NUMA-aware Optimization on Apache Spark*

Xiaochang Wu

Intel Big Data Team

Legal Disclaimer

intel Software

Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more information go to http://www.intel.com/performance.

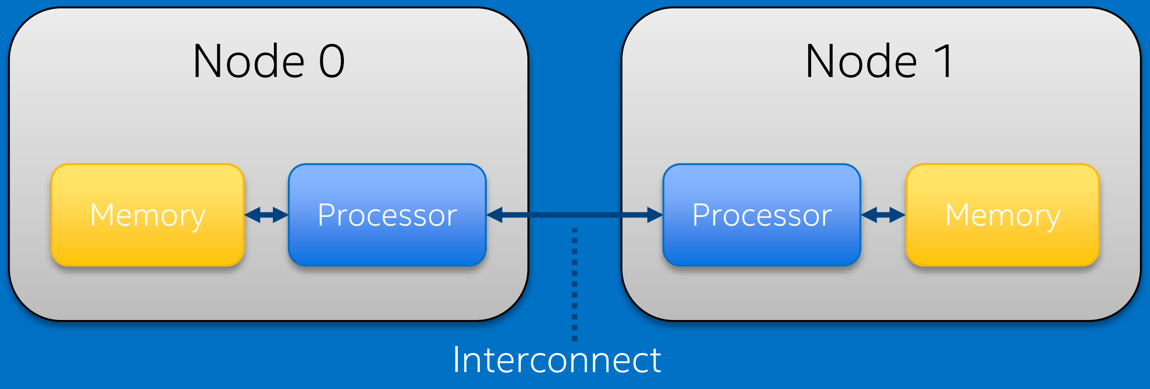
Agenda

- Introduction to NUMA
- Apache Spark* patch to enable NUMA support
- NUMA tuning for Apache Spark* applications
- Experiments on various workloads
- Conclusion

What is NUMA?

- Non Uniform Memory Access
- Address the problem of performance hit when several processors attempt to address the same memory.
- Multiple physical CPUs in a system. Each CPU has
  - Local memory: memory attached to it, fast.
  - Remote memory: memory attached not directly, slower.
- Developed commercially during the 1990s. Intel announced NUMA compatibility for its x86 servers in late 2007 with its Nehalem CPU.
- Trend: bring memory nearer to processor

Typical 2-Sockets Intel® Xeon® NUMA Topology



NUMA performance considerations:

- Latency: higher latency of accessing remote memory
- Bandwidth: interconnect contention

Avoid remote memory accesses!

```
$ numactl --hardware

available: 2 nodes (0-1)

node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65

node 0 size: 98207 MB

node 0 free: 92847 MB

node 1 cpus: 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87

node 1 size: 98304 MB

node 1 free: 94228 MB

node distances:

node   0   1

  0:  10  21

  1:  21  10
```

NUMA on Linux*

- Linux* kernel's default NUMA policy allocates pages from local node of the processor which makes the request, and fall back to other nodes if no local memory available.
- When a task (run by thread) is scheduled to a node, all previously allocated pages in other nodes will not be migrated to this node, from then on, the accessing to those pages will be remote and cost much more time.
- Solution: Manual binding using numactl command

```
 $ numactl --cpunodebind=$numa_node_bind --preferred=$numa_node_bind
<application command line>
```

*Other names and brands may be claimed as the property of others.

