OPEN ACCESS

University of BRISTOL

Fekry, A., Carata, L., Pasquier, T., Rice, A., & Hopper, A. (2019). Towards Seamless Configuration Tuning of Big Data Analytics. In *2019 IEEE International Conference on Distributed Computing Systems (ICDCS 2019)* Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/ICDCS.2019.00189

Peer reviewed version

Link to published version (if available):
10.1109/ICDCS.2019.00189

Link to publication record in Explore Bristol Research
PDF-document

This is the author accepted manuscript (AAM). The final published version (version of record) is available online via IEEE at https://ieeexplore.ieee.org/document/8885361 . Please refer to any applicable terms of use of the publisher.

University of Bristol - Explore Bristol Research

General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available:
http://www.bristol.ac.uk/pure/about/ebr-terms

Towards Seamless Configuration Tuning of Big Data Analytics

Ayat Fekry[1]*, Lucian Carata[1]*, Thomas Pasquier[2], Andrew Rice[1] and Andy Hopper[1]
[1] *Computer Laboratory, University of Cambridge*
[2] *Department of Computer Science, University of Bristol*

***Abstract*—The execution of distributed data processing workloads (such as those running on top of Hadoop or Spark) in cloud environments presents a unique opportunity to explore multiple trade-offs between elasticity (and types of resources being allocated), overall runtime and total costs. However, beyond high-level constraints and objectives, it's not the end-users who should be mainly concerned with those optimizations, but the cloud providers. They have both the vantage point to collect actionable information, economies of scale and position to adjust parameters when dynamic conditions change, in order to fulfil SLOs that go beyond classic measures of latency and throughput.**

This is at odds with the existing approach of making software (including the interfaces to the cloud and the processing frameworks) as *configurable* as possible. We propose that rather than configurability, self-*tunability* (or the illusion of it as far as the end-user is concerned) is a better long-term goal.

I. Introduction

The task of deriving insights from increasing quantities of data (estimated at 1.7 MB/s/person in 2020 [2]) is poised to remain actual for some time to come, with systems evolving to keep up with the data volume either through technology advancements or optimisations of known pipelines, while also improving cost-efficiency.

The idea of processing data by distributing work over a cluster of machines is well established, with Data Intensive Scalable Computing (DISC) systems such as Hadoop [4], Spark [1] and Flink [3] being widely deployed even today. What is understood less is how such systems should be configured for running a given workload optimally, and deployed in the cloud using the appropriate resources (number of VMs, CPUs, memory, disk) to meet given targets of runtime or cost. Until now, those frameworks have been designed to operate mostly in environments where the resource allocation is static (as opposed to elastic) and the configuration of their many parameters is left as an exercise to the expert end-user.

Existing results show that misconfiguration is expensive: plausible but under-provisioned cluster setups can slow the analytics pipelines by up to 12X [10] while suboptimal framework configurations can lead to 89X performance degradation [31]. At the same time, a push for democratizing data implies allowing people with fewer resources and less expertise in debugging pipelines or low-level cloud bottlenecks to perform complex analytics.

As an end-user of commoditized computing infrastructures, the competitive advantage should lean towards having insights into *how* to process the data rather than into ways to *optimally run* systems that do the data processing. We therefore propose transparent self-tuning of data processing pipelines, offered as a cloud service with user-settable, high level objectives (SLOs) such as runtime or cost.

Existing efforts in this space have focused on the automation of configuration for individual components [19], [26], [25], [28], [10], [30]. In particular, they address cloud and DISC systems configuration separately. Of course, real-world scenarios imply that such optimisations need to be done jointly, considering elements such as: cloud resource allocation and scheduling (co-location with other workloads), workload characteristics (type, input data distribution, frequency), DISC configuration, etc. Optimal choices for some of those elements are not absolute but dependent on the others (a basic example would be the relationship between the number of virtual CPUs allocated and the number of Spark executor cores).

Thinking about the joint optimisation is not as simple as extending what has been already done: current configuration tuning strategies are based on the exploration of the configuration search space, which quickly grows as more dimensions are added. This means that tuning costs increase beyond what is feasible for single clients to run while maintaining efficiency: for example, a recent search-based Spark workload tuning required around 500 workload executions [35]; model-based approaches require thousands of executions to build *prediction models* that estimate the cost of running with a particular configuration [31].

