Apache Kafka on IBM zSystems – Performance Experiences, Hints and Tips

Marc Beyerle (marc.beyerle@de.ibm.com)
Senior Java Performance Engineer, IBM Mainframe Specialist

Document version: 1.0
Document date: 2022-06-24

Notices and disclaimers

© 2022 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights – use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Notices and disclaimers, *continued*

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and DS8000, Easy Tier, IBM zSystems, and z/OS are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names and logos might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml

Agenda

- Introduction to Apache Kafka
- Performance measurement and tuning approach
- Observations and recommendations
- Summary

About Apache Kafka

- Quote from the official homepage: *"Apache® Kafka® is an open-source distributed **event streaming platform** used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications."*

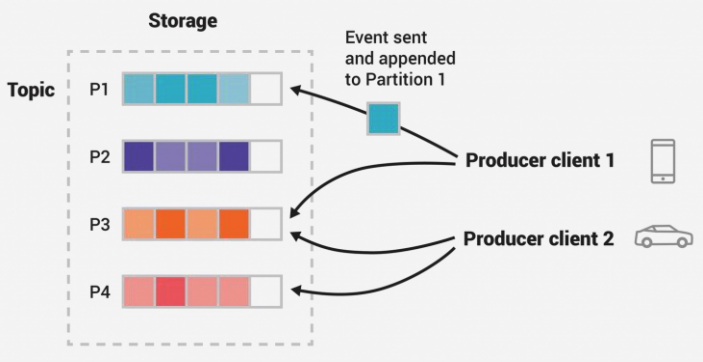
- At first sight, Kafka seems to be like a ***message queueing system***, but there are some ***major differences***
  - Messages are called ***events;*** single events cannot be deleted; ***pub-sub*** approach, etc.
  - At its core, Kafka is a ***distributed and partitioned commit log***

- Originally developed at ***LinkedIn®***, now part of the *Apache Software Foundation* (ASF)
  - ***Confluent® Inc.***: main contributor to Kafka and service / add-on provider

- Written in Java® and Scala

About Apache Kafka, *continued*

- Events are stored in ***topics***, and topics are ***partitioned***
- Data is replicated across multiple Kafka ***brokers***
  - Replication is based on partitions
- Use cases (https://kafka.apache.org/uses):
  - Messaging
  - Website activity tracking
  - Metrics (operational monitoring data)
  - Near real-time analytics, fraud detection, pricing anomalies, etc.



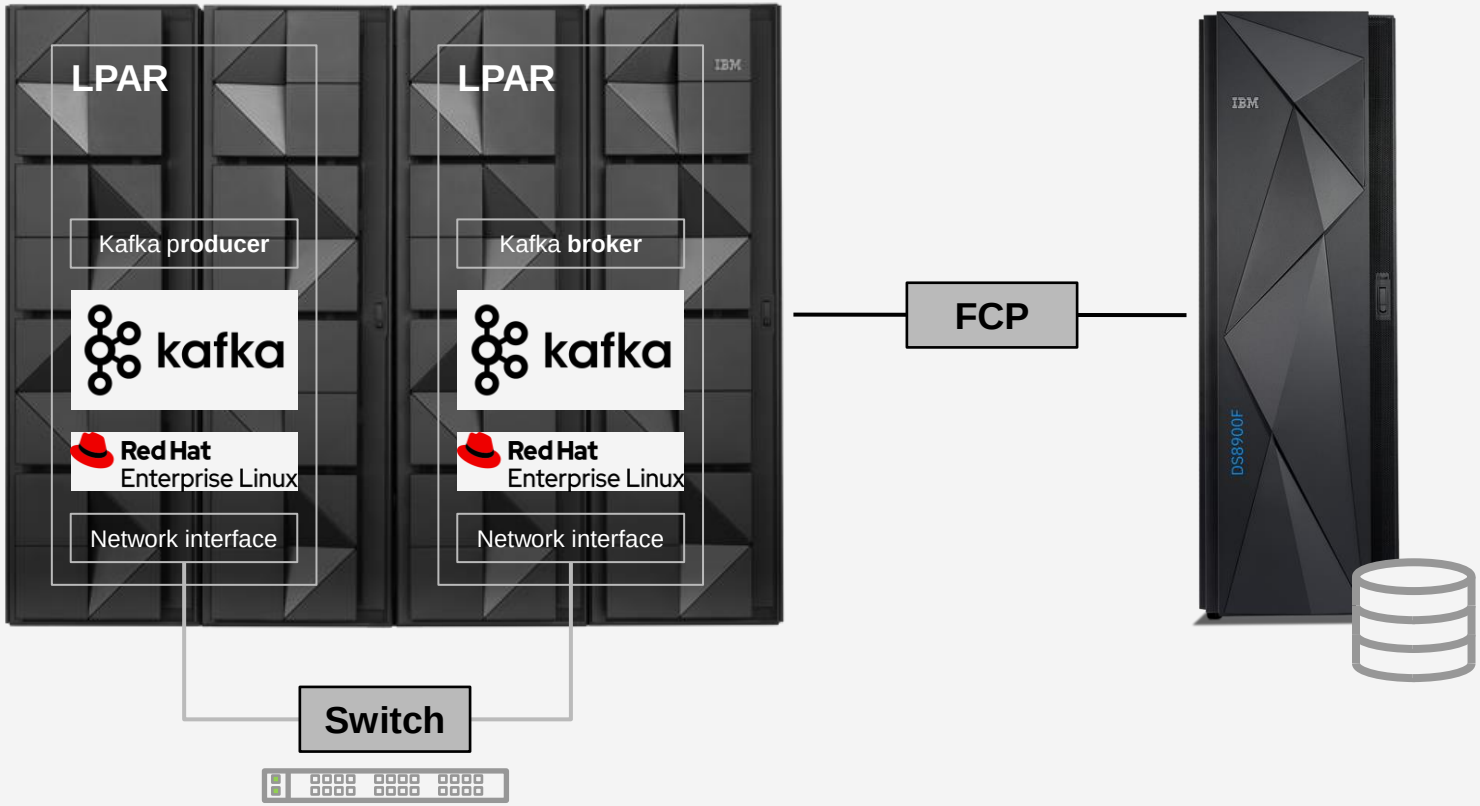
Source: https://kafka.apache.org/documentation

- Countless prominent ***"brand" customers***: HolidayCheck®, The New York Times®, Pinterest®, PayPal®, Atruvia®
  - See here: https://kafka.apache.org/powered-by

Agenda

- Introduction to Apache Kafka
- Performance measurement and tuning approach
- Observations and recommendations
- Summary

High-level environment setup



Approach

- First step was to ***download*** and ***build*** Kafka
  - "Recipe" available here: https://github.com/linux-on-ibm-z/docs/wiki/Building-Apache-Kafka
- Kafka comes with ***built-in performance test scripting*** for both producers and consumers
  - Very popular benchmarking article on LinkedIn from one of the founders of Confluent
  - Corresponding command lines for the benchmark runs are available on GitHub® (see the link here) – need some adjustment for latest Kafka version
- Got familiar with both Kafka ***operations / configuration*** and performance scripting command line ***syntax***
  - Kafka: configure IP addresses, start ZooKeeper™, start broker, number of partitions, etc.
  - Performance scripting: specify topic, number of events, etc.