Software

NUMA memcpy test on Linux*

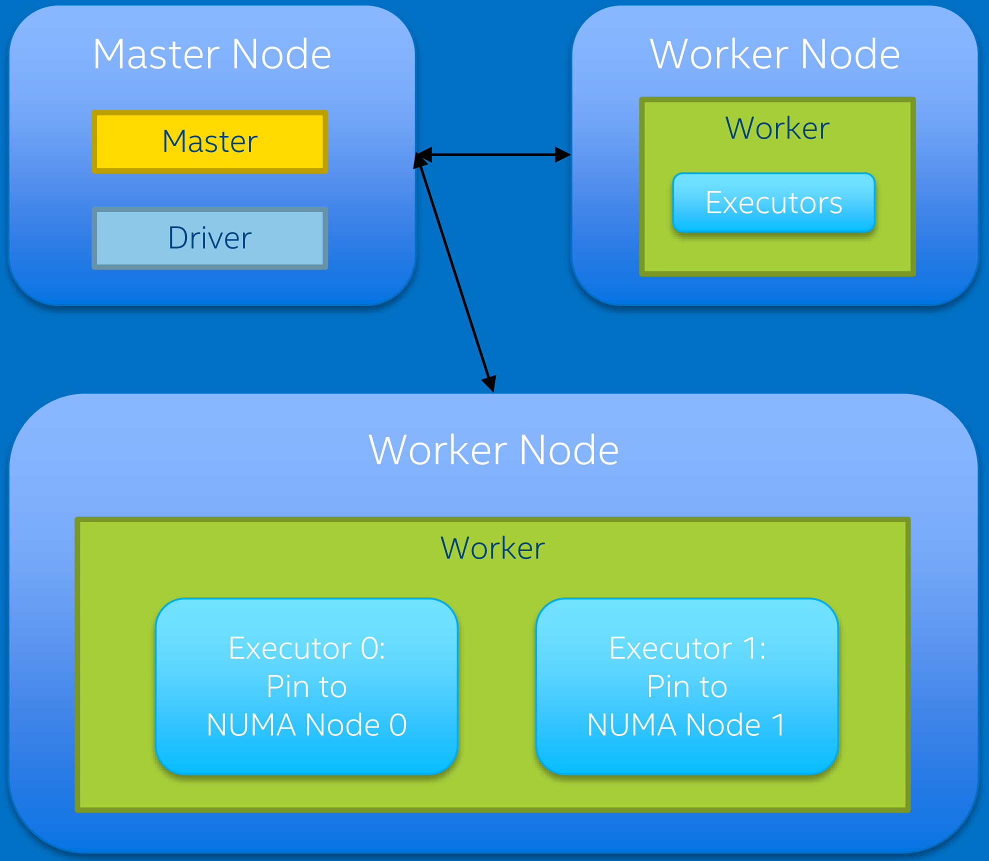
- memcpy test of numademo* tool
  - First allocate specified size of memory based on NUMA policy and copy the first half data to the rest half
- Default policy allocates pages on the node of the CPU that triggers the allocation.
- Set cpu-bind to node 0 and preferred memory-bind to node 0. Compare the performance between "alloc on node 1" vs. "local allocation" to simulate the worst case differences.

Data Size	4GB	8GB	16GB	32GB	64GB	128GB
Scaling	1.086x	1.074x	1.124x	1.116x	1.134x	1.079x

"local allocation" vs "alloc on node 1"

* numademo is part of Linux system tool numactl. Other names and brands may be claimed as the property of others.

Apache Spark Deployment Revisited



Binding Executors:

- Executors only run on binding CPU node
- Memory for executors allocated on binding CPU node preferably

Issues:

- Sometimes binding node may be too busy
- Chances to allocate memory on remote node
- Can’t predict workloads on application-level

No Silver Bullet!

Apache Spark Patch to enable prefix command and NUMA binding

- Patch: https://github.com/apache/spark/pull/16411
- This patch will support adding a prefix command to the original executor launch command line. Eg:

Prefix Command	Original Executor Command Line
spark-numa.sh	{{JAVA_HOME}}/bin/java -server -Xmx4096m -Djava.io.tmpdir={{PWD}}/tmp '-Dspark.driver.port=49187' --driver-url spark://CoarseGrainedScheduler@10.0.2.192:49187 --executor-id 26 --hostname sr593

