Join

最终结果

最终结果(左表10000行，右表500行)

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8              320          3733598 ns/op         3676954 B/op      21898 allocs/op
BenchmarkJoinExample-8              322          3694551 ns/op         3677277 B/op      21900 allocs/op
BenchmarkJoinExample-8              322          4119363 ns/op         3677224 B/op      21899 allocs/op
BenchmarkJoinExample-8              324          3661290 ns/op         3676779 B/op      21897 allocs/op
BenchmarkJoinExample-8              327          3660249 ns/op         3676939 B/op      21897 allocs/op
BenchmarkJoin-8                     499          2115215 ns/op         2894243 B/op      21870 allocs/op
BenchmarkJoin-8                     558          2136380 ns/op         2894387 B/op      21871 allocs/op
BenchmarkJoin-8                     570          2104509 ns/op         2894133 B/op      21869 allocs/op
BenchmarkJoin-8                     560          2119230 ns/op         2894341 B/op      21871 allocs/op
BenchmarkJoin-8                     506          2104018 ns/op         2894492 B/op      21871 allocs/op
BenchmarkJoinSort-8                 381          3121695 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 382          3114081 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 379          3111003 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 381          3166979 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 385          3129726 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinMultiThread-8          255          4295962 ns/op         3705358 B/op      21885 allocs/op
BenchmarkJoinMultiThread-8          300          3958031 ns/op         3704992 B/op      21884 allocs/op
BenchmarkJoinMultiThread-8          301          3956462 ns/op         3705071 B/op      21885 allocs/op
BenchmarkJoinMultiThread-8          300          3971926 ns/op         3705119 B/op      21885 allocs/op
BenchmarkJoinMultiThread-8          301          3960465 ns/op         3704806 B/op      21883 allocs/op
```

最终结果(左表7790892行，右表7790892行)

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8                 1       13170333314 ns/op      4745788656 B/op 44464553 allocs/op
BenchmarkJoinExample-8                 1       13304634721 ns/op      4745916744 B/op 44465447 allocs/op
BenchmarkJoinExample-8                 1       14465854950 ns/op      4745919664 B/op 44465351 allocs/op
BenchmarkJoinExample-8                 1       11827192078 ns/op      4745750592 B/op 44464304 allocs/op
BenchmarkJoinExample-8                 1       15400937599 ns/op      4745722264 B/op 44463979 allocs/op
BenchmarkJoin-8                        1        6952821003 ns/op      3491165616 B/op 41884097 allocs/op
BenchmarkJoin-8                        1        4641746283 ns/op      3491112296 B/op 41883726 allocs/op
BenchmarkJoin-8                        1        4919519377 ns/op      3491166656 B/op 41884103 allocs/op
BenchmarkJoin-8                        1        5078487157 ns/op      3491146456 B/op 41883961 allocs/op
BenchmarkJoin-8                        1        8150041166 ns/op      3491164048 B/op 41884215 allocs/op
BenchmarkJoinSort-8                    1       13958175600 ns/op      3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1       13789610982 ns/op      3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1       13173931097 ns/op      3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1       13932589627 ns/op      3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1       14946823252 ns/op      3595449264 B/op 46745425 allocs/op
BenchmarkJoinMultiThread-8             1       15352202407 ns/op      5161280016 B/op 41864384 allocs/op
BenchmarkJoinMultiThread-8             1       13563929009 ns/op      5161257040 B/op 41864355 allocs/op
BenchmarkJoinMultiThread-8             1       22295248847 ns/op      5161320600 B/op 41864673 allocs/op
BenchmarkJoinMultiThread-8             1       17483838728 ns/op      5161324816 B/op 41864820 allocs/op
BenchmarkJoinMultiThread-8             1       18012298539 ns/op      5161341608 B/op 41864810 allocs/op
```

在大数据集上最优的优化了55%左右。

细节优化

使用`bytes.Buffer`代替`[]byte`

在`join_example.go/buildHashTable`和`join_example.go/prob`中，原先使用`[]bytes`通过`apppend`创建`key`，现在使用`bytes.Buffer`来创建`key`。