Approach, *continued*

- Focus is on (1) ***producer*** performance, not on consumer performance, and (2) ***small*** message sizes
  - When it comes to performance problems, most people seem to struggle with ***producer*** performance
  - For message queueing systems, it is typically ***harder*** to achieve high throughput rates when using ***smaller*** message sizes compared to using larger message sizes
- Put the *System Under Test* (SUT) under load, just to see how it behaves at run-time
  - First impression: compared to other Java workloads, Kafka is pretty ***network and disk I/O intensive***, but ***not too CPU-intensive***
- Started with a simple, ***out-of-the-box*** configuration (baseline) and tried to identify bottlenecks in the entire stack – hardware, operating system, middleware, application – ***step by step***
  - Bottlenecks were ***moving*** from one component / area to the other

Agenda

- Introduction to Apache Kafka
- Performance measurement and tuning approach
- Observations and recommendations
- Summary

Disclaimers

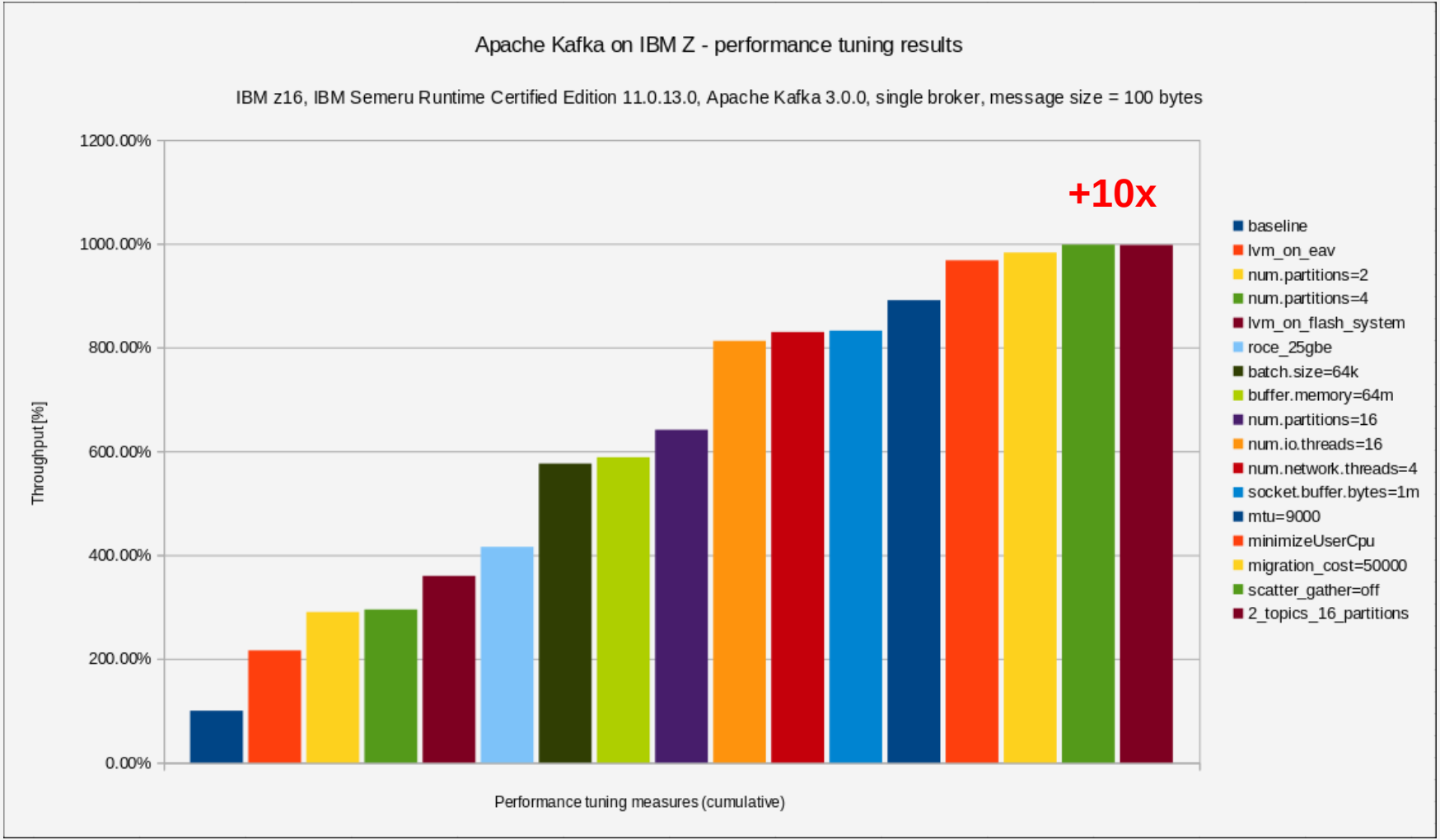
The following is ***important*** – please read it carefully

- The performance test results in the following charts were obtained in a ***controlled lab environment*** natively in a *Logical Partition* (LPAR). The measured differences in throughput might not be observed in real-life scenarios and environments other than native LPAR.

- All test runs were performed with Red Hat® Enterprise Linux® 8, IBM® Semeru Runtime Certified Edition 11.0.13.0, and Apache Kafka 3.0.0. ***Other*** product versions might produce ***different*** performance results.

- All of the tests were specifically executed for ***Kafka***. The impact of the recommendations in this chart deck on ***other*** message queueing systems might be ***totally different***, including ***adverse*** performance effects.

- All of the tests were specifically executed for a heavy ***producer*** workload with ***small*** message sizes. The impact of the recommendations in this chart deck on other types of workloads – consumer, for example – might be ***totally different***, including ***adverse*** performance effects.


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
+10x
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #1

- Throughput recommendation #1: use a ***striped logical volume*** for storing Kafka data

- When analyzing disk I/O related performance data, I noticed that the device holding Kafka ***log data*** was constantly at > 95% utilization
  - Can be seen with `sadc` and / or `iostat`
  - Look for the `%util` column

```
01/26/2022 09:53:06 AM
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm r_await w_await aqu-sz rareq-sz wareq-sz  %util
dasda            0.00    4.00      0.00    308.00     0.00     0.60   0.00  13.04    0.00    1.85   0.01     0.00    77.00   0.20
dasdb            0.00    0.00      0.00      0.00     0.00     0.00   0.00   0.00    0.00    0.00   0.00     0.00     0.00   0.00
dasdd            0.00  667.00      0.00 259184.80     0.00     4.20   0.00   0.63    0.00   24.10  16.07     0.00   388.58  98.40
...
```

- **Note**: some columns were cut in the above output in order to make it fit the page

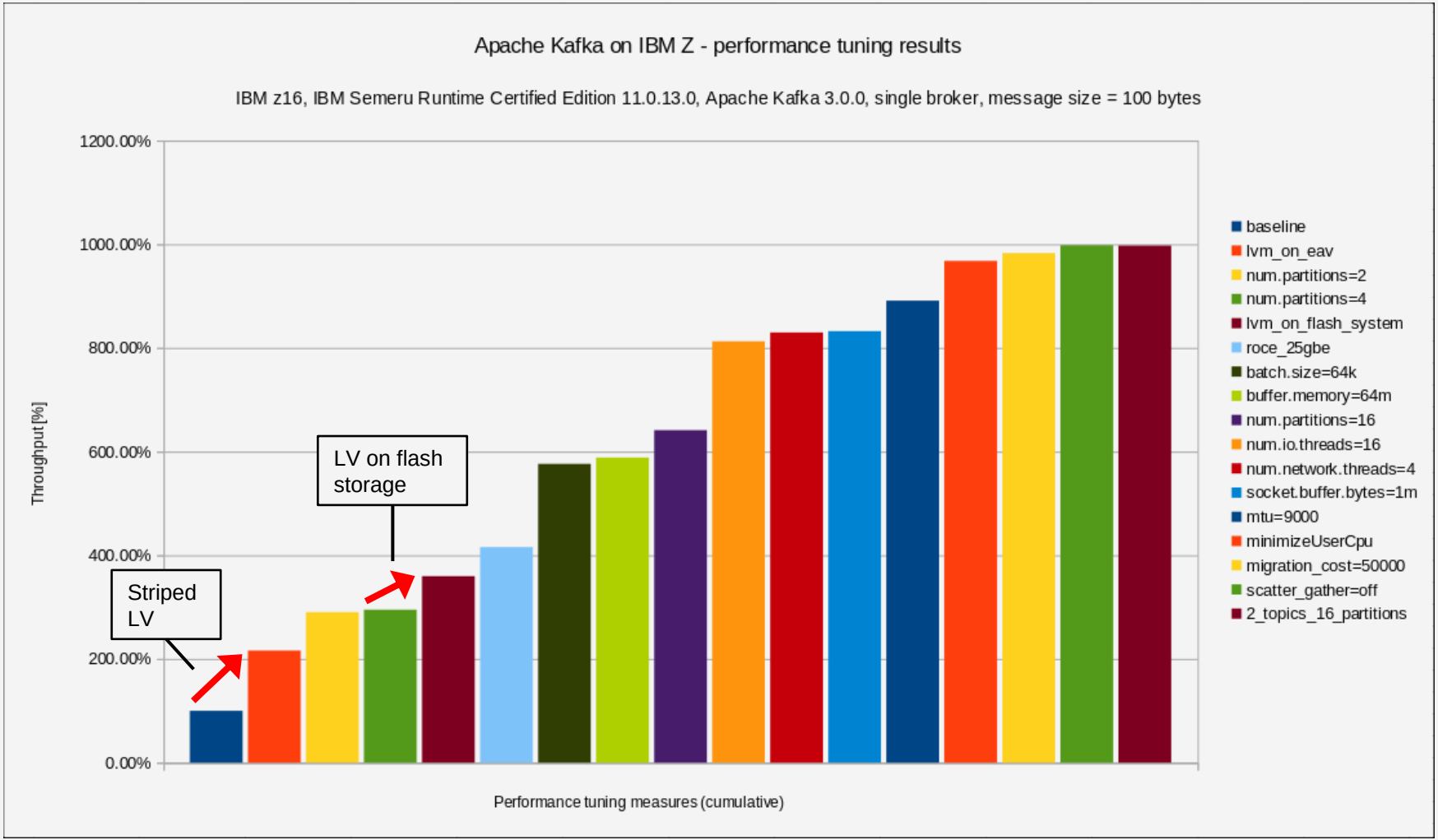
Throughput recommendation #1, *continued*

