Flink 使用 RocksDB 和 Gemini 的性能对比实验

发布于: 2020 年 08 月 06 日

微博机器学习平台使用 Flink 实现多流 join 来生成在线机器学习需要的样本。时间窗口内的数据会被缓存到 state 里，且 state 访问的延迟通常决定了作业的性能。开源 Flink 的状态存储主要包括 RocksDB 和 Heap 两种，而在去年的 Flink Forward 大会上我们了解到阿里云 VVP 产品自研了一款更高性能的状态存储插件 Gemini，并对其进行了测试和试用。

在本篇文章中我们将对 RocksDB、Heap 和 Gemini 在相同场景下进行压测，并对其资源消耗进行对比。测试的 Flink 内核版本为 1.10.0。

测试场景

我们使用真实的样本拼接业务作为测试场景，通过将多个流的数据union后对指定key做聚合（keyby），在聚合函数里从各个流中获取相应的字段，并将需要的字段重新组合成一个新的对象存储到 value state 里。这里对每个新的对象都定义一个 timer，用 timer 功能来替代 TimeWindow，窗口结束时将数据发射到下游算子。使用 timer 功能的主要原因是 timer 更灵活，更方便用户自定义，在平台的实用性，可扩展性上表现更好。

MemoryStateBackend vs. RocksDBStateBackend

首先需要说明的是，MemoryStateBackend 不建议在线上使用，这里主要是通过测试量化一下使用 Heap 存储 state 的资源消耗。

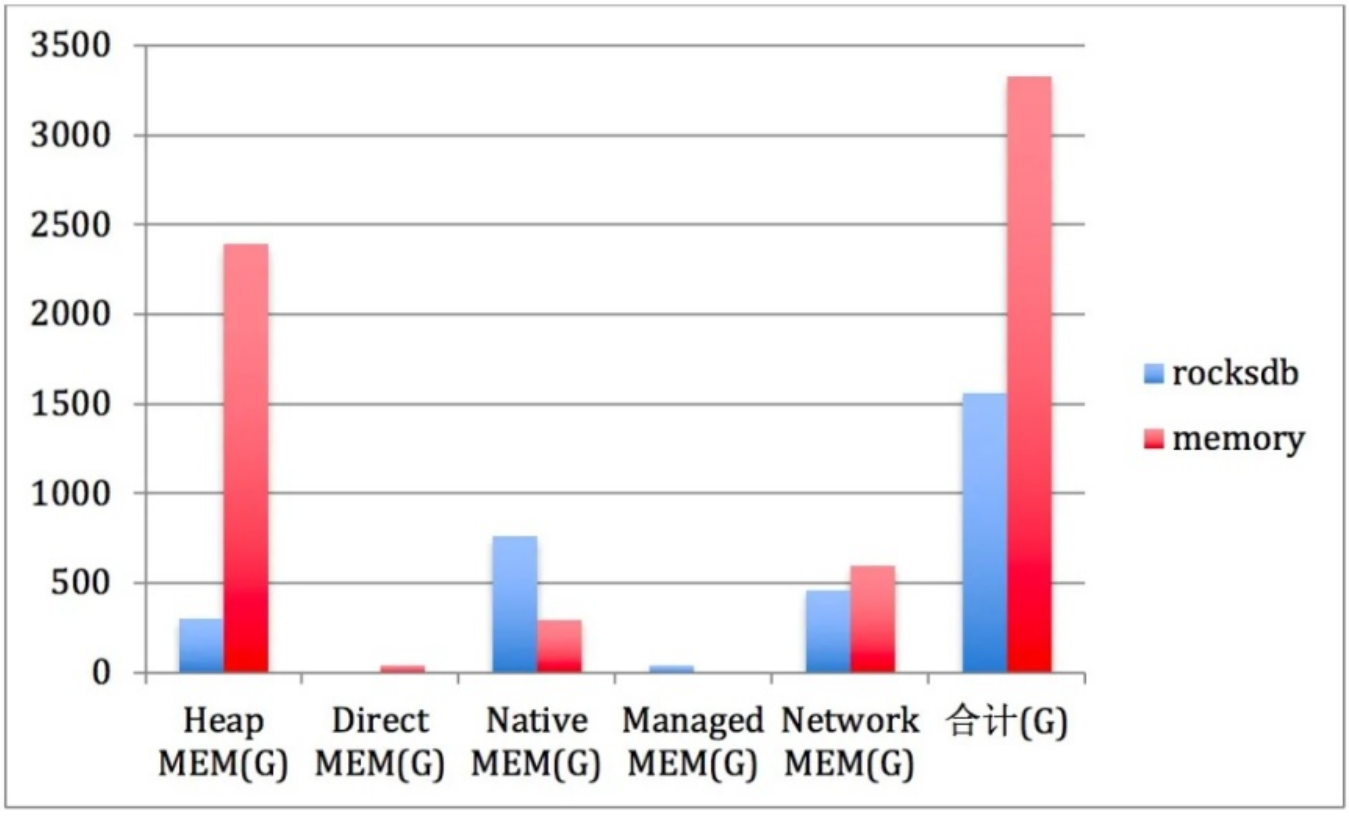
我们在测试中对 checkpoint 的配置如下：

```
CheckpointInterval:10分钟
CheckpointingMode: EXACTLY_ONCE
CheckpointTimeout:3分钟
```