Even worse, changes in workload and environment characteristics (input size, data, VM migration) mean that re-tuning might be needed, with less time to amortize the costs of finding previous tuned configurations.

The end-users should neither incur those high tuning costs, nor be concerned with detecting any change in workload characteristics for re-tuning in order to avoid missing opportunities related to efficiency and cost. Instead, configuration tuning should be *fully* automated by the cloud providers. They witness numerous workload execution metrics under different configurations and are aware of any underlying changes in workload co-location, network congestion, etc. Not only can they build tuning models of better accuracy using collected execution metrics, but also can instantly detect any change in workload characteristics and adapt resource provisioning and DISC system configuration accordingly.

Research in this direction is more tractable than understand-

* equal contribution

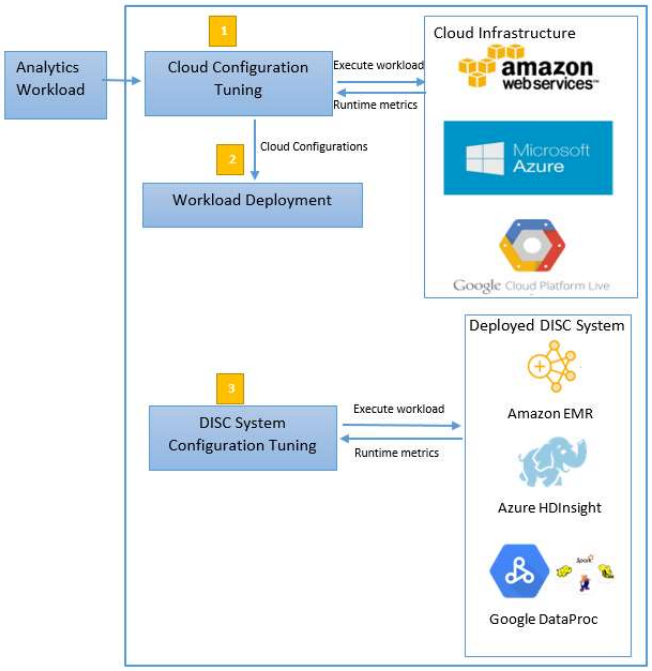

Fig. 1. Workload Configuration tuning

ing the general workload interference problem (which needs solving for achieving stable self-tuning systems), but will set the scene for results in the particular case of comparatively long-running, repeated computation. Making this class of computation predictable and offering explicit trade-offs between cost and runtime is set to decouple the need for large-scale computation from the requirement of in-depth expertise in low-level aspects of software optimization.

II. Background

Most frequently, end-users tune system configurations manually based on expertise, measurement and/or trial-and-error. However, the manual approach doesn't guarantee not missing further optimization opportunities and it is difficult to apply in a high-dimensional configuration space. Likewise, a number of general methods (USE, TSA)[1] have been suggested as recipes for figuring out the location of bottlenecks and explaining the time spent by applications. Those are ways of guiding tuning or debugging performance issues rather than automating the process for dynamic workloads. However, some automation strategies have been proposed, and we summarize how those could be applied to the case of real DISC systems in cloud environments. Our own work in this space has led to the development of a self-tuning Spark prototype, which we use as an experimental testbed for various tuning strategies and exploratory measurements. We base our discussion in this paper both on usage of this prototype as well as on related research in the configuration tuning space. Irrespective of the particular strategy, the tuning process takes place in two stages Fig. 1: In the first stage, the end-user uses a tuning system to select the characteristics of the virtual cluster used to run the DISC workload. By executing the workload one or more times, the tuning system tries to determine its sensitivity to changable parameters such as the number of CPUs and VM instances as well as to requirements of memory/disk and network bandwidth, etc. Based on this information and previously built models, the system suggests how an optimal infrastructure would look like, and the user instantiates it and deploys the workload. In the second stage, the tuning system identifies the optimal DISC system configuration for the given workload. In this section, we briefly discuss both stages starting from existing work or plausible extensions of it based on current directions and discuss essential limitations.

A. Cloud Configuration

During cloud configuration tuning, let's assume that the workload executes several times on different types of instances provided by different cloud infrastructures such as Amazon EC2, Microsoft Azure and Google Compute Engine. On each infrastructure, the workload is executed under different cloud configurations to find which one is best (e.g. instance family, instance type, number of instances).

