Kafka Streams Performance Summary

April 6, 2016
Alex Cook (cooka@us.ibm.com)

Intro

This series of performance tests was conducted to compare the difference in performance between the Streams 4.1 consumer (toolkit version 3.0) and the rewrite of the consumer using the Kafka 0.9 consumer API (toolkit version 4.0). Performance tests were also run to benchmark the Kafka Producer (which did not change from toolkit version 3.0 to 4.0).

Executive Summary

- The Kafka 0.9 Consumer is faster or equivalent while using less CPU and putting less strain on the Kafka broker.
- For messages 1k or larger, the Kafka broker was the limiting factor (broker couldn't handle more messages than could be consumed).
- Consuming and producing messages of type blob is ~10% faster than rstring/ustring.
- The effect of consistent region on performance is negligible for message sizes over 512 bytes. For message size of 100 bytes (consumption rates ~200k), there is a 10% slow-down when checkpointing on 30-second intervals.
- We observed fluctuating performance when benchmarks were run over a long period of time. We concluded that the fluctuation was disk I/O related on the Kafka broker server, as changing the location of the message log to /dev/shm (ramdisk) led to steady results.
- Increasing the size of the max.partition.fetch.bytes consumer config led to significant improvement in throughput for 512-byte messages, while increasing the value for receive.buffer.bytes was observed to make a very small improvement.

