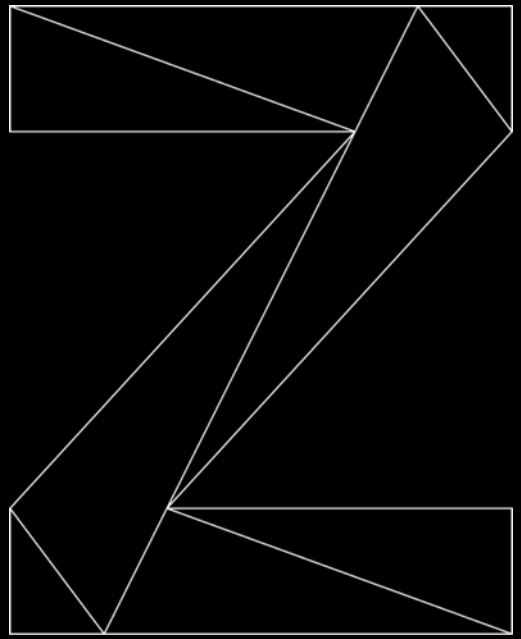
Elasticsearch on IBM Z – Performance Experiences, Hints and Tips

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

Document version: 1.1
Document date: 2021-05-07



IBM

Notices and disclaimers

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 Z, IBM z15, 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 Elasticsearch
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
- Summary

About Elasticsearch

- Elasticsearch™ is a ***search engine*** based on the ***Apache® Lucene®*** library
  - First released in 2010
- Written in ***Java®***, provides ***RESTful*** interfaces
  - Allows for a plethora of very diverse clients
- Data can be both ***distributed*** and ***replicated*** across nodes
  - Elasticsearch indexes are divided into *shards*, replicas are based on shards
  - Shards in turn are Lucene indexes

About Elasticsearch, *continued*

- Elasticsearch ***use cases***:
  - Full text search: regular documents, HTML pages, source code, etc.
  - Logging and log analysis: cloud solutions from Amazon™, Google®, and Microsoft®, and of course Red Hat® OpenShift® Container Platform, etc.
  - Metrics: performance, availability, application usage statistics, etc.
  - Business analytics: additional components from the ELK stack are required
- Countless prominent ***"brand" customers***: ebay®, Rabobank, Ticketmaster®, Uber®, Walmart®, etc.
  - See here: https://www.elastic.co/customers/success-stories?usecase=enterprise-search

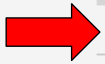
Database popularity ranking

370 systems in ranking, April 2021

Rank			DBMS	Database Model	Score		
Apr 2021	Mar 2021	Apr 2020			Apr 2021	Mar 2021	Apr 2020
1.	1.	1.	Oracle	Relational, Multi-model	1274.92	-46.82	-70.51
2.	2.	2.	MySQL	Relational, Multi-model	1220.69	-34.14	-47.66
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1007.97	-7.33	-75.46
4.	4.	4.	PostgreSQL	Relational, Multi-model	553.52	+4.23	+43.66
5.	5.	5.	MongoDB	Document, Multi-model	469.97	+7.58	+31.54
6.	6.	6.	IBM Db2	Relational, Multi-model	157.78	+1.77	-7.85
7.	7.	↑ 8.	Redis	Key-value, Multi-model	155.89	+1.74	+11.08
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	152.18	-0.16	+3.27
9.	9.	9.	SQLite	Relational	125.06	+2.42	+2.87
10.	10.	10.	Microsoft Access	Relational	116.72	-1.41	-5.19

Source: https://db-engines.com/en/ranking

Search engine popularity ranking



☐ include secondary database models

21 systems in ranking, April 2021

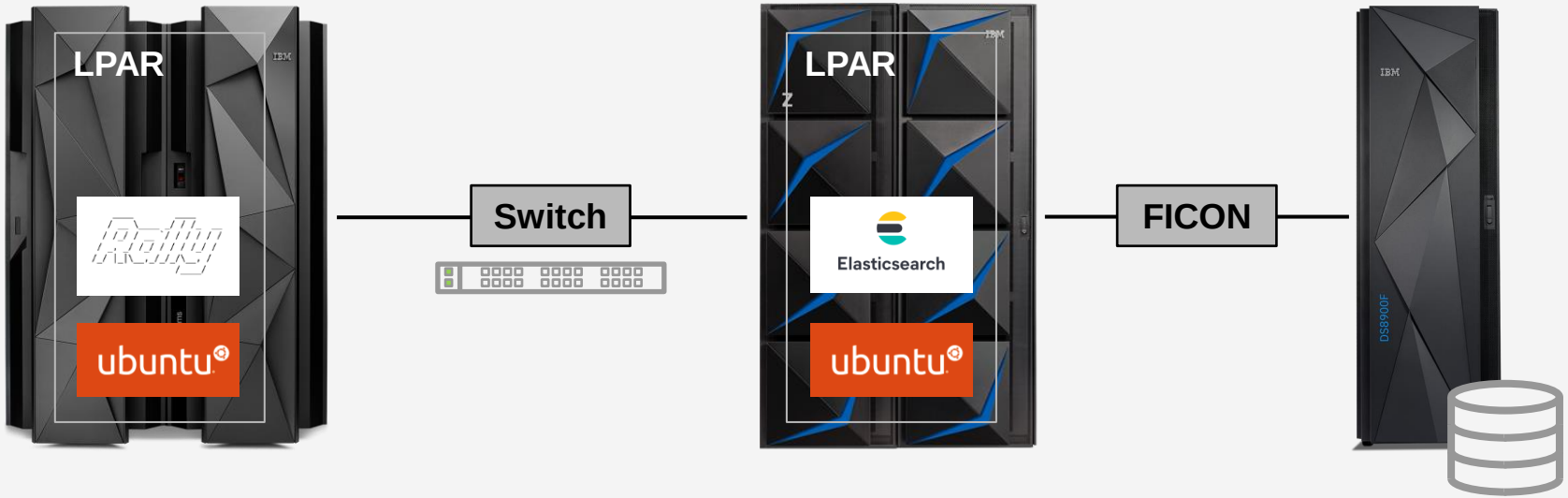
Rank			DBMS	Database Model	Score		
Apr 2021	Mar 2021	Apr 2020			Apr 2021	Mar 2021	Apr 2020
1.	1.	1.	Elasticsearch	Search engine, Multi-model	152.18	-0.16	+3.27
2.	2.	2.	Splunk	Search engine	88.49	+1.56	+0.41
3.	3.	3.	Solr	Search engine, Multi-model	50.60	+0.39	-2.99
4.	4.	4.	MarkLogic	Multi-model	9.92	+0.28	-1.34
5.	5.	↑ 8.	Algolia	Search engine	7.87	+0.23	+3.41
6.	6.	↓ 5.	Sphinx	Search engine	7.04	+0.13	+0.55
7.	7.	↓ 6.	Microsoft Azure Search	Search engine	6.53	+0.11	+0.31
8.	8.	↓ 7.	ArangoDB	Multi-model	4.77	-0.28	-0.11
9.	9.	↑ 10.	Virtuoso	Multi-model	3.18	+0.30	+0.56
10.	10.	↓ 9.	Amazon CloudSearch	Search engine	2.22	-0.02	-0.47

Source: https://db-engines.com/en/ranking/search+engine

Agenda

- Introduction to Elasticsearch
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
- Summary

High-level environment setup



Approach

- First step was to ***download*** and ***build*** Elasticsearch
  - "Recipe" available here: https://github.com/linux-on-ibm-z/docs/wiki/Building-Elasticsearch
- Second step was to get ***Rally*** up and running, which is the ***de-facto standard*** benchmarking tool
  - Quickstart guide: https://esrally.readthedocs.io/en/stable/quickstart.html
  - Benchmarking an existing cluster: https://esrally.readthedocs.io/en/stable/recipes.html#benchmarking-an-existing-cluster
- Got familiar with both Elasticsearch ***operations / configuration*** and Rally command line ***syntax***
  - Elasticsearch: check index state, delete indexes, view thread pools, etc.
  - Rally: select track, select challenge, etc.