The workload is then deployed to the cloud provider where it showed the best runtime under the optimized configuration, using an existing "native" DISC-deployment service such as Amazon EMR [5], Microsoft Azure HDInsigh [6], or Google DataProc [7]. Such approaches are inherently static (once the cluster setup and cloud provider are decided, it is assumed they remain constant) and miss the opportunity of using the cloud's elasticity features when the workload changes. Furthermore, their choices could be biased due to transient co-location of test workload runs with other resource-intensive workloads or (at the other end) with atypically low contention for resources.

Three systems have the potential of realizing the "static" vision today: **Cherrypick** [10] finds near-optimal cloud configurations by leveraging Bayesian optimization to build a performance model for recurring jobs using a small number of execution samples. **PARIS** [30] is a system for selecting the best VM for certain workloads based on user-defined metrics. It uses offline profiling for benchmarking various VM types, then combines this with an online fingerprint of each workload. The combined data is used to build a decision tree and a random forest-based performance model to select the best VM type. **Ernest** [28] tunes the cloud configurations for machine learning-based analytics workloads. It builds a performance model based on the particular structure of machine learning jobs, but has poor adaptivity to other types of workloads [10].

B. DISC systems Configuration

The tuning of DISC systems, which are notoriously difficult to configure optimally due to the large search space, has been approached by specializing for particular frameworks: Several solutions address tuning for Hadoop/MapReduce workloads:

[1] http://www.brendangregg.com/methodology.html

MROnline[25] proposed a modified Hill climbing technique to find good configurations; it limits the search space using predefined tuning rules. StarFish [19] uses an What-If engine that attempts to predict the cost of different configurations given profiled data. For example, the engine can answer queries like "Given the profile of a job A, input data x, cluster resources c1, what will the performance of job B be with input data y and cluster resources c2". Here, finding good configurations hinges on the accuracy of the what-if engine itself; it showed less accuracy when tried with heterogeneous applications and cloud workloads [26]. AROMA [24] is a system for Hadoop resource provisioning and configuration tuning. It uses the k-medoids algorithm to cluster the executed jobs based on CPU, network and IO, then leverages Support Vector Machine (SVM) for tuning the configuration. Similarly, Bu et el. [11] proposed to tune Web systems such as Apache server and Tomcat configuration using reinforcement learning. They tuned 8 configuration parameters using 25 executions. Those approaches fit Hadoop and systems with limited number of configuration parameters, as the number of *tuned* configuration parameters (6-12) is significantly smaller than in other systems (such as Spark).

For more parameters, Yu et al. proposed DAC [31], a data-size aware Spark configuration tuning system, using a hierarchical modelling approach to approximate workload execution time as a function of its input data-size and configuration. It then leverages Genetic algorithms to search for good configurations based on the execution time estimated by the model. The high costs of workload executions to build this model are hard to amortize before re-tuning is needed. However, for frequently run static workloads, DAC improves performance by 30-89X with respect to the default configuration and tunes 41 configuration parameters. Wang et al. [29] leverage regression trees to tune Spark configurations; they tune 16 configuration parameters and improve performance by 36%. This approach also needs a significant number of execution samples to build a regression tree model of a good accuracy. BestConfig [35] leverages a divide-and-diverge sampling method and a recursive bound-and-search algorithm to tune configurations. It was used to tune 30 spark configuration parameters using 500 execution samples achieving 80% runtime performance improvement with respect to the default configuration.

C. Discussion

The existing isolated tuning paradigm does not even start to address the configuration tuning challenges, namely, *high overheads* (because of the need to re-run workloads numerous times, making cost amortization difficult and the whole idea of tuning impractical), *poor adaptivity* (in case of any changes of the environment or of the workload, rendering previous tuned configurations obsolete), *limited accuracy* (due to models which do not take into account what the workload actually does but considers them as black-boxes) and the *lack of transparency* for the end-user (who needs to have expertise in tuning).