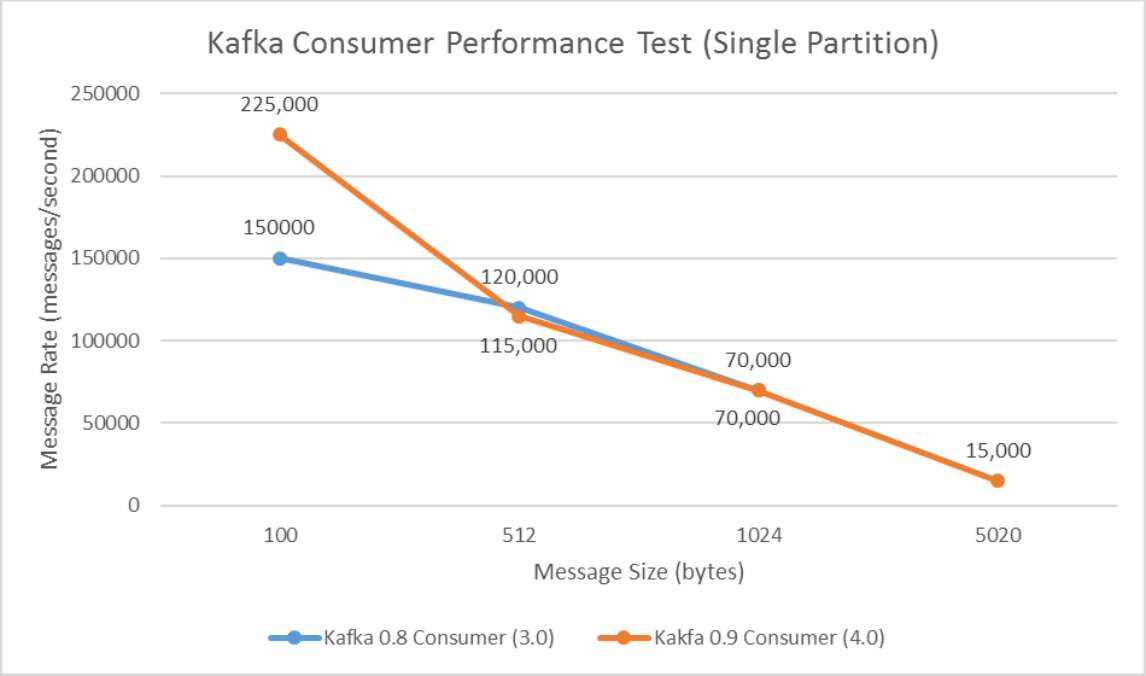
Results and Conclusions

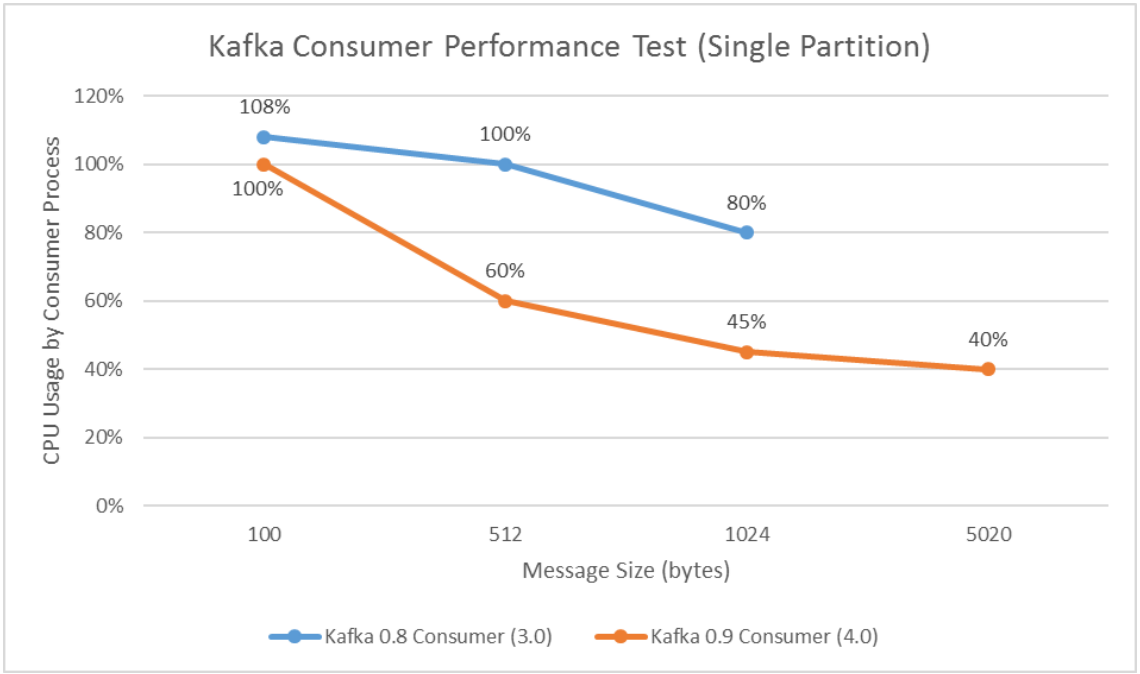
Basic Setup:

- 3 dedicated 8-core(Intel(R) Xeon(R) CPU X5570 @ 2.93GHz) , rhel6, x86 hosts (1 Kafka 0.9.0 broker server, 1 Streams management host, 1 Streams application host). Disk: Seagate Constellation ST9500530NS (500GB, 7200 RPM, 32MB Cache, SATA 3.0Gb/s, 2.5")
- Used non-dedicated XIV Fibre Channel storage server for Kafka/Streams Zookeeper. The disks were commodity 7200 RPM drives with non-volatile cache for fast storage.
- 1 Gb shared network switch
- Default server configurations were used except for log retention was set to 1,073,741,824 bytes.

Kafka Consumer Single Partition Test

Test Overview: The following two charts show the results for a single Kafka consumer in Streams reading byte array (blob) messages from a single partition. The Streams host contains the Kafka Consumer operator, as well as performance measurement operators. Measurements of message rate are aggregated over 30 second intervals.