Approach, *continued*

- Focus is on ***indexing*** – i.e. write – performance, not query – i.e. read – performance
  - When it comes to performance problems, most customers seem to struggle with indexing performance
- Put the *System Under Test* (SUT) under load, just to see how it behaves at run-time
  - First impression: compared to other Java workloads, Elasticsearch is pretty ***disk I/O intensive*** (and CPU-intensive, of course)
- In the beginning, I used the `geopoint` track, but switched to `http_logs` later in the process
  - Advantage of `geopoint`: allows for ***quick turnaround times***
  - Advantage of `http_logs`: allows for (a) ***longer-lasting runtimes*** for reference runs and (b) comes very close to real-world customer scenarios

Agenda

- Introduction to Elasticsearch
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
- Summary

Disclaimers

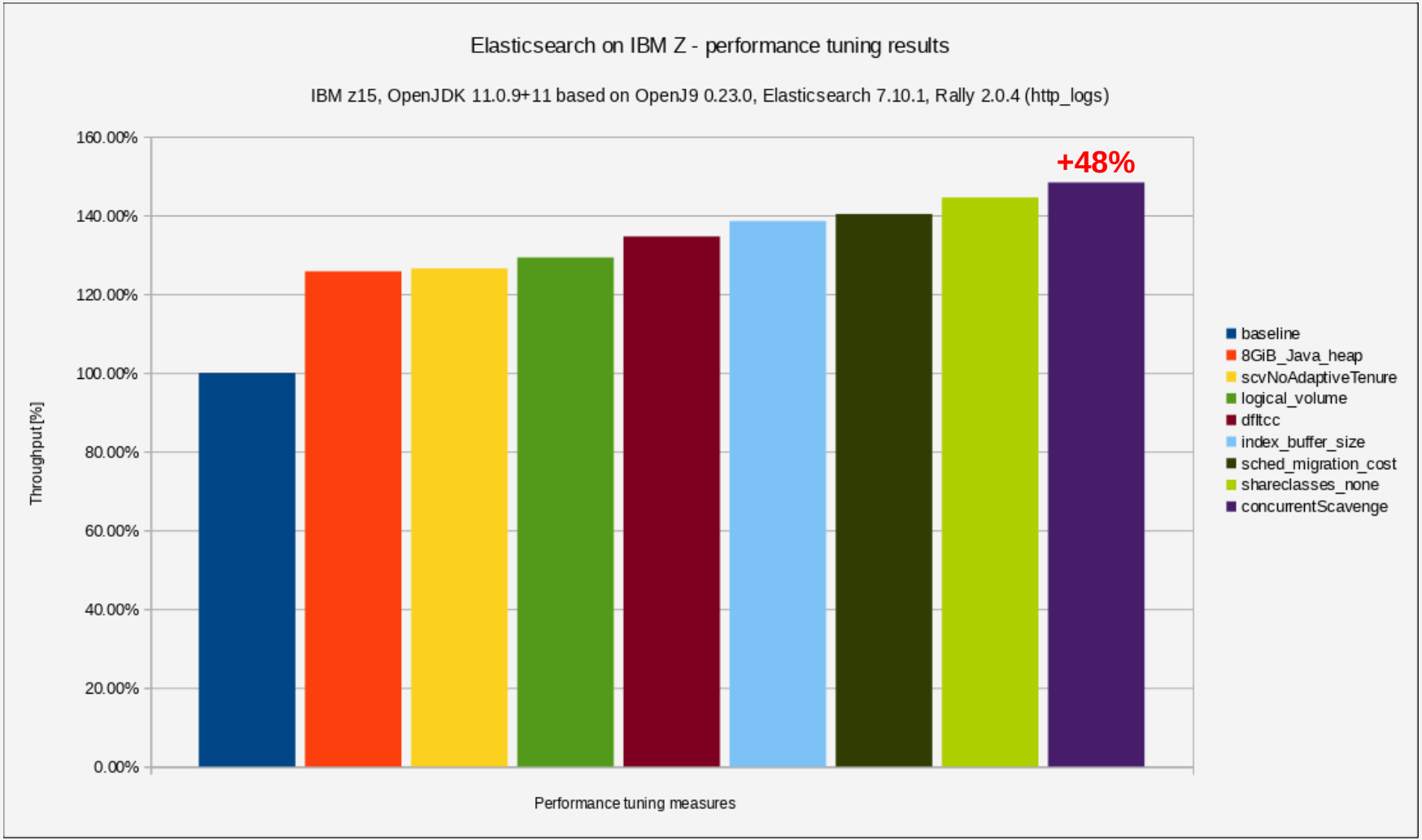
The following is ***important*** – please read it carefully

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

- All of the test runs were performed with Ubuntu® 20.04.2 LTS, Elasticsearch 7.10.1, and Rally 2.0.4. ***Other*** product versions might produce ***different*** performance results.

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

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


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
+48%
Performance tuning measures
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge

Throughput recommendation #1

- Throughput recommendation #1: increase the ***Java heap size*** to at least 8 GiB
- Be ***careful***, however, with the absolute number presented here, because the actually required heap size heavily depends on the application workload: # of parallel requests, size of the data, think time, etc.
- For sure, the default heap size of 1 GiB is ***too small*** for most indexing-heavy workloads
- In order to determine the optimal Java heap size, perform an ***analysis*** of ***verbose garbage collection data*** gathered during typical workload execution times
- Marc's recommended tool for this task: *Garbage Collection and Memory Visualizer* (GCMV)
  - Standalone version for Eclipse® IDE:
    https://marketplace.eclipse.org/content/ibm-monitoring-and-diagnostic-tools-garbage-collection-and-memory-visualizer-gcmv
  - Integrated into IBM® Support Assistant:
    https://www.ibm.com/support/home/product/C100515X13178X21/other_software/ibm_support_assistant

Throughput recommendation #1, *continued*

- **Before (1 GiB Java heap)**

Summary

Concurrent collection count	135
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	8.81
Global collections - Mean interval between collections (ms)	2129
Global collections - Number of collections	140
Global collections - Total amount tenured (MB)	42969
Largest memory request (bytes)	1039160
Number of collections triggered by allocation failure	8140
Nursery collections - Mean garbage collection pause (ms)	5.59
Nursery collections - Mean interval between collections (ms)	36.9
Nursery collections - Number of collections	8135
Nursery collections - Total amount flipped (MB)	321297
Nursery collections - Total amount tenured (MB)	73031
Proportion of time spent in garbage collection pauses (%)	16.39
Proportion of time spent unpaused (%)	83.61
Rate of garbage collection (MB/minutes)	264466

- **After (8 GiB Java heap)**

Summary

Concurrent collection count	5
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	27.2
Global collections - Mean interval between collections (ms)	52323
Global collections - Number of collections	5
Global collections - Total amount tenured (MB)	7635
Largest memory request (bytes)	4194312
Number of collections triggered by allocation failure	465
Nursery collections - Mean garbage collection pause (ms)	40.6
Nursery collections - Mean interval between collections (ms)	645
Nursery collections - Number of collections	465
Nursery collections - Total amount flipped (MB)	223275
Nursery collections - Total amount tenured (MB)	14824
Proportion of time spent in garbage collection pauses (%)	6.51
Proportion of time spent unpaused (%)	93.49
Rate of garbage collection (MB/minutes)	277481