模拟`join`中构建`key`的过程的benchmark：

```
func BenchmarkBytesBuffer(b *testing.B) {
  buf := &bytes.Buffer{}
  datas := []string{"12345678", "12345678"}
  for i := 0; i < 100000000; i++ {
    for _, data := range datas {
      buf.WriteString(data)
    }
    buf.Reset()
  }
}

func BenchmarkAppend(b *testing.B) {
  buf := make([]byte, 0, 8)
  datas := []string{"12345678", "12345678"}
  for i := 0; i < 100000000; i++ {
    for _, data := range datas {
      buf = append(buf, []byte(data)...)
    }
    buf = buf[:0]
  }
}
```

测试结果如下：

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkBytesBuffer-8                 1       1419387000 ns/op              64 B/op          1 allocs/op
BenchmarkBytesBuffer-8                 1       1381776296 ns/op              64 B/op          1 allocs/op
BenchmarkBytesBuffer-8                 1       1379020850 ns/op              64 B/op          1 allocs/op
BenchmarkBytesBuffer-8                 1       1383089713 ns/op              64 B/op          1 allocs/op
BenchmarkBytesBuffer-8                 1       1376875481 ns/op              64 B/op          1 allocs/op
BenchmarkAppend-8                      1       2086955441 ns/op              16 B/op          1 allocs/op
BenchmarkAppend-8                      1       1821564079 ns/op              16 B/op          1 allocs/op
BenchmarkAppend-8                      1       1848822686 ns/op             664 B/op          4 allocs/op
BenchmarkAppend-8                      1       1816490786 ns/op              16 B/op          1 allocs/op
BenchmarkAppend-8                      1       1809934171 ns/op              16 B/op          1 allocs/op
PASS
ok      join    17.076s
```

可以看到使用`byte.Buffer`有大约30%的性能提升。

在MVMap中存储第一列数值代替存储行数

在`join_example.go/probe`的时候取出符合**Join**条件的行数之后需要对表再访问存储一次，对于此次任务来说可以通过直接存储左表的第一列的值来避免这次访问。不过这样做会导致`buildHashTable`的逻辑和**Join**逻辑耦合。

框架优化

朴素的Sort + Scan Join

使用`Sort + Scan`实现Join算法，分析表中的行的特点可以用`radix sort`来加速`sort`过程，将连接条件中的列数值用`_`拼接，在最后加上两个表的`id`，左表为`0`，右表为`1`，这样能保证排序后满足**Join**条件的左表的行在右表的上面，最后在Id后面加上需要做计算的列。

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8              321          3979700 ns/op         3677127 B/op      21898 allocs/op
BenchmarkJoinExample-8              325          3677553 ns/op         3677046 B/op      21898 allocs/op
BenchmarkJoinExample-8              322          3676268 ns/op         3676944 B/op      21898 allocs/op
BenchmarkJoinExample-8              325          3685063 ns/op         3676990 B/op      21898 allocs/op
BenchmarkJoinExample-8              322          3661654 ns/op         3676863 B/op      21897 allocs/op
BenchmarkJoinSort-8                 388          3080915 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 388          3077406 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 382          3074975 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 386          3104763 ns/op         1730400 B/op      31538 allocs/op
BenchmarkJoinSort-8                 386          3089899 ns/op         1730400 B/op      31538 allocs/op
PASS
ok      join    16.844s
```

可以看到在左表10000行，右表500行的时候，sort的性能比原有的Hash Probe的方法好一点，主要原因是建立Hash表的时间大于小数据集上的`radix sort`时间。