同时对 RocksDB 增加了如下配置：

```
setCompressionType: LZ4_COMPRESSION
setTargetFileSizeBase: 128 * 1024 * 1024
setMinWriteBufferNumberToMerge: 3
setMaxWriteBufferNumber: 4
setWriteBufferSize: 1G
setBlockCacheSize: 10G
setBlockSize: 4 * 1024
setFilter: BloomFilter(10, false)
```

测试发现，相同作业处理相同的数据量时，使用 MemoryStateBackend 的作业吞吐和 RocksDB 类似（输入 qps 为 30 万，聚合后输出 qps 为 2 万），但所需要的内存(taskmanager.heap.mb)是 RocksDB 的 8 倍，对应的机器资源是 RocksDB 的 2 倍。



由此我们得出以下结论：

使用 MemoryStateBackend 需要增加非常多的 Heap 空间用于存储窗口内的状态数据（样本），相对于把数据放到磁盘的优点是处理性能非常好，但缺点很明显：由于 Java 对象在内存的存储效率不高，GB 级别的内存只能存储百兆级别的真实物理数据，所以会有很大的内存开销，且 JVM 大堆 GC 停机时间相对较高，影响作业整体稳定，另外遇到热点事件会有 OOM 风险。
使用 RocksDB 则需要较少的 Heap 空间即可，加大 Native 区域用于读缓存，结合 RocksDB 的高效磁盘读写策略仍然有很好的性能表现。

GeminiStateBackend vs. RocksDBStateBackend

可以通过如下方式，在 Ververica Platform 产品中指定使用 Gemini state backend:

```
state.backend=org.apache.flink.runtime.state.gemini.GeminiStateBackendFactory
```

同时我们对 Gemini 进行了如下基础配置：

```
// 指定Gemini存储时的本地目录
kubernetes.taskmanager.replace-with-subdirs.conf-keys= state.backend.gemini.local
state.backend.gemini.local.dir=/mnt/disk3/state,/mnt/disk5/state
// 指定Gemini的page压缩格式（page是Gemini存储的最小物理单元）
state.backend.gemini.compression.in.page=Lz4
// 指定Gemini允许使用的内存占比
state.backend.gemini.heap.rate=0.7
// 指定Gemini的单个存储文件大小
state.backend.gemini.log.structure.file.size=134217728
// 指定Gemini的工作线程数
state.backend.gemini.region.thread.num=8
```

机器配置

集群规模(台)	cpu(core)	disk	memory(G)	network
16	32	ssd	128G	万兆网卡

作业使用资源对应参数

state	tm	tm.slot	tm.cpu.core	tm.native.memory	tm.heap	jm.heap
Gemini	128	1	1	1G	10G	8G
RocksDB	128	1	1	11G	2G	8G

内存相关参数

	BlockCacheSize	writeBuffer	heap
RocksDB	10G	1G	
Gemini	0	0	8G

对比结果

	抽样比例	样本生成量	used memory	io-util%	cpu	结论
Gemini	50%	15800条/s	89%	5%	69%	样本量正常符合预期且稳定
	55%	17280条/s	97%	9%	72%	样本量正常符合预期且稳定
	60%	11924条/s	99%	100%	58%	反压导致生成样本量减少且不稳定
RocksDB	20%	7200条/s	52%	97%	43%	样本量正常符合预期且稳定
	25%	5771条/s	65%	100%	30%	有反压并且样本量减少
	30%	367条/s	66%	100%	10%	反压导致样本量异常甚至为每秒几百条

Note：全量的样本拼接负载使用 16 台机器无法完全服务，因此我们通过对数据进行不同比例的抽样来进行压测。当出现反压时，我们认为作业已经达到性能瓶颈。

结论

由以上对比可以看出，在数据、作业处理逻辑、硬件配置等都相同的前提下，使用 Gemini 成功处理的数据量是 RocksDB 的 2.4 倍（17280 vs 7200 条/s）。同时通过硬件资源消耗的对比可知，RocksDB 更快达到磁盘 IO 瓶颈，而 Gemini 则具备更高的内存和 CPU 利用率。

作者简介：

曹富强、晨馨，微博机器学习研发中心–高级系统工程师。现负责微博机器学习平台数据计算/数据存储模块，主要涉及实时计算 Flink、Storm、Spark Streaming，数据存储Kafka、Redis，离线计算 Hive、Spark 等。目前专注于Flink/Kafka/Redis在微博机器学习场景的应用，为机器学习提供框架，技术，应用层面的支持。