Throughput recommendation #1, *continued*

- Turned out to be the major tuning knob: ***25% more throughput***

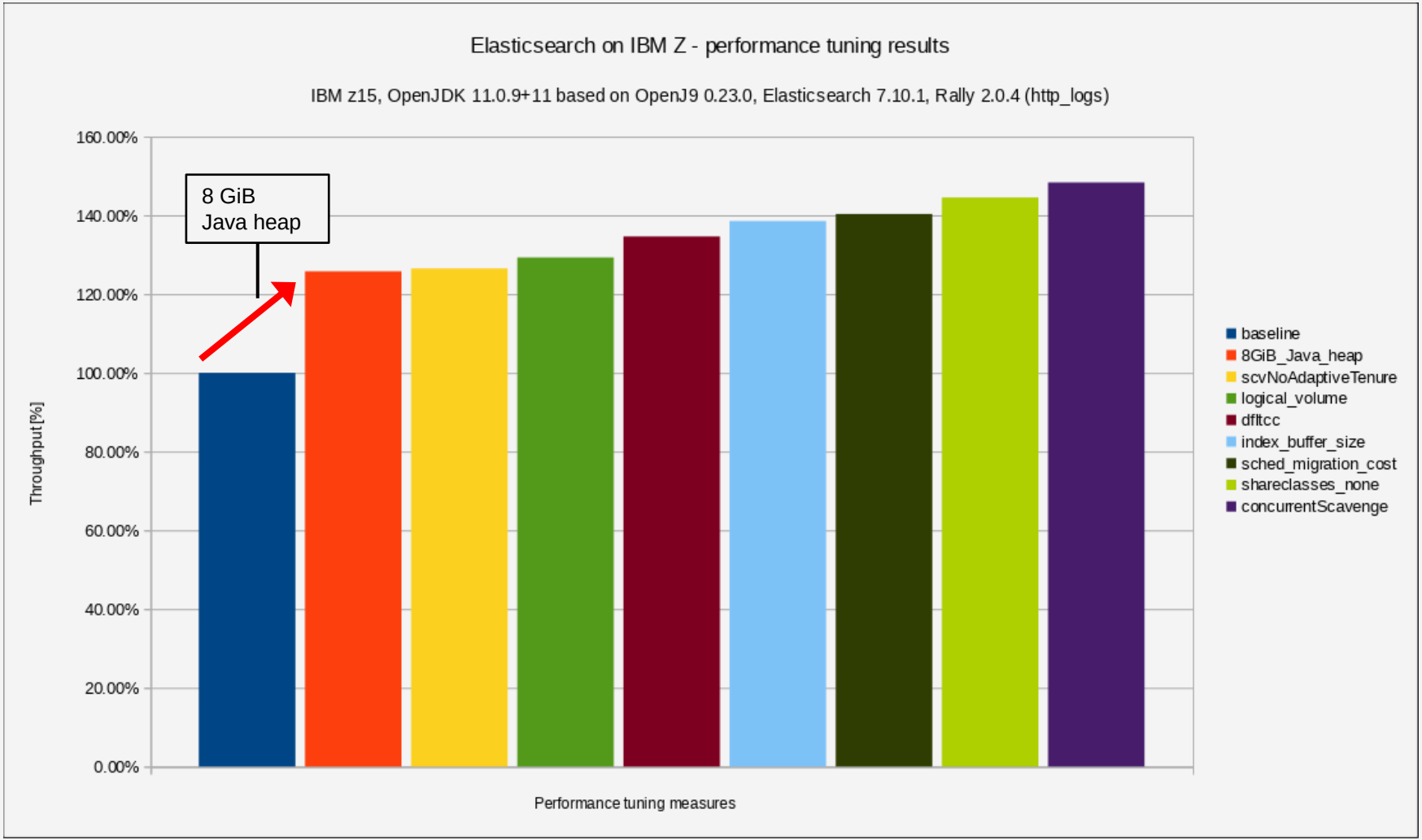
- Recommended for performance: configure the ***minimum*** heap size (`-Xms`) ***equal*** to the ***maximum*** heap size (`-Xmx`) in order to avoid *Java Virtual Machine* (JVM™) heap resize operations

  – Elasticsearch does this already out-of-the-box in the `jvm.options` configuration file

- ***Important***: always leave some ***free space*** in the system for other applications / daemons, etc. and the Linux® kernel itself, otherwise you will see activity from the Linux *out-of-memory* (OOM) killer

  – Holds true not only for Elasticsearch, but for all Java-based workloads

  – Memory pages that constitute the Java heap are regularly touched by Java garbage collection ⇨ memory is "active" from an operating system point of view


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
8 GiB
Java heap
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
Performance tuning measures
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge

Throughput recommendation #2

- Throughput recommendation #2: use a ***striped logical volume*** for storing Elasticsearch data

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

```
03/16/2021 09:17:15 AM
Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz  aqu-sz  %util
dasda            0.00      0.00     0.00   0.00    0.00     0.00    5.00    982.40     1.60  24.24    3.72   196.48    0.00   0.60
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
dasdc            0.00      0.00     0.00   0.00    0.00     0.00 1312.00 365877.60    18.60   1.40    4.62   278.87    2.89  95.40
…
```

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

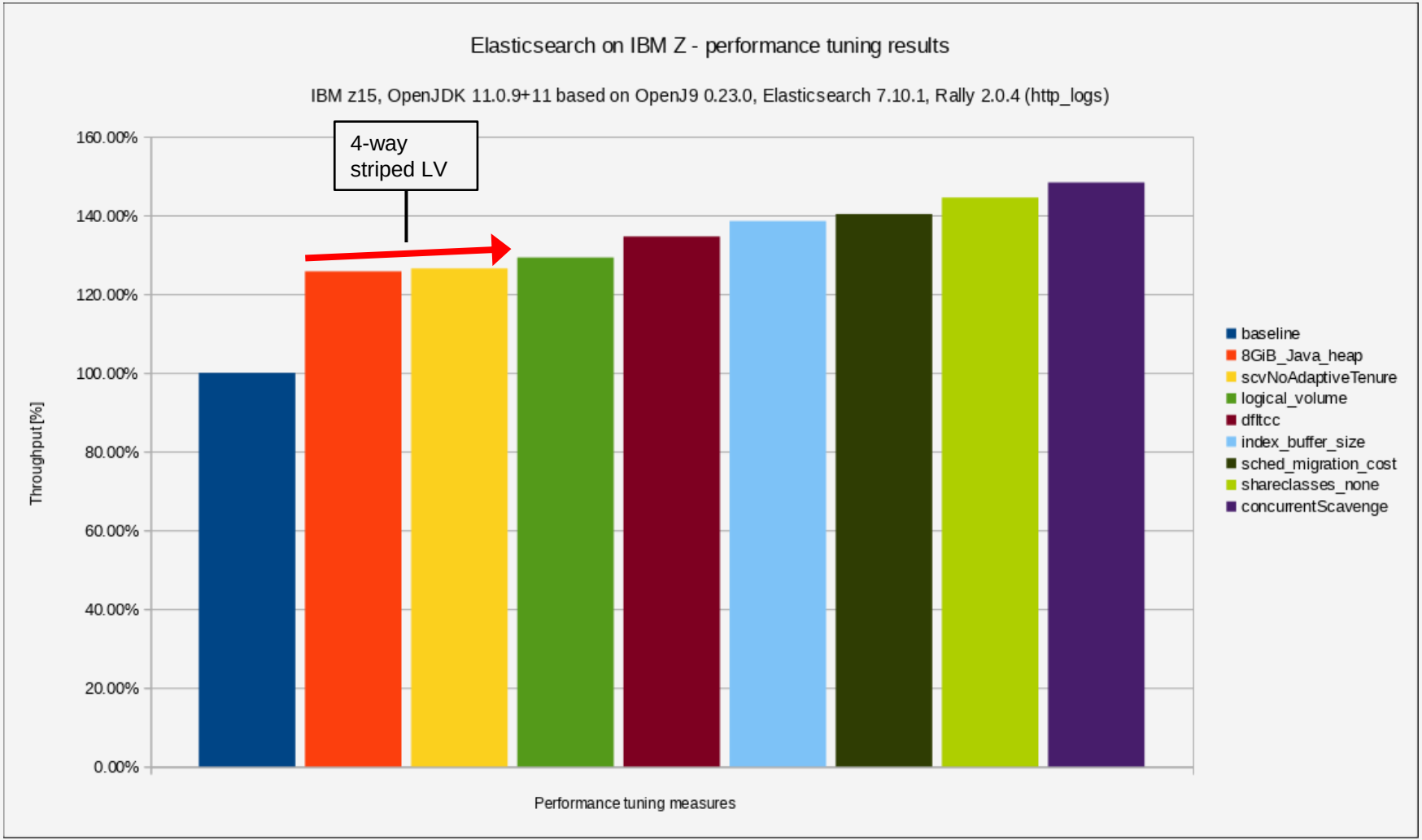
Throughput recommendation #2, *continued*

