**Spark**

**测试目的**

1. 调研当前竞品CPU在spark上运行状况并给出评测数据报告
2. 针对naqu进行性能摸底，分析芯片在spark运行的优劣，并针对执行状况进行系统调优

**基线测试**

**硬件规格**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 服务器名称 | Ampere | Intel | AMD ZEN3 | AMD ZEN4 | HJ01 |
| 处理器 | Ampere Altra Max | Intel(R) Xeon(R) Platinum 8352Y | AMD EPYC 7763 64-Core Processor | AMD EPYC 9754 | HJ01 |
| 核数 | 128\*2 Core | 64\*2 Core（超线程） | 128\*2 （超线程) | 128\*2（超线程） | 128\*2 core |
| 主频 | 3.0GHz(锁频) | 2.2GHz(默频)  3.4GHz(Turbo) | 3.3 GHz(Turbo) | 3.0 |  |
| 内存 | 256+256(3200MT/s) | 256+256(3200MT/s) | 256 + 256(3200MT/s) | 256+256(4800MT/s) |  |
| 网卡 | Mellanox ConnectX-4 Lx 10Gb | Intel X710  10Gb | Intel X710 10Gb | Mellanox ConnectX-4 Lx 10Gb |  |
| 硬盘 | Intel P5520 1.92T\*2 | Intel P5520 1.92T\*2 | Intel P5520 1.92T\*2 | Intel P5520 1.92T\*2 | Intel P5520 1.92T\*2 |

在ARM 和AMD的机器中还需要额外测试 跨numa 跨socket等不同情景的延迟

**软件规格**

|  |  |  |
| --- | --- | --- |
| 软件类别 | 软件名称 | 版本 |
| benchmark | TPC-DS | 2.4（测试100SQL） |
|
| Hibench | 7.1.1 (测试terasort 排序) |
| TPC-H | 2.4 |
| runtime&compiler | hadoop | 3.4.0 |
| yarn | 3.4.0 |
| spark | 3.3.1 |
| GCC | 10.2 |
| jdk | 1.8.0\_341(x86) **open-jdk8(arm)** |
| glibc | 2.33 |
| gluten | 1.0 |
| OS | CentOS | 7.9 |
| Kernel 5.10.0 |
|
|
|
|
|

**操作系统配置**

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **配置项** | **值** | **备注** |
| 内存 | 默认页大小 | x86：4k  ARM：64k | CentOS8 在不同架构机器上安装完成后，默认配置即符合要求，无需手动调整 |
| 透明大页是否开启 | 不开启 | CentOS8 默认配置是always |
| 静态大页 | 无 |  |
| 是否开启KSM | 否 | 在x86环境下，CentOS8.0默认开启，**需手动关闭**，命令如下：   |  | | --- | | Bash // 关闭KSM服务 systemctl stop ksmtuned.service // 关闭KSM服务的开机启动 systemctl disable ksmtuned.service | |
| SWAP交换是否配置 | 关闭 | CentOS8.0默认已关闭 |
| CPU | CPU缓存预取 | 打开 | **在BIOS中查看**，因机型而异 |
| CPU工作模式 | Performance | CentOS8.0默认配置不是Performance，**需手动配置**，命令如下：   |  | | --- | | Bash cpupower frequency-set -g performance | |
| 中断 | irqbalance是否开启 | 关闭 | 在x86环境下，CentOS8.0默认开启该服务，**需手动关闭**，命令如下：   |  | | --- | | Plain Text // 关闭irqbalance服务 systemctl stop irqbalance |  |  | | --- | | Plain Text // 开机默认不启用irqbalance服务 systemctl disable irqbalance | |
| OS调度 | 进程调度算法 | 默认 | - |
| 调度相关的内核配置（一般这个配置不会动） | 默认 | - |
| 网络 | 防火墙是否开启,配置的规则如何 | 关闭 | CentOS8默认已关闭 |
| 网络协议栈处理中相关过滤模块 | 开启 | CentOS8默认已开启 |
| MTU | 默认值 | CentOS8默认值1500 |
| RingBuffer | 默认值 | CentOS8环境下，x86机器默认值为512k，ARM机器默认值为1024k |
| 网卡LRO | 打开 | **x86环境下，网卡LRO被硬件固定禁用，无法在软件层面启用；ARM环境下，该选项默认值为off，需手动配置，命令如下：**   |  | | --- | | Bash ethtool -K <interface> lro on | |
| 网卡TSO | 打开 | CentOS8默认已开启 |
| 存储 | 存储IO是否走pagecache | 是 | CentOS8默认已开启 |
| 文件系统 | Ext4 | CentOS8 默认的文件系统即为Ext4 |
| IO | IOMMU/SMMU是否开启 | 关闭 | CentOS8 默认已开启 |

**测试方案**

**功能测试**

**单元测试**

在源码编译时执行mvn test 便可执行所有单元测试

**集成测试**

在模块代码下包含众多的功能集成测试的代码，但由于spark跟众多外部系统存在交互除非部署众多组件才能进行完成的功能验证测试

**性能测试**

**单socket测试**

单机测试主要采用以下架构在裸机上部署单机hadoop后将测试任务提交到yarn中，再由yarn负责后续任务的调度。执行任务的数据量根据测试规格选择1000、3000

|  |  |
| --- | --- |
| 单机部署图 | yarn 单机任务提交执行示意图 |

单机测试方法：

单socket启动hadoop集群的所有节点采用yarn 方式提交测试任务，同时TPC-DS选择scale factor 1000(1T数据)

1. yarn启动绑定socket0
2. 确保nvme和内存网卡在相同socket上

**关键参数**

|  |  |
| --- | --- |
| **dfs.block.size** | **256M** |
| **yarn.scheduler.minimum-allocation-mb** | **1024** |
| **yarn.scheduler.maximum-allocation-mb** | **65536** |
| **yarn.scheduler.minimum-allocation-vcores** | **1** |
| **yarn.scheduler.maximum-allocation-vcores** | **125** |
| **yarn.nodemanager.resource.cpu-vcores** | **128** |
| **yarn.nodemanager.resource.memory-mb** | **258048** |
| **yarn.nodemanager.numa-awareness.enabled** | **true** |
| **mapreduce.map.java.opts** | **2048M** |
| **spark master** | **yarn** |
| **spark.executor.core** | **5** |
| **spark.executor.memory** | **9G** |
| **spark.default.parallelism** | **125** |
| **spark.executor.instances** | **25** |
| **spark.sql.shuffle.partitions** | **2500** |

其中executor的提交相关参数计算参考[HiBench Introduction. Carson Wang Software & Services Group - PDF Free Download](https://docplayer.net/18523833-Hibench-introduction-carson-wang-carson-wang-intel-com-software-services-group.html)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| socket配置 | executor-core | number | executor-memory | 数据量 | parallelism | partitonSize | 备注 |
| 单socket | 5 | 25 | 9 | 1T/2T | 125 | 125\* | number=(128-1)/5  mem<=256/25\*0.9  parallelism= 5\*25  partitonSize=n\*parallelism |

