Spark vs. Virtualized Spark: A Performance Analysis

Wenjing Jin[1] and Jae W. Lee[2]
[1] Department of Electrical and Computer Engineering, Sungkyunkwan University
2066, Seobu-ro, Jangan-gu, Suwon-si, Gyeonggi-do 16419, Korea
[2] Department of Semiconductor Systems Engineering, Sungkyunkwan University
2066, Seobu-ro, Jangan-gu, Suwon-si, Gyeonggi-do 16419, Korea
E-mail: {wenjing90, jaewlee}@skku.edu

Abstract: Apache Spark is an open-source framework for scalable big data processing. OpenStack is a popular virtualization framework that provides Infrastructure as a Service (IaaS) on cloud. Deploying Spark on OpenStack provides many benefits such as on-demand resource scaling, greater availability and flexibility. However, this virtualized Spark is likely to have very different performance characteristics from the native Spark. This paper aims to quantize the cost of virtualization on a Spark cluster. Our experiments dem-onstrate that (i) the virtualized Spark with four nodes is about 1.58X slower than the native Spark, (ii) all of network, CPU and GC cause this slowdown. Overall, the network waiting time and CPU time contribute the most to the increased execution time, and the GC time has the highest increasing rate.

Keywords-- **Apache Spark, OpenStack, Virtualization**

1. Introduction

Apache Spark [1] is one of the most popular large-scale data processing engines today. It is an open-source project firstly developed by UC Berkeley AMP Lab. Spark can achieve much higher performance than Hadoop MapReduce [2] (10X faster on disk, 100X faster in memories) by placing data inside memory. Spark uses Resilient Distributed Datasets (RDDs) [3], which allows programmer cache the intermediate data in memory rather than writing them back to disk.

OpenStack [4] is an open-source platform for Infrastructure as a Service (IaaS) on cloud. OpenStack can help enterprises build their own private and public clouds, in a similar way to Amazon EC2 and S3.

As Spark is becoming more widely adopted, there are growing interests in deploying it on OpenStack. Many enterprises try deploying Spark on OpenStack provides many benefits such as on-demand resource scaling, greater availability and flexibility. However, this virtualized Spark is likely to have very different performance characteristics from the native Spark.

This paper aims to quantize the cost of virtualization on a Spark cluster. By running 6 workloads from Intel HiBench [5] on both virtualized and native Spark clusters of the same configuration, we first measure how much their performance differs. Besides, using a trace analysis tool, we identify the reasons for this difference. Our analysis shows that all of network, CPU and GC are attributed to the performance difference between the two clusters.

2. Experimental Setup

Cluster Setup: We employ a 4-node homogeneous cluster

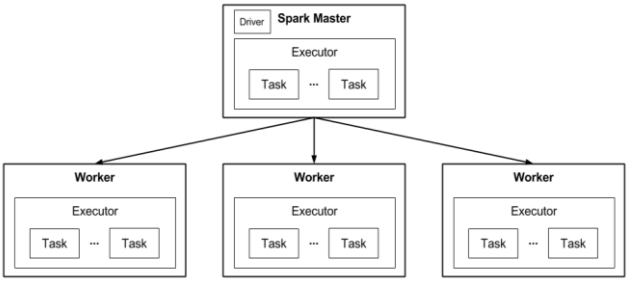

Figure 1. Cluster Configuration

Table 1. Workloads Characterization

Type	Workloads	Input Data Size
Job based	WordCount	31.96 GB
	TeraSort	32 GB
SQL	Scan	1.82 GB
	Join	1.97 GB
Web Search	PageRank	2.99 GB
Machine Learning	Bayes	1.82 GB

for our experiments. Each node has an Intel® i7-4790 CPU, which has 4 cores with 8 threads running at 4 GHz, 2 TB HDD, and 32 GB of physical memory. The OS is 64-bit Ubuntu 14.04 LTS. Nodes are connected by a 10 Gbps Ethernet switch.

We deploy the Liberty version of OpenStack [4]. In addition to the 4 compute nodes, a separate controller node is set up. We use the provider networks option for the OpenStack networking service, which means all instances attached directly to the public network.

We use Spark version 1.5.1 [1] running in standalone mode on HDFS 1.2.1 [2]. As shown in Figure 1, we deploy Spark cluster on 4 nodes. For the virtual Spark cluster, we deploy a Spark cluster on the four OpenStack compute nodes with one OpenStack VM instance per node. Each VM has 7 vCPU cores, 500 GB disk, and 31 GB memory. Each Spark executor has 30 GB memory and 6 threads, which means it can run 6 tasks in parallel.

Workloads: We run 6 workloads from Intel HiBench [5] on both clusters as summarized in Table 1: WordCount, TeraSort, Scan, Join, PageRank, and Bayesian Classification (Bayes). HiBench is a popular big data benchmark suite. We set the data scaled to *huge* size.

