Kafka Compression Performance Tests

Backgroud

`Kafka` use `End-to-End` compression model which means that `Producer` and `Consumer` are doing the compression and de-compression jobs. This feature enables the reduction of on-the-fly network costs and the `Broker` will increase its cpu load.

Environment

Hardware Box

CPU	Memory	Disk
2.5 GHz Intel Core i7	16GB	512GB SSD

Software Box

Kafka	JDK	Scala	Broker	Producer	JVM
0.8.2.1	1.7.0u75	2.11	1	1	-Xms4G -Xmx4G -Xmn2G

Kafka Configuration

Replica	Partition
1	1

Messages Content

The content I used to send to `Kafka` is a nginx log which contains 607,781 lines and 200MB. Each line is like below:

```
1   127.0.0.1 - - [24/Mar/2015:15:57:09 +0800] "GET /login?gotype=2 HTTP/1.1" "0.002" 200 3177 "http://abc.com/URLhtml" "Mozilla/4.0 (compatible
```

```
1   $wc -l passport.access.log
2     607781 passport.access.log
3   ~/Downloads
4   $du -sh passport.access.log
5   200M    passport.access.log
```

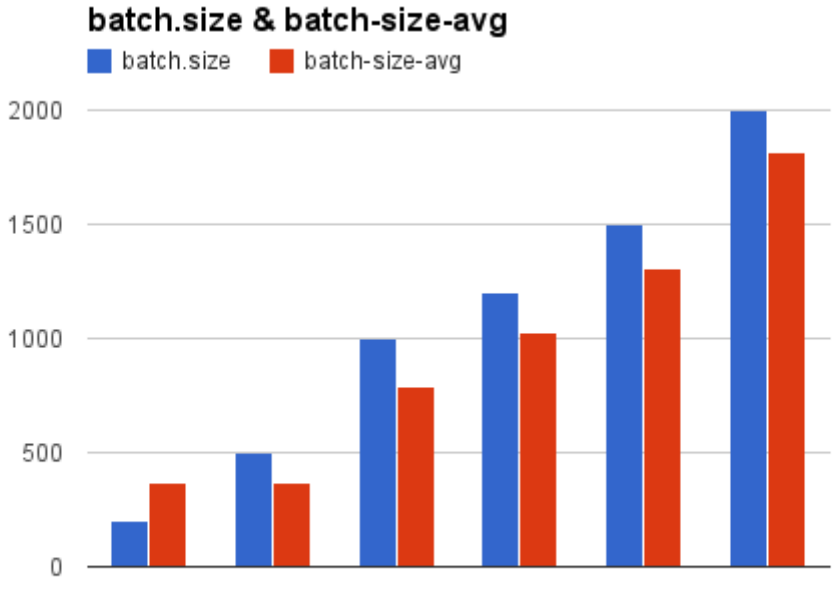
Baseline Test