- ***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 4 for my tests
  - Double-check with `lvdisplay -m`
- Verify that the disk I/O load is really spread over all 4 devices:

```
03/16/2021 10:35:58 AM
Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz  aqu-sz  %util
dasda            0.20      0.80     0.00   0.00    4.00     4.00    4.00   1028.80     3.20  44.44    2.15   257.20    0.00   0.60
dasdb            0.00      0.00     0.00   0.00    0.00     0.00 1647.80  93600.00   178.40   9.77    1.37    56.80    0.00  91.60
dasdc            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      0.00     0.00   0.00    0.00     0.00 1488.80  92883.20   179.60  10.76    1.45    62.39    0.00  91.40
dasde            0.00      0.00     0.00   0.00    0.00     0.00 1576.20  93319.20   177.40  10.12    1.43    59.21    0.00  91.80
dasdf            0.00      0.00     0.00   0.00    0.00     0.00 1655.80  93764.80   179.00   9.76    1.37    56.63    0.00  91.60
dm-0             0.00      0.00     0.00   0.00    0.00     0.00 7083.00 373567.20     0.00   0.00    1.74    52.74   12.36  92.00
…
```

Throughput recommendation #2, *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*** Elasticsearch `data` directories – and thus, multiple devices and / or logical volumes – for storing index data
  - Makes things more complicated from an overall setup / configuration perspective
  - ***Shard*** data is spread over `path.data` entries in `elasticsearch.yml` configuration file
  - Files belonging to a single shard will all be stored in the same data directory


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
4-way
striped LV
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
Performance tuning measures
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge

Throughput recommendation #3

- Throughput recommendation #3: with IBM z15™, exploit the **Integrated Accelerator for *zEnterprise Data Compression* (zEDC)**, also known as *deflate* or `DFLTCC`
  - New instruction that was introduced with IBM z15

- During workload analysis with the Linux `perf` tool, I noticed that a measurable amount of CPU cycles in Elasticsearch was spent for ***compression*** – enabled by default, based on the LZ4 algorithm
  - Idea: let's make use of `DFLTCC`, if possible

- Searched the Elasticsearch and Lucene source code for a possibility to use GZIP-based compression
  - Reason: `DFLTCC` only supports compression based on the *deflate* algorithm such as GZIP

- 2 changes are needed: (1) Elasticsearch ***configuration file change*** and (2) Linux ***shell environment*** setup
  - See the next page for details

Throughput recommendation #3, *continued*

- Add ***an additional entry*** in the Elasticsearch configuration file called `elasticsearch.yml`:

```
index.codec: best_compression
```

- Add ***2 environment variables*** to your configuration – add a file in `/etc/profile.d`, modify `.profile` for the user running Elasticsearch, add `export` statements to `elasticsearch-env` script, etc.

```
export DFLTCC=1
export DFLTCC_LEVEL_MASK=0x01fe
```

- This enables hardware-accelerated compression ***for compression levels 1-8***

  – `0x01fe` as a bit mask: `0000 0001 1111 1110`

- See also Eberhard's blog entry: https://linux.mainframe.blog/zlib-acceleration

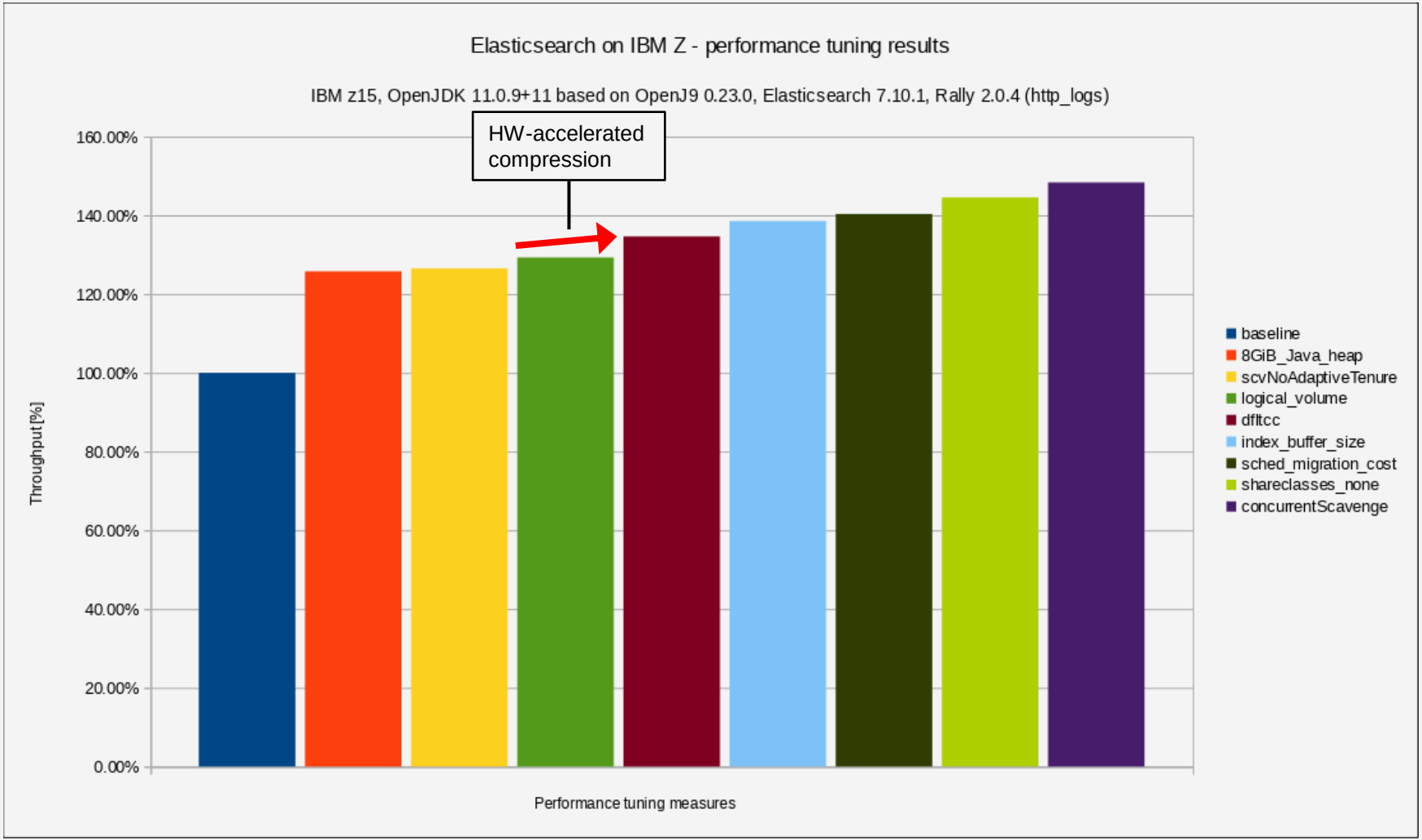
Throughput recommendation #3, *continued*

- ***Double-check*** the usage of `DFLTCC` instruction with the following `perf` command line:

```
perf stat -e DFLT_ACCESS,DFLT_CYCLES,DFLT_CC,DFLT_CCFINISH -a -p <JAVA_PID> -- sleep 10
```

- In the above command line, `<JAVA_PID>` is the ID of the Java process running Elasticsearch
- After enabling hardware-accelerated compression, counters ***increase significantly*** (ca. 6x – 8x)
- ***Pre-requisite***: LPAR has to be ***authorized*** for the corresponding counters
- Double-check LPAR authorization with:

```
lscpumf -c | grep DFLT
```


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
HW-accelerated compression
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
Performance tuning measures
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge

