Machine-Learning Based Spark and Hadoop Workload Classification Using Container Performance Patterns

Mikhail Genkin(✉), Frank Dehne, Pablo Navarro, and Siyu Zhou

School of Computer Science, Carleton University, Ottawa, Canada
michael.genkin@carleton.ca
https://carleton.ca/scs/

Abstract. Big data Hadoop and Spark applications are deployed on infrastructure managed by resource managers such as Apache YARN, Mesos, and Kubernetes, and run in constructs called containers. These applications often require extensive manual tuning to achieve acceptable levels of performance. While there have been several promising attempts to develop automatic tuning systems, none are currently robust enough to handle realistic workload conditions. Big data workload analysis research performed to date has focused mostly on system-level parameters, such as CPU and memory utilization, rather than higher-level container metrics. In this paper we present the first detailed experimental analysis of container performance metrics in Hadoop and Spark workloads. We demonstrate that big data workloads show unique patterns of container creation, completion, response-time and relative standard deviation of response-time. Based on these observations, we built a machine-learning-based workload classifier with a workload classification accuracy of 83% and a workload change detection accuracy of 74%. Our observed experimental results are an important step towards developing automatically tuned, fully autonomous cloud infrastructure for big data analytics.

Keywords: Big data cloud performance · On-line automatic tuning · YARN · Hadoop · Spark

1 Introduction

1.1 Background

Key big data technologies such as Hadoop map-reduce jobs, Spark applications, Hive, Hbase and others run on hardware clusters that are managed by open-source and commercial resource managers, such as YARN, Mesos, and Kubernetes. Resource managers arbitrate resources available to different applications, to form a key architectural layer in the cloud computing paradigm. Resource managers use constructs called *containers* to manage analytic and other applications running on the cluster. Containers manage, and gate, CPU memory and

C. Zheng and J. Zhan (Eds.): Bench 2018, LNCS 11459, pp. 118–130, 2019.
https://doi.org/10.1007/978-3-030-32813-9_11

disk resources assigned by the resource manager to the application. Containers make sure that a given application does not exceed it's allotted share of resources. Applications, and analytic frameworks, interact with the resource manager to schedule tasks to run in these containers. The way a container is implemented by the resource manager varies among resource managers. In some cases, such as in YARN, containers are explicitly defined data structures used by the resource manager for resource arbitration and internal book-keeping. In other cases, such as with Mesos, the resource manager uses an operating-system-level technology, such as Docker, to attach container semantics to it's internal resource arbitration mechanism. Regardless of how the container concept is implemented, the end result is essentially the same - application tasks run in containers, and thus examining container performance can provide insights into application performance.

1.2 Problem

Apache Hadoop and Spark each have dozens of configurable parameters that can significantly affect performance of analytic jobs. A poorly tuned configuration can result in order-of-magnitude slower performance than for an optimally-tuned one. Manual tuning involves experimenting with different combinations of tunable parameters. Considering that each experiment, at multi-Terabyte data scale, can take hours to days to run, this can turn into a very long and expensive procedure.

1.3 Limitations of Previous Approaches

To solve this problem, there have been a number of attempts to automatically tune big data applications [1,2,4–13]. These focused primarily on automatically tuning the Hadoop MapReduce framework. More recently there have been attempts to automatically tune Spark as well. In our previous work we developed KERMIT - the first on-line automatic tuning engine for YARN, capable of automatically tuning CPU and memory for both Apache Hadoop and Apache Spark [3]. KERMIT was able to demonstrate better tuning efficiency for standard Hadoop and Spark benchmarks.

Most studies demonstrated improvements only on small data sets that are not representative of data volumes in big data applications. Furthermore, performance improvements were only documented on very simple, single-user workloads based on a few sample applications. For real-life big data applications, systems such as Hadoop or Spark need to be dynamically tuned to handle large scale, big data workloads that arise from a multitude of applications and user requirements, and change over time. Such real-life scenarios are not addressed by any of the published automatic tuning approaches.

For on-line automatic tuning applications - "to tune or not to tune?" - is the key question that the tuning engine needs to be able to answer accurately in order to achieve optimal performance. Too much tuning causes an overhead that can sometime cancel out any performance benefit from optimizing tunable

parameters. Not enough tuning results in jobs running slower due to sub-optimal tuning.

