DOI: 10.2507/30th.daaam.proceedings.049

INFLUENCE OF NUMA AND MEMORY LOCALITY ON MICROSOFT SQL SERVER 2019 PERFORMANCE

Vedran Dakić

This Publication has to be referred as: Dakic, V[edran] (2019). Influence of NUMA and Memory Locality on Microsoft SQL Server 2019 Performance, Proceedings of the 30th DAAAM International Symposium, pp.0364-0369, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/30th.daaam.proceedings.049

Abstract

Memory locality in relation to which CPU core (*Central Processing Unit*) a certain virtual machine is using is an essential part of extracting maximum performance out of our NUMA-aware (*Non Uniform Memory Access)* applications. That means that designing and configuring our virtual server infrastructure correctly can be a complex process. In this paper, we're going to objectively analyze the impact of correct NUMA settings on Microsoft SQL Server Enterprise 2019 in-memory database performance on VMware's vSphere 6.0 hypervisor. We are going to use these findings to try to reach a conclusion as to what's the correct way to design server and virtual machine infrastructure for optimum performance so that our SQL queries aren't paying the unnecessary latency price because of the wrong memory locality and virtual machine design.

Keywords: NUMA; performance; in-memory database; locality; memory

1. Introduction

In Uniform Memory Access (UMA) systems, typically only one CPU can access system memory at a time [1]. This can lead to significant performance reductions in SMP systems, with more severe degradation occurring as the number of processors in a system is increased [1].

In a shared memory setting the multi-socket CPUs are equipped with their own memory module, and access memory modules across sockets in a NUMA pattern [2]. Memory access across socket is relatively expensive compared to memory access within a socket [2]. And although the interconnect interfaces between CPU sockets are much faster than what they were, CPUs are also much faster, their built-in memory controllers are also faster, and memory is also faster. That means that, no matter the fact that Intel and AMD lowered the latencies and increased the speed of these interconnects (for example, QPI or *QuickPath Interconnect* on Intel architectures, introduced in 2007.), that speed isn't something that is going to be able to compensate for all the increases in CPU, memory controller and memory speed.

High-performance servers are non-uniform memory access (NUMA) machines [3]. NUMA moves away from a centralized pool of memory and introduces topological properties. By classifying memory location based on the signal path length from the processor to the memory, latency and bandwidth bottlenecks are avoided [4]. This also means that we need to do everything we can to keep the memory traffic local to the CPU, as that's going to be the fastest way – both in terms of latencies, and in terms of pure memory bandwidth.

Databases are the perfect type of service to test this problem, as they are heavily memory-limited, and memory locality plays a significant role in database performance. That means that a I/O heavy database query in a non-NUMA configuration will lose a lot of time going across interconnect interfaces and back to be able to access data that it needs to read from memory, and to write data that it needs to write to memory. With time, this becomes an even bigger problem, as databases grow in size, which increases the overhead for regular operations, like reading data from tables and writing data to them. Correct virtual machine configuration in terms of NUMA should help these problems a lot, as it will eliminate any expensive intra-socket interconnect communication, which means that it won't suffer the same latency and bandwidth restrictions that incur with incorrect NUMA virtual machine configuration.

In 2014, Microsoft introduced a new feature for its SQL Server databases, called In-Memory Database, which was in line with what was happening on the market at the time. For example, SAP HANA and Oracle databases also have similar features. This feature should be able to cope with a huge growth in data being generated and committed to databases, while at the same time offering low latencies and high throughput for database queries.

By using VMware Hypervisor and Microsoft SQL NUMA awareness, In-Memory database and various NUMA configurations – we will try to find a correct way to configure our virtual machines running Microsoft SQL so that we have the highest levels of performance. Our premise is that only one NUMA configuration should consistently offer the highest levels of performance, and that should be the single-node NUMA virtual machine configuration with all memory local to the CPU (connected directly to the CPU's memory controller). All other configurations should offer lower levels of performance, and the performance gap should grow as we grow SQL query's sample size.