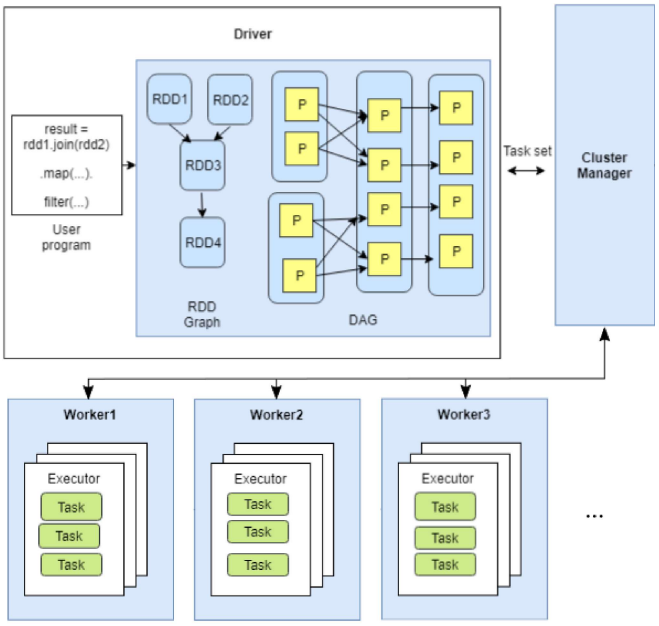

Fig. 2. Spark internal architecture, redrawn with further edits from [23]

For example, Cherrypick works well to tune the configuration for recurring analytics jobs but needs the end-user to define a representative workload for his recurring job, which requires intensive exploration efforts and technical background (*user intervention*). Ernest effectively tunes the machine-learning based jobs but does not adapt to SQL-based jobs (*poor adaptivity*). Starfish shows limited prediction accuracy when tried under different configurations [26] (*limited accuracy*). Lastly, most of the existing solutions incur high tuning *overhead*, by spending a significant amount of time exploring the search space, either to identify good configurations or to build predictive models for workload performance [35], [25], [24].

Providing an end-to-end configuration tuning as a cloud service has the potential to overcome these tuning challenges. Since the cloud is a centralized place for executing various workloads, it can leverage gathered execution metrics across multiple clients to offer tuning as a service, *fully* automating the configuration tuning of the analytics workloads in an adaptive way and with minimal end-user involvement.

III. A Motivating Example

We will refer to Apache Spark [1] as a concrete example for showing how a world in which self-tuning is a building-block would look like. Spark has been widely adopted for in-memory big data analytics. In this section we highlight its complex internal structure and numerous tunable parameters. This complexity, together with the variety of workloads it caters to, makes it a prime candidate for experimenting non-trivial strategies for automated, adaptive, accurate and efficient configuration tuning.

A. Spark Internals

Spark was developed to overcome the limitations of the MapReduce [14] paradigm in handling iterative workloads. MapReduce forces Mappers to write data to disk for reducers to read, which consumes significant IO resources for iterative applications (but also leads to more predictable resource usage patterns). Spark makes the design choice of keeping data in memory as RDDs [32] (Resilient Distributed Datasets), saving significant IO costs and speeding up iterative job execution time by up to 10x compared to Hadoop [33]. RDDs are immutable collections distributed over a cluster of machines to form a restricted shared memory, with each RDD consisting of a set of partitions. Fig. 2 shows how Spark works internally. Users write a program and submit it to the Spark Driver, which is a separate process that executes user applications and schedules them into executable jobs. The Spark programming model is based on two types of function, namely, transformations and actions. Transformations represent lazy computations on the RDD that create a new RDD (e.g., map, filter, etc.). Actions trigger computation on an RDD and produce an output (e.g., count, collect, etc.). When an application invokes an action on an RDD, it triggers a Spark job. Each job has an RDD dependency graph which is a graph of all parent RDDs of an RDD, representing a logical execution plan for a set of transformations and the lineage of RDDs. In terms of runtime, this complexity allows for various critical paths and bottlenecks which can vary from one workload to the next, making the system as a whole more difficult to model as a black box.

The RDD graph is mapped into a Directed Acyclic Graph (DAG) that represents the physical execution plan of how a job will be split into stages, the dependencies between stages and the partitions processed in each stage. The Driver uses the DAG to define the set of tasks to execute at each stage. Typically, an RDD partition is given as an input to a stage and is processed by a Spark task. Finally, the driver sends the task set to the cluster manager which then assigns tasks to worker nodes. A worker node can have multiple executors, with each of them being a process executing an assigned task and sending the result back to the driver.

B. Spark Internal Choices

Each Spark executor has a number of configuration parameters that significantly influence global performance, with tuned configuration parameters being able to improve the performance by up to 89X compared to the default configuration [31]. Spark has 200 configuration parameters [8], with the search space to tune just 30 of them exceeding 10^{40} possible configurations. For each workload, the user needs to find the best choice for configuration parameters covering different execution aspects such as processing, memory, networking and data shuffling. Example to these choices are: how many executor instances? what is the size of memory per executor? what is the number of cores per executor? how many partitions within RDD? what is the size of shuffled data buffer? should the shuffled data be compressed?