- ***Best practice***: for a striped logical volume, use as many ***stripes*** as there are ***physical*** volumes in the volume group that is used for the logical volume – I used 3 stripes initially and 8 for later tests
  - Double-check with `lvdisplay -m`
- Verify that the disk I/O load is really spread over all 3 devices:

```
01/26/2022 10:10:18 AM
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm r_await w_await aqu-sz rareq-sz wareq-sz  %util
dasda            0.00    5.00      0.00    336.00     0.00     1.40   0.00  21.88    0.00    1.52   0.01     0.00    67.20   0.20
dasdb            0.00  403.20      0.00 155387.20     0.00  2078.20   0.00  83.75    0.00    8.62   3.48     0.00   385.38  80.20
dasdd            0.00  399.80      0.00 155379.20     0.00  2078.40   0.00  83.87    0.00    9.04   3.62     0.00   388.64  82.40
dasdc            0.00  402.00      0.00 155389.60     0.00  2079.80   0.00  83.80    0.00    9.13   3.67     0.00   386.54  82.20
...
dm-17            0.00 7440.00      0.00 466019.20     0.00     0.00   0.00   0.00    0.00   10.85  80.69     0.00    62.64  82.80
...
```

Throughput recommendation #1, *continued*

- Other best practices for DS8000® based storage servers:
  - Use ***storage pool striping*** (or IBM Easy Tier®)
  - Use DASDs from ***different*** *Logical Control Units* (LCUs)
  - Use ***HyperPAV*** aliases from different LCUs
- For ***really*** large amounts of data, you might consider using ***multiple*** entries for Kafka `log` directories – and thus, multiple devices and / or logical volumes – for storing log data
  - Makes things more complicated from an overall setup / configuration perspective
  - ***Partition*** data is spread over `log.dirs` entries in `server.properties` configuration file
  - Files belonging to a single partition will all be stored in the same log directory


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
Throughput [%]
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Striped LV
LV on flash storage
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #2

- Throughput recommendation #2: configure ***multiple partitions*** for storing your log data
  - Setting is called `num.partitions` and can be found in the `server.properties` configuration file
  - Default value is `1`
  - Started with `2`, but increased to `16` in later tests
  - Optimal value depends on the actual workload / ***hardware resources***, etc.
- When analyzing `javacore` files, I noticed that there was 1 ***monitor*** – the Java term for *lock* – that was held by a so-called Kafka ***request handler thread*** that was blocking all other request handler threads
  - Stack trace revealed that this lock was being used for ***writing partition data***
  - See the next slide for a sample stack trace

Throughput recommendation #2, *continued*

Snippet from a `javacore` file:

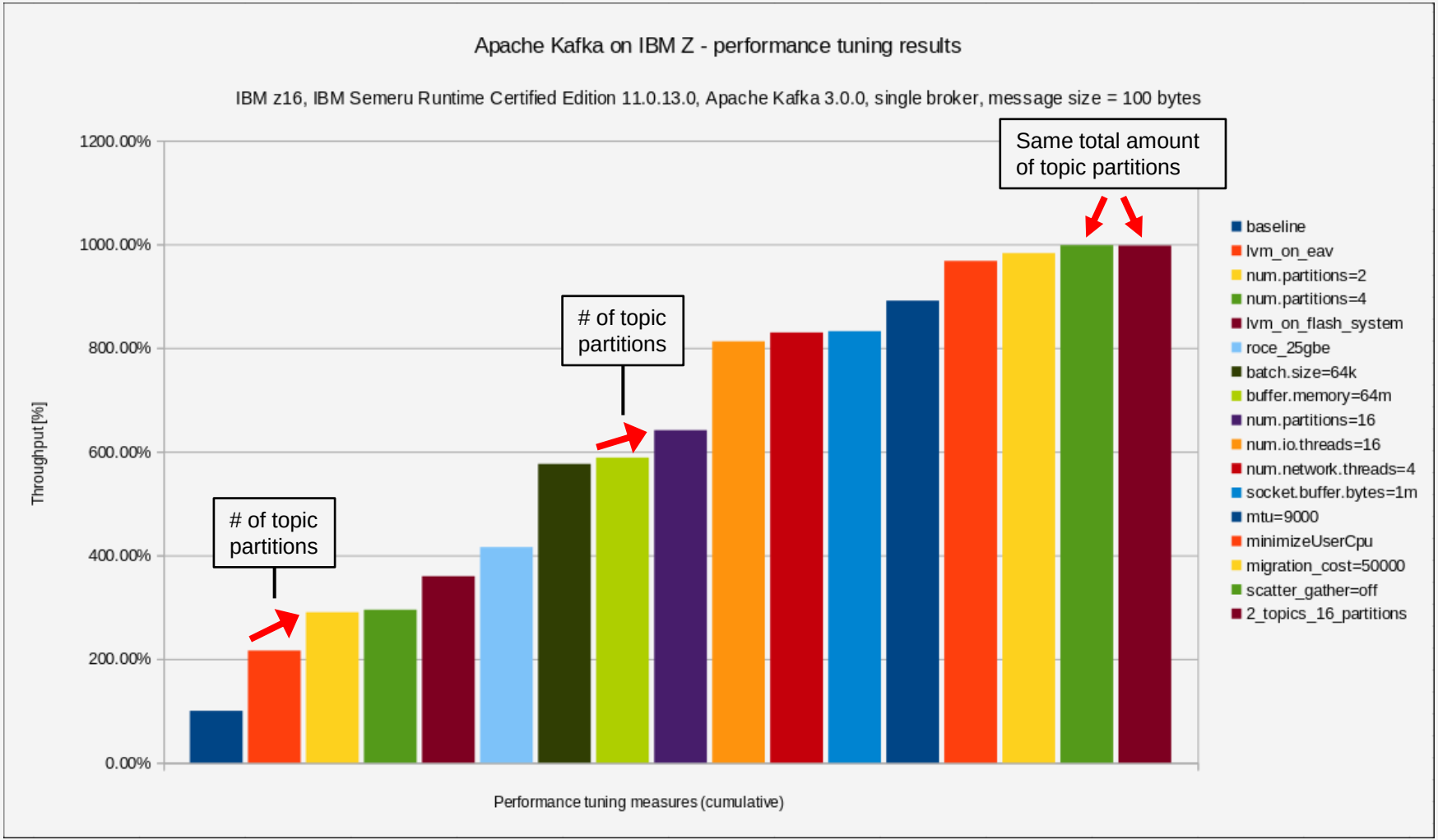
```
...
3XMTHREADINFO      "data-plane-kafka-request-handler-5" J9VMThread:0x0000000000DF7300, omrthread_t:0x000003FF9DB76440, java/lang/Thread:0x000000001F88EBC0, state:B, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x47, isDaemon:true)
3XMTHREADINFO1            (native thread ID:0x3FCE8, native priority:0x5, native policy:UNKNOWN, vmstate:B, vm thread flags:0x00000201)
3XMTHREADINFO2            (native stack address range from:0x000003FF4FD7E000, to:0x000003FF4FDBE000, size:0x40000)
3XMCPUTIME               CPU usage total: 48.297743025 secs, current category="Application"
3XMTHREADBLOCK     Blocked on: java/lang/Object@0x000000001F877850 Owned by: "data-plane-kafka-request-handler-1" (J9VMThread:0x0000000000DECC00, java/lang/Thread:0x000000001F>
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=5471576 (0x537D58)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at kafka/log/Log.append(Log.scala:801(Compiled Code))
4XESTACKTRACE                at kafka/log/Log.appendAsLeader(Log.scala:741(Compiled Code))
4XESTACKTRACE                at kafka/cluster/Partition.$anonfun$appendRecordsToLeader$1(Partition.scala:1042(Compiled Code))
4XESTACKTRACE                at kafka/cluster/Partition$$Lambda$1277/0x0000000000000000.apply(Bytecode PC:20(Compiled Code))
4XESTACKTRACE                at kafka/utils/CoreUtils$.inLock(CoreUtils.scala:231(Compiled Code))
4XESTACKTRACE                at kafka/utils/CoreUtils$.inReadLock(CoreUtils.scala:237(Compiled Code))
4XESTACKTRACE                at kafka/cluster/Partition.appendRecordsToLeader(Partition.scala:1030(Compiled Code))
...
```