2. Methodology

Measuring performance of an IT system is all about repeatable methodology so, that was the primary focus of this research. We used the standard AdventureWorks database which is available from Microsoft and github (you can find it on this shortened link: https://bit.ly/2zsevpc). Having in mind that we need a fixed sample size to get repeatable, yet complex database procedure, we cleaned the database from the unnecessary tables and left just two of them (Production.TransactionHistory and Sales.SalesOrderDetail). These tables have more than 100.000 rows each, which is enough of a sample to do our testing. Keep in mind that modern server applications are expected to process a huge number of requests simultaneously without noticeable degradation of performance, e.g. response times and throughput [5], which is specifically what we're measuring in our paper.

Testing was done on an isolated HP ProLiant DL380 Gen9 server, 2xIntel Xeon E5-2698v3 16-core CPU, 192GB memory (96GB per CPU) and Intel DC P3500 PCI-Express SSD, where database was stored. Operating system (Windows Server 2016) was stored on a separate Intel DC P3500 PCI-Express SSD, to minimize any kind of influence that OS might have on database performance, which is also in line with Microsoft SQL Server design best practices. Virtual machine that we used for testing used 64GB of memory, with "Reserve all guest memory (All locked)" option turned on. This was done so that we avoid any kind of memory sharing, paging, or any kind of external influence on the amount of real memory that was used to feed this testing virtual machine with memory. Also, by using *bcdedit* and SQL server configuration, we tuned the way SQL uses CPU and memory so that it's in accordance to our testing scenarios.

During testing, we measured in-memory database performance with a very memory-intensive test – a *cross join* between our two tables. Result of a cross join SQL is a cartesian product, which combines every row from first table with every row from the second table. So, if a first table has X number of rows and the second one has Y number of rows, the resulting table will have X*Y rows. Having in mind that the subject of this research is memory performance specifically, we used this procedure as it has a couple of very useful characteristics:

- It doesn't use a lot of CPU time and it's not CPU-limited;
- It uses a lot of memory I/O as it needs to read all the rows from table X and table Y, and then create a resulting, cartesian product table which is X*Y in dimensions.

The way in which cross join works is illustrated on the following picture:

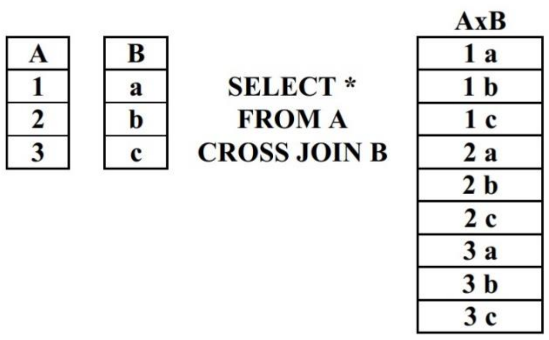

Fig. 1. SQL cross join

Testing has been divided in to multiple runs:

- A 500x500 sample from the top of first and second table
- A 1000x1000 sample from the top of first and second table
- A 2000x2000 sample from the top of first and second table
- A 3000x3000 sample from the top of first and second table

All of these tests have been done three times (our table is going to use median values), with various NUMA node configurations:

- 1 socket x 16 CPU cores
- 2 sockets x 8 CPU cores
- 4 sockets x 4 CPU cores
- 8 sockets x 2 CPU cores
- 16 sockets x 1 CPU cores

By measuring all of these scenarios we want to learn how NUMA node configuration (which is different in all five configurations) influences the memory performance of our SQL cross join procedure. We also used a setting called CPU affinity to lock virtual machine CPU execution on physical cores only, as the CPU in our server also supports Hyperthreading, which we didn’t want to use. By specifying a CPU affinity setting for each virtual machine, you can restrict the assignment of virtual machines to a subset of the available processors in multiprocessor systems [6]. This option will tell the VMware Hypervisor kernel to ignore all virtual machine execution calls for all of the other logical cores that it can see on the server. We’re using “Cores per socket” setting to create a different amount of logical NUMA nodes to simulate real-life NUMA node behaviour. By default, ESXi NUMA scheduling and related optimizations are enabled only on systems with a total of at least four CPU cores and with at least two CPU cores per NUMA node [7].

For measuring the time needed to do cross join procedure, we used the *set statistics time* statement, which measures the time necessary to parse, compile and execute a SQL procedure in two distinctive phases : CPU time (how much work is being done by CPU, in milliseconds), and elapsed time (time that’s needed for a complete procedure to finish, in milliseconds).