A number of researchers studied big data workload characteristics [4,6,9]. However, virtually all of these studies focused on lower-level operating system metrics such as CPU, memory and disk utilization, and even lower-level hardware counters such as L1, L2 and L3 cache hit rates. While these data provide important insights about the workload, it is difficult to relate them directly to Hadoop and Spark tunable parameters and develop a tuning strategy at the resource manager level. No published research to date has focused on container performance analysis. However, as discussed above, resource managers operate on and interact with *containers*. Optimizing performance of big data applications deployed on containerized cloud infrastructure requires optimizing the performance of the containers in which they run.

1.4 Our Contribution

We present the first experimental study of container performance patterns observed in Hadoop and Spark workloads. We focus on the following container performance metrics:

- container duration (response-time),
- container response-time relative standard deviation (RSD),
- container creation rate,
- container completion rate.

We demonstrate that for realistic big data workload sizes (e.g. 2 TB data sets) all important workload changes that are relevant for on-line automatic tuning are accompanied by:

- order of magnitude changes in container creation rate,
- statistically significant changes in RSD.

Our experiments demonstrate that the above metrics provide very clear statistical markers that can be used by automatic tuning systems to detect changes in workload characteristics and initialize local and global parameter searches. We also observed that many Hadoop MapReduce and Spark jobs have distinctive signatures that can be used by machine learning systems to identify jobs on the fly and apply effective tuning parameters.

Based on these observations, we built a machine-learning based workload classifier with a workload classification accuracy of 83% and a workload change detection accuracy of 74%. Our observed experimental results are an important step towards developing automatically tuned, fully autonomous cloud infrastructure for big data analytics.

1.5 Resource Managers and Containers

YARN, Mesos and Kubernetes are the most popular open-source resource managers used today. Resource managers arbitrate system resource such as CPU,

memory and disk among different applications that run on a cluster. Resource managers use containers to assign and track resource allocations to different applications. In the context of this study the term container refers to a construct the resource manager uses to track resources allocated to an application. The container may be an abstract construct, or it may be backed by a technology such as Docker that enforces resource utilization at the operating system level and ensures isolation of one application from another. Most resource managers available today implement this container concept even though it is not called the container in all cases. YARN, Mesos and Kubernetes provide Docker integration.

2 Evaluation Methodology

Our evaluation methodology focused on simulating common Hadoop and Spark workloads and workload transitions using well-understood big data benchmarks. Container performance metrics were compiled by analyzing log data.

Before capturing container performance statistics for each workload transition experiment, runs were performed to establish the optimal sampling window length. The sampling window duration was chosen so that the majority of windows had a statistically valid number of containers recorded. For example, if all container creation and completion events were recorded during a single, very long, window then this would not make for a compelling analysis.

2.1 Container Performance Metrics

As part of our experiments, the following container performance metrics were collected and analyzed:

1. **Container Creation Rate.** This is the number of containers created during a given observation window.
2. **Container Completion Rate.** This is the number of containers that finish execution during a given observation window.
3. **Container Average Response-Time.** This is the average response-time calculated for all containers that complete execution during a given observation window.
4. **Container Response-Time Relative Standard Deviation (RSD).** This metric measures the degree of scatter among container response time measurements in a given observation window. It is defined as the standard deviation of container response-times, divided by the average container response-time for container response-times in a given observation window. Small RSD indicates tightly clustered data while large RSD indicates widely scattered data. Increase in the RSD value across a workload transition can indicate the introduction of a bottleneck due to a change in processing.

Our analysis focuses on calculating both the absolute values for container metrics at steady state, and the relative amount of change that occurs as the workload passes through each transition. The relative amount of change equals

the average metric value observed in two observation windows after the transition, divided by the average metric value in the two windows immediately before the transition.

2.2 Workloads and Workload Transitions

Table 1 summarizes the different workloads and workload transitions analyzed in this study, with the benchmarks, data size, and procedure used in each case.