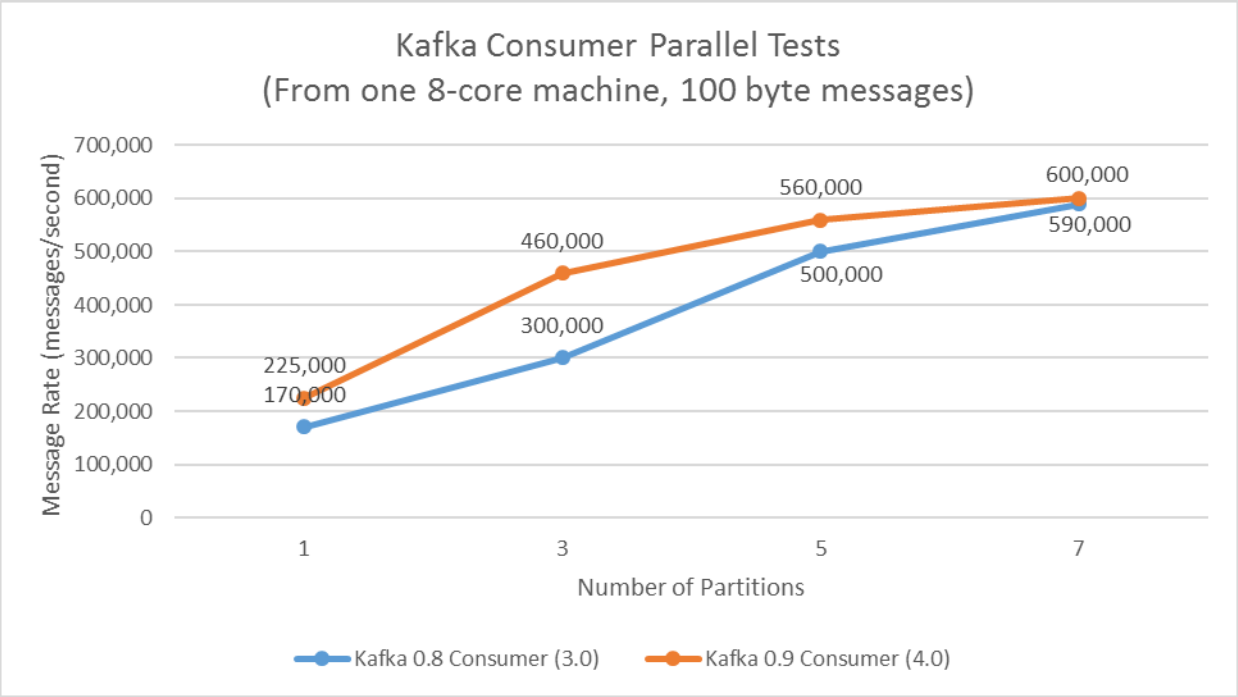




- Kafka 0.9 consumer is faster, or about the same as the Kafka 0.8 consumer.
- For the same rate of consumption, Kafka 0.9 consumer utilizes significantly less CPU.
- Kafka 0.9 consumer puts a significantly smaller strain on the Kafka broker it is consuming from than 0.8 consumer (we observed lower broker CPU use for equivalent data rates).
- For messages 1k or larger, the Kafka broker was the limiting factor (we could no longer push more messages than could be consumed).
- Results are equivalent for the String version of this test, but with 10% lower performance.

Kafka Parallel Consumption

Test Overview: The following two charts show the results for parallel consumers reading from multiple partitions. Topics of partition size 1, 3, 5, and 7 were tested with the corresponding number of consumers located on the single Streams host. The Streams host also included performance measurement operators. 7 Kafka brokers were setup, resulting in at most one partition per broker.



- Parallel consumption on a single, 8-core host (multiple Kafka brokers) scales relatively well for first 3 partitions, but then slows. We hypothesize that the slowdown is related to the limitations of using a 1 Gb network switch.