**Hadoop配置文件**

hdfs-site.yml

|  |
| --- |
| YAML <configuration> <property>  <name>dfs.block.size</name>  <value>268435456</value>  </property> <property>  <name>dfs.namenode.secondary.http-address</name>  <value>hadoop1:9001</value>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>file:/data1/dfs/name,/data2/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:/data1/dfs/data,/data2/dfs/data</value>  </property>  <property>  <name>dfs.replication</name>  <value>3</value> <property>  <name>dfs.datanode.max.transfer.threads</name>  <value>8192</value> </property> <property>  <name>dfs.datanode.max.xcievers</name>  <value>65536</value> </property>  </property>  <property>  <name>dfs.webhdfs.enabled</name>  <value>true</value>  </property> <property>  <name>dfs.client.read.shortcircuit</name>  <value>true</value>  </property>  <property>  <name>dfs.domain.socket.path</name>  <value>/var/lib/hadoop-hdfs/dn\_socket</value>  </property> <property>  <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>  <value>true</value> </property> <property>  <name>dfs.client.file-block-storage-locations.timeout</name>  <value>30000</value> </property> </configuration> |

yarn-site.xml

|  |
| --- |
| YAML <?xml version="1.0"?> <configuration>  <!-- Site specific YARN configuration properties --> <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>  <value>org.apache.hadoop.mapred.ShuffleHandler</value>  </property>  <property>  <name>yarn.resourcemanager.address</name>  <value>hadoop1:8032</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>hadoop1:8030</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>hadoop1:8035</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>hadoop1:8033</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>hadoop1:8088</value>  </property>  <property>  <name>yarn.nodemanager.aux-services.spark2\_shuffle.class</name>  <value>org.apache.spark.network.yarn.YarnShuffleService</value>  </property>   <property>  <name>yarn.nodemanager.aux-services.spark\_shuffle.class</name>  <value>org.apache.spark.network.yarn.YarnShuffleService</value>  </property> <property>  <name>yarn.nodemanager.pmem-check-enabled</name>  <value>false</value> </property> <property>  <name>yarn.nodemanager.vmem-check-enabled</name>  <value>false</value> </property> <property>  <description>The minimum allocation for every container request at the RM,  in MBs. Memory requests lower than this won't take effect,  and the specified value will get allocated at minimum.</description>  <name>yarn.scheduler.minimum-allocation-mb</name>  <value>1024</value> </property> <property>  <description>The maximum allocation for every container request at the RM,  in MBs. Memory requests higher than this won't take effect,  and will get capped to this value.</description>  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>**65536**</value> </property>  <property>  <description>Amount of physical memory, in MB, that can be allocated  for containers.</description>  <name>yarn.nodemanager.resource.memory-mb</name>  <value>258048</value> </property> <property>  <description>Ratio between virtual memory to physical memory when  setting memory limits for containers. Container allocations are  expressed in terms of physical memory, and virtual memory usage  is allowed to exceed this allocation by this ratio.  </description>  <name>yarn.nodemanager.vmem-pmem-ratio</name>  <value>2.1</value> </property> <property>  <description>Amount of physical memory, in MB, that can be allocated  for containers.</description>  <name>yarn.app.mapreduce.am.resource.mb</name>  <value>258048</value> </property> <property>  <name>yarn.nodemanager.resource.cpu-vcores</name>  <value>128</value> </property> <property>  <name>yarn.scheduler.maximum-allocation-vcores</name>  <value>125</value> </property> <property>  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>258048</value>  </property> <property> <property>  <name>yarn.nodemanager.numa-awareness.enabled</name>  <value>true</value>  </property> <property>  <name>yarn.nodemanager.local-dirs</name>  <value>/data1/tmp/nm-local-dir,/data2/tmp/nm-local-dir</value> </property> <property> <name>yarn.log-aggregation-enable</name> <value>true</value> </property> <property> <name>yarn.log-aggregation.retain-seconds</name> <value>6048000</value> </property> <property> <name>yarn.nodemanager.remote-app-log-dir</name> <value>/tmp/logs</value> </property> <property>  <name>yarn.nodemanager.delete.debug-delay-sec</name>  <value>864000</value> <!-- one day --> </property> <property>  <name>yarn.log.server.url</name>  <value>http://hadoop1:19888/jobhistory/logs/</value> </property> </configuration> |

hadoop-damon.sh

|  |
| --- |
| Shell exec numactl -C 0-127 -m 0 "$hdfsscript" --config "${HADOOP\_CONF\_DIR}" --daemon "${daemonmode}" "$@" |

yarn-daemon.sh

|  |
| --- |
| Shell exec numactl --physcpubind=0-127 -m 0 nice -n $YARN\_NICENESS "${HADOOP\_YARN\_HOME}/bin/yarn" \ --config "${HADOOP\_CONF\_DIR}" --daemon "${daemonmode}" "$@" |

**Spark 任务脚本**

TPC-DS测试提交脚本

|  |
| --- |
| Shell ssh root@hadoop1 "echo 3 > /proc/sys/vm/drop\_caches" if [ ! -n "$1" ] ;then  queries="q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14a,q14b,q15,q16,q17,q18,q19,q20,q21,q22,q23a,q23b,q24a,q24b,q25,q26,q27,q28,q29,q30,q31,q32,q33,q34,q35,q36,q37,q38,q39a,q39b,q40,q41,q42,q43,q44,q45,q46,q47,q48,q49,q50,q51,q52,q53,q54,q55,q56,q57,q58,q59,q60,q61,q62,q63,q64,q65,q66,q67,q68,q69,q70,q71,q72,q73,q74,q75,q76,q77,q78,q79,q80,q81,q82,q83,q84,q85,q86,q87,q88,q89,q90,q91,q92,q93,q94,q95,q96,q97,q98,q99" else  queries=$1 fi echo "clean cache" echo "execute $queries" export SPARK\_HOME=/opt/modules/spark-3.3.2 ${SPARK\_HOME}/bin/spark-submit \ --name "spark-one-socket" \ --class com.databricks.spark.sql.perf.tpcds.RunTPCDS \ --master yarn \ --deploy-mode cluster \ --num-executors 25 \ --executor-cores 5 \ --executor-memory 9G \ --conf spark.sql.warehouse.dir=/user/root/warehouse \ --conf spark.sql.catalogImplementation=hive \ --conf spark.sql.shuffle.partitions=2500 \ --conf spark.serializer=org.apache.spark.serializer.KryoSerializer \ --conf spark.network.timeout=1200s \ --conf spark.speculation=true \ --conf spark.speculation.interval=1000 \ --conf spark.speculation.quantile=0.75 \ --conf spark.speculation.multiplier=1.5 \ --conf spark.memory.fraction=0.75 \ --conf spark.storage.storageFraction=0.4 \ --conf spark.yarn.appMasterEnv.JAVA\_HOME=/opt/modules/jdk8 \ --conf spark.executorEnv.JAVA\_HOME=/opt/modules/jdk8 \ /opt/modules/spark/spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar --scaleFactor 1000 --location /home/tpcds-performance-data --format parquet --dbPrefix hdfs1000g\_ -i 1 -q $queries |