3. Results

For our results, we created a table and five charts that show the influence of NUMA configuration on time needed to complete SQL query for various sample sizes (in rows of first and second table). Our expectation was:

- If we use CPU affinity to “pin” VM workload to physical cores, the fastest performance should always come from a scenario in which we have one NUMA node (physical CPU), if that NUMA node has enough memory to handle the complete VM workload (which it does, as 96GB per socket in our testing systems leaves more then 30GB of free memory on that CPU);
- Every other configuration should have smaller, non-linear performance drop, as our SQL query will spend more time in “overhead”, accessing data that might not be on the same NUMA node via QPI interface;
- The bigger the number of simulated sockets is, the bigger the difference should be (again, as SQL query will spend more time waiting for overhead tasks to complete, no matter the logical NUMA mapping.

Here’s our table with results:

Sample size	**1 socket x 16 cores**	**2 socket x 8 cores**	**4 sockets x 4 cores**	**8 sockets x 2 cores**	**16 sockets x 1 cores**
500x500	4994	5003	5017	5093	5130
1000x1000	19354	19590	19906	19965	20350
2000x2000	76592	76920	78152	78487	79324
3000x3000	173080	173126	175256	176099	178193

Table 1. SQL table row sample size (rows) vs CPU/NUMA configuration (columns), measurement in milliseconds

Also, are our corresponding charts, per sample size:

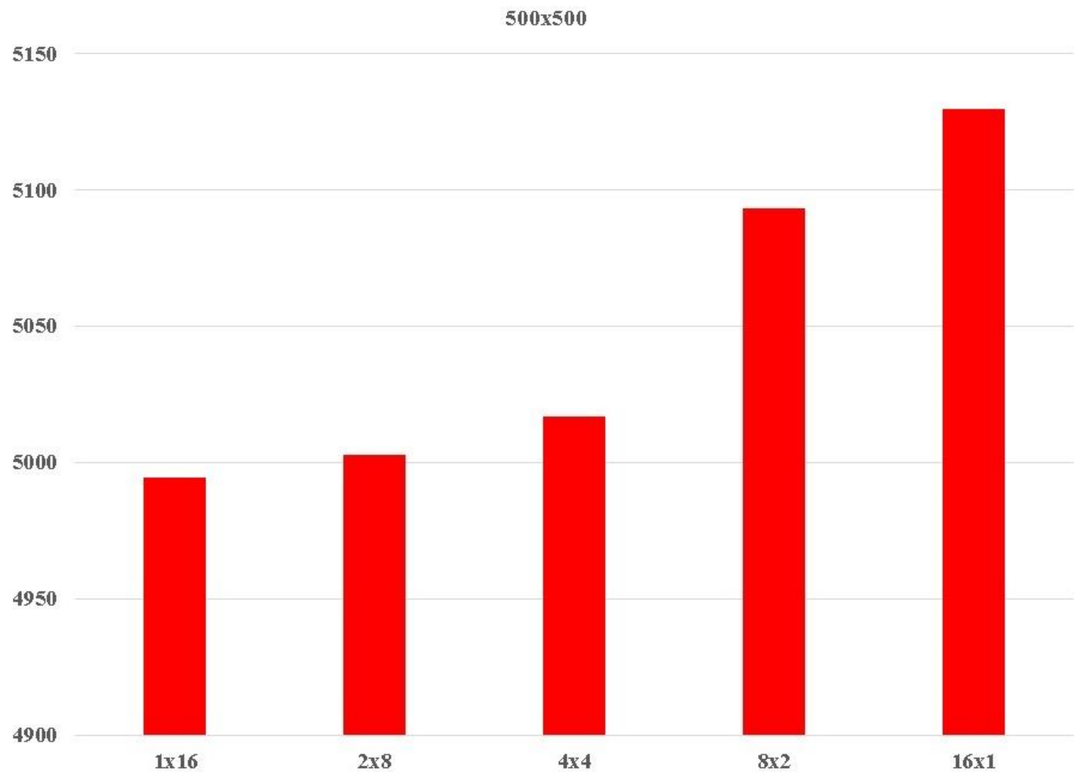

Fig. 2. Performance for 500x500 sample size in milliseconds (lower number = better performance)