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8                 1        13727104449 ns/op       4745792184 B/op 44464443 allocs/op
BenchmarkJoinExample-8                 1        12338778452 ns/op       4745842136 B/op 44464803 allocs/op
BenchmarkJoinExample-8                 1        14284431679 ns/op       4745744864 B/op 44464263 allocs/op
BenchmarkJoinExample-8                 1        11064060866 ns/op       4745704824 B/op 44463984 allocs/op
BenchmarkJoinExample-8                 1        14324512819 ns/op       4745738696 B/op 44464097 allocs/op
BenchmarkJoinSort-8                    1        13771684966 ns/op       3595449360 B/op 46745426 allocs/op
BenchmarkJoinSort-8                    1        12114632105 ns/op       3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1        12468568548 ns/op       3595449360 B/op 46745426 allocs/op
BenchmarkJoinSort-8                    1        12168030353 ns/op       3595449264 B/op 46745425 allocs/op
BenchmarkJoinSort-8                    1        12164721885 ns/op       3595449264 B/op 46745425 allocs/op
PASS
ok      join    130.694s
```

上图为在200M大数据集（左表右表都是200M的数据集）进行**Join**的测试结果，发现Sort Join和原有的Hash Probe性能差别不大。

并行Probe

对于**No Partition Hash Join**先排除并行`BuildHashTable`的做法，因为Golang没有一个类似Java的ConcurrentMap的好的实现，这将会导致对`Map`的并发写入性能十分低下。而`Probe`过程是个只读过程，不同线程的读取不会影响其他线程，所以可以将右表切分为多块，然后并行probe，最后获得结果。

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8               297          4066145 ns/op         3677359 B/op      21900 allocs/op
BenchmarkJoinExample-8               321          3730632 ns/op         3676961 B/op      21898 allocs/op
BenchmarkJoinExample-8               313          3876992 ns/op         3677103 B/op      21898 allocs/op
BenchmarkJoinExample-8               314          3877282 ns/op         3676921 B/op      21898 allocs/op
BenchmarkJoinExample-8               319          3690531 ns/op         3677182 B/op      21899 allocs/op
BenchmarkJoinMultiThread-8           306          4057527 ns/op         3704834 B/op      21883 allocs/op
BenchmarkJoinMultiThread-8           300          3989178 ns/op         3705179 B/op      21884 allocs/op
BenchmarkJoinMultiThread-8           302          4045798 ns/op         3704850 B/op      21883 allocs/op
BenchmarkJoinMultiThread-8           298          3981878 ns/op         3705211 B/op      21885 allocs/op
BenchmarkJoinMultiThread-8           301          3969436 ns/op         3704875 B/op      21883 allocs/op
PASS
ok      join    17.859s
```

这是左表10000行右表500行的测试结果，发现性能甚至比单线程`prob`更慢，这是因为右表过小，多线程的优势没有体现出来，交换一下左右表的大小效果会很明显。

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8                 1        13608223176 ns/op       4745708792 B/op 44463863 allocs/op
BenchmarkJoinExample-8                 1        10892037663 ns/op       4745908352 B/op 44465272 allocs/op
BenchmarkJoinExample-8                 1        16225355084 ns/op       4745771352 B/op 44464320 allocs/op
BenchmarkJoinExample-8                 1        13758195150 ns/op       4745864776 B/op 44464970 allocs/op
BenchmarkJoinExample-8                 1        16280310981 ns/op       4745924168 B/op 44465502 allocs/op
BenchmarkJoinMultiThread-8             1        8653106695 ns/op        5161266056 B/op 41864288 allocs/op
BenchmarkJoinMultiThread-8             1        8034771773 ns/op        5161292824 B/op 41864480 allocs/op
BenchmarkJoinMultiThread-8             1        8118463895 ns/op        5161286224 B/op 41864432 allocs/op
BenchmarkJoinMultiThread-8             1        7742048654 ns/op        5161244984 B/op 41864142 allocs/op
BenchmarkJoinMultiThread-8             1        8721361106 ns/op        5161255064 B/op 41864345 allocs/op
PASS
ok      join    114.559s
```

这是左表200M右表200M的测试结果，此时`probe`过程占比更多，多线程优化了`probe`过程，导致整体性能提升了30%左右。

边读边计算