**整机不绑核测试**

hadoop-damon.sh

|  |
| --- |
| Shell exec "$hdfsscript" --config "${HADOOP\_CONF\_DIR}" --daemon "${daemonmode}" "$@" |

yarn-daemon.sh

|  |
| --- |
| Shell exec $YARN\_NICENESS "${HADOOP\_YARN\_HOME}/bin/yarn" \ --config "${HADOOP\_CONF\_DIR}" --daemon "${daemonmode}" "$@" |

hadoop参数配置和spark提交参数与前边保持一致

**整机扩展测试**

跨socket的测试以单机双socket作为整机测试的代表。相比于单socket 内存和cpu多了一倍，但在启动时不需要绑定socket内存和cpu

**关键参数**

|  |  |
| --- | --- |
| **dfs.block.size** | **256M** |
| **yarn.scheduler.minimum-allocation-mb** | **1024** |
| **yarn.scheduler.maximum-allocation-mb** | **65536** |
| **yarn.scheduler.minimum-allocation-vcores** | **1** |
| **yarn.scheduler.maximum-allocation-vcores** | **250** |
| **yarn.nodemanager.resource.cpu-vcores** | **256** |
| **yarn.nodemanager.resource.memory-mb** | **520192** |
| **yarn.nodemanager.numa-awareness.enabled** | **true** |
| **mapreduce.map.java.opts** | **2048M** |
| **spark master** | **yarn** |
| **spark.executor.core** | **5** |
| **spark.executor.memory** | **9G** |
| **spark.default.parallelism** | **125** |
| **spark.executor.instances** | **25** |
| **spark.sql.shuffle.partitions** | **2500** |

Executor相关参数计算参考[HiBench Introduction. Carson Wang Software & Services Group - PDF Free Download](https://docplayer.net/18523833-Hibench-introduction-carson-wang-carson-wang-intel-com-software-services-group.html)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| socket配置 | executor-core | number | executor-memory | 数据量 | parallelism | partitonSize | submitTask | 备注 |
| 不绑 | 5 | 25 | 9 | 1T | 125 | 125\*n | 2 |  |
| 不绑 | 5 | 50 | 9 | 2T | 250 | 250\*n | 1 | number=(256-1)/5-1  mem<=512/50\*0.9  parallelism=5\*50  partitonSize=n\*parallelism |

**Hadoop配置文件**

hdfs-site.xml

|  |
| --- |
| YAML <configuration> <property>  <name>dfs.block.size</name>  <value>268435456</value>  </property> <property>  <name>dfs.namenode.secondary.http-address</name>  <value>hadoop1:9001</value>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>file:/data1/dfs/name,/data2/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:/data1/dfs/data,/data2/dfs/data</value>  </property>  <property>  <name>dfs.replication</name>  <value>3</value> <property>  <name>dfs.datanode.max.transfer.threads</name>  <value>8192</value> </property> <property>  <name>dfs.datanode.max.xcievers</name>  <value>65536</value> </property>  </property>  <property>  <name>dfs.webhdfs.enabled</name>  <value>true</value>  </property> <property>  <name>dfs.client.read.shortcircuit</name>  <value>true</value>  </property>  <property>  <name>dfs.domain.socket.path</name>  <value>/var/lib/hadoop-hdfs/dn\_socket</value>  </property> <property>  <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>  <value>true</value> </property> <property>  <name>dfs.client.file-block-storage-locations.timeout</name>  <value>30000</value> </property> </configuration> |

yarn-site.xml配置

|  |
| --- |
| JavaScript <?xml version="1.0"?> <configuration>  <!-- Site specific YARN configuration properties --> <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>  <value>org.apache.hadoop.mapred.ShuffleHandler</value>  </property>  <property>  <name>yarn.resourcemanager.address</name>  <value>hadoop1:8032</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>hadoop1:8030</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>hadoop1:8035</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>hadoop1:8033</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>hadoop1:8088</value>  </property>  <property>  <name>yarn.nodemanager.aux-services.spark2\_shuffle.class</name>  <value>org.apache.spark.network.yarn.YarnShuffleService</value>  </property>   <property>  <name>yarn.nodemanager.aux-services.spark\_shuffle.class</name>  <value>org.apache.spark.network.yarn.YarnShuffleService</value>  </property> <property>  <name>yarn.nodemanager.pmem-check-enabled</name>  <value>false</value> </property> <property>  <name>yarn.nodemanager.vmem-check-enabled</name>  <value>false</value> </property> <property>  <description>The minimum allocation for every container request at the RM,  in MBs. Memory requests lower than this won't take effect,  and the specified value will get allocated at minimum.</description>  <name>yarn.scheduler.minimum-allocation-mb</name>  <value>1024</value> </property> <property>  <description>The maximum allocation for every container request at the RM,  in MBs. Memory requests higher than this won't take effect,  and will get capped to this value.</description>  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>**65536**</value> </property>  <property>  <description>Amount of physical memory, in MB, that can be allocated  for containers.</description>  <name>yarn.nodemanager.resource.memory-mb</name>  <value>**520192**</value> </property> <property>  <description>Ratio between virtual memory to physical memory when  setting memory limits for containers. Container allocations are  expressed in terms of physical memory, and virtual memory usage  is allowed to exceed this allocation by this ratio.  </description>  <name>yarn.nodemanager.vmem-pmem-ratio</name>  <value>2.1</value> </property> <property>  <description>Amount of physical memory, in MB, that can be allocated  for containers.</description>  <name>yarn.app.mapreduce.am.resource.mb</name>  <value>**520192**</value> </property> <property>  <name>yarn.nodemanager.resource.cpu-vcores</name>  <value>256</value> </property> <property>  <name>yarn.scheduler.maximum-allocation-vcores</name>  <value>250</value> </property> <property>  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>520192</value>  </property> <property> <property>  <name>yarn.nodemanager.local-dirs</name>  <value>/data1/tmp/nm-local-dir,/data2/tmp/nm-local-dir</value> </property> <property> <name>yarn.log-aggregation-enable</name> <value>true</value> </property> <property> <name>yarn.log-aggregation.retain-seconds</name> <value>6048000</value> </property> <property> <name>yarn.nodemanager.remote-app-log-dir</name> <value>/tmp/logs</value> </property> <property>  <name>yarn.nodemanager.delete.debug-delay-sec</name>  <value>864000</value> <!-- one day --> </property> <property>  <name>yarn.log.server.url</name>  <value>http://hadoop1:19888/jobhistory/logs/</value> </property> </configuration> |

**Spark任务脚本**

TPC-DS测试脚本

|  |
| --- |
| Shell ssh root@hadoop1 "echo 3 > /proc/sys/vm/drop\_caches" if [ ! -n "$1" ] ;then  queries="q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14a,q14b,q15,q16,q17,q18,q19,q20,q21,q22,q23a,q23b,q24a,q24b,q25,q26,q27,q28,q29,q30,q31,q32,q33,q34,q35,q36,q37,q38,q39a,q39b,q40,q41,q42,q43,q44,q45,q46,q47,q48,q49,q50,q51,q52,q53,q54,q55,q56,q57,q58,q59,q60,q61,q62,q63,q64,q65,q66,q67,q68,q69,q70,q71,q72,q73,q74,q75,q76,q77,q78,q79,q80,q81,q82,q83,q84,q85,q86,q87,q88,q89,q90,q91,q92,q93,q94,q95,q96,q97,q98,q99" else  queries=$1 fi echo "clean cache" echo "execute $queries" export SPARK\_HOME=/opt/modules/spark-3.3.3 ${SPARK\_HOME}/bin/spark-submit \ --name "spark-two-socket" \ --class com.databricks.spark.sql.perf.tpcds.RunTPCDS \ --master yarn \ --deploy-mode cluster \ --num-executors 50 \ --executor-cores 5 \ --executor-memory 9G \ --conf spark.sql.warehouse.dir=/user/root/warehouse \ --conf spark.sql.catalogImplementation=hive \ --conf spark.sql.shuffle.partitions=2500 \ --conf spark.serializer=org.apache.spark.serializer.KryoSerializer \ --conf spark.network.timeout=1200s \ --conf spark.speculation=true \ --conf spark.speculation.interval=1000 \ --conf spark.speculation.quantile=0.75 \ --conf spark.speculation.multiplier=1.5 \ --conf spark.memory.fraction=0.75 \ --conf spark.storage.storageFraction=0.4 \ --conf spark.yarn.appMasterEnv.JAVA\_HOME=/opt/modules/jdk8 \ --conf spark.executorEnv.JAVA\_HOME=/opt/modules/jdk8 \ /opt/modules/spark/spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar --scaleFactor 1000 --location /home/tpcds-performance-data --format parquet --dbPrefix hdfs1000g\_ -i 1 -q $queries |