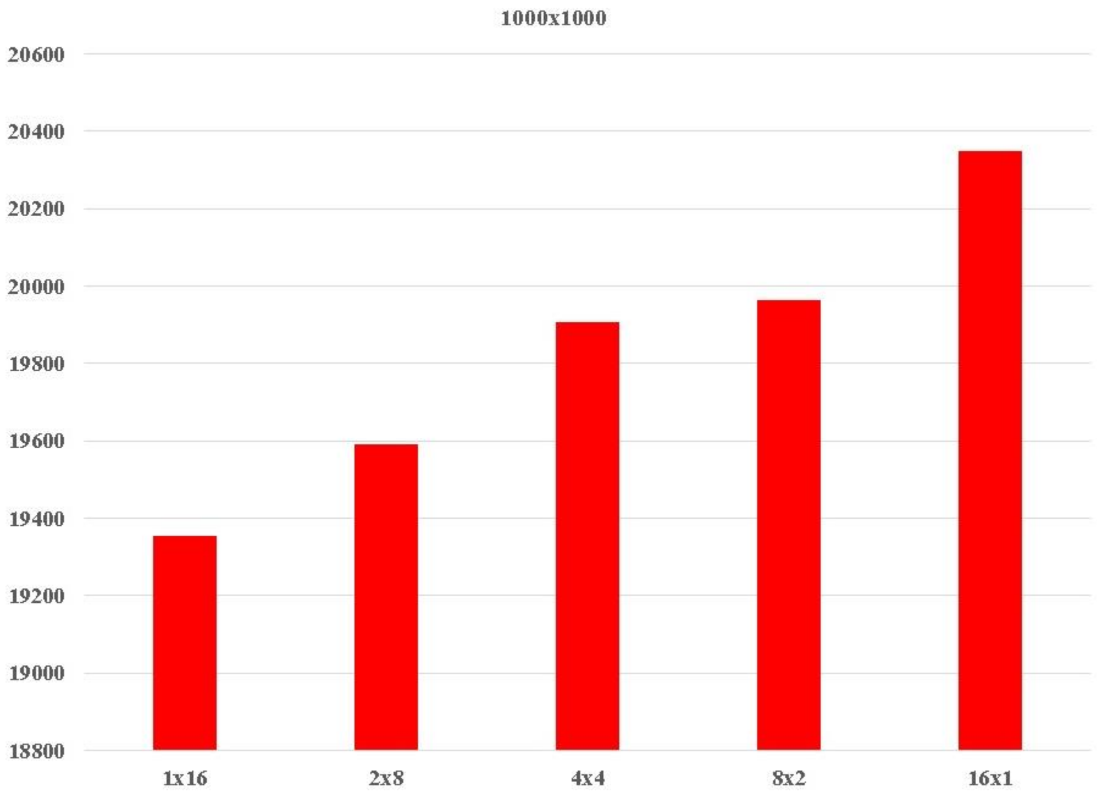

Fig. 3. Performance for 1000x1000 sample size in milliseconds (lower number = better performance)