Exposing all of those knobs makes the system flexible. It also makes it difficult to run efficiently without a mountain of expertise and measurement. Even so, human expertise is of little help in dynamic tuning. Consider web services: they can make direct use of cloud elasticity because of architectures sharing little state between instances and clear metrics such as number of requests per second and latency. Fewer metrics are available to make similar decisions for long-running tasks or for their stateful execution as part of a large DAG.

IV. Vision

Our vision is to enable the fully automated, accurate and adaptive tuning of analytics workloads while bounding costs on the end-user side. This will facilitate workload deployment and save effort, time and money for cloud users. Consider a user wanting to deploy his Spark analytics workload to the cloud, while lacking facilities of self-tuning: he needs to use prior knowledge about his workload to efficiently pick the best cloud instance family and type for deployment (risking either higher costs or long runtimes and crashes when choosing incorrectly). He then starts the time consuming task of figuring out which of the default Spark parameters he should change to make things better. Any failed test execution is expensive and has a long fix-execute-debug cycle.

A recent study shows that 40% [9] of the analytics jobs are recurring while the remaining are changing over time. Furthermore, even recurring jobs have characteristics that change (different input, increasing data sizes). Therefore, it is expected that in the usual case, the configuration will need regular re-tuning [10]. Triggering this could be as simple as detecting relative performance degradation over time while running the same workload type on the same cluster configuration. Currently, it is the end-user's responsibility to detect this degradation and change the configuration accordingly.

In an ideal world, we've claimed that it's the cloud provider who needs to address resource provisioning and DISC system tuning, also detecting the need for re-tuning with minimal end-user intervention.

Feasibility: The cloud is a centralized place for executing numerous workloads, and the cloud providers witnesses abundant execution metrics under different cloud and DISC systems configurations. They can leverage the gathered execution metrics to build tuning models of better accuracy. Moreover, the cloud providers are aware of any underlying changes in workload co-location, network congestion, etc. This allows them to instantly detect any change in workload characteristics and adapt resource provisioning and DISC system configuration accordingly. However, providing an integrated tuning service could be seen as a risk on the cloud providers' revenues, since it accelerates workload deployment and execution. On the other hand, providing such a service also has significant potential benefits: It will minimize the configuration tuning burden on the user and require less expertise for running complex workloads at lower price points, eventually leading to a wide democratization of cloud to non-technical users, offering a path of growth for cloud providers in an already

competitive environment. We illustrate the principles of our proposed vision in the following subsections. All observations are based on our experience in running our own self-tuning Spark prototype in clouds from two major providers, totalling more than 6 months of continued execution for clusters from 4 VMs to 20 VMs, with more than 2000 configurations tested across 5 types of workloads.

A. Seamless configuration tuning with little human intervention

The types of users running cloud-based big data analytics workloads will in time become more diverse. For example, there is no reason why part of the jobs running today in specialized HPC clusters will not be able to run in the cloud, making use of large pools of resources but accepting some form of co-location or shared infrastructure. There is a big group of data scientists in domains from biology [18], [27] to physics [21] and astronomy [34] which have spent a significant amount of time tuning their workloads for specific HPC cluster environments. Those configurations will not necessarily port well to new shared environments, so anything aiding the transition will help bring new customers for cloud providers.

Similarly, lower entry-point prices will diversify the number of people wanting to run complex data processing jobs or machine learning algorithms to test hypotheses or analyse existing public datasets. Research in Universities without resources for private clusters or realizing the costs in managing and keeping those up to date might also partially make use of cloud infrastructures. All of this means a new "generation" of users not necessarily focused on optimizing workloads but interested in the results of their execution will see manual configuration as a barrier, and running with the default configurations as prohibitively expensive.

Thirdly, if the cloud is to become a commodity which people use either directly or indirectly as a fact-of-life (to get reports about their IoT infrastructure, to understand their impact on the world around them by collecting data from sensors in the environment, etc), it is clear that complete transparency into how configuration details are chosen is important. As a comparison with the past, making a phone call today no longer requires the manual configuration of electrical circuits along the line or talking to an operator. Why should the cloud-of-tomorrow be any different?