**集群测试**

集群测试方法：

利用本地裸机进行性能测试，三台机服务器成hadoop集群，单socket的绑定操作和单机类似，提交executor\_number\*3其他的参数基本不变 集群测试数据量初步定位3T/6T

测试过程采用三次测试取平均值的方法来计算每个SQL的时间。

|  |  |
| --- | --- |
| 集群部署图 | yarn 集群任务执行示意图 |

集群配置

|  |  |
| --- | --- |
| **dfs.block.size** | **256M** |
| **yarn.scheduler.minimum-allocation-mb** | **1024** |
| **yarn.scheduler.maximum-allocation-mb** | **65536** |
| **yarn.scheduler.minimum-allocation-vcores** | **1** |
| **yarn.scheduler.maximum-allocation-vcores** | **250** |
| **yarn.nodemanager.resource.cpu-vcores** | **256** |
| **yarn.nodemanager.resource.memory-mb** | **520192** |
| **yarn.nodemanager.numa-awareness.enabled** | **true** |
| **mapreduce.map.java.opts** | **2048M** |
| **spark master** | **yarn** |
| **spark.executor.core** | **5** |
| **spark.executor.memory** | **9G** |
| **spark.default.parallelism** | **5\*25\*3** |
| **spark.executor.instances** | 25**\*3** |
| **spark.sql.shuffle.partitions** | **2500\*3** |

集群hadoop配置与整机socket配置一样 提交参数略有不同

TPC-DS提交脚本

|  |
| --- |
| Shell ssh root@hadoop1 "echo 3 > /proc/sys/vm/drop\_caches" ssh root@hadoop2 "echo 3 > /proc/sys/vm/drop\_caches" ssh root@hadoop3 "echo 3 > /proc/sys/vm/drop\_caches" if [ ! -n "$1" ] ;then  queries="q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14a,q14b,q15,q16,q17,q18,q19,q20,q21,q22,q23a,q23b,q24a,q24b,q25,q26,q27,q28,q29,q30,q31,q32,q33,q34,q35,q36,q37,q38,q39a,q39b,q40,q41,q42,q43,q44,q45,q46,q47,q48,q49,q50,q51,q52,q53,q54,q55,q56,q57,q58,q59,q60,q61,q62,q63,q64,q65,q66,q67,q68,q69,q70,q71,q72,q73,q74,q75,q76,q77,q78,q79,q80,q81,q82,q83,q84,q85,q86,q87,q88,q89,q90,q91,q92,q93,q94,q95,q96,q97,q98,q99" else  queries=$1 fi echo "clean cache" echo "execute $queries" export SPARK\_HOME=/opt/modules/spark-3.3.3 ${SPARK\_HOME}/bin/spark-submit \ --name "spark-cluster" \ --class com.databricks.spark.sql.perf.tpcds.RunTPCDS \ --master yarn \ --deploy-mode cluster \ --num-executors 150 \ --executor-cores 5 \ --executor-memory 9G \ --conf spark.sql.warehouse.dir=/user/root/warehouse \ --conf spark.sql.catalogImplementation=hive \ --conf spark.sql.shuffle.partitions=2500 \ --conf spark.serializer=org.apache.spark.serializer.KryoSerializer \ --conf spark.network.timeout=1200s \ --conf spark.speculation=true \ --conf spark.speculation.interval=1000 \ --conf spark.speculation.quantile=0.75 \ --conf spark.speculation.multiplier=1.5 \ --conf spark.memory.fraction=0.75 \ --conf spark.storage.storageFraction=0.4 \ --conf spark.yarn.appMasterEnv.JAVA\_HOME=/opt/modules/jdk8 \ --conf spark.executorEnv.JAVA\_HOME=/opt/modules/jdk8 \ /opt/modules/spark/spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar --scaleFactor 1000 --location /home/tpcds-performance-data --format parquet --dbPrefix hdfs1000g\_ -i 1 -q $queries |

**评测工具和指标**

**评测工具**

本次采用的评测工具包含如下:

TPC-H由一系列模拟数据和一组22个复杂查询组成,涉及大量的数据聚合和连接操作。其中:

1. 数据模型模拟一家国际级制造商的业务,包括8个表(用于存储订单、部件、供应商等信息)。
2. 22个SQL查询代表了实际工作负载中的常见场景,如供应链分析、定价分析、销售预测等。
3. 查询涉及大量复杂的查询操作,如连接、子查询、聚合函数、窗口函数等。
4. 可以通过扩大数据规模(Scale Factor)来测试DBMS处理大数据的能力。

TPC-DS是TPC(Transaction Processing Performance Council)针对决策支持系统设计的一个较新的基准测试套件,全称是TPC Decision Support Benchmark。

与传统的TPC-H基准测试相比,TPC-DS具有以下几个主要特点:

1. 更复杂的数据模型和查询工作负载 TPC-DS的数据模型包含24个表和7个视图,比TPC-H更贴近现实世界中的数据仓库设计。查询工作负载涵盖99个SQL查询,比TPC-H的22个查询更加全面和复杂。
2. 模拟零售行业的业务场景 TPC-DS模拟的是一个虚构的零售产品供应商的业务数据,包括产品目录、存货、供应商、客户、促销活动等数据。这与当前许多企业实际面临的业务场景和数据特征非常贴近。
3. 更好地支持当前的数据仓库架构和工具 TPC-DS旨在与现代化的数据仓库框架、技术栈和工具相匹配,如Hadoop、Spark、MPP数据库等。它支持构建混合数据仓库,将关系数据与非结构化数据相结合。
4. 更大的数据规模 TPC-DS的数据规模可以扩展到远大于TPC-H,有利于测试系统处理海量数据的能力。
5. 定期更新维护 TPC-DS由TPC委员会持续维护和更新,以确保其保持与时俱进,符合最新的数据处理技术趋势。