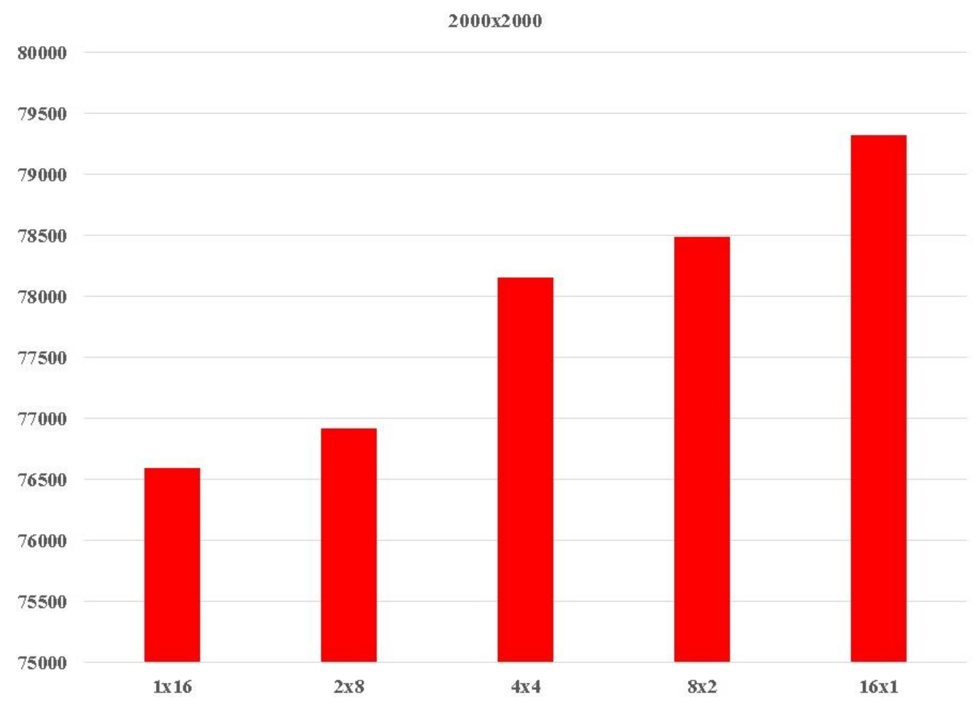

Fig. 4. Performance for 2000x2000 sample size in milliseconds (lower number = better performance)

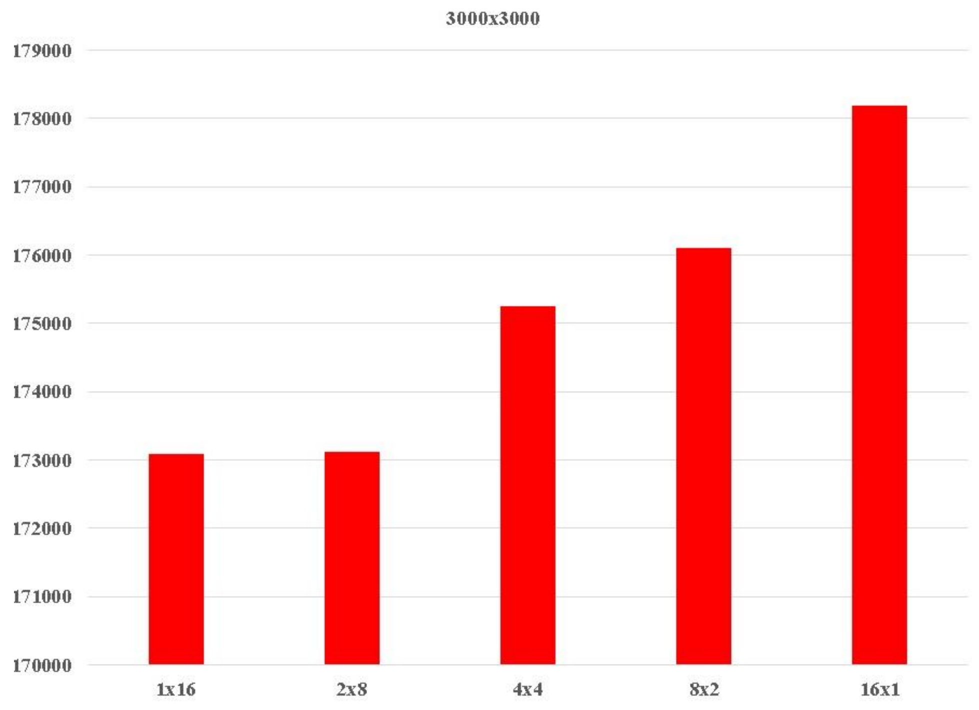

Fig. 5. Performance for 3000x3000 sample size in milliseconds (lower number = better performance)

4. Conclusions

There are two key questions that we're trying to answer with this paper:

1. What are the performance levels of in-memory SQL Server database with correctly configured NUMA vs incorrectly configured NUMA in virtual machines on VMware vSphere Hypervisor?
2. After seeing performance impact of NUMA correct/incorrect configurations, what is the correct approach to design physical servers and virtual machines for in-memory Microsoft SQL database when using VMware vSphere Hypervisor?

We can use results of this research to introduce a standardized way of designing physical and virtualized infrastructure for Microsoft SQL Server database, regardless of the way we're consuming our database (in-memory, via virtual disk or via Raw Device Mapping). There are three main conclusions that we can draw from our results:

1. NUMA configuration for any given database virtual machine plays a significant role in the overall database performance;
2. It's a good policy (by VMware) to automate these settings so that correct NUMA configuration is always presented to the virtual machine (starting with vSphere Hypervisor 6.5);
3. There is a simple design pattern that needs to be followed when designing physical servers and virtual machines in order to be able to extract maximum sustainable performance from our infrastructure, which is to configure our physical and virtual servers so that all I/O for a virtual machine running Microsoft SQL database is kept local to the CPU socket where the virtual machine is running.