Kafka Producer Configuration

compression.type	buffered.memory	acks	linger.ms
None	32MB	1	0

Test Result

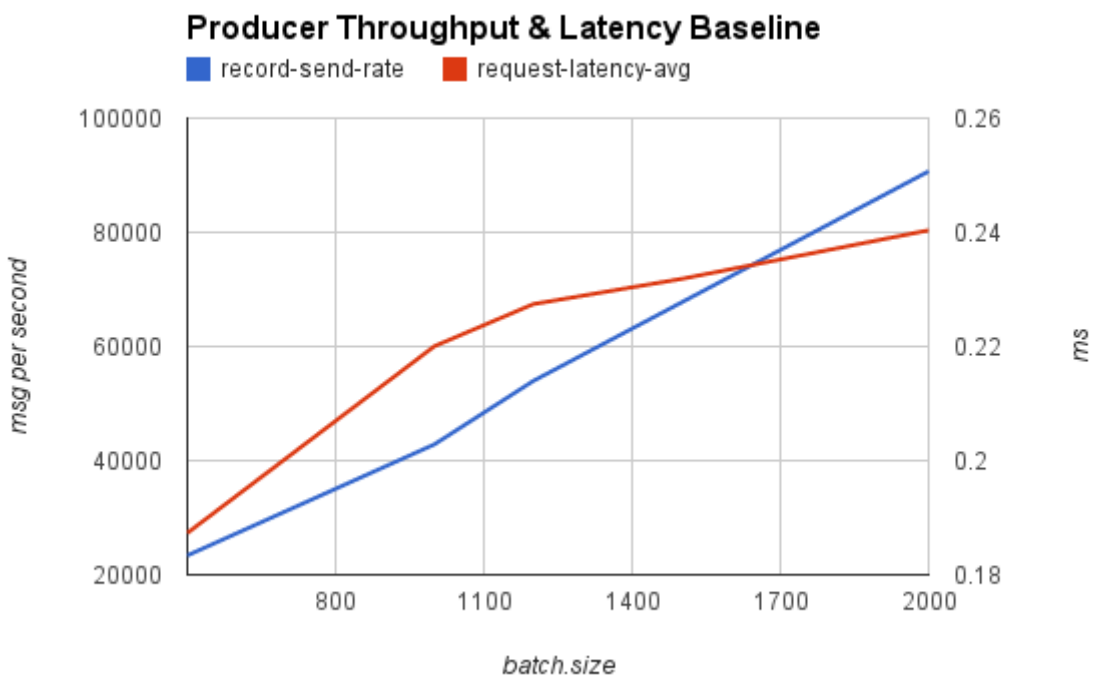
The first test is going to figure out the fact that the value we defined in `batch.size` is not the exactly batch size `Producer` will use. I got the real batch size of `Producer` by using its metrics API. Tests have been done with `batch.size` of 200, 500, 1000, 1200, 1500, 2000. Chart below is the result:



We can see clearly that `batch.size` is a smaller than `batch-size-avg` (it is the excatly batch size `Producer` used). And while the `batch.size` is 200 and 500, the `batch-size-avg` is almost the same - aroud 370. That is to say, `batch-size-avg` is not only settled by the parameter `batch.size`, furthermore, it is also related to other factor.

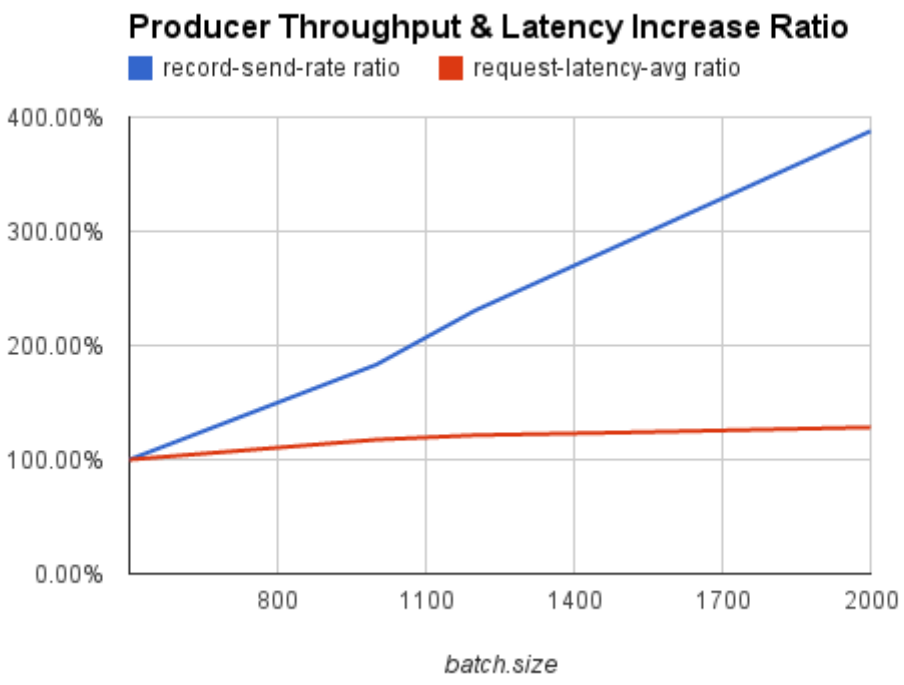
All tests have been done in different `batch.size` of 500, 1000, 1200, 1500 and 2000. Why I didn't use `batch.size` of 200 because the phenomenon in the above paragraph.