Throughput recommendation #4

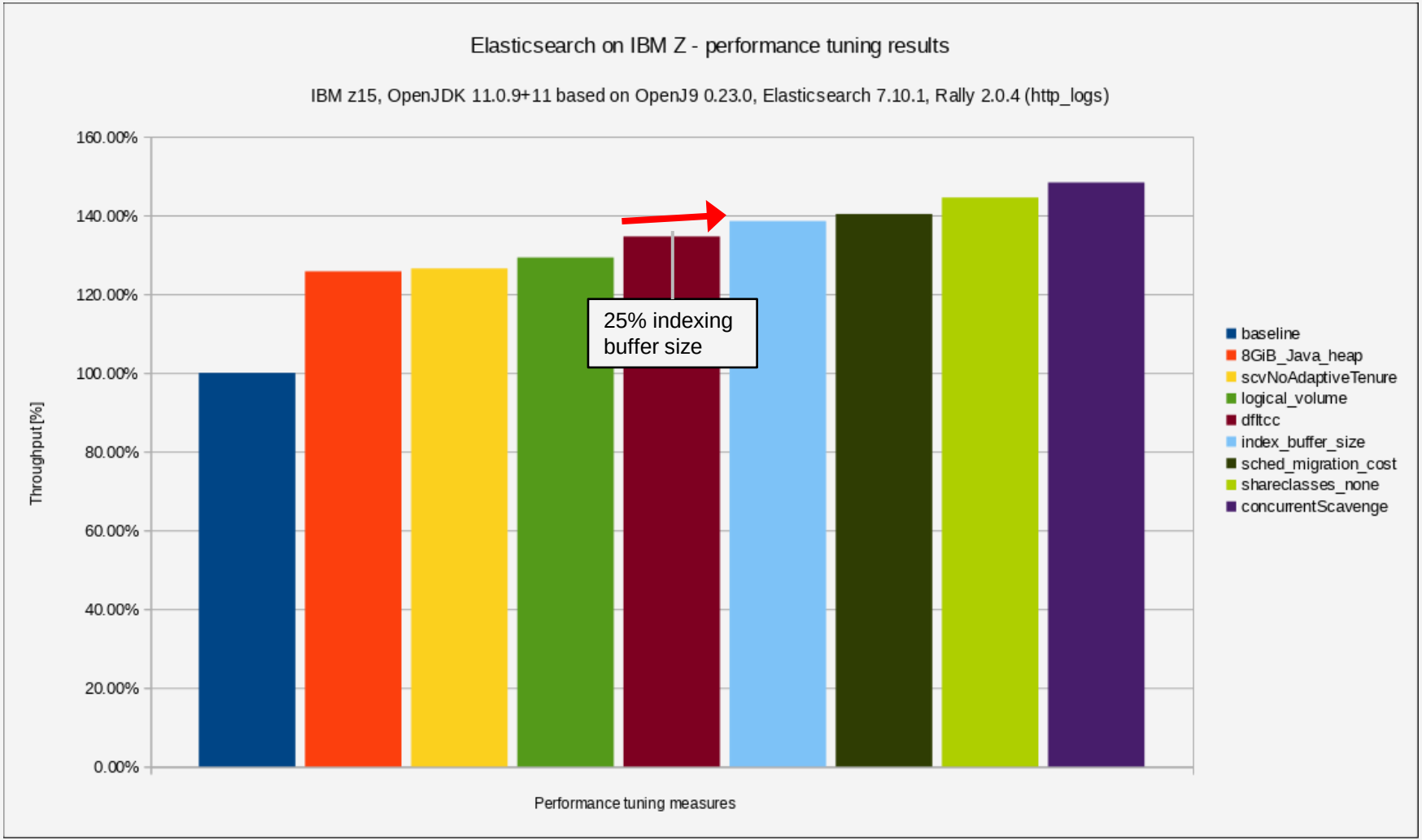
- Throughput recommendation #4: increase the size of the ***Elasticsearch indexing buffer*** to 25% of the available Java heap
  - In my test setup, this results in (25% of 8 GiB =) 2 GiB indexing buffer size
- Increase in size is recommended on a couple of Internet articles / blogs, etc.
  - Be careful: ***not*** all recommendations did actually increase indexing performance
- Add ***an additional entry*** in `elasticsearch.yml`:

```
indices.memory.index_buffer_size: 25%
```

- Some people recommend ***even higher*** values for the indexing buffer size
  - 25% seems ***reasonable*** because (a) higher values did ***not*** improve indexing performance any further in my test setup and (b) reserving too much memory for indexing might lead to `OutOfMemoryError`

Throughput recommendation #4, *continued*

- According to the Elasticsearch reference documentation *"…The indexing buffer is used to **store newly indexed documents**. When it fills up, the documents in the buffer are written to a segment on disk."*
  - On top of the indexing buffer, index data is also written to the ***translog*** in order to avoid data loss
- According to how the Elasticsearch indexing process works, the indexing buffer size should be configured taking into account the ***threshold size*** for a translog ***flush***
  - Indexing buffer is configured for the ***entire node***, whereas the translog flush threshold size is configured on a ***per index*** basis
- In my test setup, increasing the indexing buffer size was ***sufficient***, increasing the translog flush threshold size did not increase indexing performance any further
  - Depends on the ***overall indexing activity*** on the node: amount of incoming data per second, number of indexes running in parallel, write performance of the disks, etc.


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
25% indexing buffer size
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge
Performance tuning measures

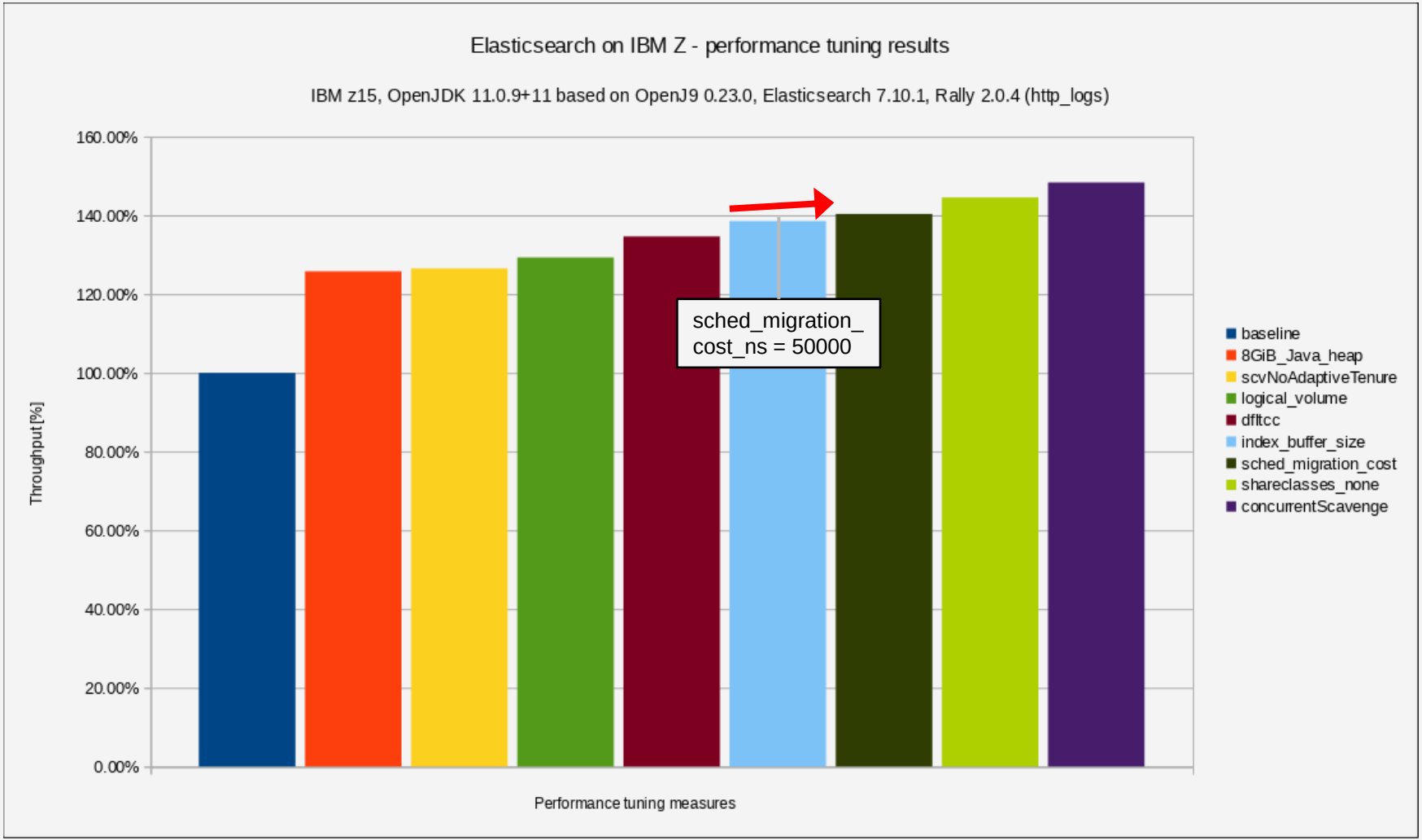
Throughput recommendation #5

- Throughput recommendation #5: consider lowering the Linux kernel ***scheduler migration cost***
- Linux `sysctl` setting that configures the ***number of nanoseconds*** the kernel will wait ***before considering moving*** a thread to another CPU
  - Default setting is appropriate for most workloads
- The ***higher*** this migration cost is, the ***longer*** the scheduler will wait before considering moving a thread to another CPU
  - Makes a lot of sense for large Linux images and / or scattered Linux images that span multiple PU chips and / or multiple nodes / clusters or even multiple drawers
- ***Additional entry*** in `/etc/sysctl.conf`:

```
kernel.sched_migration_cost_ns=50000
```