Hibench: terasort 生成数据进行排序

选择多个工具是参考了ampere和arm blog中均有介绍过也是业界通常用来测试性能的工具

**评测指标**

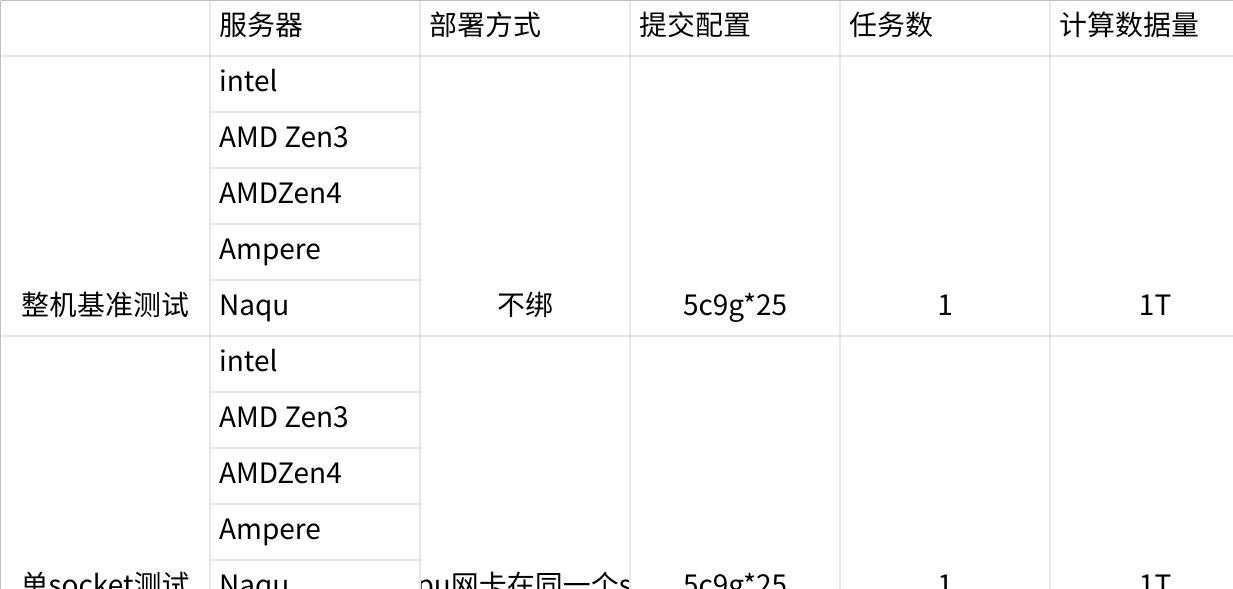
TPC-H: 执行时间

TPC-DS: 执行时间

Hibench: 执行时间

整机功耗：消耗的watt

**性能测试用例汇总**



**点击图片可查看完整电子表格**

**测试要点**

1. Spark-SQL转换为执行计划代码过程中是否存在JDK负优化(关注窗口函数)
2. 向量化ARM neon指令的执行效率(CPU密集型SQL计算效率的提升)
3. 跨numa访问对Spark执行性能的影响(测试yarn numa感知功能对性能的影响以及跨socket带来的延迟)
4. 网络IO和磁盘IO处理latency跟CPU的关联关系(IO密集型SQL)

**数据采集和分析**

**资源利用率**

cpu利用率 系统调用 用户 idle等在执行过程的占比

Mem: 内存使用率，

Io: iops和io latency

Network: 网络带宽

**热点函数**

利用hj-devkit采集 所有container的热点函数和堆栈并最终合并生成火焰图。

主要采用CPU密集型作业

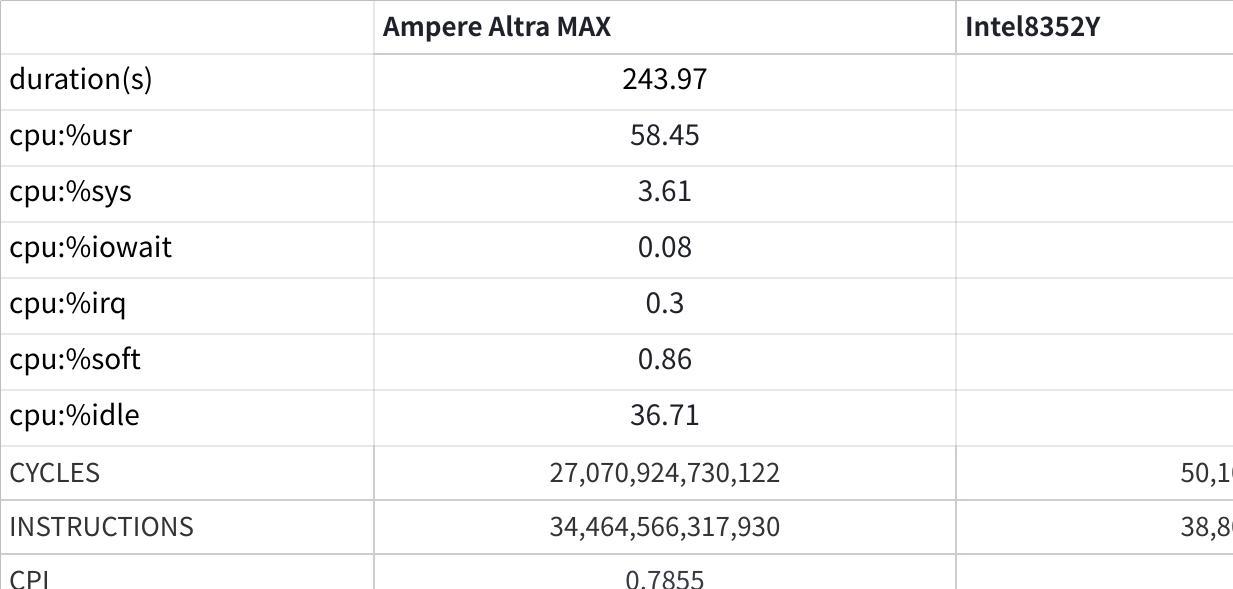


非CPU密集型SQL考虑采集Offcpu火焰图

**微架构数据**

微架构主要看CPU密集性作业

整体



**点击图片可查看完整电子表格**

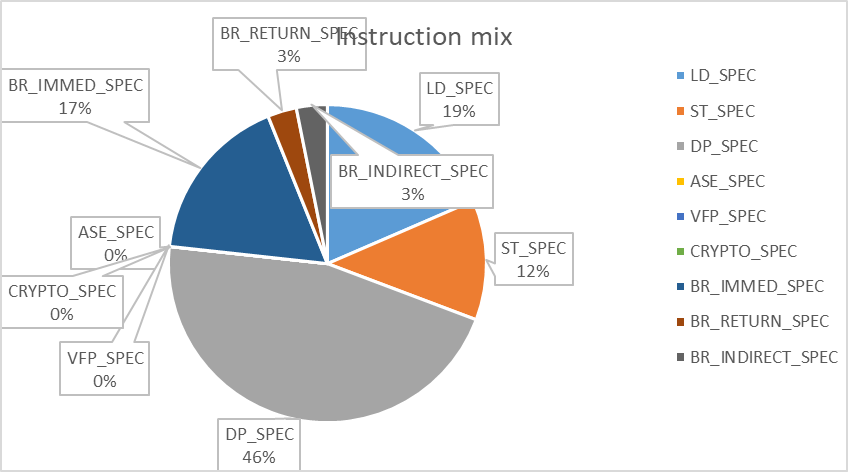
DTLB

|  |  |  |
| --- | --- | --- |
|  | Ampere Altra MAX host1 | Intel8352Y |
| DTLB WALKS | 2,157,193,982 | 5,217,947,528 |
| DTLB ACCESSES | 15,303,790,147,003 | 25,025,026,389,929 |
| DTLB MPKI | 0.0561 | 0.0804 |
| DTLB WALK RATE | 0.0001 | 0.0001 |