Our assumption was confirmed – if we *pin* virtual machine to a single NUMA node (single CPU socket), use only its physical cores and locally connected memory, performance will be the best. If we spread our virtual CPU execution cycles across multiple NUMA nodes, it will have significant influence on the memory performance of our SQL cross join procedure. To use our 3000x3000 rows as an example, the difference in performance between a 1x16 NUMA configuration and 16x1 NUMA configuration is more than five seconds. This difference will only get bigger the bigger our query is, as the trend is clearly visible in our results, as well.

As a counterpoint to our research, it would be fair to mention that there are applications that don't use NUMA at all, and therefor can't benefit from any kind of special configuration. For example, Exchange Server 2019 is not a NUMA-aware application and performance tests have shown no significant performance improvements by enabling vNUMA [9], so there would be no point in trying to configure something NUMA-related on a virtual machine that's running Microsoft Exchange. Furthermore, scaling up virtual machines with Exchange is not the best practice scenario. It doesn't scale very well with additional CPU cores, and its memory scale-out performance benefits also flatten out rather quickly, depending on a company/mailbox database size and scenario. So, the relationship between application performance, scale-out or scale-up gains and NUMA configuration is something that needs to be researched before finishing any kind of design that includes these applications.

No matter which version of VMware's vSphere Hypervisor you're using, generally it's a good idea to design your physical hardware CPU count amount the future SQL virtual machine CPU count. That means that, if your plan is to have a 16-core virtual machine running SQL, it would be a good idea to have at least 18 physical cores in your server, as additional cores will be used for hypervisor's work, as well. If the number of cores per virtual socket on a vNUMA-enabled VM is set to any value other than the default of one and that value doesn't align with the underlying physical host topology, performance might be slightly reduced [9]. That way, you won't split performance across NUMA nodes and you won't use fast, but still slower QPI inter-CPU links for memory access. Memory locality plays a significant role in SQL performance, and the "closer" memory is to the CPU, the less overhead we'll experience, as well as less un-evenness in terms of long-term performance.

Accepting this kind of design paradigm will have a long-term effect on both purchasing decisions and database performance. In the IT industry, companies mostly buy servers as an aggregate of resources coupled with their cost, less so for application and performance-based criteria primarily. This problem has deeper roots - in the long-term unfamiliarity of how applications work, how they scale, which resources are they hungry for, and – most importantly – how to properly design environments, which should be both down-up and up-down process.

5. References

[1] Hollowell, C.; Caramarcu, C.; Strecker-Kellogg W.; Wong, A. & Zaytsev, A. (2015). The Effect of NUMA Tunings on CPU Performance, Proceedings of 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015), IOP Publishing, doi:10.1088/1742-6596/664/9/092010

[2] Gawade, M.; Kersten, M. (2015). NUMA obliviousness through memory mapping, Proceedings of DaMon 2015, Sigmod, at Melbourne, Australia, ISBN 978-1-4503-3638-3

[3] Calciu, I.; Sen, S.; Balakrishman, M. & Aguilera M. (2017). Black-box Concurrent Data Structures for NUMA architectures, ASPLOS 2017., at Xi'an, China, ISBN 978-1-4503-4465-4

[4] Dennerman, F.; Hagoort, N. (2017). VMware vSphere 6.5 Host Resources Deep Dive, Frank Dennerman and Niels Hagoort, ISBN 9781540873064

[5] Randic, M.; Jednakovic, H.& Blaskovic, B. (2010.) Dynamic thread assignment in a tandem of threadpools inspired by the adaptetion mechanism in honeybee foraging, Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Vienna, Austria, ISBN 978-3-901509-73-5.

[6] VMware (2019). vSphere Resource Management, VMware.

[7] VMware (2015). Performance Best Practices for VMware vSphere 6.0, VMware.

[8] VMware (2019). Virtualizing Microsoft Exchange Server on VMware Best Practices Guide, VMware.

[9] Elder, K.; Kusek, Cristopher; Sarkar, P. (2017). vSphere High Performance Cookbook, Packt Publishing, ISBN: 978-1-78646-462-0