Throughput recommendation #2, *continued*

- ***Alternative approach***: if you cannot increase the number of ***partitions*** of your Kafka topic for some reason, consider increasing the amount of ***topics***

- ***Rationale***: in the Scala code, there is ***1 lock per partition*** for synchronizing the writing of data, regardless of whether you are using 1 topic, 2 topics, or more
  - Important is to increase the ***total number of locks*** per *Java Virtual Machine* (JVM®) instance – i.e. per broker instance – in order to remove the pressure from the individual locks
  - From a ***pure locking perspective***, using 1 topic with 16 partitions is equal to using 2 topics with 8 partitions each – both configurations will use 16 partitions – and therefore, 16 locks – in total

- Verification experiment: in my test environment, the throughput of a 1 topic / 16 partitions configuration was ***identical*** to a 2 topics / 8 partitions each configuration (see next chart)

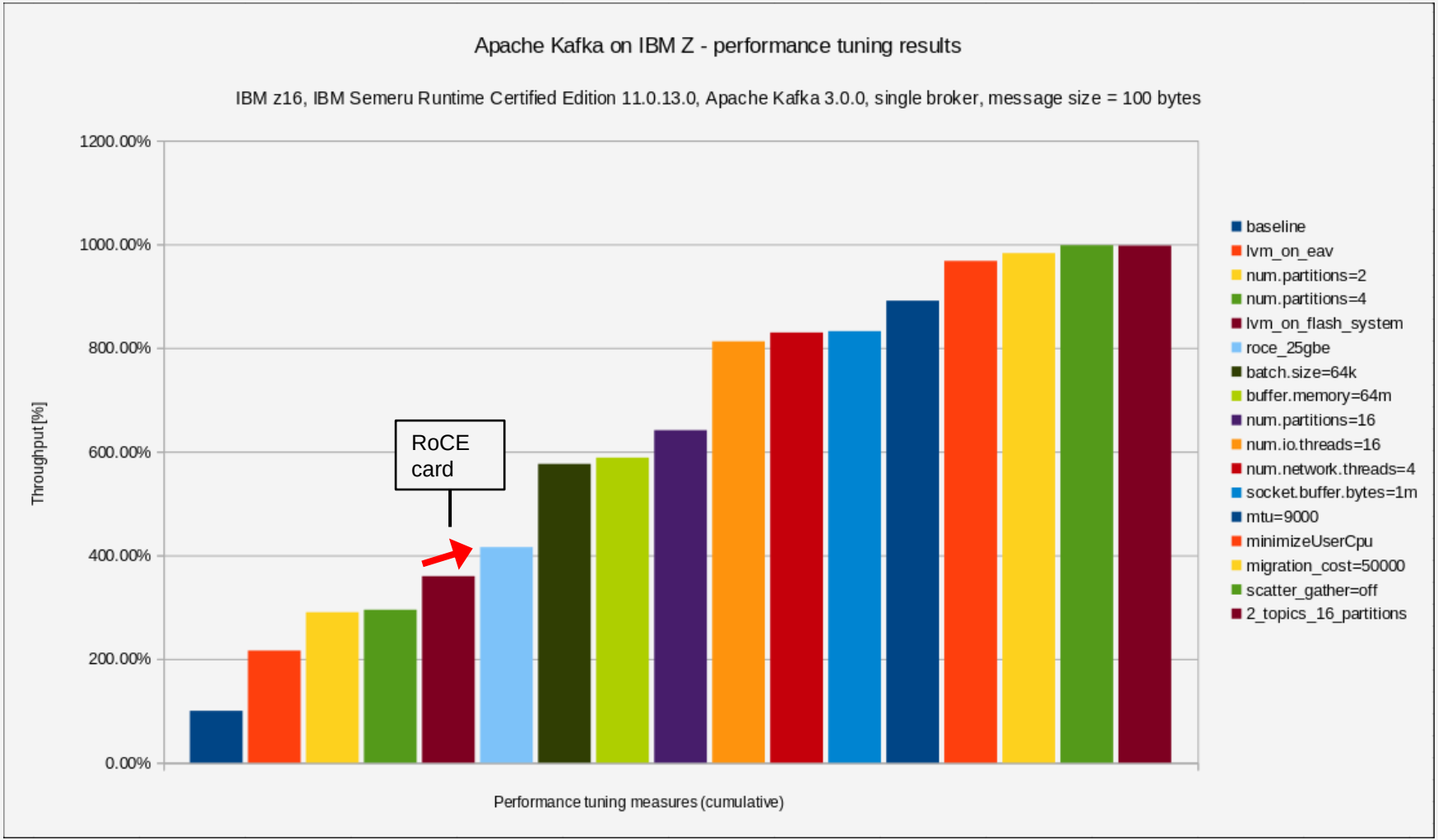

Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
Throughput [%]
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Same total amount of topic partitions
# of topic partitions
# of topic partitions
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #3

- Throughput recommendation #3: use ***RoCE cards*** to increase Kafka producer throughput
  - Network performance is an ***art*** – there are countless aspects that must be taken into account
  - When it comes to networking, Apache Kafka exhibits a ***transactional*** – i.e. request / response – workload pattern
  - IBM internal tests have shown that RoCE cards ***outperform*** OSA-Express cards for transactional workload patterns that do not saturate the physical network link, in particular when it comes to ***latency***
- Remember that raw performance is ***only one aspect*** when choosing your network attachment option
  - You might have a requirement that does not allow you to make use of RoCE cards, for example
  - A good ***overview*** comparing RoCE cards with OSA-Express cards can be found here

Throughput recommendation #3, *continued*

- ***Don't get confused*** by the supposedly small increase in throughput on the next chart
  - The net increase in end-to-end throughput might seem small at first sight
  - However, all follow-on recommendations ***heavily depend*** on a low-latency network
  - In other words: the optimizations that I discovered after recommendation #3 would not have been possible without switching over to RoCE
- The OSA-Express7S 10 GbE card that I was using before switching over to RoCE was already running at ***more than 80% utilization***
  - Didn't try with OSA-Express7S 25 GbE though