2.3 Parameter Settings

Unless stated otherwise, the Hadoop MapReduce and Spark configurations used default values. For YARN, the yarn.nodemanager.resource.cpu-vcores parameter in the yarn-site.xml file was set to the total number of CPUs shown by the operating system on each of the cluster nodes. The yarn.nodemanager.resource.memory-mb and yarn.scheduler.maximum-allocation-mb parameters were set to the total amount of memory on each data node. In mapred-site.xml, the parameter mapreduce.job.reduces was set to 36. The parameters mapreduce.output.fileoutputformat.compress and mapreduce.map.output.compress were set to true. The parameters mapreduce.output.fileoutputformat.compress.codec and mapreduce.map.output.compress.code were set to org.apache.hadoop.io.compress.Default in order to avoid running out of space in the HDFS during bigger runs. The parameter mapred.child.java.opts was modified to increase the maximum JVM heap size setting from the default to 850 MB. This was done to remove the possibility of a memory bottleneck impacting container performance. On the Spark side, the spark.executor.memory configuration parameter was set to 6G to ensure that most memory on our nodes was utilized.

2.4 Hardware and Software

All measurements were performed on a 8-node cluster comprising 1 management node and 7 compute/data nodes (all KVM virtual machines running on IBM S822L Power8 with Dual 10-core Power8 3.42 GHz; one bare metal server was used for every two VMs). Each node was equipped with a 100 GB SSD drive for operating system and Hadoop stack installation. All the nodes shared access to a 12 TB network shared drive connected through a 10Gb fiber switch. Each node was also equipped with 48 GB RAM and 10 virtual cores. All nodes were running the Ubuntu 16.04 ppc64le operating system. The test cluster topology is shown in Fig. 1. We used Hadoop 2.7.3 and Spark 2.1.1. In order to facilitate container metric collection, a jar file containing the YARN resource manager and our KERMIT library [3] was built and deployed to replace the standard YARN jar.

Table 1. Workloads, workload transitions, and benchmarks.

Transition	Description	Benchmarks, procedure and data size
Hd-sj-1	Transition from map to reduce processing in a single Hadoop map-reduce job	HiBench WordCount benchmark. 2 TB
Hd-sj-2	Transition from map to reduce processing in a single Hadoop map-reduce job	TeraSort benchmark. 2 TB
Hd-sj-3	Transition from reduce-shuffle to reduce processing in a single Hadoop map-reduce job	TeraSort. 2 TB
Hd-sufl-4	Transition from TeraGen to TeraSort processing in a Hadoop single-user job flow	TeraGen-TeraSort-TeraValidate sequence of jobs. 2 TB
Hd-sufl-5	Transition from TeraSort to TeraValidate processing in Hadoop single-user job flow	TeraGen-TeraSort-TeraValidate sequence of jobs. 2 TB
Hd-sj-6	Transition from one iteration to another within Hadoop K-Means machine learning job	HiBench K-Means. 2 TB
Hd-sufl-7	Transition from Hadoop WordCount reduce processing to TeraSort map processing in a single-user job flow	HiBench WordCount-TeraSort-K-Means job flow. 2 TB
Hd-sufl-8	Transition from TeraSort reduce processing to K-Means processing in a single-user job flow	HiBench WordCount-TeraSort-K-Means job flow. 2 TB
Hd-mufl-9	Multi-user transition from TeraSort shuffle to K-Means	2 users (1 running TeraSort, and 1 K-Means) 2 TB
Hd-mufl-10	Multi-user transition from K-Means iteration back to TeraSort reduce phase	2 users (1 running TeraSort, and 1 K-Means) 2 TB
Hd-mufl-11	Multi-user transition from TeraSort map phase to K-Means iteration	2 users (1 running TeraSort, and 1 K-Means) 2 TB
Hd-mufl-12	Multi-user transition from K-Means iteration to TeraSort map phase	2 users (1 running TeraSort, and 1 K-Means) 2 TB
Hd-mufl-13	Multi-user transition from TeraSort map phase to K-Means iteration	3 users (1 running TeraSort, and 2 K-Means) 2 TB
Hd-mufl-14	Multi-user transition from K-Means iteration to TeraSort reduce phase	3 users (1 running TeraSort, and 2 K-Means) 2 TB
Sp-sj-1	Transition from map() to reduceByKey() processing in a single Spark job	Spark ARL TeraSort. 2 TB
Sp-sufl-2	Transition from Spark K-Means processing to TPC-DS-inspired Q3	SMB-2 1 user, use case 2 (batch analytics) Spark job sequence. 2 GB per application
Sp-sufl-3	Transition from Spark TPC-DS-inspired Q3 to Q53	SMB-2 1 user, use case 2 (batch analytics) Spark job sequence. 2 GB per application
Sp-sufl-4	Transition from Spark TPC-DS-inspired Q53 to Q89	SMB-2 1 user, use case 2 (batch analytics) Spark job sequence. 2 GB per application
Sp-sufl-5	Transition from Spark TPC-DS-inspired Q89 to Q8	SMB-2 1 user, use case 2 (batch analytics) Spark job sequence. 2 GB per application
Sp-mufl-6	Transition from Spark single-user batch processing to multi-user (3 interactive users)	SMB-2 1 batch user + 3 interactive users, use case 3 (mixed analytics) Spark job sequence. 2 GB per application
Sp-mufl-7	Transition from Spark multi-user (3 interactive users) to single-user batch processing	SMB-2 1 batch user + 3 interactive users, use case 3 (mixed analytics) Spark job sequence. 2 GB per application
Sp-sufl-8	Initiation of Spark streaming	spark-perf benchmarking suite
Sp-sufl-9	Completion of Spark streaming	spark-perf benchmarking suite
Sp-sufl-10	Transition from Spark aggregateByKey to aggregateByKey(Int)	spark-perf benchmarking suite, data scale 3
Sp-sufl-11	Transition from Spark aggregateByKey to sortByKey()	spark-perf benchmarking suite, data scale 3
Sp-sufl-11	Transition from Spark count() to filter()	spark-perf benchmarking suite, data scale 3