Dcache

|  |  |  |
| --- | --- | --- |
|  | Ampere Altra MAX | Intel8352Y |
| L1D CACHE MPKI | 4.49 | 2.5093 |
| L1D CACHE MISS RATE | 0.01 | 0.0099 |
| L2 CACHE MPKI | 5.54 | 0.3856 |
| L2 CACHE MISS RATE | 0.26 | 0.1536 |
| LLC CACHE MPKI | **242.34** | 7.8154 |
| LLC CACHE MISS RATE | 0.87 | 0.3821 |

**指令占比**



**优化方案**

**向量化**

Gluten是一个spark-sql引擎加速插件。当前有很多原生数据处理引擎如arrow,velox，clickhouse,这些本地库的性能远超spark上的jvm处理引擎，但是目前这些库都是大部分都是在单机上执行的。然而spark是具备易扩展性的数据处理的引擎，gluten将本地向量化引擎库的优势和spark-sql的优势同时结合起来达到加速sql执行的目的。gluten当前主要实现的功能有以下几点：

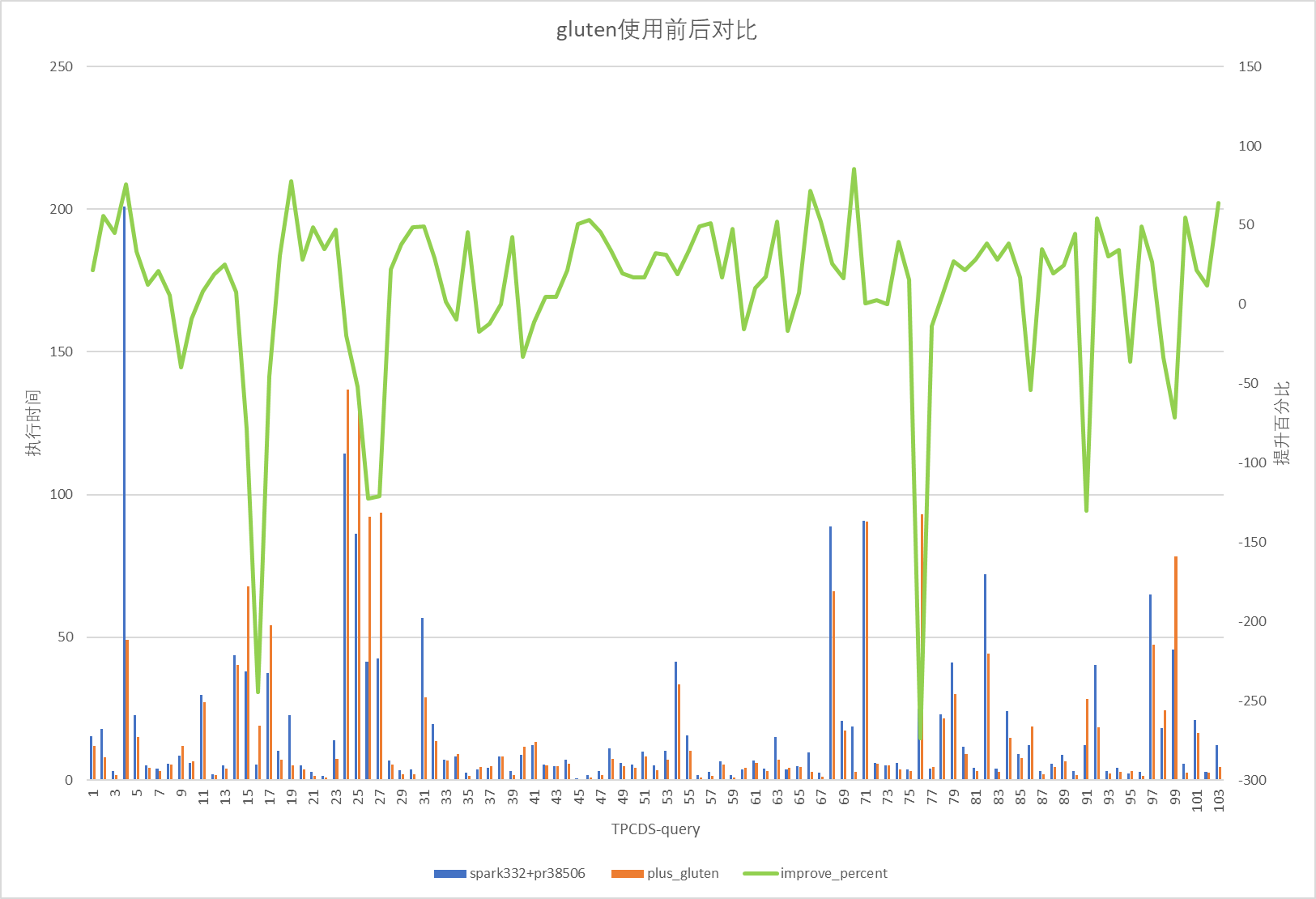
1. 将spark-sql的物理执行计划转换为Substrait计划并发送给本地引擎执行
2. 将关键数据处理卸载到本地库执行
3. 管理本地库和jvm之间数据共享

**gluten应用现状**

Gluten 当前主要应用于CPU密集型SQL。gluten提供了完整columnar Shuffle，使用gluten的时候可以同时借助celeborn的remoteshuffle 服务，将gluten shuffle 的native Partitioner, zero copy优势和cleleborn shuffle的稳定可扩展的优势同时结合。

**gluten在ARM上运行结果**

在前面应用层优化的基础上应用gluten插件，执行的结果如下图。 TPC-DS 80%的SQL都有提升，CPU密集型SQL提升超过50%,具体效果如下图，整体提升**5.5%**。



应用Gluten中间件之后，q4的运行时间**缩短了约75%(从201s缩短到49s）**。下图为整形指令火焰图，其中代码编译生成(图中右边黄色部分）占比48%，其余绿色部分主要为Spark计算流程，针对这部分的向量化，基于gluten实现。