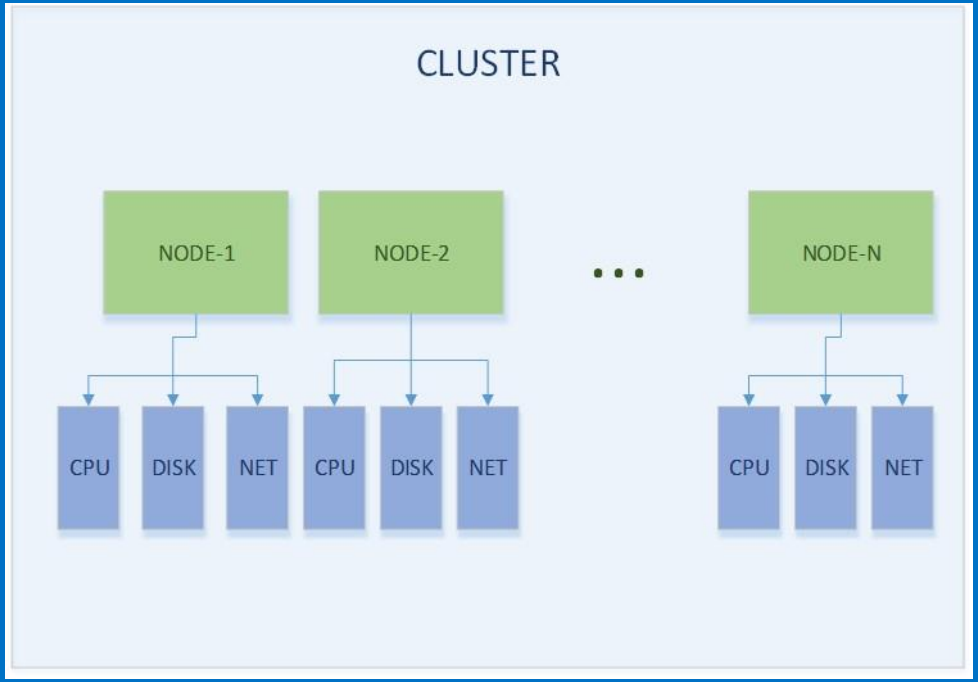
- This prefix command (*spark-numa.sh* in this case) can be a user customized script which invokes *numactl* to bind each executor to a specific node according to executor id.
- In our experiments, a simple interleave binding is applied to balance task load, users need to fine tune script for more complex situation.

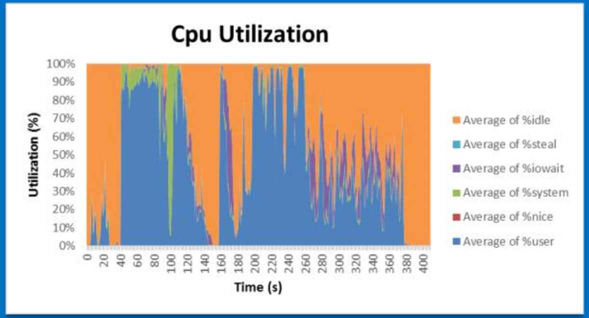
intel Software

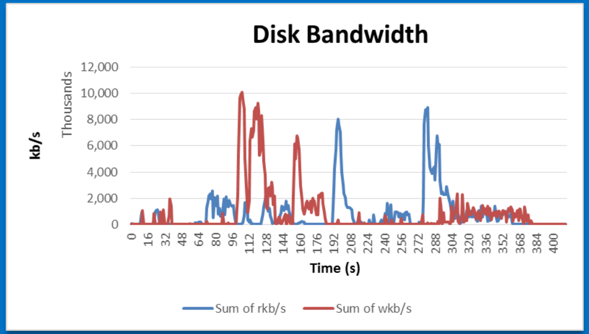
NUMA Tuning Workflow

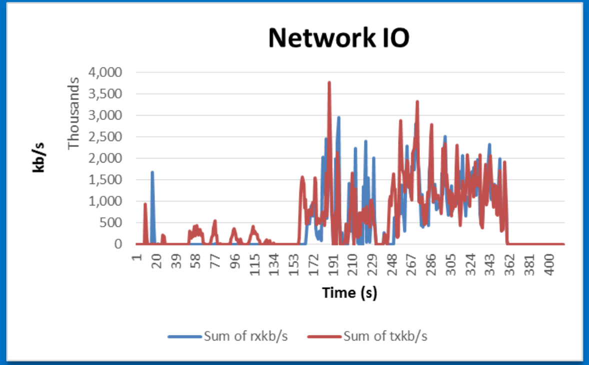
- Prepare Workloads
- Benchmark baseline results with no NUMA support
  - Use Apache Spark* Web UI to check how Spark application executes and Use PAT check CPU, Memory, Disk IO and Network IO usages and tune Spark parameters for optimization.
  - Run each workload several times until the result doesn't deviate much from previous ones. The first several runs will warm up cache.
  - Calculate average of last 3 results.
- Checking real-time NUMA metrics with numatop*
- Use Intel® VTune™ for Linux to do memory access analysis. Check remote memory access (RMA) and local memory access (LMA) ratios.
- If RMA/LMA ratio is high, apply NUMA-support patch and benchmark to see if there is any performance gain.