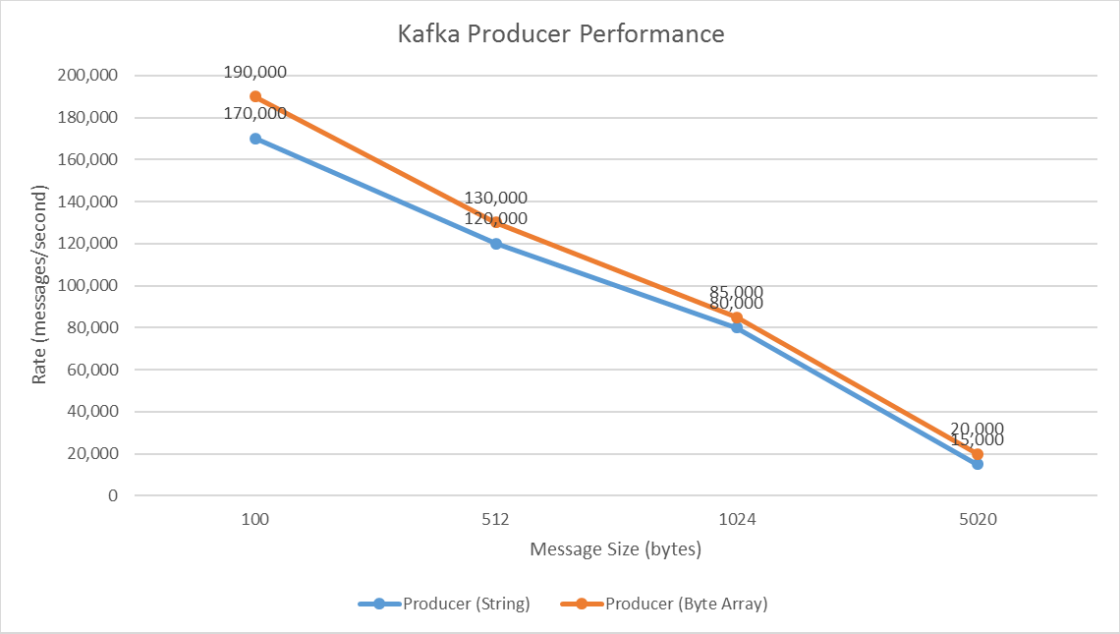
Kafka Consistent Region Consuming

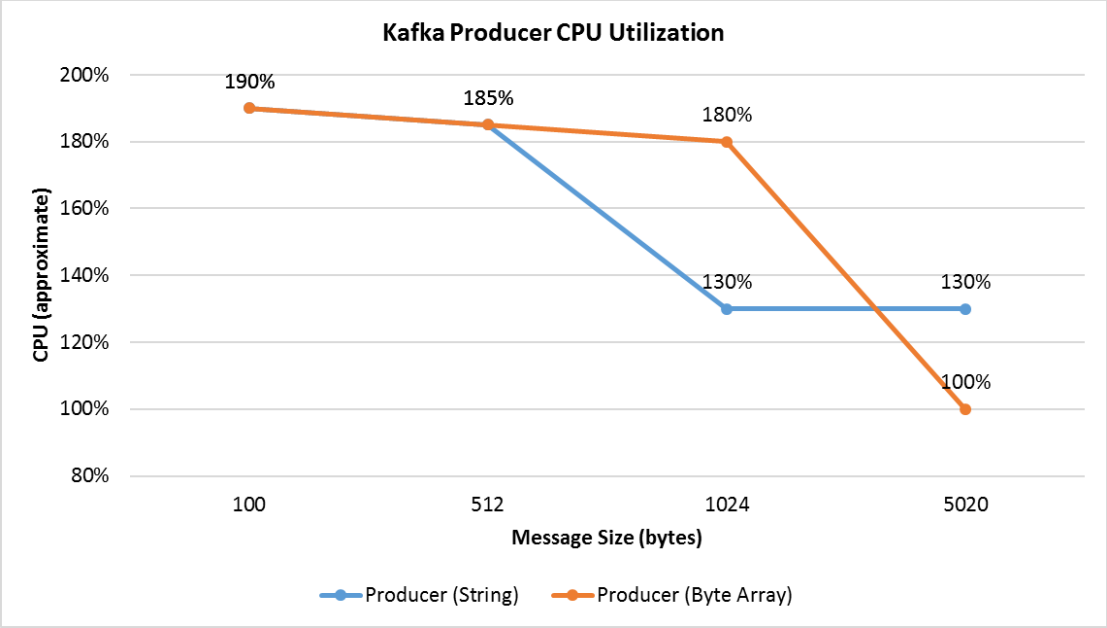
Test Overview: The single-partition consumer tests described above were run using consistent region with a checkpointing period of 30 seconds.

- For a checkpointing period of 30 seconds, the effect of consistent region on the performance is negligible for message sizes over 512 bytes. For message size of 100 bytes (consumption rates ~200k), there is a 10% slow-down.

Kafka Producer

Test Overview: The following chart shows the results for a single producer sending byte array (blob) and String (rstring) messages to a single partition without message acknowledgment. The Streams host contains the Kafka Producer operator, as well as performance measurement operators.