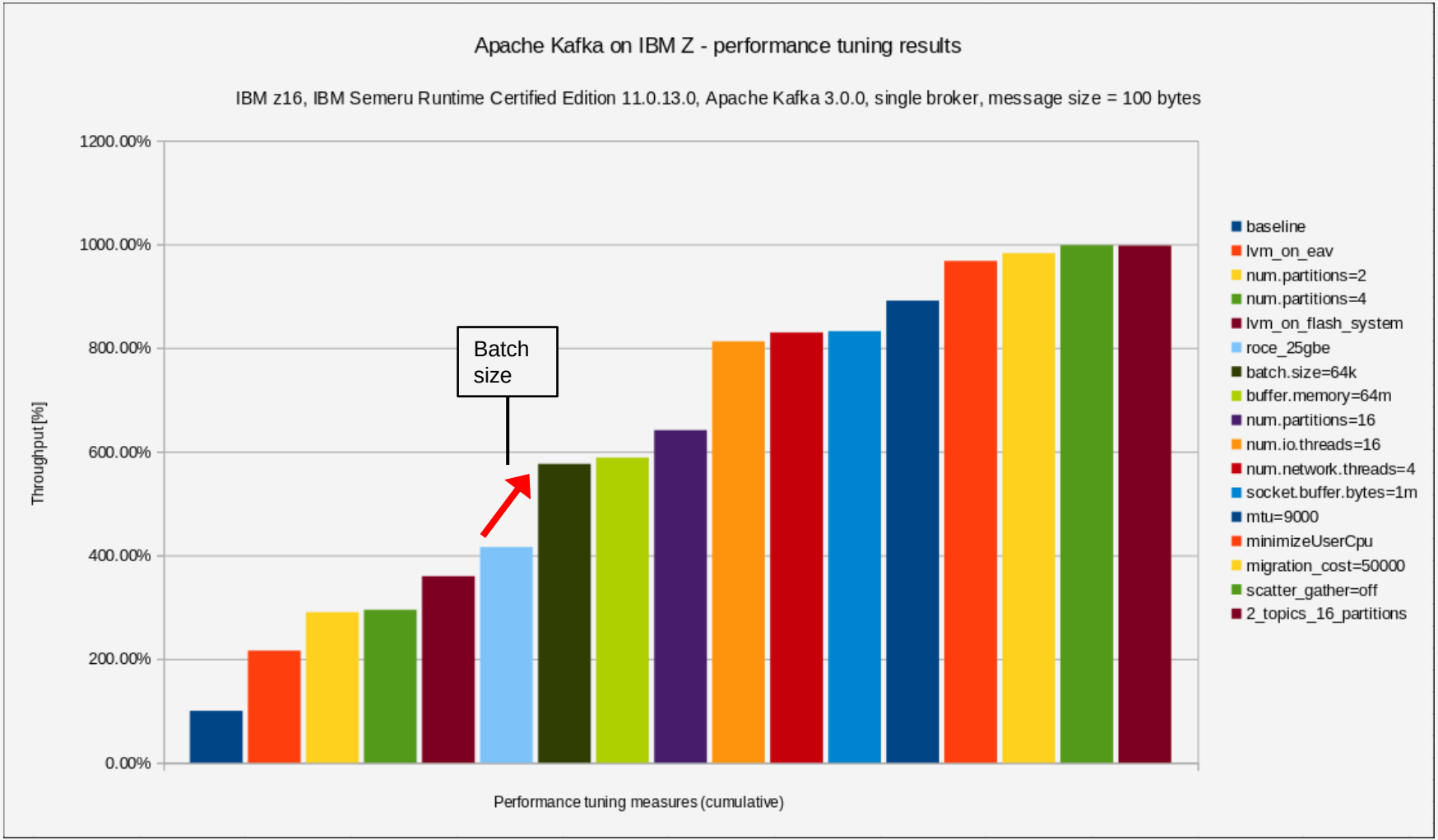

Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
RoCE card
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #4

- Throughput recommendation #4: increase the ***batch size*** for the Kafka producers
  - This is the amount of data that gets merged into one request for the broker
  - Configuration setting is called `batch.size` and the default is `16384` (16 KiB)
  - Passed on the command line for the performance test scripting via `--producer-props`
- At the end of the day, the `batch.size` setting can be a trade-off between ***increased throughput*** (good) and ***increased latency*** (bad)
  - Fewer network transfers with larger amounts of data means higher overall throughput
  - Waiting for the buffers to be filled ***can*** lead to increased latency, depending on how you configure the `linger.ms` parameter (see next page)

Throughput recommendation #4, *continued*

- If ***latency is not critical***, double-check if you need to adjust the value of `linger.ms` (default is `0`)
  - This setting controls the maximum amount of time that is spent ***waiting for the buffers to be filled***
  - If (a) your buffers are ***not fully utilized*** and if (b) you can tolerate a slightly increased latency, you might consider increasing the value for this parameter
  - Producers running at high clock speeds and at full CPU capacity will fill up the buffers very quickly, so you don't even need to take `linger.ms` into account
- ***In my test environment***, a batch size of `65536` (64 KiB) turned out to be optimal
  - Increasing the batch size beyond 64 KiB did ***not*** increase throughput substantially
- Optimal value depends on your ***requirements*** and the ***compute power*** of the producers

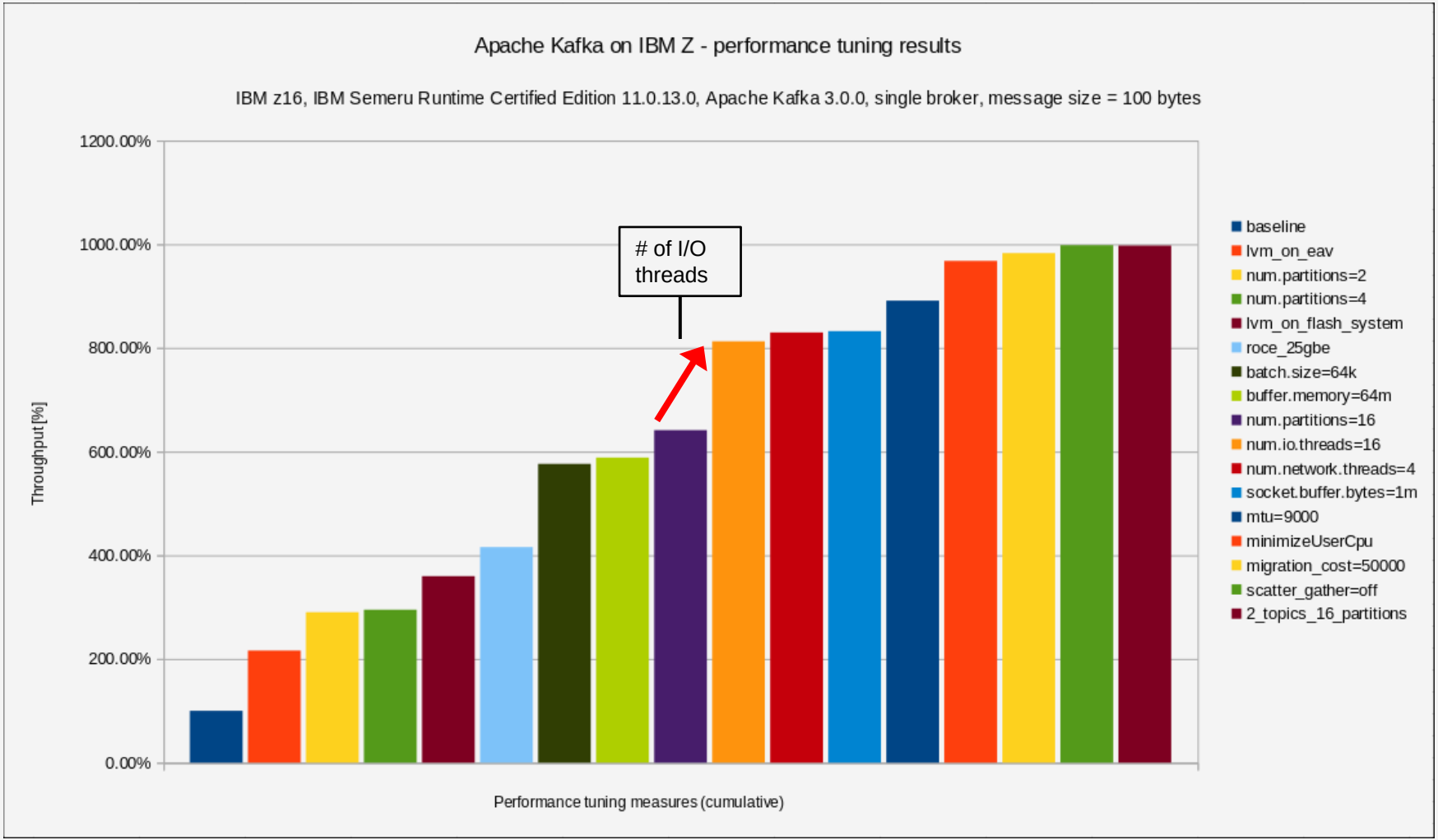

Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
Throughput [%]
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Batch size
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #5

- Throughput recommendation #5: increase the number of ***I/O threads***
  - According to Kafka documentation, this setting controls *"The number of threads that the server uses for processing requests, which **may** include disk I/O"*
  - Setting is called `num.io.threads` and can be found in the `server.properties` configuration file
- In the beginning of my study, I did not pay too much attention to this setting, but it turned out to influence performance ***quite measurably*** (see performance bar chart)
- My analysis of Kafka's runtime behavior revealed that the optimal value depends on the ***amount of CPUs*** available in the Linux® image running Kafka (see next chart)
  - In my test environment, a value ***slightly larger*** than the actual number of Linux CPUs – hardware threads in case of native LPAR – turned out to be optimal
  - Increasing the amount of I/O threads beyond this number did ***not*** increase throughput substantially