Intel® PAT – Intel® Performance Analysis Tool









- Image from : https://github.com/intel-hadoop/PAT

Checking real-time NUMA metrics with numatop

- numatop: alternative tool to check real-time NUMA stats
- NUMA access stats for TPCx-BB* q01 sample

NumaTOP v1.0, (C) 2013 Intel

Monitoring 767 processes and 4685 threads (interval: 5.3s)

PID	PROC	RMA(K)	LMA(K)	RMA/LMA	CPI	*CPU%
77070	java	708.8	422.3	1.7	6.23	19.4
77049	java	396.6	411.8	1.0	4.84	17.9
77053	java	280.4	335.1	0.8	2.70	17.3
77050	java	298.6	534.4	0.6	5.57	16.4
77081	java	248.5	498.4	0.5	2.76	16.1
77051	java	250.2	324.5	0.8	5.94	11.8
77045	java	441.6	82.9	5.3	6.17	11.7
77726	java	82.1	34.3	2.4	5.36	2.3
77623	java	15.1	66.0	0.2	4.90	2.2
77686	java	1.0	39.0	0.0	5.44	1.1
77069	java	25.7	7.8	3.3	5.81	0.8
77096	java	15.7	3.3	4.7	5.76	0.3
77047	java	94.9	66.2	1.4	1.28	0.2
77406	java	48.6	88.0	0.6	1.19	0.1
77052	java	17.1	38.9	0.4	1.18	0.1
77046	java	30.3	44.1	0.7	1.49	0.1
40609	java	67.3	154.2	0.4	1.69	0.0
76522	numatop	53.9	131.7	0.4	1.40	0.0
131618	kworker/u38	17.3	12.5	1.4	1.48	0.0
40862	java	0.4	0.3	1.3	3.32	0.0
436	kworker/85:	0.2	0.3	0.5	4.12	0.0

NUMA-Unaware

NumaTOP v1.0, (C) 2013 Intel

Monitoring 780 processes and 4700 threads (interval: 5.1s)

PID	PROC	RMA(K)	LMA(K)	RMA/LMA	CPI	*CPU%
102248	java	1863.1	64107.2	0.0	2.17	23.4
102259	java	531.1	15450.1	0.0	1.40	4.4
102287	java	730.2	13051.9	0.1	1.35	3.5
102735	java	512.4	13898.0	0.0	1.39	3.1
102250	java	3016.2	16790.2	0.2	1.57	3.0
102269	java	1204.3	13621.6	0.1	1.18	2.7
102279	java	1675.9	17376.5	0.1	1.22	2.6
102281	java	3292.1	16176.6	0.2	1.19	2.4
102243	java	984.4	15440.8	0.1	1.21	2.4
102245	java	977.0	11265.1	0.1	1.15	2.3
102280	java	788.9	15347.1	0.1	1.36	2.3
102239	java	226.0	15974.2	0.0	1.22	2.2
102295	java	199.0	15943.5	0.0	1.27	2.1
102753	java	1854.8	23355.3	0.1	1.43	2.0
102247	java	289.6	14809.5	0.0	1.20	1.9
102242	java	198.9	14412.5	0.0	1.22	1.8
76522	numatop	117.2	249.9	0.5	1.09	0.1
40609	java	35.8	211.6	0.2	1.56	0.0
1600	rngd	8.4	157.7	0.1	1.86	0.0
10	rcu_sched	1.1	19.4	0.1	1.56	0.0
101919	java	7.2	118.9	0.1	1.50	0.0

Applied NUMA-aware patch

Software

VTune™ Memory Access Analysis Example

- Identify memory access issues with Intel® Vtune™
- Collect data with Linux command line and analyze with Windows GUI
- Capture 5 secs data on worker node:

```
$ amplxe-cl -collect memory-access --duration 5
```

