



Goldengate 性能优化

杜兴春

2019.4.12



CONTENTS

目录

- 01 如何监控延迟（心跳脚本）
- 02 可能的性能瓶颈有哪些？
- 03 Extract进程的优化手段
- 04 Replicat进程的优化手段
- 05 集成模式下的优化手段



如何监控延迟（心跳脚本）

监控*latency*的脚本

■ *Lag*指令

配置*NoTcpSourceTimer*参数：保留原始时间戳发送到目标，这样统计更准确（*pump*进程中配置）
（第5课审计案例中可以配置该参数，统计会更准确。）

进程*startup*

- *DynamicResolution*
- *WildcardResolve Dynamic*（通配符方式会降低性能，但是更方便）



可能的性能瓶颈有哪些？

性能瓶颈的诊断过程

- 性能诊断是从`rep`到`ext`反向进行（从目标开始分析起）
- 借助`checkpoint`来分析读和写检查点是否`moving`
- `Mgr`进程`report lag`
- 从吞吐量分析（`reportcount, stats`）

当然如果是正常的进程`stop`（业务维护场景下）或者大事务导致`rep`慢都属于正常。

分析性能瓶颈

- *CPU*
- *DISK IO*
- *Memory、Pageing*
- *Network*
- *Database*

网络瓶颈

- 可以拆分多个 *datapump*
- 增加 *flushsecs*（默认1秒）、*tcpflushbytes*
- 增加 *tcpbufsize*
- 提升 *cachemgr*

Disk瓶颈

■ Disk读

- EOFDelay、EOFDelayCSecs（控制读*redo log*或读*trail*文件的频率，提高该值，降低频率）
- 日志所在磁盘的性能以及日志文件的独立分布

■ Disk写

- 写*trail*文件、写*checkpoint*信息
- 更高性能的*disk*、*RAID*的配置（*raid 1+0*）
- *Checkpointsecs*（降低*checkpoint*频率，默认10秒）
- *Replicat: GroupTransOps*（控制多少*operation group*到一个事务，降低*checkpoint*）
- *Compressupdates*、*compressdeletes*（减少*trail*文件的大小）

Manager调优

- *Autorestart*
- *Autostart*
- *BootDelayMinutes* (windows环境下, 如果ogg配置成service自动启动, 控制在DB还未启动好时的autostart行为)



Extract进程的优化手段

Extract进程 – 寻找瓶颈

- 使用系统工具监控Extract运行的CPU/IO/Mem

- 使用TESTMAPPINGSPEED（配置一路测试ext进程，加上参数，

使用`stats extract <extract_name>, totalonly *, reportrate sec`或者`min`来监控处理速度）

- 可以使Extract只解析日志但不写队列，用于测试日志抽取速度

- 如果加入该参数使Extract性能能够显著提高，则写队列为可能瓶颈

- 缩小Extract包含的表范围

只保留一个数据变化较小表或者加一个测试表，观察Extract处理速度是否得到提高，如有显著提高则可能瓶颈在数据库fetch，可通过查询运行期间存在哪些select语句予以验证

- Extract的瓶颈一般在于LCR转换为UDF

Extract进程的性能 - 调优

- 过滤和转换建议不在`extract`进程中配置
- 最直接的优化（特别是需要回库`fetch`数据），建议拆分出单独的`ext`进程（有关联的表放在一个进程中）
- 对于大表（`LOB`字段多），可以用`@RANGE`进行拆分
- `DB`写日志和`extract`读日志可能引起的`IO`竞争（对存储的性能要求提升）
- 如果配置了`SQLEXEC`，`MAXFETCHSTATEMENTS`参数控制源端`extract`可以`open`的`cursor`数量

`FETCHOPTIONS MAXFETCHSTATEMENTS 150`

- 如果系统`IO`出现瓶颈
 - 增大日志读取间隔 `EOFDELAY 3` //间隔3秒，缺省为1秒
 - 增大内存刷新闻隔 `FLUSHSECS 3` //间隔为3秒，缺省为1秒

Pump进程 - 调优

- 使用PASSTHRU避免与源数据库交互；不加userid和password参数
- 加入数据压缩： *COMPRESS*
- 增大tcp缓存大小： *TCPBUFSIZE*和*TCPFLUSHBYTES*
- 增大队列读取间隔和内存刷新闻隔： *EOFDELAY*、*FLUSHSECS*



Replicat进程的优化手段

Replicat进程 – 寻找瓶颈

- 使用系统工具监控Extract运行的CPU/IO/Mem
- 使用TESTMAPPINGSPEED（配置一路测试rep进程，加上参数，使用stats replicat <group>, totalonly *, reportrate sec或者min来监控处理速度）
 - 可以使replicat只解析队列但不写数据库，用于测试读队列速度
 - 如果加入该参数使replicat性能能够显著提高，则写数据库为瓶颈
- Replicat的瓶颈一般在于写数据库

*Replicat*进程的性能

- 加速目标行的定位 – 使用`keycols`

- 前提：逻辑`key`，也必须保证是唯一的，否则可能定位错误
- 该列必须在附加日志中有，如果没有，源：`add trandata, cols(<column>)`
- 在目标表上，为该列创建`index`