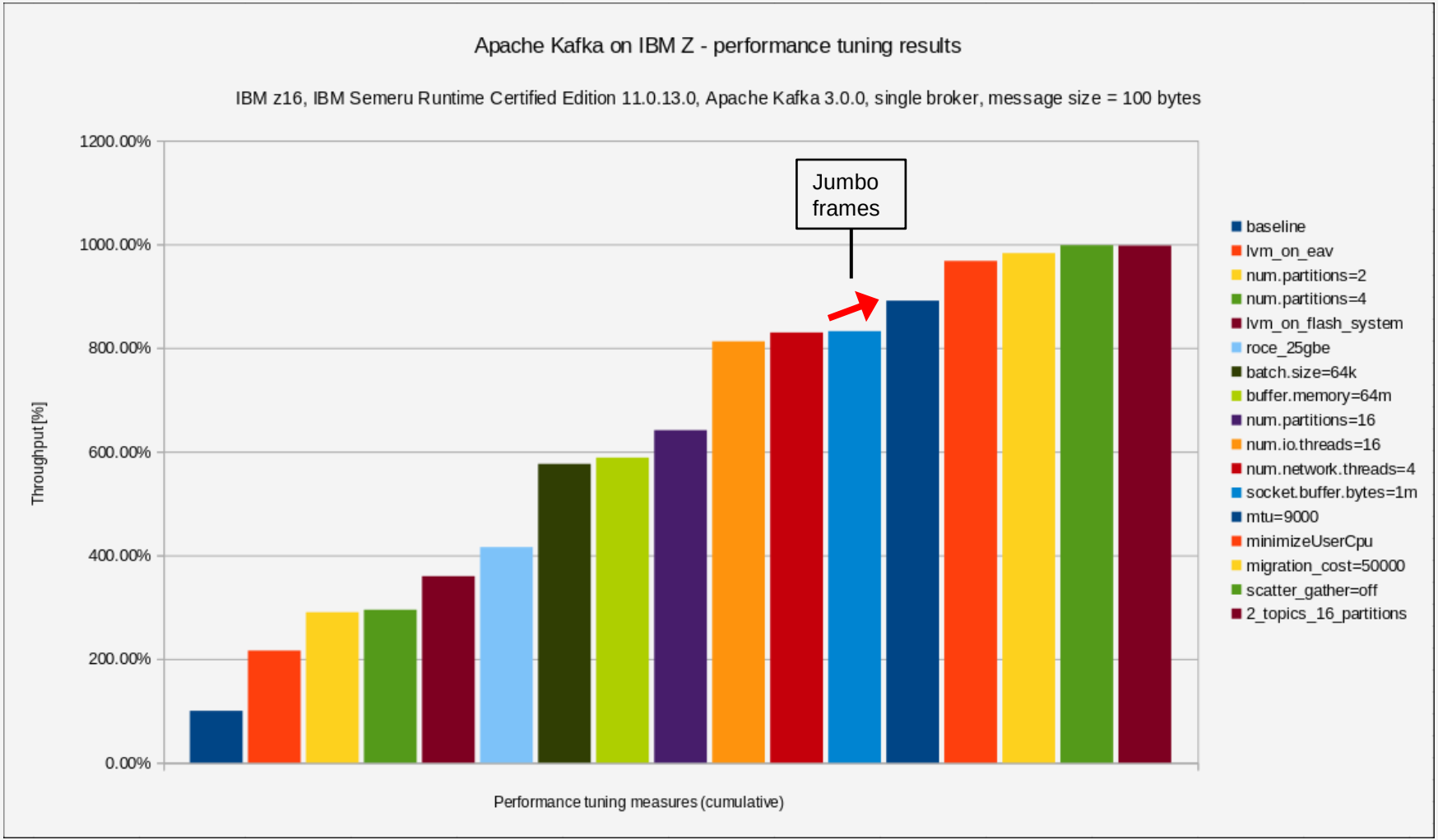
Throughput recommendation #5, *continued*

- Talking in JVM terms, this setting controls the amount of ***request handler threads***
  - From my observations, the request handler threads ***do*** drive disk I/O, at least for producer workloads – see recommendation #2 regarding the JVM locks
- ***Important***: this setting goes ***hand in hand*** with the total number of locks available in the broker instance – and therefore, the overall number of partitions
  - If your total amount of locks is smaller than the amount of request handler threads, then there will ***always*** be a request handler waiting for a lock – ***guaranteed***
  - Even if you configure the ***same amount*** of locks and request handler threads, you may ***still encounter*** locking situation situations, namely when 2 request handler threads want to write to the same partition
  - Therefore, make sure that the total number of locks in your broker instance is ***at least equal*** to the number of request handler threads (the more locks, the better)


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
# of I/O threads
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #6

- Throughput recommendation #6: configure your Ethernet® network to make use of ***jumbo frames***
  - Standard *Maximum Transmission Unit* (MTU) is 1500 bytes, jumbo frames use 9000 bytes
- ***Rationale***: especially when using batching, the producers are sending ***large packets*** over the network
  - Use `iptraf`, for example, in order to figure out the currently used average packet size
- Fewer packets per second sent over the wire means ***less packet processing*** in the entire network stack and hence improved end-to-end throughput
  - In general: jumbo frames ***might*** have an impact on latency when analyzing individual packets
  - In my test environment: end-to-end latency ***improved*** with jumbo frames turned on, simply because the producers were sending less packets


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
Jumbo frames
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Throughput recommendation #7

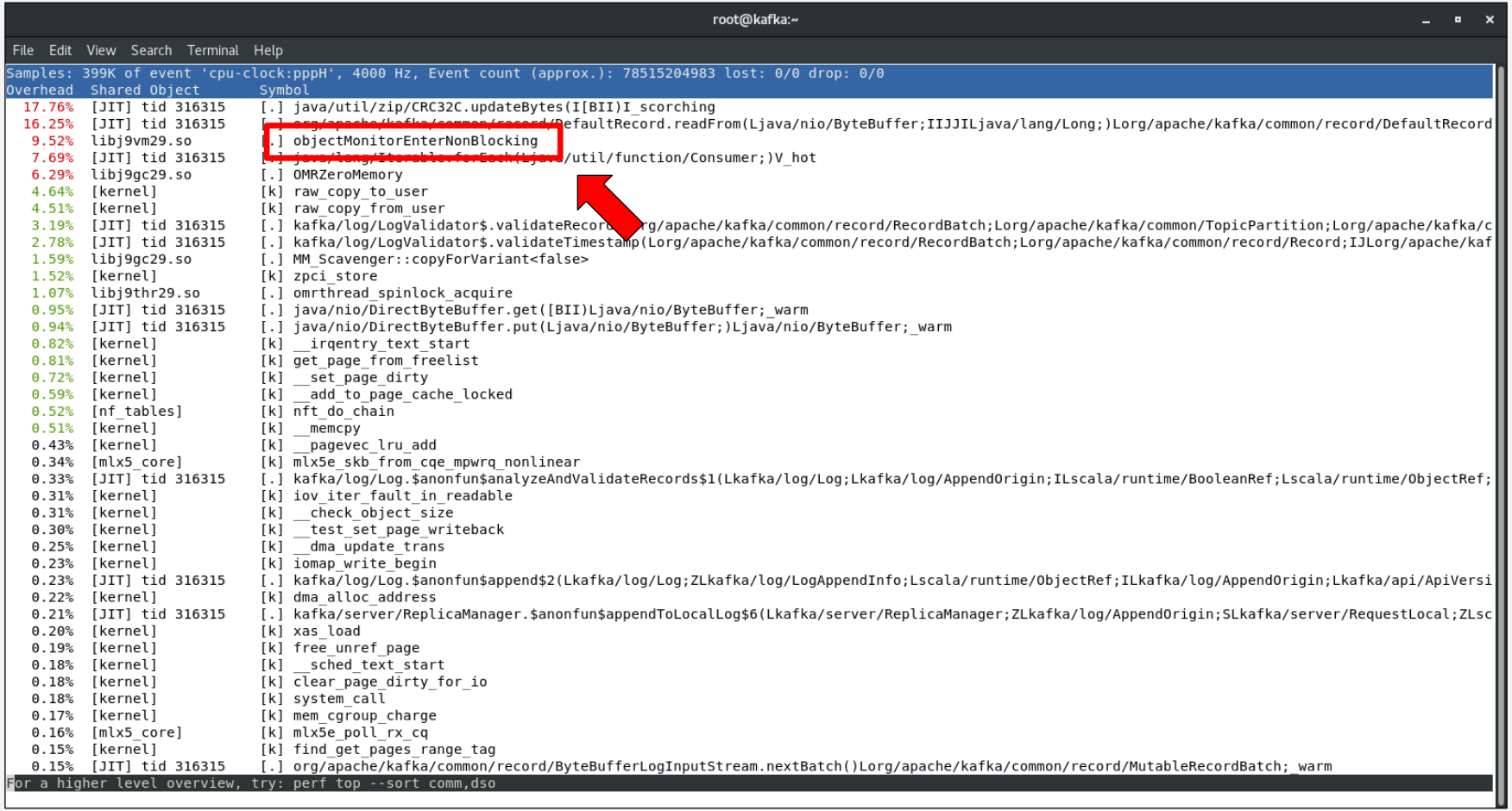
- Throughput recommendation #7: change the ***locking(*) behavior*** of the JVM