NUMA-unaware VTune™ data		VTune™ data after applying NUMA-aware patch	
Elapsed Time:	5.006s	Elapsed Time:	5.007s
CPU Time:	349.969s	CPU Time:	374.452s
Memory Bound:	28.5%	Memory Bound:	25.3%
L1 Bound:	14.6%	L1 Bound:	15.6%
L2 Bound:	1.3%	L2 Bound:	1.5%
L3 Bound:	6.0%	L3 Bound:	5.3%
DRAM Bound:	7.9%	DRAM Bound:	4.1%
Memory Bandwidth:	11.4%	Memory Bandwidth:	12.7%
Memory Latency:	59.3%	Memory Latency:	61.2%
Remote / Local DRAM Ratio:	0.675	Remote / Local DRAM Ratio:	0.000
Loads:	275,790,673,472	Loads:	347,496,824,592
Stores:	91,926,178,872	Stores:	109,500,842,488
LLC Miss Count:	208,806,264	LLC Miss Count:	144,804,344
Average Latency (cycles):	9	Average Latency (cycles):	9
Total Thread Count:	3,497	Total Thread Count:	3,660
Paused Time:	0s	Paused Time:	0s

Memory access VTune™ analysis for TPCx-BB q01 sample

Cluster Configurations for TPC-DS* and TPCx-BB* experiments

Hardware	
CPU	Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz 22 cores 44 threads * 2 sockets 2 NUMA Nodes
Memory	Kinston* DDR4-2133 8G * 24 slots = 192G 96G for each NUMA node
Storage	For OS: - SSD 1 X Intel® SSD DC S3510 Series（480GB,2.5in SATA 6Gb/s) for Data: - HDD：7 X Seagate* Constellation.2 ST9500620NS 500GB 7200 RPM 64MB Cache SATA 6.0Gb/s
Network	Intel® Ethernet Controller 10 Gigabit X540-AT2
Software	
OS	CentOS* Linux release 7.1.1503 (Core) Kernel version: 3.10.0-514.2.2.el7.x86_64
Java	OpenJDK* Runtime Environment (build 1.8.0_111-b15)
Hadoop	Hadoop* 2.7.3
Spark	Spark* 2.1

- 1 master node + 2 worker nodes
- Spark SQL parameters:

--master spark://sr592:7077

--driver-memory 16g

--num-executors 32

--executor-memory 11520m

--executor-cores 5

--conf "spark.sql.shuffle.partitions=480"

Software

Workloads for Experiments

- Query samples from TPC-DS
  - TPC-DS models the decision support functions of a retail product supplier. User queries convert operational facts into business intelligence
  - 1000 GB text database compressed to 277.4 GB parquet format and stored on HDFS
- Query samples from TPCx-BB
  - TPCx-BB has analytical queries in the context of retailers with physical and online store presence. The queries are expressed in SQL for structured data and in machine learning algorithms for semi-structured and unstructured data.
  - 1000 GB text database compressed to 198.1 GB ORC format and stored on HDFS
- A typical Telco SQL workload
  - SQL queries to summarize customers consumption characteristics utilizing billing data.
  - 10GB text database, compressed to 3GB parquet format and stored on HDFS.

TPC-DS Results

- Query Samples:
  - Query 55: interactive query
  - Query 58: deep reporting
  - Query 73: data mining
- Run TPC-DS Queries:

```
$ spark-sql --database $database_name -f $query_name.sql
```

- Results:

Sample	NUMA-Unaware(s)	NUMA-Aware(s)	Scaling
q55	24	23.6	1.017x
q58	52	52	1.000x
q73	28	27	1.037x

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

TPCx-BB Results

- Query Samples:
  - Query 01: UDF/UDTF on structured data
  - Query 05: machine learning on semi-structured data
  - Query 27: UDF/UDTF/NLP on un-structured data
- Run TPCx-BB Queries