Throughput recommendation #5, *continued*

- Before applying this setting, double-check that:
  - There are ***more Linux processes / threads*** actively running on the system compared to the ***number of Linux CPUs*** available – can be verified with `ps`, `top`, or `vmstat`
  - There is still some ***idle time*** left although the system is under heavy load
- If the above conditions are ***not*** met, chances are high that the setting ***won't have*** any positive impact
- Besides the increased throughput, another ***indication*** that the Linux image is performing ***more useful work*** with this setting applied is that the amount of ***user time*** (%usr) increases measurably
  - Can be verified with `top` and / or `sar` data


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
sched_migration_
cost_ns = 50000
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge
Performance tuning measures

Throughput recommendation #6

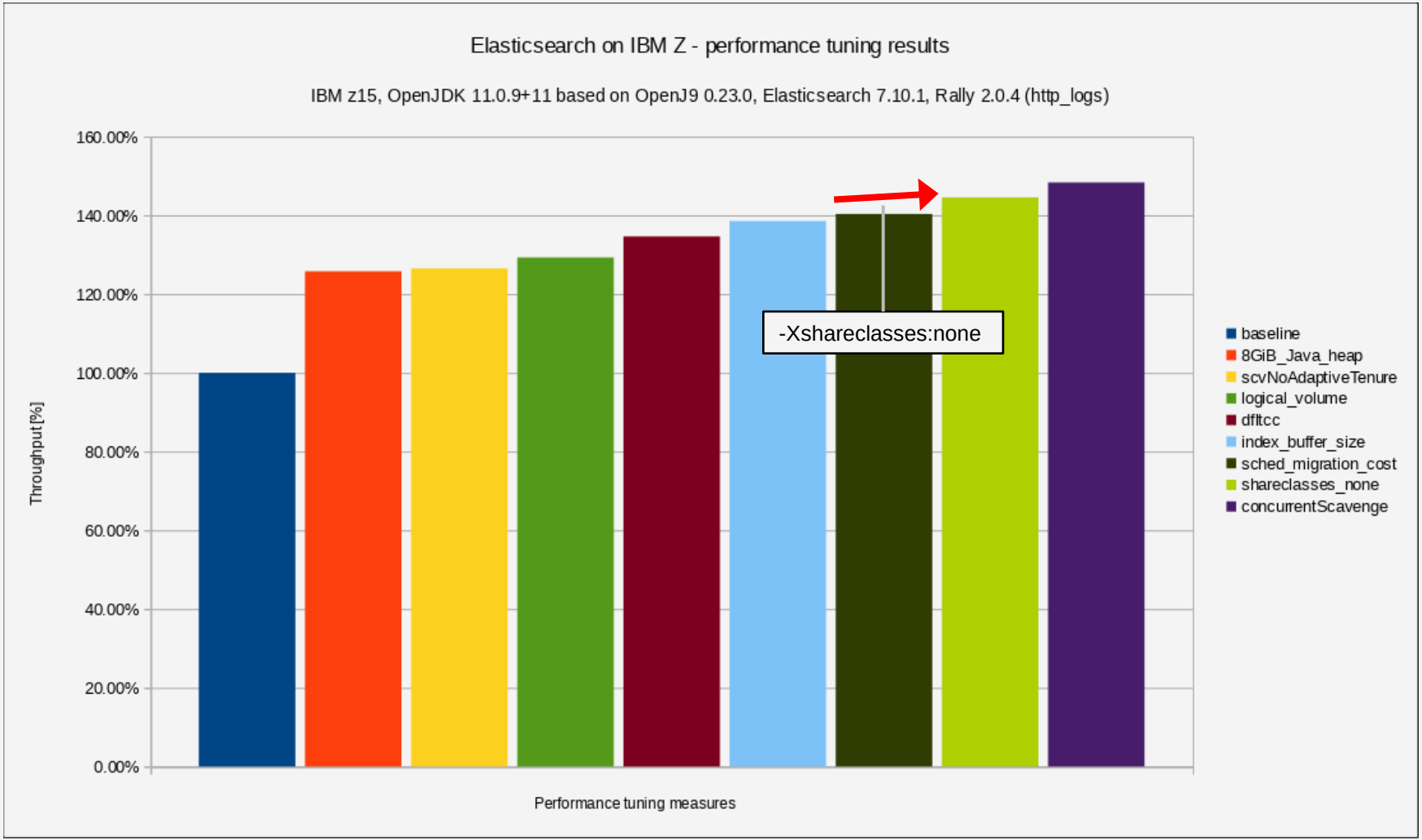
- Throughput recommendation #6: disable Java's ***shared class cache***
- ***Extremely useful*** option for multi-JVM scenarios, in particular for ***microservices*** type of applications
  - Only the ***first*** JVM needs to perform Java bytecode verification, native compilation, etc.
  - Subsequently started JVMs can ***re-use*** pre-verified Java classes and pre-compiled native code
  - Greatly improves JVM start-up times
- Since version 0.16.0 of OpenJ9, class data sharing is ***enabled by default***
  - Blog entry: https://blog.openj9.org/2019/10/15/openj9-class-sharing-is-enabled-by-default
  - Release history: https://www.eclipse.org/openj9/docs/version0.16

Throughput recommendation #6, *continued*

- Quote from an IBM Developer article written by the *Just-In-Time* (JIT) compiler team: *"One caveat is that … the code quality for AOT compilations is **somewhat lower** than the quality for JIT compilations."*
  - Article: https://developer.ibm.com/languages/java/articles/optimize-jvm-startup-with-eclipse-openjj9
  - AOT stands for *Ahead-Of-Time* compiler, used for native code that is placed into the shared cache
- OpenJ9 addresses this by ***recompiling*** frequently executed AOT bodies
  - Recompile is done by the "regular" JIT compiler
- If the workload is made up by ***many not-so-frequently*** executed AOT bodies, then those methods will ***stay*** at the mentioned lower compilation quality level
  - Heavily depends on the application

Throughput recommendation #6, *continued*

- ***Caveat:*** disabling the shared class cache will ***increase*** JVM start-up time
  - Shouldn't be an issue for long-running applications
  - Not recommended in scenarios where JVM start-up time is a significant contributor to overall application throughput and / or responsiveness
- Realized by adding an additional Java ***command line option***: `-Xshareclasses:none`
  - Add this option to the `jvm.options` configuration file


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
-Xshareclasses:none
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge
Performance tuning measures