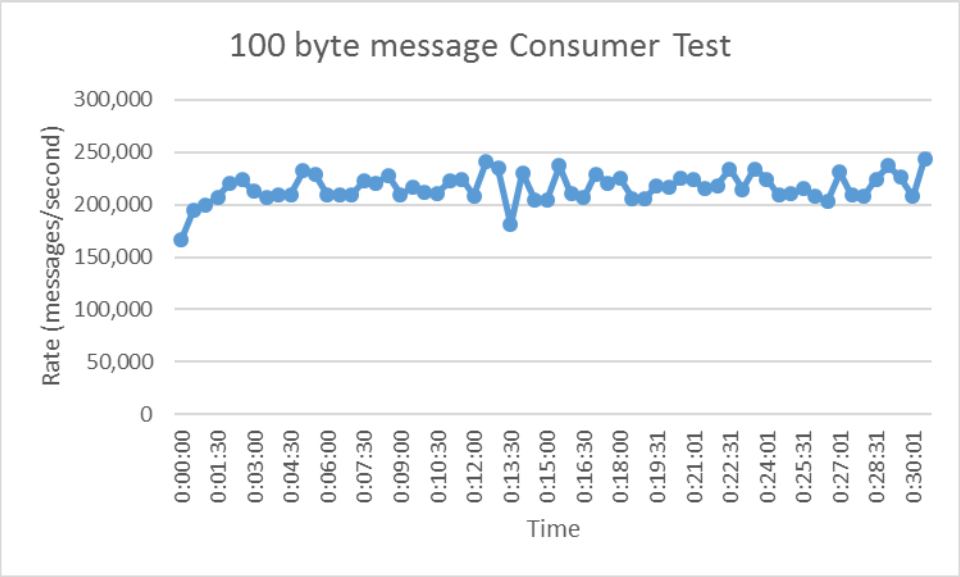




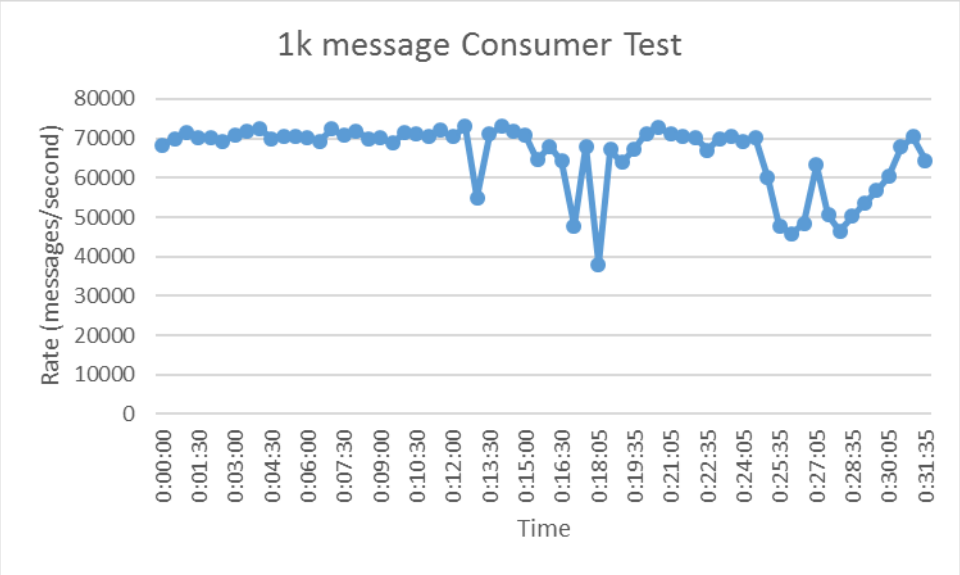
- For messages 1k and larger, the Kafka broker is the limiting factor. CPU for the producer goes down since the server can't keep up with what the producer is capable of.

Kafka Server Performance

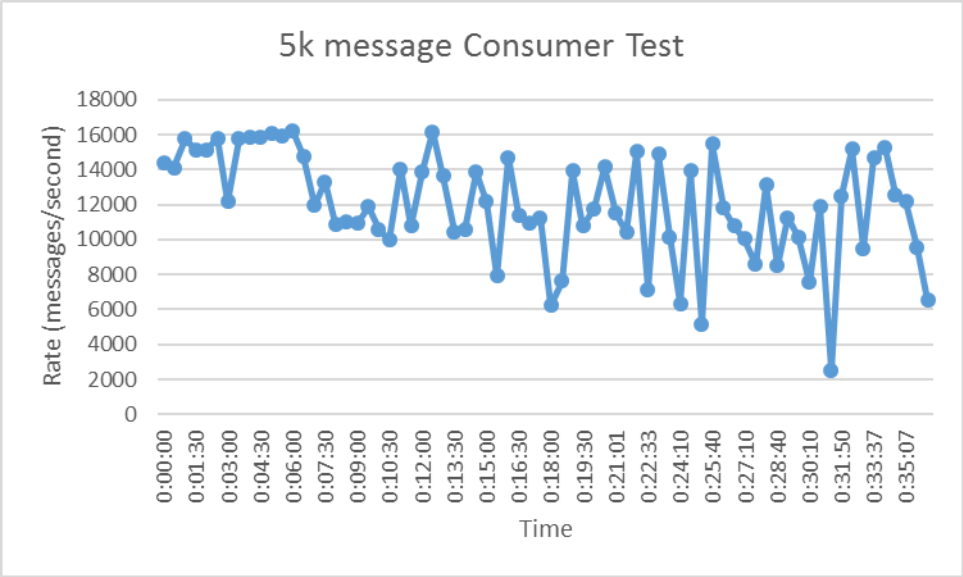
Test Overview: To observe how steady performance was over time, we ran consumer test measurements for at least 30 minutes. In many cases, we ran tests for an entire weekend. **Variability in message rate was concluded to be the result of disk I/O operations on the Kafka broker host.**