Replicat进程的性能

■ 操作合并

➤ *BatchSQL*: 相同的表、相同的操作类型、相同的字段操作在内存中合并成一个数组array

BATCHSQL BATCHESPERQUEUE 100, OPSPERBATCH 8000

说明: 只适用于小表, 对于列特别多或者字段特别长的表反而可能降低性能; 对于少量表重复进行操作的情景例如批处理比较有效。可以通过对两个子参数的组合进行尝试获取最佳性能(子参数控制内存的开销); 如果无pk的表, 开启batchsql

参考ogg12c reference guide 3-26 p232

Replicat进程的性能

- 密集小事务使用事务合并

- *GroupTransOps*: 事务的合并（下图中的normal mode, non-batchsql）

GroupTransOps 1500

三个交易，分别有200/400/500个记录，假如grouptransops 为1000 (缺省值)，则Replicat一直要等待到第三个交易时 $200 + 400 + 500 > 1000$ 才后Commit

Replicat进程的性能

■ 大事务的拆分

➤ *MaxTransOps* （集成模式不支持）

MAXTRANSOPS 10000

事务拆分实际是破坏了事务一致性，可能出现不一致情况。

建议设置为相同值 *MAXTRANSOPS = GROUPTRANSOPS*

■ 加速Insert速度

➤ *INSERTAPPEND*: 非集成模式下，仅对oracle数据库，相当于*Append Hint*；一般用在一个事务中大量的insert操作到同一张表，如果小事务多，可能反而会使性能下降。

Replicat进程的性能

■ 不同的表拆进程

- 有关联关系的表放在一个进程（如：同一个`schema`下的表）
- 拆分了之后，对于DDL复制要注意，只能一路配置`ddl`复制（或者`ddl`是`include mapped`）；或者直接限制只复制DML

拆分进程过程，参考《进程拆分》

Replicat进程的性能

■ 一张大表进行分区: @RANGE

➤ 如果在源端:

Pump:

```
rmtrail ./dirdat/aa
```

```
Table scott.t1, filter (@range (1, 2));
```

```
rmtrail ./dirdat/bb
```

```
Table scott.t1, filter (@range (2, 2));
```

目标上配置两个replicat进程分别读aa和bb trail文件

➤ 如果在目标端配置:

```
Map scott.t1, target gtj.t1, filter (@range (1, 2, ID));
```

```
Map scott.t1, target gtj.t1, filter (@range (2, 2, ID));
```

需要是PK或keycols列

进程拆分注意事项

- 各进程间没有同步机制，应尽量确保同一交易涉及表在一个进程；以业务或Schema进行区分
- 单个extract进程可处理日志一般为30-50G/小时，单个replicat进程一般只能处理1G队列/小时，可采用一个extract对多个replicat的模式
- 由于extract在catch up（追赶）模式需要读取归档日志，速度慢且耗费资源高，建议extract一旦出现较大延迟则立即进行拆分
- Replicat拆分可能临时造成各进程间不同步，但是
 - 多个Replicat性能会得到很大提高，可以保证数据复制始终是实时的
 - 是同一个队列，当它们应用队列数据完毕后是可以达到数据一致的
 - 当源端出现灾难后，由于Extract可以保证源端抽取时数据的一致性，而目标端多Replicat读取的

Replicat进程的性能

- 降低checkpoint频率，但是latency会变大

CheckpointSecs

- 数据库

➤ 数据库级别的优化、*segment*的优化

➤ *MAXSQLSTATEMENTS*参数，加大*cursor*的分配，最大250

MAXSQLSTATEMENTS 200



集成模式下的优化手段

集成模式下的性能

- 集成模式下，源和目标的多线程提升性能

- 集成capture

TRANLOGOPTIONS INTEGRATEDPARAMS (max_sga_size ***, parallelism *)

- 集成apply

DBOPTIONS INTEGRATEDPARAMS(parallelism N)

- Range的配置

MAP <source>, target <table>, THREADRANGE(1-N);

MAP <source>, target <table>, THREAD(1);

集成模式下的性能

■ 多线程下可能出现的问题

- 事务的依赖关系，导致ora-1403 (*no data found*)
- 死锁
- 性能下降

■ *COMMIT_SERIALIZATION* 配置目标上事务提交的顺序

- *dependent_transactions*: 按照依赖关系提交
- *Full*: 和源保持完全一致；速度会下降

DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION DEPENDENT_TRANSACTIONS

集成模式下的性能

- *BatchSQL*可以通过*inbound server*来支持

默认*batchtransops*为50, 如果*apply*出现: *wait for depenency*过高时, 需要减少*batchtransops*值

如果*batchsql*出错时, 会回滚*batch*的事务, 转成*normal mode*进行*apply*

- *Maxtransops*不支持
- *Grouptransops*: 在*parallelism=1*时支持

集成模式下的视图

■ 配置相关

Db_goldengate_inbound

Db_goldengate_process

Db_gg_inbound_process

Db_apply

....

■ 运行时

v\$GG_APPLY_Server

v\$GG_APPLY_Reader

V\$goldengate_table_stats。 。 。

A graphic featuring a large, solid dark blue diamond in the center. Inside the diamond, the words "THANK YOU" are written in a white, bold, sans-serif font. Behind the diamond is a light blue square outline. To the left and right of the diamond, there are two small, solid dark green triangles pointing outwards.

THANK YOU