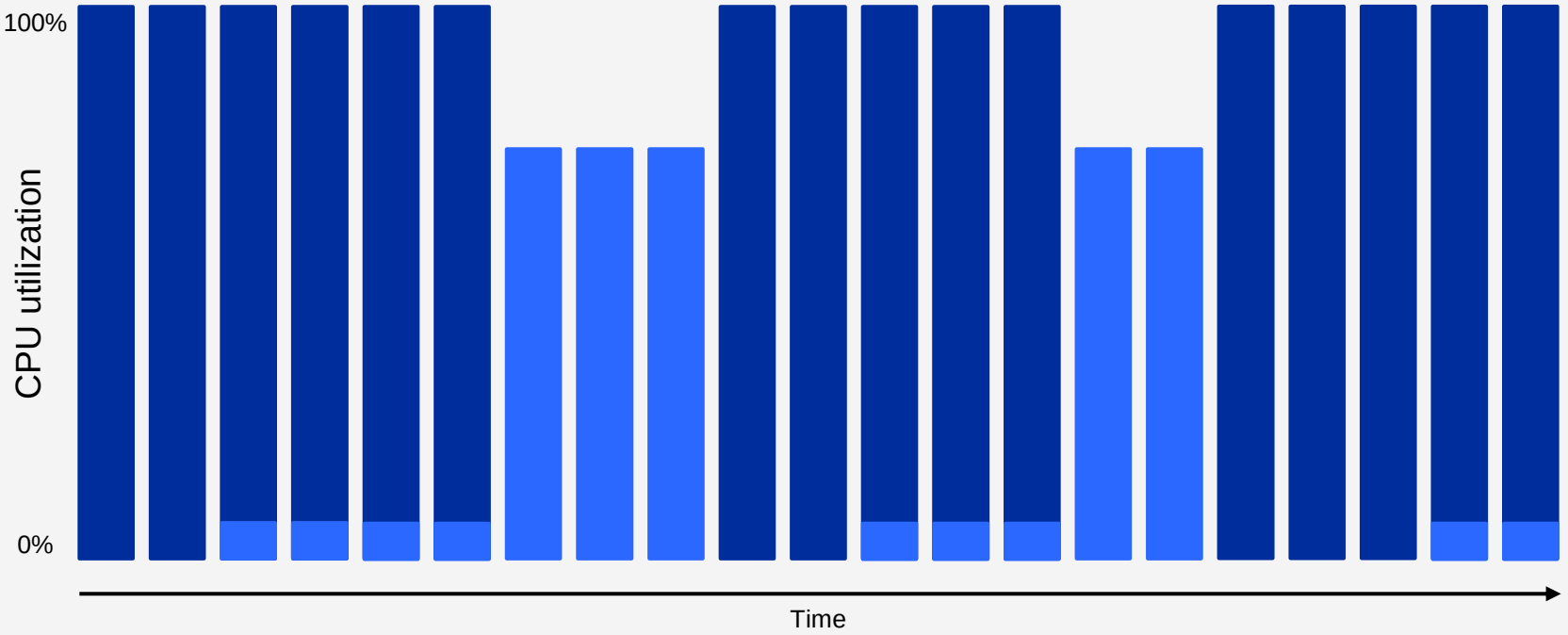
Throughput recommendation #7

- Throughput recommendation #7: enable ***pause-less garbage collection***
- Add an ***additional entry*** to the `jvm.options` configuration file:

  `-Xgc:concurrentScavenge`

- ***Important***: pause-less garbage collection does ***not*** "by magic" improve the CPU cost of garbage collection
  - Amount of garbage is still the same, even with pause-less garbage collection enabled
  - Very probably, CPU cost is even slightly higher with pause-less garbage collection enabled
- What it actually does is that is ***improves*** the ***pause times*** significantly
  - Achieved by running most of the garbage collection activity ***in parallel*** to the application threads

Regular garbage collection with gencon

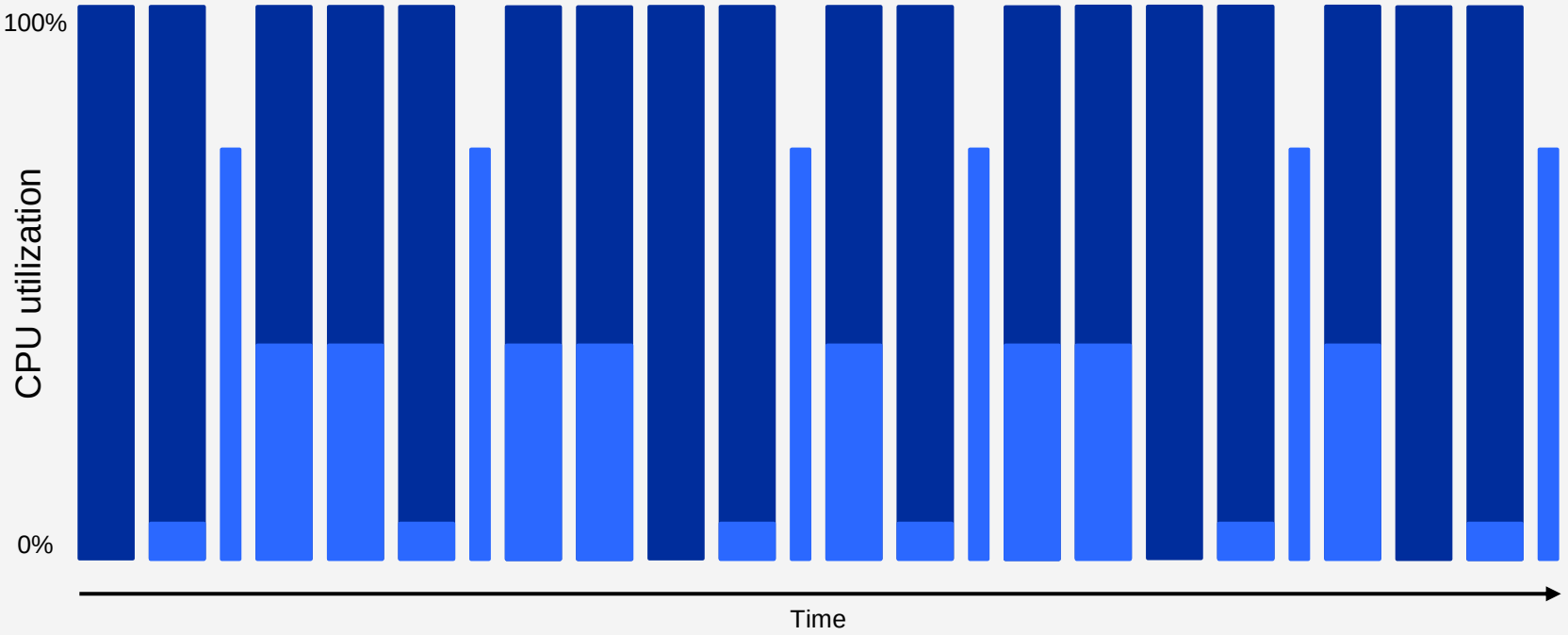


Picture is only illustrative and does not reflect any particular real-world application and / or CPU utilization values. Observations only hold true for large OS images.