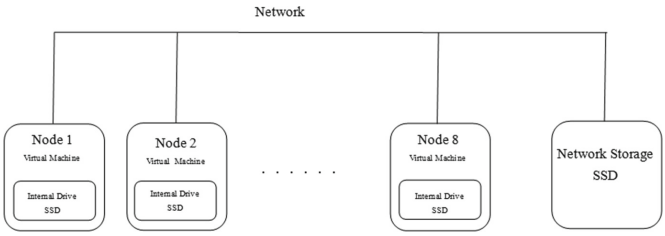

Fig. 1. Test-bed topology.

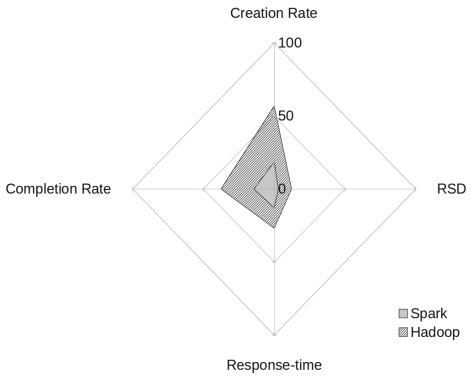

Fig. 2. Radar chart showing average Hadoop and Spark workload metric values.

3 Results

Below we present our workload analysis, workload classification and workload transition detection findings.

3.1 Steady State Workload Characteristics

Figure 2 shows a radar chart that compares container performance metric averages observed for Hadoop and Spark workloads. To construct this chart, a random sampling of observation windows for Hadoop and Spark observed during steady state conditions were selected for analysis. Container performance statistics, including maximum, minimum, average, and standard deviation were calculated for all metrics. Although averages are shown in Fig. 2, maximum values were also examined and found to show almost exactly the same trend as averages. For brevity, only averages are shown.

It was observed that for Hadoop workloads, average container metric values showed much greater range than for Spark workloads. Average container creation rate, container completion rate, container response-time and RSD were all observed to be about 3x greater for Hadoop than for Spark. There is an area on the radar chart where Hadoop and Spark workloads do overlap, but there is a much larger area where they do not overlap.