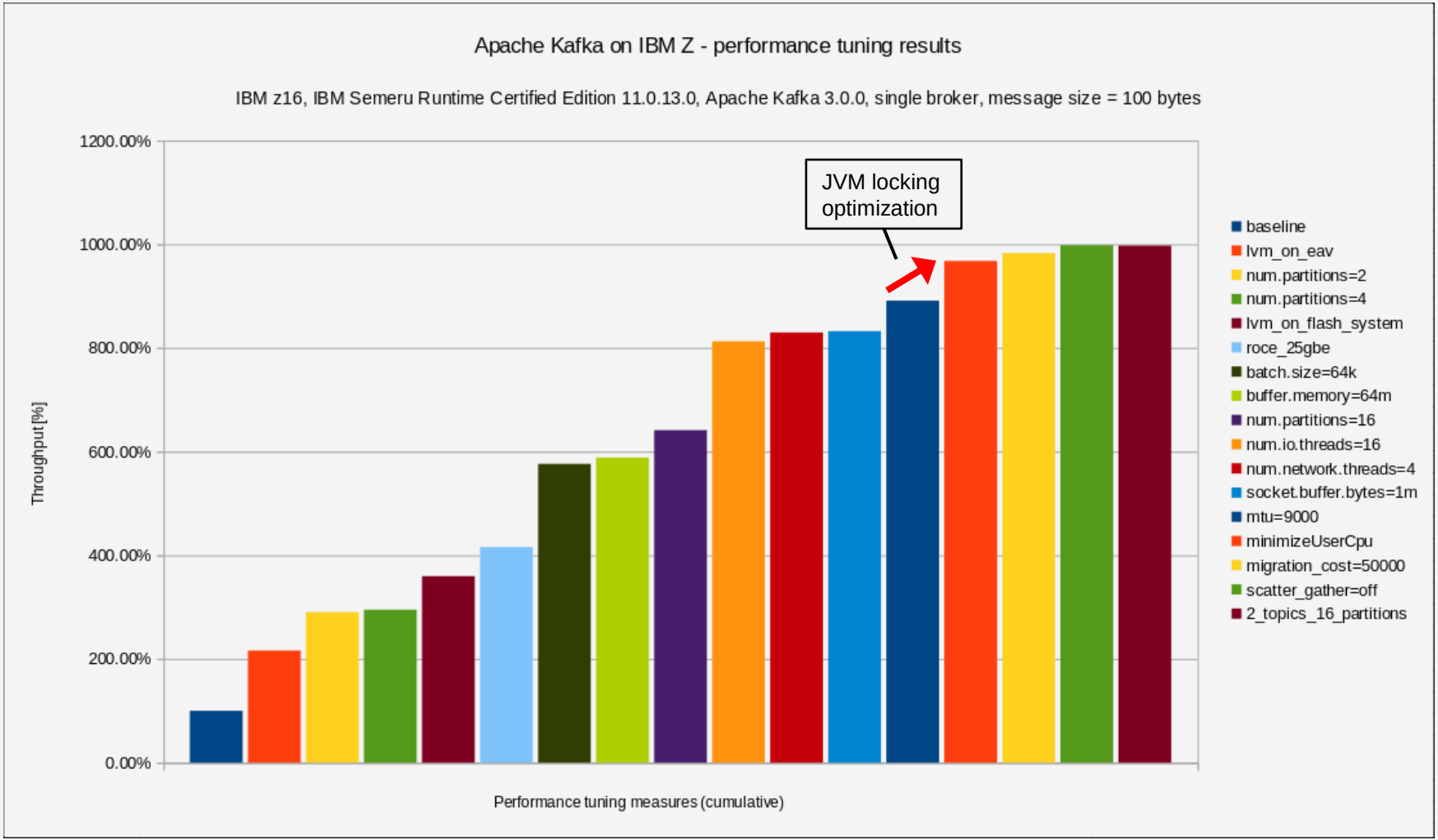
- The OpenJ9 JVM has a very ***sophisticated*** locking approach based on multiple tiers of lock ***spinning***
  - Great blog entry for the technically interested: https://blog.openj9.org/2019/04/02/openj9-locking-synchronization

- After ***experimenting*** with lots of different settings, one particular Java command line option for thread synchronization turned out to be ***optimal*** for Apache Kafka: `-Xthr:minimizeUserCpu`
  - Documented here: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=options-xthr
  - Add this option to your ***Java command line*** when starting the Kafka broker

- This setting did actually ***decrease CPU consumption*** in my producer workload, which in turn opened the door for further optimizations

(*)Technically correct, this should read *object synchronization* or Java *monitor* behavior. Nevertheless, the term *locking* was chosen for better overall consumability.


root@kafka:~
File Edit View Search Terminal Help
Samples: 399K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 78515204983 lost: 0/0 drop: 0/0
Overhead Shared Object Symbol
17.76% [JIT] tid 316315 [.] java/util/zip/CRC32C.updateBytes(I[BII)I_scorching
16.25% [JIT] tid 316315 [.] org/apache/kafka/common/record/DefaultRecord.readFrom(Ljava/nio/ByteBuffer;IIJJILjava/lang/Long;)Lorg/apache/kafka/common/record/DefaultRecord
9.52% libj9vm29.so [.] objectMonitorEnterNonBlocking
7.69% [JIT] tid 316315 [.] java/lang/Iterable.forEach(Ljava/util/function/Consumer;)V_hot
6.29% libj9gc29.so [.] OMRZeroMemory
4.64% [kernel] [k] raw_copy_to_user
4.51% [kernel] [k] raw_copy_from_user
3.19% [JIT] tid 316315 [.] kafka/log/LogValidator$.validateRecord...rg/apache/kafka/common/record/RecordBatch;Lorg/apache/kafka/common/TopicPartition;Lorg/apache/kafka/c
2.78% [JIT] tid 316315 [.] kafka/log/LogValidator$.validateTimestamp(Lorg/apache/kafka/common/record/RecordBatch;Lorg/apache/kafka/common/record/Record;IJLorg/apache/kaf
1.59% libj9gc29.so [.] MM_Scavenger::copyForVariant<false>
1.52% [kernel] [k] zpci_store
1.07% libj9thr29.so [.] omrthread_spinlock_acquire
0.95% [JIT] tid 316315 [.] java/nio/DirectByteBuffer.get([BII)Ljava/nio/ByteBuffer;_warm
0.94% [JIT] tid 316315 [.] java/nio/DirectByteBuffer.put(Ljava/nio/ByteBuffer;)Ljava/nio/ByteBuffer;_warm
0.82% [kernel] [k] __irqentry_text_start
0.81% [kernel] [k] get_page_from_freelist
0.72% [kernel] [k] __set_page_dirty
0.59% [kernel] [k] __add_to_page_cache_locked
0.52% [nf_tables] [k] nft_do_chain
0.51% [kernel] [k] __memcpy
0.43% [kernel] [k] __pagevec_lru_add
0.34% [mlx5_core] [k] mlx5e_skb_from_cqe_mpwrq_nonlinear
0.33% [JIT] tid 316315 [.] kafka/log/Log.$anonfun$analyzeAndValidateRecords$1(Lkafka/log/Log;Lkafka/log/AppendOrigin;ILscala/runtime/BooleanRef;Lscala/runtime/ObjectRef;
0.31% [kernel] [k] iov_iter_fault_in_readable
0.31% [kernel] [k] __check_object_size
0.30% [kernel] [k] __test_set_page_writeback
0.25% [kernel] [k] __dma_update_trans
0.23% [kernel] [k] iomap_write_begin
0.23% [JIT] tid 316315 [.] kafka/log/Log.$anonfun$append$2(Lkafka/log/Log;ZLkafka/log/LogAppendInfo;Lscala/runtime/ObjectRef;ILkafka/log/AppendOrigin;Lkafka/api/ApiVersi
0.22% [kernel] [k] dma_alloc_address
0.21% [JIT] tid 316315 [.] kafka/server/ReplicaManager.$anonfun$appendToLocalLog$6(Lkafka/server/ReplicaManager;ZLkafka/log/AppendOrigin;SLkafka/server/RequestLocal;ZLsc
0.20% [kernel] [k] xas_load
0.19% [kernel] [k] free_unref_page
0.18% [kernel] [k] __sched_text_start
0.18% [kernel] [k] clear_page_dirty_for_io
0.18% [kernel] [k] system_call
0.17% [kernel] [k] mem_cgroup_charge
0.16% [mlx5_core] [k] mlx5e_poll_rx_cq
0.15% [kernel] [k] find_get_pages_range_tag
0.15% [JIT] tid 316315 [.] org/apache/kafka/common/record/ByteBufferLogInputStream.nextBatch()Lorg/apache/kafka/common/record/MutableRecordBatch;_warm
For a higher level overview, try: perf top --sort comm,dso