B. Resilience to input data and environment changes

Running analytics workloads that process ever growing data sets in an *elastic* cloud environment implies the vulnerability of the workloads to frequent changes in their characteristics, either due to the data itself or due to changes in the underlying infrastructure.

We experimented three different workloads from a popular big data benchmark [20], we used three evolving input sizes (`DS1`, `DS2` and `DS3`) for each workload. An Amazon EMR cluster of four `h1.4xlarge` instances was used to run this experiment. If the tuning approach is not resilient to input size changes then it will tune the configuration once for `DS1` and reuse this configuration for the evolving input sizes. However, re-tuning the configuration over the growing input sizes has the potential to save workload execution time through accommodating the changes in workload characteristics. To illustrate this potential savings, for each workload and input size we ran the workload using 100 random configurations to find the best configuration. Table I shows the potential savings in execution time from the re-used configuration of `DS1` to the best configuration found for `DS2` and `DS3`. Predictably, as the input size grows, the execution time savings from re-tuning the configuration can be significant and reach up to 56%. However, the amount of execution time savings varies from workload to another and re-tuning can lead to marginal or no savings (Wordcount workload). It is crucial to *accurately* and *efficiently* define the need for configuration re-tuning to seize any optimization opportunities while accelerating the amortization of the re-tuning cost. Paradoxically, the current approaches for cloud and DISC systems tuning require the user to perform a significant number of new executions for identifying configuration that is adapted to changes in workload characteristics. Our vision of the future configuration tuning suggests that approaches should aim to be resilient to such changes, automatically detecting the need of re-tuning and finding a new close-to-optimal configuration after a minimum number of executions. This would help the end-user to accelerate the deployment of their analytics workloads and the extraction of insights from data, instead of worrying about workload tuning and optimization.

Potential savings	**Pagerank**	**Bayes Classifier**	**Wordcount**
$DS1\_best - DS2\_best$	8%	17%	0%
$DS1\_best - DS3\_best$	56%	25%	3%

TABLE I
POTENTIAL EXECUTION TIME SAVING OF RE-TUNING CONFIGURATION OVER EVOLVING INPUT SIZES.

C. Offload tuning cost to the cloud provider

The current isolated tuning paradigm has high tuning costs, possibly higher than the actual runtime cost of the workload during its lifetime (especially for non-periodic or one-off workloads). In practice, even for frequently run jobs, the number of executions required to tune a workload might exceed the number of times the workload runs before re-tuning is necessary. For example, the `BestConfig` [35] system requires 500 execution samples to identify a good Spark configuration, and this would consume more resources than the 90 "normal" runs of our exemplar workload during a 3 months period. Indeed, the cost of workload tuning should not outweigh the runtime cost of the workload before it requires re-tuning.

This is especially important as tuning itself uses multiple executions to iteratively search for good configurations, and before one is found the tuning system will inevitably explore

configurations which could yield worse performance than the initial configuration. If the time spent executing using those bad configurations is not later outweighed by the increase in speed due to the best found configuration, tuning makes no sense.

Our vision is that the high tuning and re-tuning costs should be offloaded from the user to the cloud provider. The cloud is a centralized place that is able to keep a record of the different workloads' execution history under different cloud and DISC system configurations, across users. This data can only be leveraged by the cloud provider, and it would enable efficient configuration tuning from the user's perspective. This approach to end-to-end tuning of the analytics workloads will bound the tuning costs on the user side and speed up cost amortization on the cloud provider side.

D. Jobs should run within X% of the optimal runtime

Today, cloud providers offer no guarantees about high-level properties of jobs run by users on their infrastructure. Instead, they set service-level objectives (SLOs) related to properties of infrastructure which they have full control over (network and disk throughput, latency, etc). However, in a world where some workloads become predictable enough *because* of long-term characterization and tuning, it would make sense to offer end-users guarantees based on those known properties. And beyond these, the tuning service could let users make trade-off decisions which impact things like cost: do I need the results quickly no matter the cost, or am I willing to wait a long time for the results?

Presently, such choices are implicit in the user's picked cloud configuration, but lacking tuning the impacts are unclear: Who can tell me if scaling vertically, horizontally or both gives me the best benefit vs cost ratio?

Predictability naturally leads to better cost forecasting on the user side (users want to know how much they will pay when running a system that potentially scales up and down for some periodic workloads), but just as importantly simplifies the task of cloud provider's job scheduler and should make it more efficient in selecting locations for job execution.