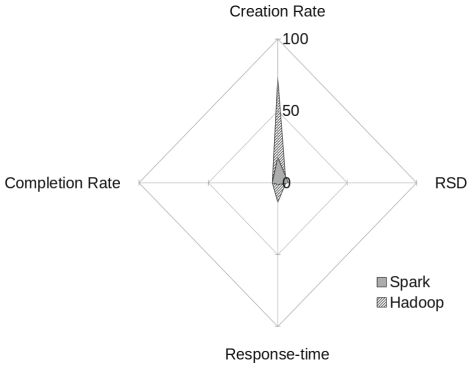

Fig. 3. Radar chart showing average Hadoop and Spark workload transition values.

3.2 Dynamic Workload Characteristics - Workload Transitions

Figure 3 shows a radar chart that compares container performance metric changes observed for Hadoop and Spark workloads. The change of a metric is defined as the average metric value observed after the workload transition divided by the average metric value observed before the workload transition. Figure 3 shows average changes observed for all transitions measured during this study. As for steady-state performance statistics, full statistics including maximum, minimum, average, and standard deviation were calculated and examined for all cases. Since maximum values were found to show almost exactly the same trend as averages, only averages are shown.

Hadoop workloads were observed to produce container creation rate changes and container response-time changes that were on average 3x greater than corresponding changes produced by Spark workloads. Changes in RSD and container completion rate were observed to show a similar trend. As with steady-state metrics, an area of overlap between Hadoop and Spark workloads can be observed in Fig. 3. However, we observe a larger area where workload transition metrics do not overlap.

4 Identifying and Classifying Workloads

A prototype classifier using several popular machine-learning algorithms was constructed. The prototype was developed in Scala, using Apache Spark Mlib to implement k-means, logistic regression, decision tree, gradient-boosted trees, and random forest algorithms. A machine learning data-set (in libsvm format) was compiled from workload transition data that were labeled as either Spark or Hadoop. The data set was randomly split into training and testing data sets using a 70-30 rule, and the accuracy of prediction for each algorithm was evaluated. The process of splitting, training and testing was repeated 100 times for each algorithm to study the variance produced by the random splits.

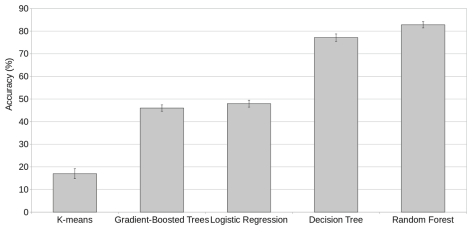

Fig. 4. Workload classification accuracy for common machine-learning algorithms.

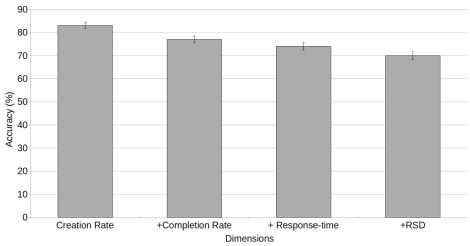

Fig. 5. Impact of using different container performance statistics on the classification accuracy of the Random Forrest algorithm.

The average classification accuracy (and standard deviation) for each algorithm is shown in Fig. 4. We observe that the Random Forest algorithm achieves the best workload classification accuracy of 83%.

To investigate how different container performance measures affect accuracy of prediction, several additional experiments were performed. The following data sets were prepared: (1) Container creation rate data only. (2) Container creation rate data plus container completion rate data. (3) Container creation rate data plus container completion rate data plus container response-time data. (4) Container creation rate data, plus container completion rate data, plus container response-time data, plus RSD data. The same random split procedure as described above was performed on each data set. The accuracy of classification was evaluated for the Random Forest algorithm. Results are shown in Fig. 5.

The findings are surprising. We observe that using container creation rate data alone resulted in the best classification accuracy. Adding data from other dimensions reduced rather than enhanced the classification accuracy.