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
JVM locking optimization
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

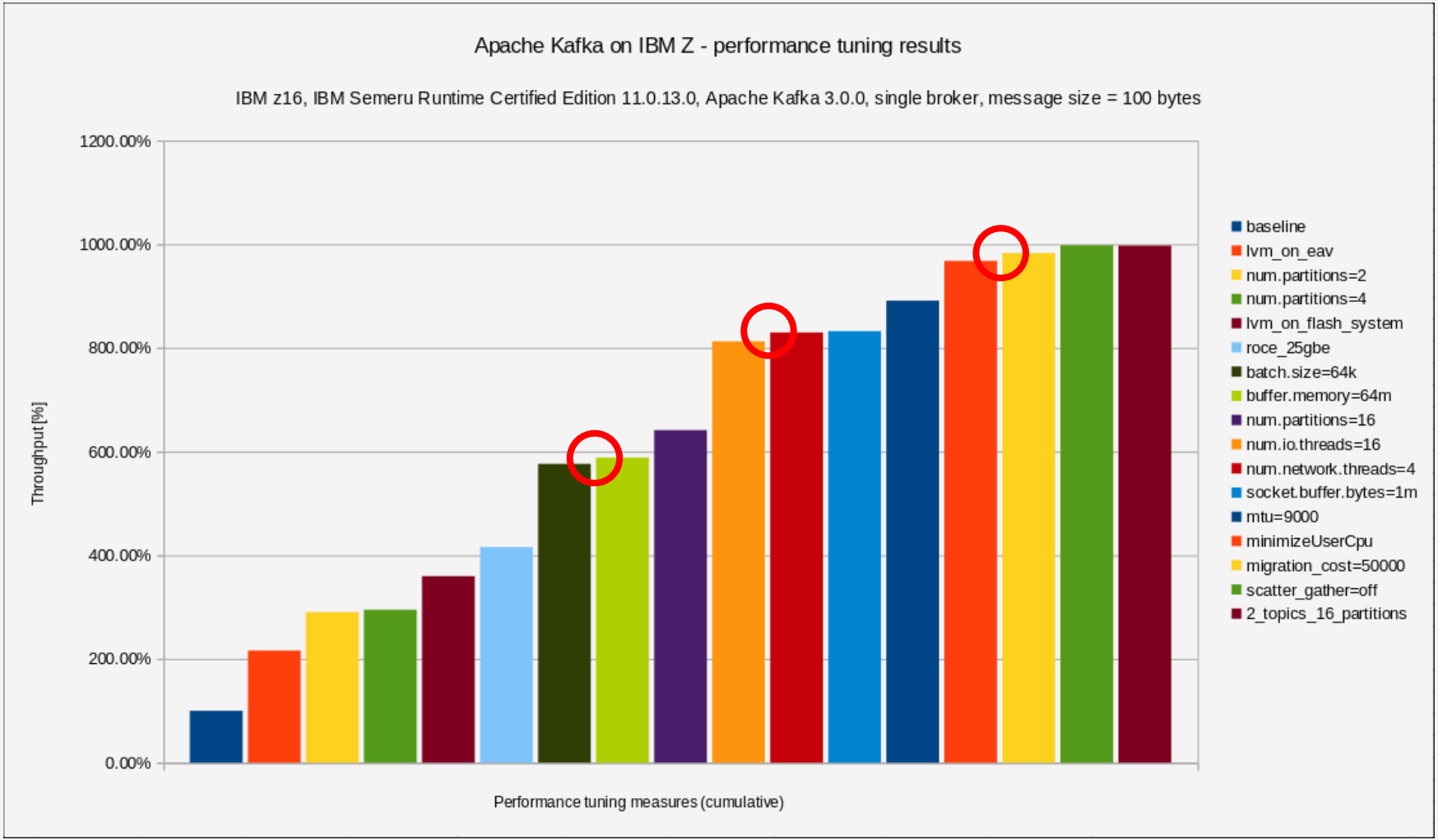
The remainder

- There is a number of minor tuning recommendations left that each only have a ***small*** impact on performance, but lead to a ***double-digit percentage increase*** in summary

- `buffer.memory` setting for the Kafka producers (optimum in my environment: 64 MiB)
  - Apache Kafka documentation: *"The total bytes of memory the producer can use to buffer records waiting to be sent to the server"*

- `num.network.threads` in the `server.properties` configuration file (my environment: `4`)
  - Documentation: *"The number of threads that the server uses for receiving requests from the network and sending responses to the network"*

- Linux kernel scheduler migration cost: `kernel.sched_migration_cost_ns=50000`
  - Linux `sysctl` setting that configures the ***number of nanoseconds*** the kernel will wait ***before considering moving*** a thread to another CPU


Apache Kafka on IBM Z - performance tuning results
IBM z16, IBM Semeru Runtime Certified Edition 11.0.13.0, Apache Kafka 3.0.0, single broker, message size = 100 bytes
1200.00%
1000.00%
800.00%
600.00%
400.00%
200.00%
0.00%
Throughput [%]
Performance tuning measures (cumulative)
baseline
lvm_on_eav
num.partitions=2
num.partitions=4
lvm_on_flash_system
roce_25gbe
batch.size=64k
buffer.memory=64m
num.partitions=16
num.io.threads=16
num.network.threads=4
socket.buffer.bytes=1m
mtu=9000
minimizeUserCpu
migration_cost=50000
scatter_gather=off
2_topics_16_partitions

Agenda

- Introduction to Apache Kafka
- Performance measurement and tuning approach
- Observations and recommendations
- Summary

Summary

- Apache Kafka is a very popular Open Source ***event streaming platform***, used across a variety of software stacks and implemented by a huge number of customers worldwide

- If you follow the recommendations in this presentation, you can ***greatly improve performance*** of your Kafka installation on Linux on IBM zSystems®

  – Recommendations are spread across the ***entire stack***: RoCE cards, striped logical volume, Linux kernel scheduler migration cost, Java object synchronization tuning, number of partitions

- 3 ***high-level areas*** that had the largest impact: disk I/O, network I/O, and locking

- ***Outlook***: if you have further Java-based enterprise solutions that need an in-depth performance analysis, let me know

Thank you!

Resources

- Linux on IBM zSystems and IBM LinuxONE
  - Official homepage: https://www.ibm.com/it-infrastructure/z/os/linux
  - Documentation: https://www.ibm.com/docs/en/linux-on-systems?topic=linuxone-library-overview
  - Tuning hints and tips: https://www.ibm.com/docs/en/linux-on-systems?topic=performance-tuning-hints-tips

- IBM Java SDK downloads: https://www.ibm.com/support/pages/java-sdk-downloads

- Java on z/OS®: https://www.ibm.com/support/pages/java-sdk-products-zos

- IBM Runtimes for Business: https://www.ibm.com/support/pages/ibm-runtimes-business

- OpenJDK® V11 based on HotSpot – available from Eclipse® Adoptium™: https://adoptium.net/releases.html?variant=openjdk11

- OpenJDK V11 based on OpenJ9 – available from IBM Semeru Runtimes: https://developer.ibm.com/languages/java/semeru-runtimes/downloads

- User Guide (includes diagnostics information) for IBM Java V8: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=downloadable-documentation