The objective stated in the title of the section is challenging to achieve as stated (mostly because measuring it requires knowing the optimal execution time, but that could be replaced with "the runtime of similar workloads ever run in the cloud"). However, we believe it's a good goal to aspire to in terms of the language in which the new type of SLOs should be formulated. In the end, this should represent a commonly agreed metric for the *efficiency of the tuning system.*

V. Challenges

In this section we illustrate the challenges and open questions associated with offering a fully autonomous configuration tuning service by the cloud provider.

A. Develop models that can transfer their tuning knowledge

It is important to define the tuning models in a way that can transfer the acquired tuning knowledge to similar workloads. In our context, the key knowledge to transfer is the correlation between the different configuration parameters and the workload performance. However, it is challenging to extract this information from complex machine learning models, which usually work as a *black-box* and do not explain underlying mechanisms. For example, Gaussian process optimization has been applied successfully to enable data-efficient cloud configuration tuning [10]. However, it is challenging to extract the acquired tuning knowledge from Gaussian process. Some work has been proposed to increase the interpretability of black-box approaches while maintaining the modelling accuracy. Duvenaud et al. [16] proposed the *Additive* Gaussian processes, which decomposes the model into a sum of low-dimensional functions, each depending on only a subset of the input variables, potentially enabling the interpretation of input interactions and their influence on the variance of the overall model.

It is similarly possible to move beyond black-box modelling, perhaps using static analysis of submitted workloads to predict critical execution paths and bottlenecks and learn the corresponding configuration parameters which eliminate them.

B. Leverage the tuning knowledge across workloads

Usually, configuration tuning takes place by building one model predicting the relationship between configuration and runtime per workload. A single such model cannot generalize runtime predictions across workloads of different characteristics. However, different workloads might still share behaviours or type of sensitivity to particular configuration parameters. Using this information can lead to significant improvements in the efficiency of tuning. As an example for ways of achieving this AROMA [24] proposed to cluster similar workloads and build a prediction model for each cluster. The challenge lays in finding accurate ways to i) characterize workloads and define similarity across workloads: some work has been proposed to characterize the different analytics workloads [12], [22] but further study is needed to show its effectiveness in end-to-end configuration tuning; ii) inject the acquired knowledge from one tuning workload to a similar one: this has the potential to accelerate the tuning and improve its data efficiency (required number of workload executions). Some work has proposed taking advantage of *homogeneous* transfer learning across similar tasks in NLP [15], [13]. Further work is needed to study its applicability in the context of configuration tuning. The idea here is to use a pre-trained model "template" to initialize models for workloads with similar characteristics, which are then fine-tuned to its unique properties. The promise is in a faster convergence of the tuning process.

Irrespective of the path chosen, the accurate characterization of analytic workloads is crucial in being able to detect similarities between them in the first place to avoid any negative transfer [17].

C. Define a metric for tuning accuracy as part of SLOs

Current cloud provider's SLO contain specific quantified characteristics of the provided service such as availability,

throughput, frequency, response time, or quality. With configuration tuning offered as an integral part of the cloud services, there should be an agreement on how to characterize its properties as part of the SLO. Open questions remain as to exactly what metrics of effectiveness should be picked: Can the effectiveness of the configuration be defined as a distance from the optimal configuration? What if it is not viable to find the optimal configuration in high dimensional spaces? Should that reference configuration be defined as the best configuration found for a similar workload? Alternatively, would the amount of performance improvement with respect to the default configuration be acceptable as a metric in the SLO? What if there is no default configurations (the case of cloud configuration tuning, there is no default instance family and type)?

In the end, the relevant metrics will emerge after cloud providers also have a better picture of what they *could* offer guarantees on, so prototype implementations will be required before settling for one particular option.

D. Define the need for workload re-tuning

To have a fully autonomous configuration tuning service, it is important to accurately define the need for workload re-tuning. The tuning service should be able to distinguish marginal changes in workload characteristics from dramatic ones. This will allow the service to detect the need for further optimization and avoid any long-term performance degradation and associated cost inefficiencies. Moreover, accurately defining the need for re-tuning will enable avoiding any false re-tuning. For example, simply picking fixed percentual runtime deltas as thresholds for re-tuning are likely to lead to it being done either too frequently or too late.

The current approach for detecting the need for workload re-tuning is based on fixed heuristics [10] that can have similar accuracy issues across different workloads (i.e. what is considered a marginal change for a workload, might be a significant change for another).