5 Detecting Workload Transitions

Data collected for a typical Hadoop single-user job flow are shown in Fig. 6. This flow executes the following benchmark sequence back-to-back: WordCount-TeraSort-K-Means. Workload transitions are marked with vertical dashed lines and indicated in Fig. 6. The horizontal axis records the observation window number. The job flow is divided into a series of observation windows. Window number

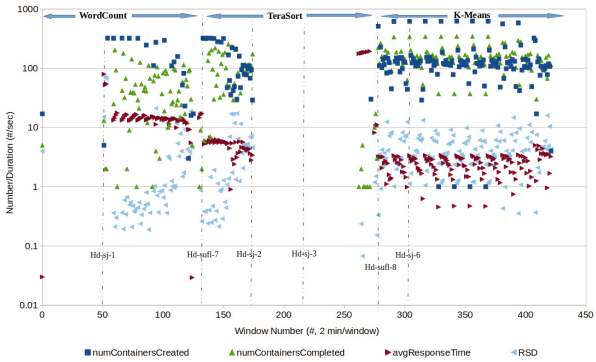

Fig. 6. Container performance metrics observed for Hadoop WordCount-TeraSort-K-Means job flow, using a 2 min observation window and 2 TB data size.

0 represents the very beginning for the entire job sequence. The duration of each window is fixed (set at the beginning of the job flow). The y-axis records the value of each container metric for a given observation window.

Figure 7 shows a multi-user Spark job flow. In this case a single-user thread was started. This thread executed the sequence of batch-type Spark jobs including a K-Means machine learning job and longer-running TPC-DS-inspired queries Q3, Q8, Q53 and Q89. After a delay of 600 s, 3 more user threads were started. Each of those user threads executed a sequence of 8 shorter TPC-DS-inspired queries running under a single Spark context. These queries were meant to simulate interactive drill-down operations initiated by a human analyst.

Container metric values measured during the course of a single observation window are shown as different symbols described in the figure legend. As we move right along the x-axis we can see drops and jumps in the patterns of symbols as we cross the workload transitions, represented by vertical dashed lines.

Observation window data collected for all data points were replayed as a real-time stream. A rolling average and standard deviation for each container metric were computed for 5 consecutive windows in the stream. During each computation, Welch's test was performed to evaluate whether a statistically meaningful difference existed between the means observed at current and previous steps. Welch's test was performed double-sided, using 95% confidence.

In those cases where a statistically meaningful difference was observed, our prototype code recorded the current observation window and noted a transition there. Transitions identified by the prototype were compared with transitions identified manually by examining YARN, MapReduce and Spark executor logs. Transition detection accuracy for each metric was calculated by dividing total transitions identified by the prototype by total transitions identified manually from logs and multiplying by 100. Results are shown in Fig. 8.

Surprisingly, workload change detection was observed to be the least accurate when using the container creation rate metric (18%), and the most accurate (74%) when using the RSD metric. Changes in nature of processing being per-

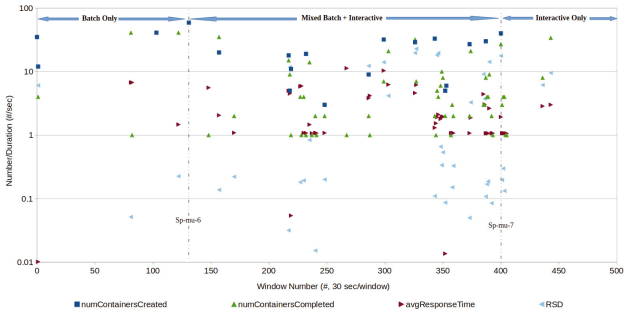

Fig. 7. Container performance metrics observed for multi-user Spark workload with batch and interactive query components, using a 30 sec observation window, 2 GB data.

formed by containers result in increased variance of the data. This is reflected in different RSD values before and after the transition even in those cases where changes in average container creation rate, container completion rate, and response-time are not statistically significant.

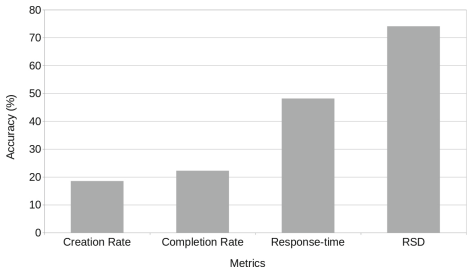

Fig. 8. Workload transition detection accuracy for different container performance metrics.