For 100 byte messages, the server is not pushed to its limits, and rate is reasonably steady.



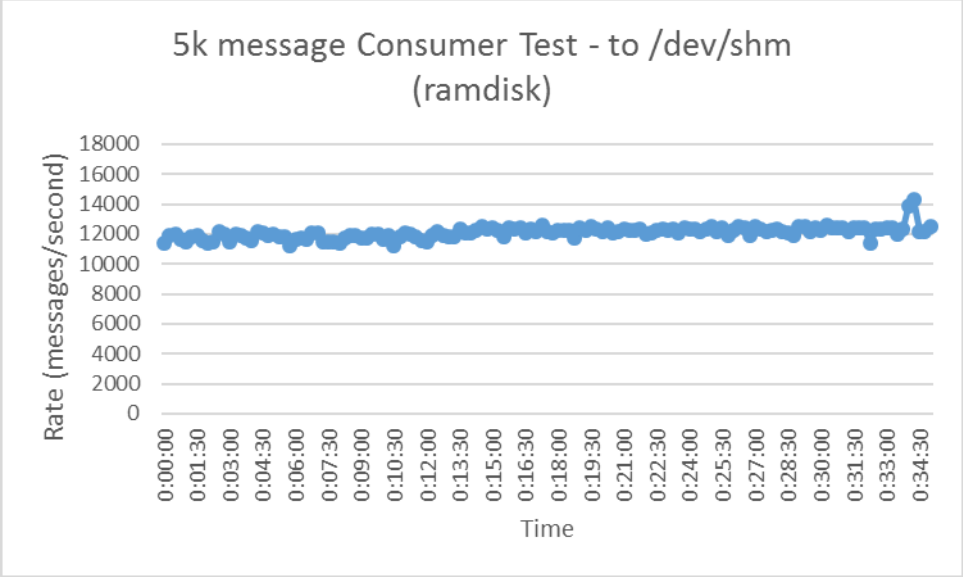
For 1k messages, the server starts to show cases of significant performance variability.



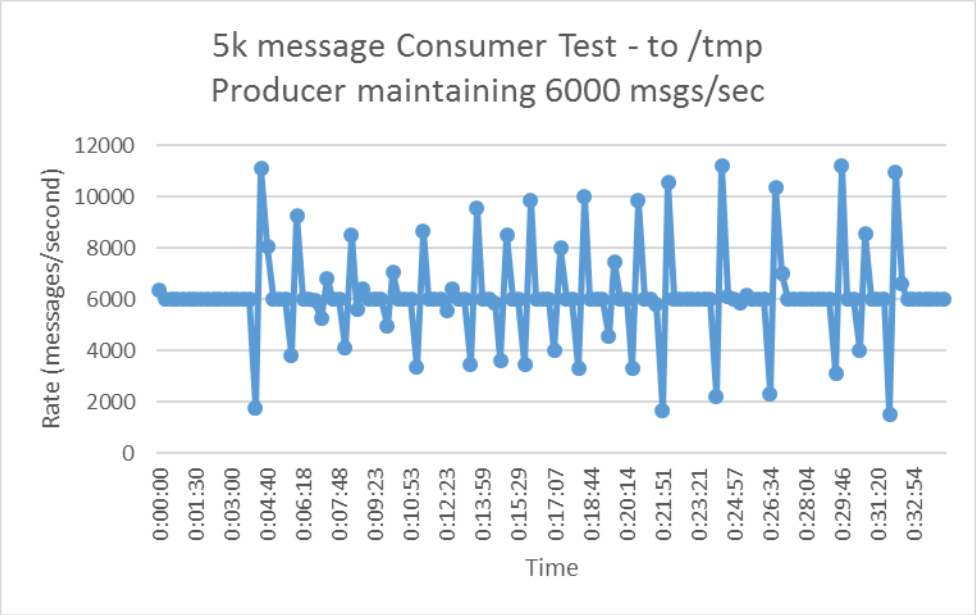
For 5k messages, the server is under an even higher load and performance becomes highly variable.