Application threads

GC threads

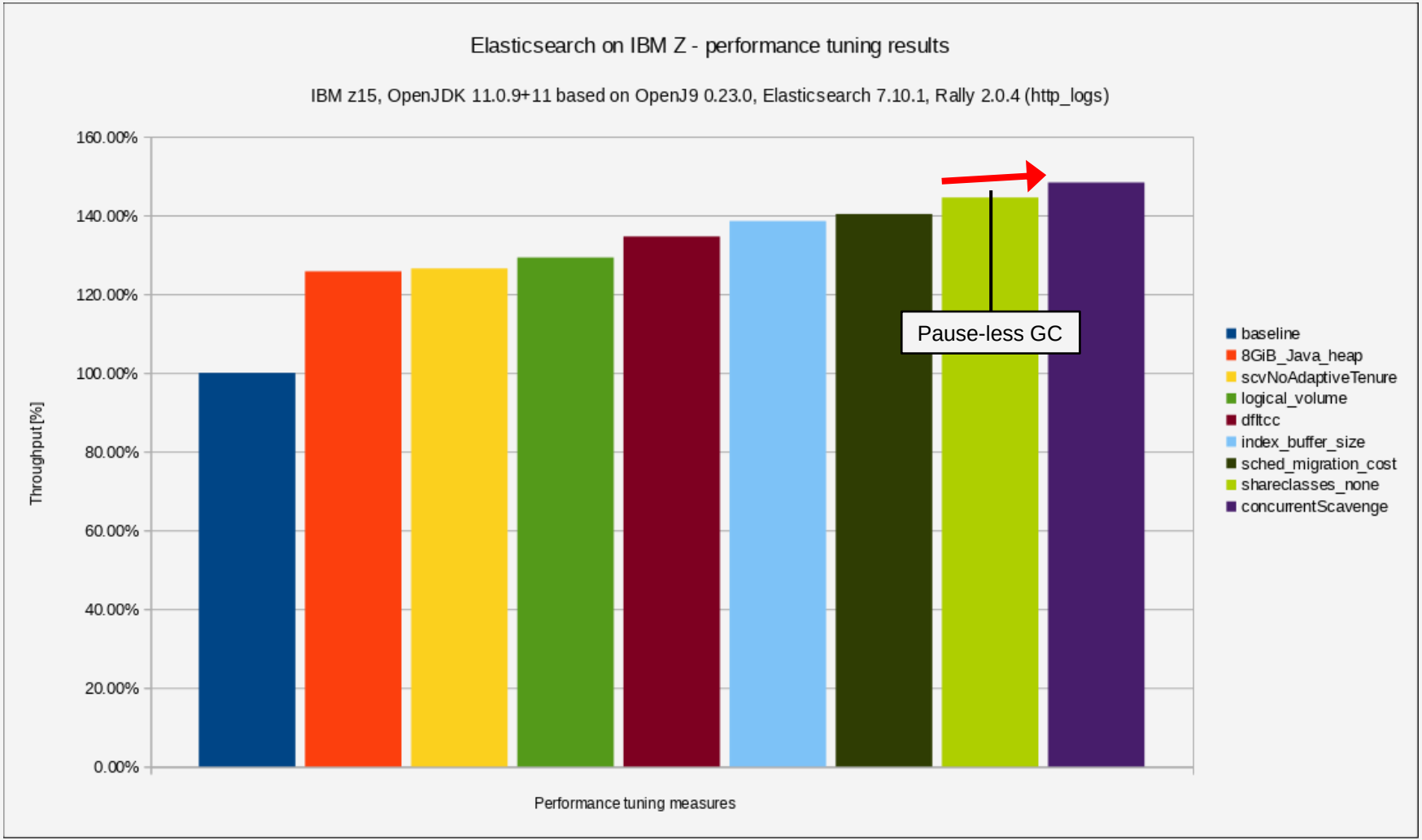
Pause-less garbage collection



Picture is only illustrative and does not reflect any particular real-world application and / or CPU utilization values. Observations only hold true for large OS images.

Application threads

GC threads


Elasticsearch on IBM Z - performance tuning results
IBM z15, OpenJDK 11.0.9+11 based on OpenJ9 0.23.0, Elasticsearch 7.10.1, Rally 2.0.4 (http_logs)
160.00%
140.00%
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Throughput [%]
Performance tuning measures
Pause-less GC
baseline
8GiB_Java_heap
scvNoAdaptiveTenure
logical_volume
dfltcc
index_buffer_size
sched_migration_cost
shareclasses_none
concurrentScavenge

Agenda

- Introduction to Elasticsearch
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
- Summary

Response time recommendation #1

- Response time recommendation #1: configure a ***fixed tenure age*** for the generational garbage collector
- Typically, response time is more important in a ***query-heavy*** scenario
  - Nonetheless, this might also be of interest for indexing, depending on the usage scenario
- The generational garbage collector – `gencon` for short – copies Java objects ***back and forth*** between the *allocate* and the *survivor* space a number of times before they get promoted into the *tenured* space
  - By design, this algorithm very elegantly performs a ***separation*** of short-lived and long-lived objects
- Without additional configuration, the amount of copy operations is adjusted ***heuristically***
  - Configuring a fixed tenure age effectively bypasses the mentioned heuristics

Response time recommendation #1, *continued*

- Add an ***additional entry*** to the `jvm.options` configuration file:

  ```
  -Xgc:scvNoAdaptiveTenure,scvTenureAge=2
  ```

- When fine-tuning (a) the different sizes of the Java heap regions and (b) the tenure age, you can think of this exercise as ***adjusting the pressure balance system*** between the nursery and the tenured heap
  - Reducing the pressure on one end means increasing it on the other
  - Lowering the tenure age effectively reduces the pressure on the nursery, because objects are moved to the tenured heap more quickly

- Reduce pause times ***even further*** by enabling pause-less garbage collection:

  ```
  -Xgc:scvNoAdaptiveTenure,scvTenureAge=2,concurrentScavenge
  ```

Response time recommendation #1, *continued*

- **8 GiB Java heap**

Summary

Concurrent collection count	5
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	27.2
Global collections - Mean interval between collections (ms)	52323
Global collections - Number of collections	5
Global collections - Total amount tenured (MB)	7635
Largest memory request (bytes)	4194312
Number of collections triggered by allocation failure	465
Nursery collections - Mean garbage collection pause (ms)	40.6
Nursery collections - Mean interval between collections (ms)	645
Nursery collections - Number of collections	465
Nursery collections - Total amount flipped (MB)	223275
Nursery collections - Total amount tenured (MB)	14824
Proportion of time spent in garbage collection pauses (%)	6.51
Proportion of time spent unpaused (%)	93.49
Rate of garbage collection (MB/minutes)	277481

- **-Xgc:scvNoAdaptiveTenure,scvTenureAge=2**

Summary

Concurrent collection count	7
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	22.9
Global collections - Mean interval between collections (ms)	41854
Global collections - Number of collections	7
Global collections - Total amount tenured (MB)	9651
Largest memory request (bytes)	1088256
Number of collections triggered by allocation failure	404
Nursery collections - Mean garbage collection pause (ms)	22.7
Nursery collections - Mean interval between collections (ms)	741
Nursery collections - Number of collections	404
Nursery collections - Total amount flipped (MB)	73742
Nursery collections - Total amount tenured (MB)	20596
Proportion of time spent in garbage collection pauses (%)	3.23
Proportion of time spent unpaused (%)	96.77
Rate of garbage collection (MB/minutes)	284299

Response time recommendation #1, *continued*

- **-Xgc:scvNoAdaptiveTenure,scvTenureAge=2**

Summary

Concurrent collection count	7
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	22.9
Global collections - Mean interval between collections (ms)	41854
Global collections - Number of collections	7
Global collections - Total amount tenured (MB)	9651
Largest memory request (bytes)	1088256
Number of collections triggered by allocation failure	404
Nursery collections - Mean garbage collection pause (ms)	22.7
Nursery collections - Mean interval between collections (ms)	741
Nursery collections - Number of collections	404
Nursery collections - Total amount flipped (MB)	73742
Nursery collections - Total amount tenured (MB)	20596
Proportion of time spent in garbage collection pauses (%)	3.23
Proportion of time spent unpaused (%)	96.77
Rate of garbage collection (MB/minutes)	284299

- **-Xgc:scvNoAdaptiveTenure,scvTenureAge=2, concurrentScavenge**

Summary

Concurrent collection count	14
Forced collection count	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	23.1
Global collections - Mean interval between collections (ms)	38542
Global collections - Number of collections	7
Global collections - Total amount tenured (MB)	28810
Largest memory request (bytes)	1549488
Number of collections triggered by allocation failure	875
Nursery collections - Mean garbage collection pause (ms)	0.63
Nursery collections - Mean interval between collections (ms)	681
Nursery collections - Number of collections	882
Nursery collections - Total amount flipped (MB)	489
Nursery collections - Total amount tenured (MB)	26.6
Proportion of time spent in garbage collection pauses (%)	0.4
Proportion of time spent unpaused (%)	99.6
Rate of garbage collection (MB/minutes)	308346

Agenda

- Introduction to Elasticsearch
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
- Summary

Summary

- Elasticsearch is a very popular Open Source ***search engine***, 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 Elasticsearch installation on Linux on IBM Z®
  - Recommendations are spread across the ***entire stack***: striped logical volume, `DFLTCC`, Linux kernel scheduler migration cost, Java heap size tuning, indexing buffer size

- Even if it's not strictly required for bulk indexing performance, you might also consider implementing the ***response time*** recommendations
  - Pause-less garbage collection is a ***great*** piece of technology

- ***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 Z and IBM LinuxONE
  - Official homepage: https://www.ibm.com/it-infrastructure/z/os/linux
  - Documentation: https://www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html
  - Tuning hints and tips: https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/tuning/tuning.htm

- Performance Radar Blog: https://ibm.biz/perfradar

- 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 with OpenJ9 on AdoptOpenJDK:
  https://adoptopenjdk.net/releases.html?variant=openjdk11&jvmVariant=openj9

- User Guide (includes diagnostics information) for IBM Java V8:
  https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/welcome/downloads.html