Let's take a look on the `Producer` baseline：



The throughput rised along with the increasing of `batch.size`. And the latency rised as well. Let me make a simple math calculation. I use the `batch.size` of 500 for the base number and get the percentage of throughput and latency increasement.

-	500	1000	1200	1500	2000
throughtput	100%	183.25%	230.74%	289.59%	387.81%
latency	100%	117.47%	121.40%	123.76%	128.31%



Conclusion

The throughtput of `Producer` can get much higher with the increasing of `batch.size`. And at the meantime, latency will increas in a very tiny level.

Producer Performance with Compression

`Kafka` support three different compression type：

- gzip
- snappy
- lz4

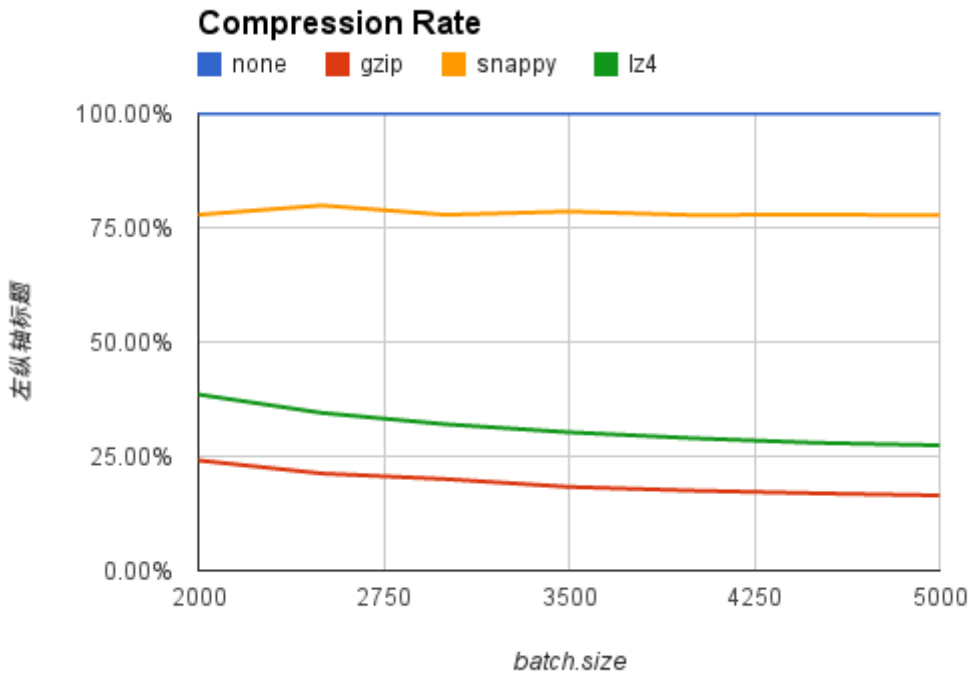
Because of the box I use to test, when I use `lz4` in testing, I got OOM exception. I think the casuse of this exception is the Java code will get all lines of the `passport.access.log` and send to `Kafka` in a very short time. So I used `batch.size` of 2000, 2500, 3000, 3500, 4000, 4500 and 5000. Using big `batch.size` will increass the latency and this will give JVM some time to do the GC jobs.