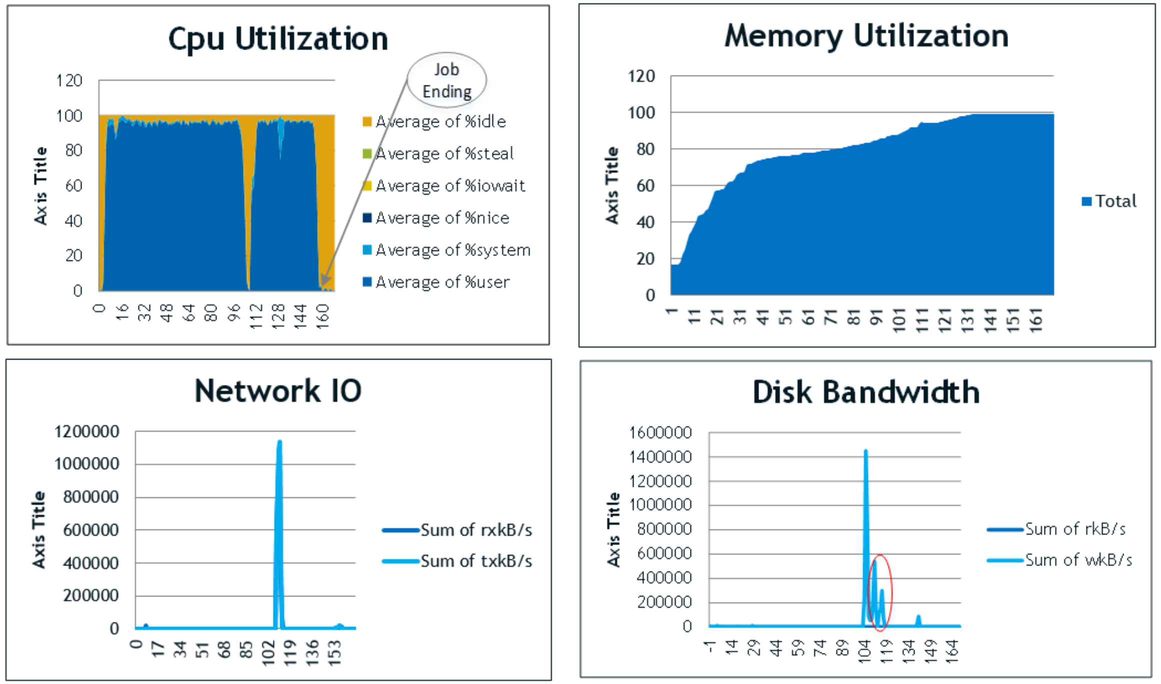
```
$ ./bin/bigBench runQuery -q $query_number -U
```

- Results:

Sample	NUMA-Unaware(s)	NUMA-Aware(s)	Scaling
q01	80	74	1.081x
q05	249	228.7	1.089x
q27	114	105	1.086x

intel Software

A Typical Telco SQL Workload Analysis and Results



Be noted: All slave nodes have same behaviors, we present one of them

Profiling with PAT

Events Count	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY	MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_DRAM	UOPS_RETIRED.RETIRE_SLOTS
w/ NUMA-Aware Patch	25,325,367,481,579	21,680,659,933,335	2,648,340,894	58,198,229	22,606,108,509,456
w/o NUMA-aware Patch	26,956,329,603,414	21,782,319,467,891	1,855,031,856	1,220,313,270	22,769,700,890,331

- Cycles Per Instruction

CPI is a little high.
NUMA-Aware: 1.2, NUMA-Unaware: 1.2

- Pipeline Slot Utilization

Utilization is low, vectorization computing may help.
NUMA-Aware: 22% , NUMA-Unaware: 21%

- L3 Miss

The L3 miss impact is minor.
NUMA-Aware: 2% Overall Cycles, NUMA-Unaware: 2.6% Overall Cycles

- Remote Node Access

Remote Memory Accessing is not the bottleneck.
NUMA-Aware: 0.07% Overall Cycles, NUMA-Unaware: 1.2% Overall Cycles

Be noted: All slave nodes have same behaviors, we present one of them

Profiling with VTune™

Result:

Sample	NUMA-Unaware(s)	NUMA-Aware(s)	Scaling
Telco SQL	163	156	1.045x

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Conclusion / Key Takeaways

- NUMA penalties varies as workloads change
  - Hard to identify memory access patterns
  - Not all big data workloads have NUMA issues, need to analyze case by case
- Leverage platform tools such as Intel® VTune™ / numatop to examine memory access
- Manual CPU binding helps when NUMA issues are big

Software

References

- NUMA on Linux: http://man7.org/linux/man-pages/man7/numa.7.html
- TPC-DS: http://www.tpc.org/tpcds/default.asp
- TPCx-BB: http://www.tpc.org/tpcx-bb/default.asp
- PAT: https://github.com/intel-hadoop/PAT
- Intel VTune for Linux: https://software.intel.com/en-us/intel-vtune-amplifier-xe/
- numatop: https://01.org/numatop