WordCount and TeraSort are job based micro benchmarks. TeraSort has a higher shuffle overhead to produce more network and disk I/O requests than WordCount, which just summarizes a large data set. Scan and Join are SQL benchmarks, where Join contains Hive queries. PageRank is an iterative web search benchmark. Bayes is a ma-

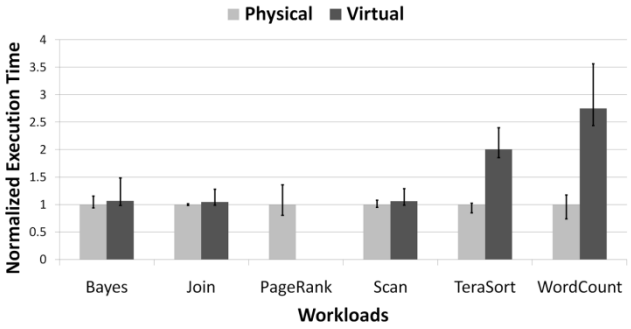

Figure 2. Execution Time

chine learning benchmark. Both PageRank and Bayes build on Spark-MLlib.

3. Results

Figure 2 compares the performance of physical and virtual Spark clusters. We take an average of 10 measurements for each workload. The Y-axis represents normalized execution time. We use a bar chart with an error bar to depict the maximum and minimum values.

As shown in Figure 2, Spark in the virtual environment is about 1.58X slower than on the physical cluster. Virtual Spark cluster exhibits performance degradation for Bayes, Join, Scan, TeraSort and WordCount, with increases in execution time of 1.07X, 1.05X, 1.06X, 2.01X, and 2.75X, respectively. Moreover, PageRank fails to run with Spark due to a timeout error on the virtual Spark cluster. This problem is caused by running out of memory space [6].

For performance profiling, we use a trace analysis tool [7], which can help us to understand the breakdown of execution time, and we use it to analyze how a virtual environment affects the performance of Spark. We first set the parameter `spark.eventLog.enabled` to `true` [1], which can enable the Spark master to write an event log with information about each completed task to a file.

Our analysis shows that network, CPU and memory are all bottlenecks in the virtual environment. Furthermore, the increasing rate of GC time is significant. We take TeraSort as an example in Figure 3 (a) and (b). The X-axis represents execution time and the Y-axis represents different tasks in a Spark job. One thing to note here is that the scales of X-axis in Figure 3 (a) and (b) are different. We choose task 480 to 492, which dominates the execution time in the job. We have found that the network waiting time and computation time in the virtualized cluster is about 2 times as much as that of the native cluster, and GC takes almost 3-4 times longer. Therefore, network waiting time and computation time have the most significant impact on the performance and the GC time has the highest increasing rate.

4. Submission Process

First, we plan to find a way to summarize the overall execution time for each execution segment. The performance profiling tool we use can only show how much time a task spends. We need not only the visualization but also more precise numbers to quantify the performance.

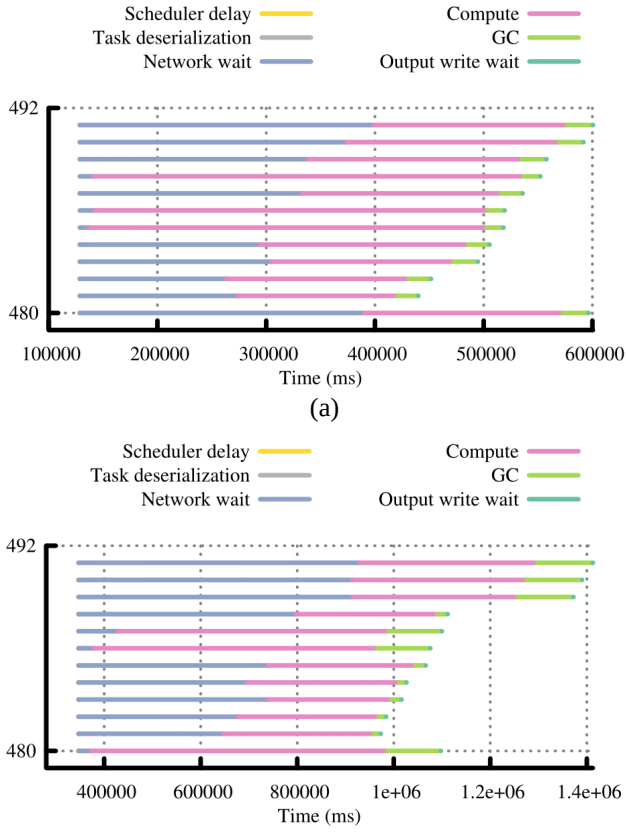

Figure 3. Performance Profiling

Second, we intend to scale out our cluster from 4 nodes to 8 nodes and test on larger scale data sets. Moreover, we plan to change from standalone mode to Yarn mode [2] for easier resource management and performance tuning.

Finally, we plan to use profiling tools for OpenStack. In this paper, we only use the profiling tool for Spark on both virtual and physical clusters to find several bottlenecks. As a next step we need more detailed analysis to figure out the sources of inefficiency and find ways to eliminate them to reduce the performance gap.

References

[1] Apache Spark. https://spark.apache.org/.
[2] Apache Hadoop. https://hadoop.apache.org/.
[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", in *Proceedings of 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.
[4] OpenStack. https://www.openstack.org/.
[5] HiBench. https://github.com/intel-hadoop/HiBench.git
[6] I. S. Choi, W. Yang, Y. Kee, "Early Experience with Optimizing I/O Performance Using High-Performance SSDs for In-Memory Cluster Computing", in *Proceedings of IEEE International Conference on Big Data (Big Data)*, 2015.
[7] K. Ousterhout. Trace analysis tool. https://github.com/kayousterhout/trace-analysis.git