VI. Conclusion

We presented the vision of seamless configuration tuning for analytics workloads. It revolves around fully automating configuration tuning and making it transparent to the end-user, from choice of cluster properties to DISC framework configurations. We illustrated and discussed the implications for the four principles of our proposed vision, which are:

1) Enabling configuration tuning for users with minimal expertise in workload optimisation,
2) Configuration tuning that is resilient to dynamic workload changes,
3) Bounded tuning cost for the end-user and its offload to the cloud provider,
4) Augmentation of the SLO with metrics for measuring the effectiveness of tuning.

Acknowledgements

We thank Google and Amazon for generously supporting us with Google Cloud and AWS research credits to experiment our Spark tuning prototype, which certainly helped in crystallizing this vision.

References

[1] Apache spark: fast and general engine for large-scale data processing., 2015. https://spark.apache.org/.

[2] Growth forecast for the worldwide big data and business analytics market through 2020, 2015. https://www.idc.com/getdoc.jsp?containerId=prUS41826116.

[3] Apache flink, 2016. http://flink.apache.org/.

[4] Apache hadoop, 2016. http://hadoop.apache.org/.

[5] Amazon EMR, 2018. https://aws.amazon.com/emr/.

[6] Google Dataproc, 2018. https://azure.microsoft.com/en-gb/services/hdinsight/.

[7] Google Dataproc, 2018. https://cloud.google.com/dataproc/.

[8] Spark Configuration parameters, 2018. https://spark.apache.org/docs/latest/configuration.html.

[9] Sameer Agarwal, Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 21–21. USENIX Association, 2012.

[10] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[11] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. A reinforcement learning approach to online web systems auto-configuration. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 2–11. IEEE, 2009.

[12] Tatsuhiro Chiba and Tamiya Onodera. Workload characterization and optimization of tpc-h queries on apache spark. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 112–121. IEEE, 2016.

[13] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. Additive gaussian processes. In *Advances in neural information processing systems*, pages 226–234, 2011.

[17] Liang Ge, Jing Gao, Hung Ngo, Kang Li, and Aidong Zhang. On handling negative transfer and imbalanced distributions in multiple source transfer learning. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(4):254–271, 2014.

[18] Runxin Guo, Yi Zhao, Quan Zou, Xiaodong Fang, and Shaoliang Peng. Bioinformatics applications on apache spark. *GigaScience*, 7(8):giy098, 2018.

[19] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *Cidr*, volume 11, pages 261–272, 2011.

[20] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[21] Kevin Jacobs and Kacper Surdy. Apache flink: Distributed stream data processing. Technical report, 2016.

[22] Zhen Jia, Jianfeng Zhan, Lei Wang, Rui Han, Sally A McKee, Qiang Yang, Chunjie Luo, and Jingwei Li. Characterizing and subsetting big data workloads. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 191–201. IEEE, 2014.

[23] Anton Kirillov. Spark Internal Architecture, 2016. http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/.

[24] Palden Lama and Xiaobo Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th international conference on Autonomic computing*, pages 63–72. ACM, 2012.

[25] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali R Butt, and Nicholas Fuller. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 165–176. ACM, 2014.

[26] Guangdeng Liao, Kushal Datta, and Theodore L Willke. Gunther: Search-based auto-tuning of mapreduce. In *European Conference on Parallel Processing*, pages 406–419. Springer, 2013.

[27] Lizhen Shi, Xiandong Meng, Elizabeth Tseng, Michael Mascagni, and Zhong Wang. Sparc: Scalable sequence clustering using apache spark. *bioRxiv*, page 246496, 2018.

[28] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *NSDI*, pages 363–378, 2016.

[29] Guolu Wang, Jungang Xu, and Ben He. A novel method for tuning configuration parameters of spark based on machine learning. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 586–593. IEEE, 2016.

[30] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 452–465. ACM, 2017.

[31] Zhibin Yu, Zhendong Bei, and Xuehai Qian. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 564–577. ACM, 2018.

[32] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[33] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[34] Zhao Zhang, Kyle Barbary, Frank Austin Nothaft, Evan Sparks, Oliver Zahn, Michael J Franklin, David A Patterson, and Saul Perlmutter. Scientific computing meets big data technology: An astronomy use case. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 918–927. IEEE, 2015.

[35] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 338–350. ACM, 2017.