Below is the result

Compression Rate

The value is better when it is smaller

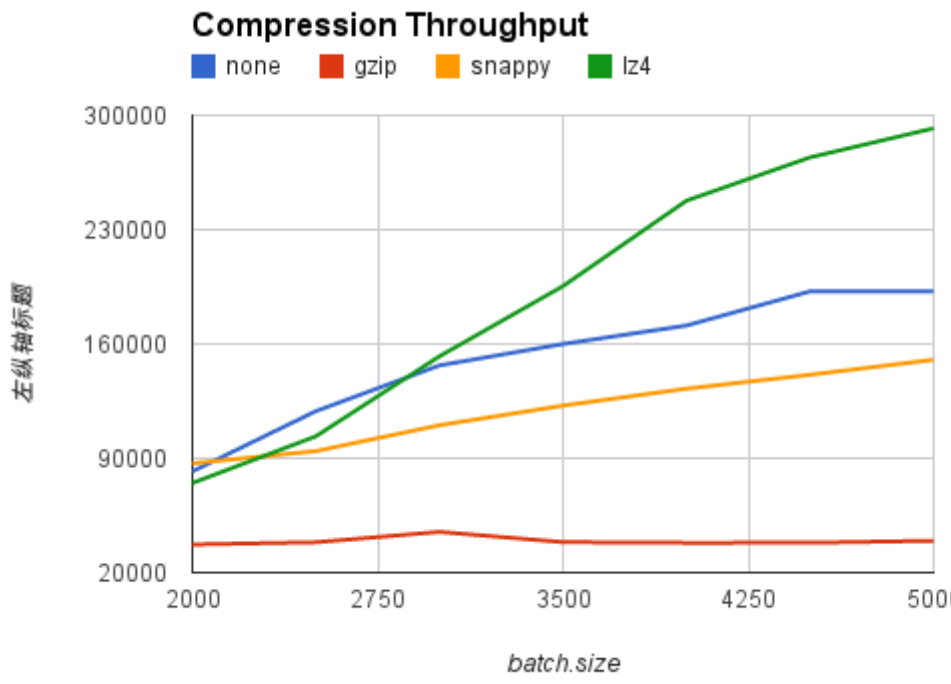
-	none	gzip	snappy	lz4
Average Compression Rate	100%	19.21%	78.19%	31.37%



Throughput

When I enabled the compression, the througput decreased. Let's see the result.

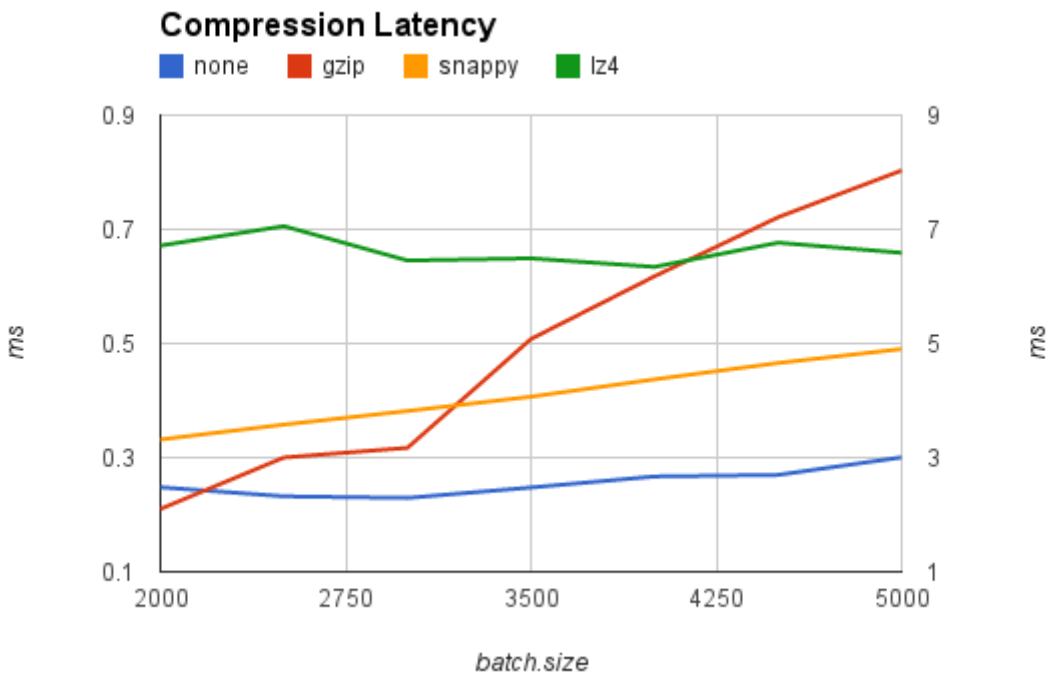
-	none	gzip	snappy	lz4
Average Throughtput	151901.1038	39346.00017	119707.5266	191469.8994