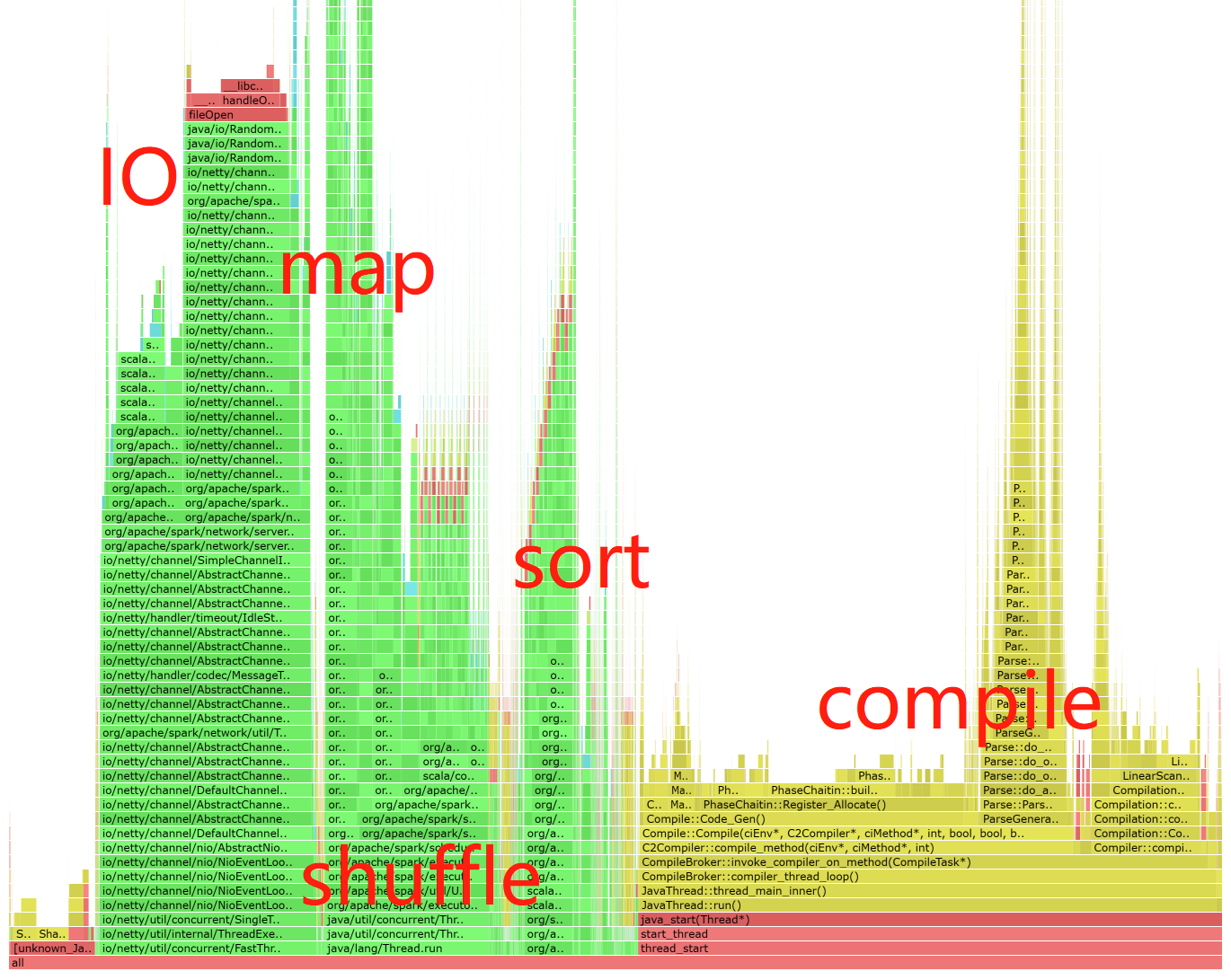


图6-47 Q4 使用gluten火焰图

下表为基于gluten插件进行向量化改造前后的指令分类对比。经过改造之后，ASE\_SPEC指令显著增加。虽然占总指令数的比例仍然较低，整形指令的占比基本上没有变化，但总指令数量降低了75%，因此具有较大的提升。

表6-6 Ameper 使用gluten前后指令占比对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metrics | Ampere计数值 | 比例 | 向量化后 | 比例 |
| LD\_SPEC | 688,541,582,797 | 18.52% | 200,167,019,970 | 20.76% |
| ST\_SPEC | 454,591,283,751 | 12.22% | 75,709,072,426 | 7.85% |
| DP\_SPEC | 1,709,633,198,145 | 45.97% | 442,718,130,452 | 45.93% |
| ASE\_SPEC | 253,804,995 | 0.01% | 1,900,481,461 | 0.20% |
| VFP\_SPEC | 433,842,823 | 0.01% | 118,540,769 | 0.01% |
| CRYPTO\_SPEC | 1,530,461 | 0.00% | 1,598,087 | 0.00% |
| BR\_IMMED\_SPEC | 638,749,299,382 | 17.18% | 162,594,326,817 | 16.87% |
| BR\_RETURN\_SPEC | 109,591,463,922 | 2.95% | 36,146,806,461 | 3.75% |
| BR\_INDIRECT\_SPEC | 116,903,142,906 | 3.14% | 44,617,340,200 | 4.63% |

