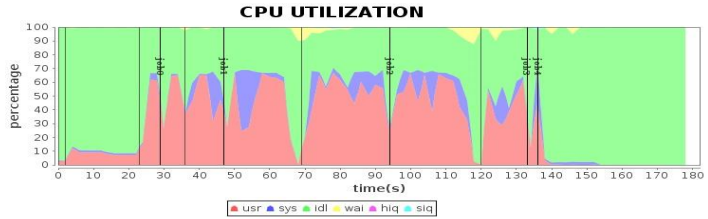
- 打开Spark作业log，分析这段时间的Spark运行状况。
- 根据log分析结果，阅读Spark相关源码。
- 通过源码发现，Spark Driver在任务分配的时候，仅针对当前已经向Driver注册过的Executor进行任务分配，而Executor的注册是有先后的，如果第一个job的任务数比较少，就会出现第一个Worker的Executor注册的时候将所有任务领走的情况。

- 优化方案：增加两个配置项，当注册Executor达到一定比例时，才开始任务分配。[SPARK-1946][SPARK-2635]
  - *spark.scheduler.minRegisteredResourcesRatio*
  - *spark.scheduler.maxRegisteredResourcesWaitingTime*
- 优化效果：stage2运行时间缩短，性能提升1.32倍









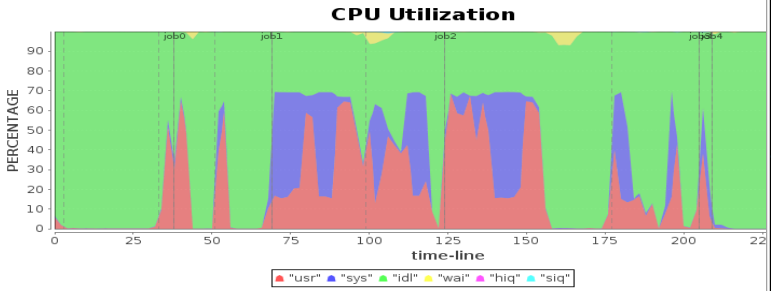
3 任务分配算法调优

- 在做log分析的时候，发现在Executor领取任务的时候，在
的任务是非lo
A[2,3,1]和B[
Executor[1]
到的任务B就
- 解决方案：对
[SPARK-219

```
+  // Improve tasks preferrd locality by sorting tasks partial ordering.
+  private def sortPendingTasksForHosts(tasksMap: HashMap[String, ArrayBuffer[Int]]) {
+    tasksMap.foreach(pair => {
+      val host = pair._1
+      var v = pair._2
+      var map = new HashMap[Int, ArrayBuffer[Int]]()
+      v.foreach(index => {
+        for (loc <- tasks(index).preferredLocations) {
+          var i = 0
+          if (loc.host == host) {
+            map.getOrElseUpdate(i, new ArrayBuffer) += index
+          }
+          i += 1
+        }
+      })
+      v.clear
+      map.foreach(kv => v ++= kv._2)
+    })
+  }
```

4 OS配置调优

- 资源分析，发现服务器大量CPU资源消耗为sys类型
- 调查发现，是因为某些Linux版本的transparent huge page默认为enable状态导致
- 优化方案：关闭OS的tra
  - *Echo never > /sys/kernel/mm/transpare*
  - *Echo never > /sys/kernel/mm/ transpare*



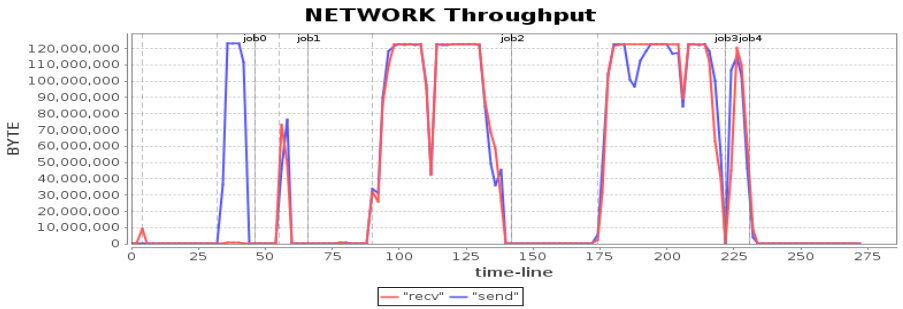

```
  30.74%            java  [kernel.kallsyms]     [k] clear_page_c_e

                    |
                    --- clear_page_c_e
                       |
                       |--99.86%-- __do_huge_pmd_anonymous_page
                       |          do_huge_pmd_anonymous_page
                       |          __handle_mm_fault
                       |          handle_mm_fault
                       |          __do_page_fault
                       |          do_page_fault
                       |          page_fault
                       |          |
                       |          |--56.93%-- oopDesc* PSPromotionManager::copy_to_survivor_space<false>(oopDesc*)
                       |          |          |
                       |          |          |--99.94%-- PSPromotionManager::drain_stacks_depth(bool)
                       |          |          |          |
                       |          |          |          |--79.15%-- StealTask::do_it(GCTaskManager*, unsigned int)
                       |          |          |          |          GCTaskThread::run()
                       |          |          |          |          java_start(Thread*)
                       |          |          |          |          start_thread
                       |          |          |          |
                       |          |          |          |--11.78%-- CardTableExtension::scavenge_contents_parallel(ObjectStartArray
*, MutableSpace*, HeapWord*, PSPromotionManager*, unsigned int, unsigned int)
                       |          |          |          |          OldToYoungRootsTask::do_it(GCTaskManager*, unsigned int)
                       |          |          |          |          GCTaskThread::run()
                       |          |          |          |          java_start(Thread*)
                       |          |          |          |          start_thread
                       |          |          |          |
                       |          |          |          |--9.06%-- ThreadRootsTask::do_it(GCTaskManager*, unsigned int)
                       |          |          |          |          GCTaskThread::run()
                       |          |          |          |          java_start(Thread*)
                       |          |          |          |          start_thread
                       |          |          |          |
                       |          |          |          --0.01%-- [...]
                       |          |          --0.06%-- [...]
```

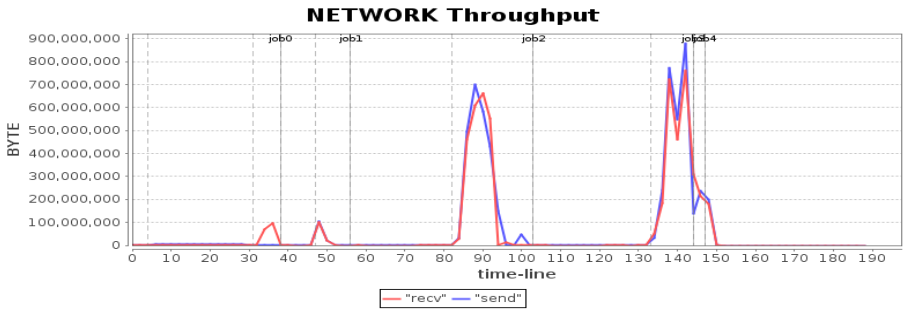
Transparent huge page 开启

5 网卡调优

- 资源分析，发现大量作业时间消耗在网络传输上。
- 解决方案：网卡带宽从1G升级到10G



1G网卡



10G网卡

Thank you for watching