Performance Variability Discussion

We were able to conclude that performance variability was the result of disk I/O operations. This was proven by setting the Kafka message log directory to a ramdisk location (/dev/shm/), which resulted in steady performance. **Note: We do not recommend writing your Kafka message logs to a ramdisk location. You risk losing your data in the case of a system reboot.**



We believe that there is a Kafka process that periodically blocks message read/writes. This temporary performance drop is observable even when message rates are reduced to levels that are easy for the server to handle, however the broker recovers quickly. The following graph shows a test where message rates were observed at a 15 second interval while a producer maintained a message rate of 6,000 messages/second. In the case of this slower message rate, the server almost immediately catches up and returns to a steady state.

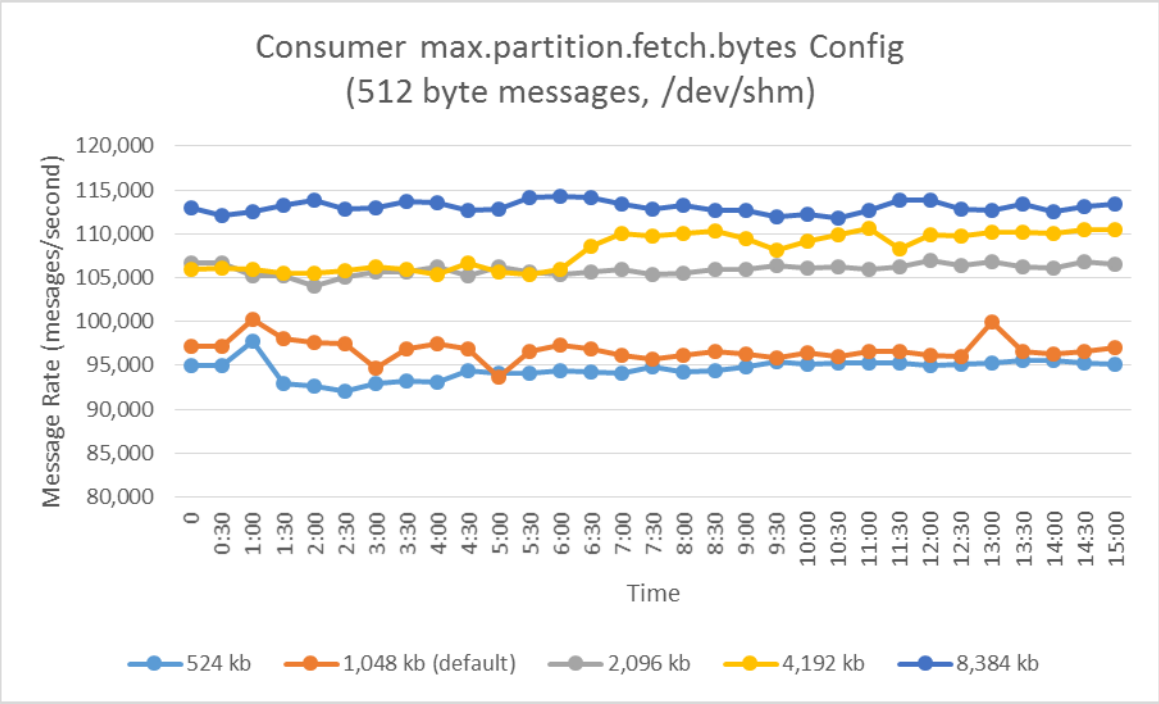


Performance Effects of Consumer Configurations

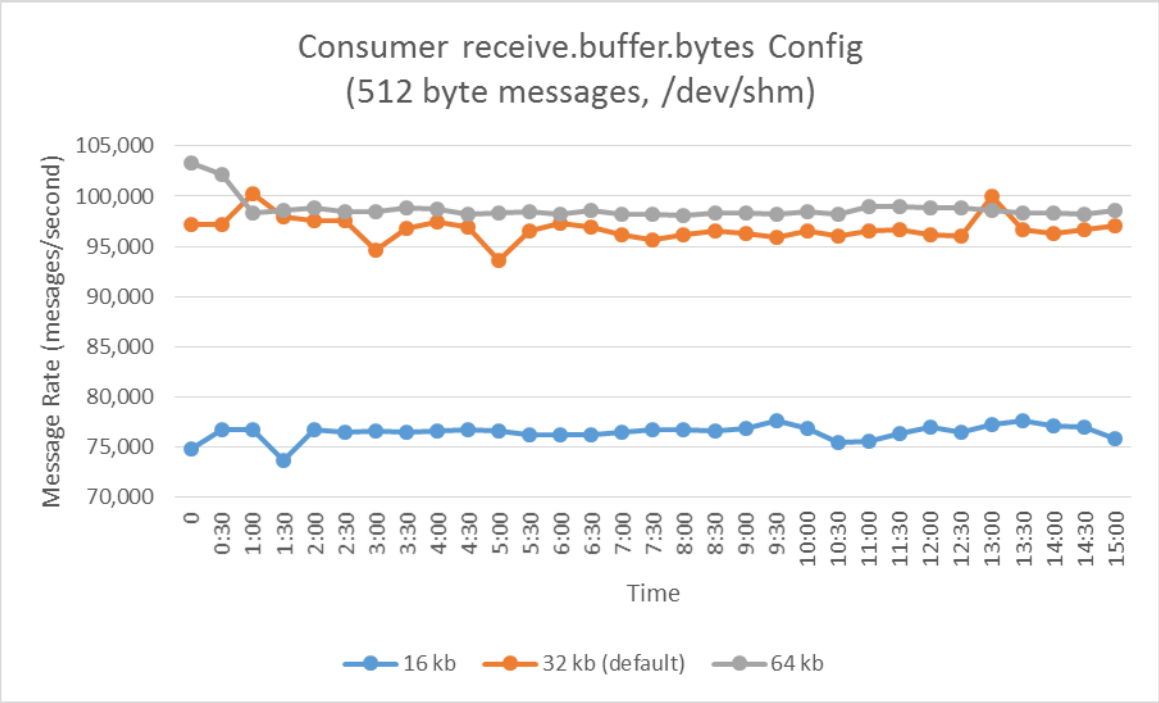
Before figuring out how to avoid the disk I/O problem discussed in the previous section, observing differences when varying consumer configurations was unreliable. Writing output to ramdisk allowed us to observe these differences.

Test Overview: We ran the Consumer test for 512-byte messages with the server writing to /dev/shm (ramdisk). We tried varying **max.partition.fetch.bytes** and **receive.buffer.bytes** (see Kafka Config documentation for details) and ran each version of the test for at least 15 minutes.

- **max.partition.fetch.bytes** - The maximum amount of data per-partition the server will return. The maximum total memory used for a request will be # of partitions * max.partition.fetch.bytes.
- **receive.buffer.bytes** - The size of the TCP receive buffer (SO_RCVBUF) to use when reading data.



Increasing the maximum partition fetch size led to increased throughput. Doubling the default value of 1,048 kb led to a ~10% improvement in throughput. Doubling again to 4,192 kb provided diminishing returns of about 5% improvement in performance. Further doubling led to further diminishing returns on throughput.



We were only able to very marginally improve results by doubling the buffer size. Halving the buffer size dropped throughput by 25-30%.

Recommendations

Based on our findings, we give the following recommendations:

1. **Move to Kafka 0.9 broker and clients.** The clients are faster, consume less CPU, and put less strain on the broker on a per-message basis.
2. **For big messages, deploy more brokers; for small messages, deploy more consumers.** For a single-partition topic, messages 1k or larger led to the Kafka broker being the limiting factor. In our case, creating multiple partitions in that topic, but keeping them on the same broker would not lead to improved performance since we are already pushing the broker to its limit.
3. **If you want to make your consumer faster, try increasing the size of max.partition.fetch.bytes.** We saw improvements in throughput of 10-15%.