```
         .          .     12:
         .          .     13:// JoinExample performs a simple hash join algorithm.
         .          .     14:func JoinExample(f0, f1 string, offset0, offset1 []int) (sum uint64) {
         .      4.27s     15:   tbl0, tbl1 := readCSVFileIntoTbl(f0), readCSVFileIntoTbl(f1)
         .      2.59s     16:   hashtable := buildHashTable(tbl0, offset0)
    1.10s      1.10s     17:   for _, row := range tbl1 {
         .      2.39s     18:           rowIDs := probe(hashtable, row, offset1)
      10ms       10ms     19:           for _, id := range rowIDs {
      30ms       30ms     20:                   v, err := strconv.ParseUint(tbl0[id][0], 10, 64)
                          21:                   if err != nil {
```

通过**pprof**的统计数据我们发现读取文件耗费的时间和`buildHashTable`，`probe`的耗费时间相当。可以考虑并行化IO和计算，使用**边读边计算**的生产者消费者模型来并行读取文件和计算过程，在读取一定数量（比如一个磁盘的Block大小）的数据后将数据发送到计算线程计算。

显然生产者线程只有一个，因为磁盘读写并行起来蠢，因为文件是连续的，相比于单线程的顺序读取，并行读取反而会造成更多的性能损失，而消费线程可以有很多个，消费线程的消耗能力大于生产线程的话也会造成消费线程空等的状态（不考虑Golang调度的影响），根据pprof的分析，在我的机器中磁盘读速度和计算速度相差不大，并且多个计算线程执行并行`buildHashTable`的话也会造成额外的同步开销，所以选择一个生产线程一个消费线程的模型。

生产者1 消费者1

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8              321          3674691 ns/op         3676906 B/op      21897 allocs/op
BenchmarkJoinExample-8              325          3668092 ns/op         3677192 B/op      21898 allocs/op
BenchmarkJoinExample-8              324          3662287 ns/op         3676975 B/op      21898 allocs/op
BenchmarkJoinExample-8              324          3657887 ns/op         3676939 B/op      21898 allocs/op
BenchmarkJoinExample-8              326          3661342 ns/op         3676942 B/op      21897 allocs/op
BenchmarkJoin-8                     555          2134215 ns/op         2894240 B/op      21870 allocs/op
BenchmarkJoin-8                     559          2355075 ns/op         2894443 B/op      21870 allocs/op
BenchmarkJoin-8                     550          2127898 ns/op         2894201 B/op      21870 allocs/op
BenchmarkJoin-8                     555          2150771 ns/op         2894355 B/op      21871 allocs/op
BenchmarkJoin-8                     559          2115890 ns/op         2894034 B/op      21869 allocs/op
PASS
ok      join    15.731s                                                                  _
```

这是左表10000行右表500行的测试结果，并行化IO和计算之后性能接近一倍的提升，这是符合预期的，因为在上面的pprof性能分析中`buildHashTable`和`probe`的时间耗费和读取文件相当，并行这个过程之后将会是总时间减少一半。

```
go test -bench Benchmark -run xx -count 5 -benchmem
go: finding github.com/pingcap/tidb v2.1.7+incompatible
go: finding github.com/pingcap/tidb v2.1.7+incompatible
goos: darwin
goarch: amd64
pkg: join
BenchmarkJoinExample-8                 1        13437257577 ns/op       4745809704 B/op 44464690 allocs/op
BenchmarkJoinExample-8                 1        12043272419 ns/op       4745856776 B/op 44465040 allocs/op
BenchmarkJoinExample-8                 1        12609029415 ns/op       4745724024 B/op 44463995 allocs/op
BenchmarkJoinExample-8                 1        12357209056 ns/op       4745838872 B/op 44464916 allocs/op
BenchmarkJoinExample-8                 1        11980866286 ns/op       4745798624 B/op 44464511 allocs/op
BenchmarkJoin-8                        1         4569365523 ns/op       3491123520 B/op 41883805 allocs/op
BenchmarkJoin-8                        1         4706474396 ns/op       3491173104 B/op 41884144 allocs/op
BenchmarkJoin-8                        1         4614517089 ns/op       3491131640 B/op 41883862 allocs/op
BenchmarkJoin-8                        1         4477663827 ns/op       3491171160 B/op 41884138 allocs/op
BenchmarkJoin-8                        1         4505070390 ns/op       3491184176 B/op 41884223 allocs/op
PASS
ok      join    87.175s                                                                  _
```

在大数据集上性能提升更加明显。