Latency

Result:

-	none	gzip	snappy	lz4
Average Latency ms	0.25	4.97	0.41	0.66
Ratio	100.00%	1937.61%	159.99%	258.62%



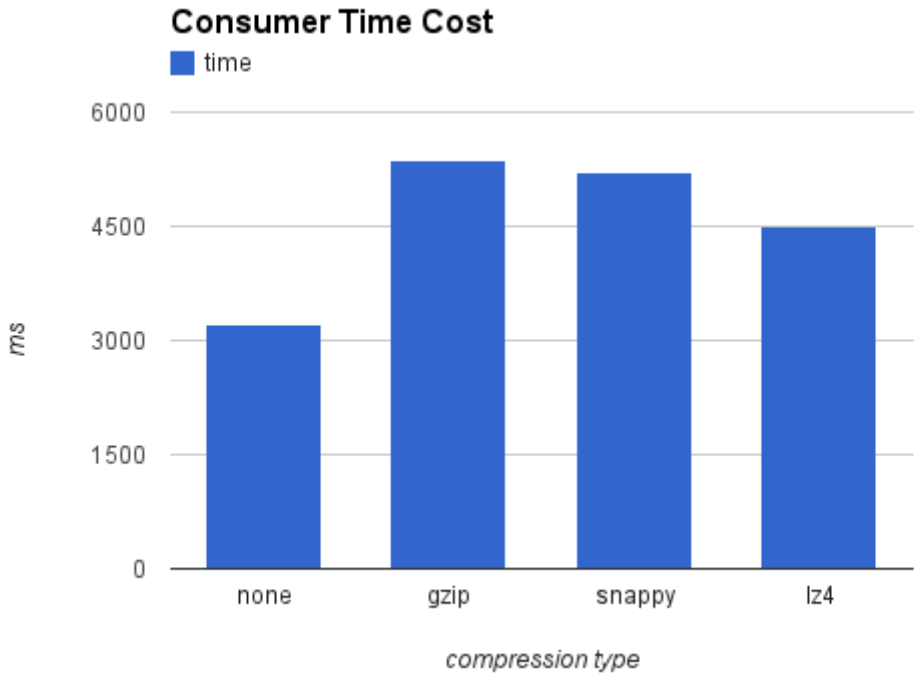
Consumer Performance with Compression

Because `Kafka` 0.8.2.1 has not enabled the new `Consumer`, so I cannot get the detailed metrics like I have got in `Producer`. So in this test, I use the total time `Consumer` used to consumes a fixed number of messages to do the benchmark.

The messages are `test.access.log` in twice size. Below is the result

-	none	gzip	snappy	lz4
Time Cost ms	3218	5374	5216	4507

-	none	gzip	snappy	lz4
Time Cost Increase Rate	100%	167%	162.09%	140.06%



We can see `lz4` is the fastest.

Conclusion

`lz4` is the best choice in compression rate and both performance of `Producer` and `Consumer`. Below is the comparison of `lz4` performance and base line performance. For simplified the chart, this only shows the result with `batch.size` of 5000.