6 Relative Value and Importance of Container Performance Metrics

Based on our findings presented above, it is possible to propose a ranking of container performance metrics:

1. **Container Creation Rate.** This metric was observed to deliver the most accurate workload classification. It is possible to achieve very good classification results using this metric alone.
2. **Container Response-Time Relative Standard Deviation (RSD).** Although less effective than the first two metrics for both workload classification, RSD was observed to be very effective for detecting important workload transitions.

3. **Container Average Response-Time.** The average container response-time was observed to allow reasonably accurate identification of workload transitions.
4. **Container Completion Rate.** Container completion rate was observed to be less useful than the first three metrics for both workload classification and workload transition detection.

7 Conclusion

In this paper we presented a new way of capturing and analyzing workload characteristics of Spark and Hadoop workloads. We demonstrated that is possible to identify and classify big data analytic workloads with high degree of accuracy using their container performance characteristics. We also demonstrated that it is possible to use container performance metrics to accurately identify important workload transitions.

The most useful metrics were found to be the container creation rate and RSD. Using these metrics, it was possible to accurately distinguish Hadoop and Spark workloads, and identify important workload transitions. Based on these observations, we built a machine-learning based workload classifier and transition monitor with a workload classification accuracy of 83% and a workload change detection accuracy of 74%.

Our observed experimental results are an important step towards developing automatically tuned, fully autonomous cloud infrastructure for big data analytics. The next generation of on-line automatic tuning systems can leverage our findings to develop tuning approaches that are more fine-grained than tuning end-to-end performance of a job. Resource managers and analytic frameworks can begin to move away from exposing large numbers of tuneable parameters, and instead focus on implementing intelligent automatic tuners.

References

1. Awan, A.J., Brorsson, M., Vlassov, V., Ayguade, E.: Micro-architectural characterization of apache spark on batch and stream processing workloads. In: 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), pp. 59–66. IEEE (2016)
2. Ding, X., Liu, Y., Qian, D.: JellyFish: Online performance tuning with adaptive configuration and elastic container in Hadoop yarn. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 831–836. IEEE (2015)
3. Genkin, M., Dehne, F., Pospelova, M., Chen, Y., Navarro, P.: Automatic, on-line tuning of yarn container memory and cpu parameters. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems, pp. 317–324. IEEE (2016)

4. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: characterization of the mapreduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW), pp. 41–51. IEEE (2010)
5. Jia, Z., et al.: Auto-tuning spark big data workloads on POWER8: prediction-based dynamic SMT threading. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, pp. 387–400. ACM (2016)
6. Jia, Z., et al.: Characterizing and subsetting big data workloads. In: 2014 IEEE International Symposium on Workload Characterization (IISWC), pp. 191–201. IEEE (2014)
7. Mishra, A.K., Hellerstein, J.L., Cirne, W., Das, C.R.: Towards characterizing cloud backend workloads: insights from google compute clusters. ACM SIGMETRICS Perform. Eval. Rev. **37**(4), 34–41 (2010)
8. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: An approach for characterizing workloads in google cloud to derive realistic resource utilization models. In: 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), pp. 49–60. IEEE (2013)
9. Mulia, W.D., Sehgal, N., Sohoni, S., Acken, J.M., Stanberry, C.L., Fritz, D.J.: Cloud workload characterization. IETE Tech. Rev. **30**(5), 382–397 (2013)
10. Wang, G., Xu, J., He, B.: A novel method for tuning configuration parameters of spark based on machine learning. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 586–593. IEEE (2016)
11. Wang, K., Tan, B., Shi, J., Yang, B.: Automatic task slots assignment in Hadoop MapReduce. In: Proceedings of the 1st Workshop on Architectures and Systems for Big Data, pp. 24–29. ACM (2011)
12. Wasi-Ur-Rahman, M., Islam, N.S., Lu, X., Shankar, D., Panda, D.K.: MR-advisor: a comprehensive tuning tool for advising HPC users to accelerate mapreduce applications on supercomputers. In: 2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 198–205. IEEE (2016)
13. Zhang, R., Li, M., Hildebrand, D.: Finding the big data sweet spot: towards automatically recommending configurations for Hadoop clusters on docker containers. In: 2015 IEEE International Conference on Cloud Engineering (IC2E), pp. 365–368. IEEE (2015)