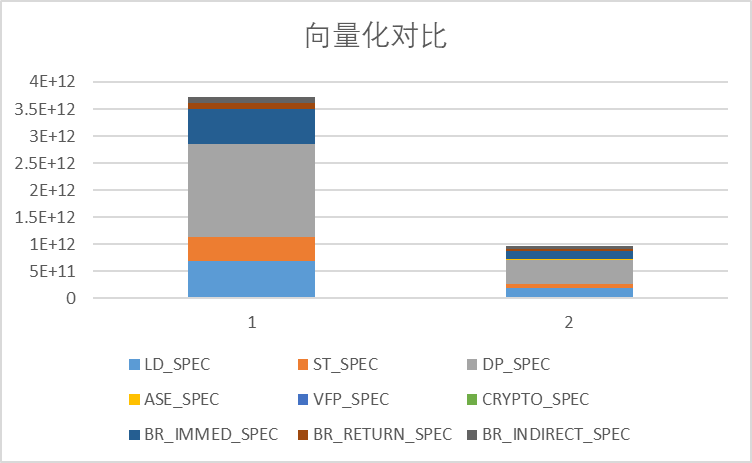


图6-44 向量化指令对比

**测试重点**

1. 评估向量化对CPU密集型和IO密集型SQL的提升效果
2. 对比不同类型机器相同指令的提升提升效率

**OS调优**

**网络调优**

**调整网卡队列**

ethtool -G eth0 rx 4096 tx 4096

**开启LRO GRO TSO GSO**

ethtool -K eth0 lro on

ethtool -K eth0 gro on  
ethtool -K eth0 tso on  
ethtool -K eth0 gso on

**配置网卡中断**

**关闭irqbalance**

停止irqbalance服务，并永久禁止开机启动

systemctl stop irqbalance.service

systemctl disable irqbalance.service

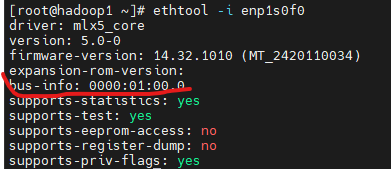
查看irqbalance状态

systemctl status irqbalance.service

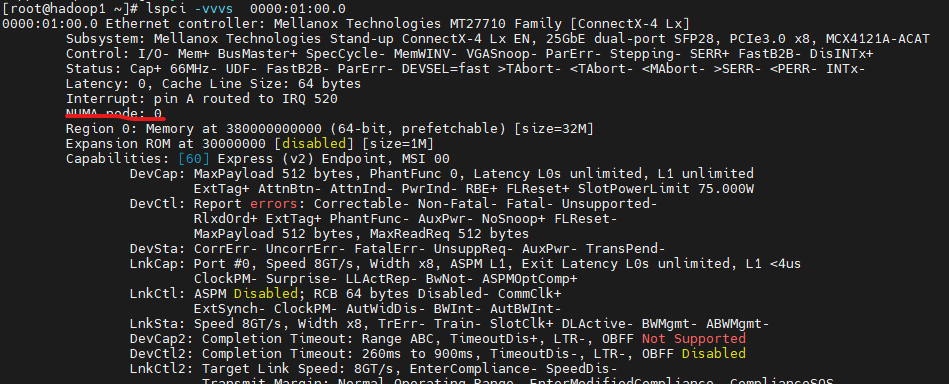
**绑定网卡中断**

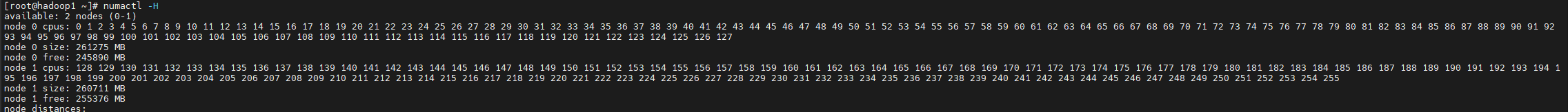
1. 查看网络设备号，并查其所在numa

ethtool -i enp1s0f0



查看设备numa





1. 绑定中断到相应的numa

bindIrq.sh

|  |
| --- |
| Bash #!/bin/bash irq\_list=(`cat /proc/interrupts | grep enp1s0f0 | awk -F: '{print $1}'`) cpunum=1 # 修改为所在node的第一个Core for irq in ${irq\_list[@]} do echo $cpunum > /proc/irq/$irq/smp\_affinity\_list echo `cat /proc/irq/$irq/smp\_affinity\_list` (( cpunum+=1 )) done |

bash bindIrq.sh

验证 verifyIrq.sh

|  |
| --- |
| Bash #!/bin/bash # 网卡名 intf=$1 log=irqSet-`date "+%Y%m%d-%H%M%S"`.log # 可用的CPU数 cpuNum=$(cat /proc/cpuinfo |grep processor -c) # RX TX中断列表 irqListRx=$(cat /proc/interrupts | grep ${intf} | awk -F':' '{print $1}') irqListTx=$(cat /proc/interrupts | grep ${intf} | awk -F':' '{print $1}') # 绑定接收中断rx irq for irqRX in ${irqListRx[@]} do cat /proc/irq/${irqRX}/smp\_affinity\_list done # 绑定发送中断tx irq for irqTX in ${irqListTx[@]} do cat /proc/irq/${irqTX}/smp\_affinity\_list |

**关闭netfilter**

在安全许可的情况下，关闭netfilter可以减少的网络收发链路

iptables --list  
iptables -F  
iptables -X  
iptables -Z  
rmmod iptable\_filter  
iptables -t mangle -F  
iptables -t mangle -X  
rmmod iptable\_mangle  
iptables -t nat -F  
iptables -t nat -X  
rmmod iptable\_nat  
rmmod xt\_conntrack  
rmmod xt\_MASQUERADE  
rmmod nf\_nat  
rmmod nf\_conntrack

**调整网络参数**

增加网络buffer 增加窗口大小 根据BDP进行适当调整

|  |
| --- |
| Shell sysctl -w net.core.rmem\_max=8388608 sysctl -w net.core.wmem\_max=8388608 sysctl -w net.core.rmem\_default=65536 sysctl -w net.core.wmem\_default=65536 sysctl -w net.ipv4.tcp\_mem=’8388608 8388608 8388608′ sysctl -w net.ipv4.tcp\_rmem=’4096 87380 8388608′ sysctl -w net.ipv4.tcp\_wmem=’4096 65536 8388608′ sysctl -w net.ipv4.route.flush=1 |

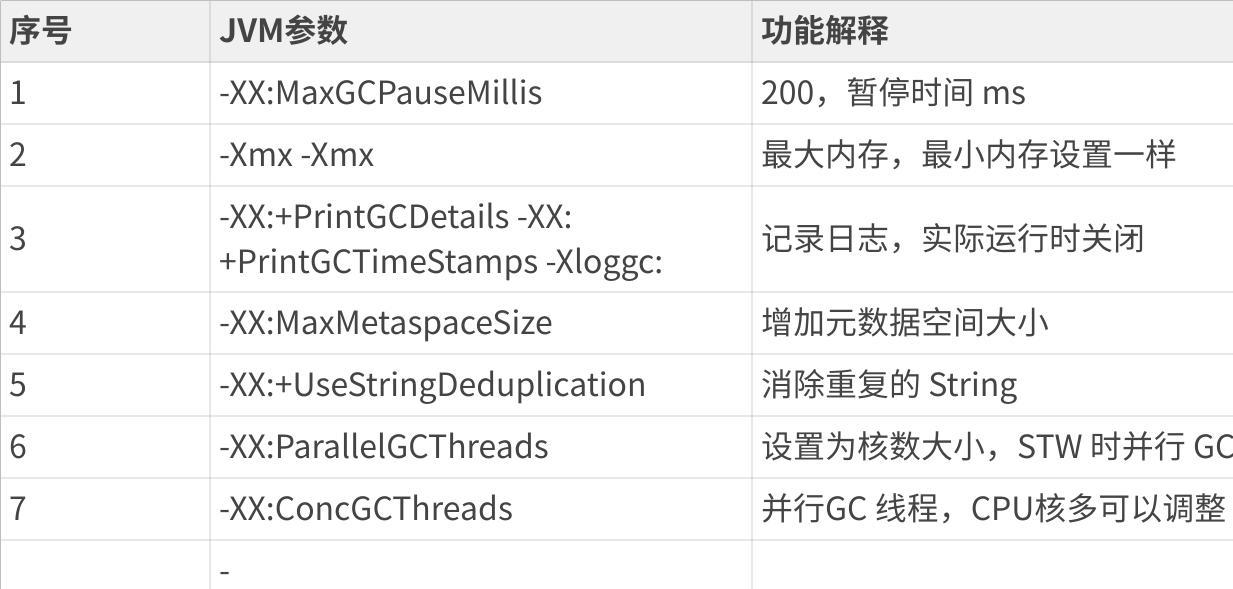
**关闭内存大页**

|  |
| --- |
| Shell echo never > /sys/kernel/mm/transparent\_hugepage/enabled echo never > /sys/kernel/mm/transparent\_hugepage/defrag |

**JVM调优**

**G1调优**

调优参数



**点击图片可查看完整电子表格**

**ZGC调优**

调优参数



**点击图片可查看完整电子表格**

**RDMA使能**

RDMA借助开源SparkUCX实现。SparkUCX 是一个高性能的 Apache Spark ShuffleManager 插件,它使用 RDMA 和其他由 UCX 支持的高性能传输方式来执行 Spark 作业中的 Shuffle 数据传输。

* SparkUCX 是一个插件,用于 Apache Spark 这个流行的大数据处理引擎
* 它专门针对 Spark 作业中的 Shuffle 过程进行了优化
* Shuffle 是 Spark 作业中一个非常关键和耗费资源的环节,需要在不同执行器之间传输中间数据
* 传统的 Shuffle 过程使用基于 TCP/IP 的网络传输,性能相对有限
* SparkUCX 使用了 RDMA (Remote Direct Memory Access) 以及其他由 UCX 框架支持的高性能网络传输技术
* RDMA 能够直接在两台主机的内存之间传输数据,避免了数据在内核和用户态的多次拷贝,大幅提升了传输效率
* 通过 SparkUCX,可以显著提高 Spark 作业中 Shuffle 阶段的性能,加速整个作业的执行速度

参考链接：

[TPCx-HS优化总结 - 记录每个瞬间](https://code0xff.org/post/2023/11/tpcx-hs%E4%BC%98%E5%8C%96%E6%80%BB%E7%BB%93/)