Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics

Ana Klimovic, *Stanford University;* Heiner Litz, *UC Santa Cruz;*
Christos Kozyrakis, *Stanford University*

https://www.usenix.org/conference/atc18/presentation/klimovic-selecta

ISBN 978-1-939133-02-1

Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics

Ana Klimovic
Stanford University

Heiner Litz
UC Santa Cruz

Christos Kozyrakis
Stanford University

Abstract

Data analytics are an important class of data-intensive workloads on public cloud services. However, selecting the right compute and storage configuration for these applications is difficult as the space of available options is large and the interactions between options are complex. Moreover, the different data streams accessed by analytics workloads have distinct characteristics that may be better served by different types of storage devices.

We present Selecta, a tool that recommends near-optimal configurations of cloud compute and storage resources for data analytics workloads. Selecta uses latent factor collaborative filtering to predict how an application will perform across different configurations, based on sparse data collected by profiling training workloads. We evaluate Selecta with over one hundred Spark SQL and ML applications, showing that Selecta chooses a near-optimal performance configuration (within 10% of optimal) with 94% probability and a near-optimal cost configuration with 80% probability. We also use Selecta to draw significant insights about cloud storage systems, including the performance-cost efficiency of NVMe Flash devices, the need for cloud storage with support for fine-grain capacity and bandwidth allocation, and the motivation for end-to-end storage optimizations.

1 Introduction

The public cloud market is experiencing unprecedented growth, as companies move their workloads onto platforms such as Amazon AWS, Google Cloud Platform and Microsoft Azure. In addition to offering high elasticity, public clouds promise to reduce the total cost of ownership as resources can be shared among tenants. However, achieving performance and cost efficiency requires choosing a suitable configuration for each given application. Unfortunately, the large number of instance types and configuration options available make selecting the right resources for an application difficult.

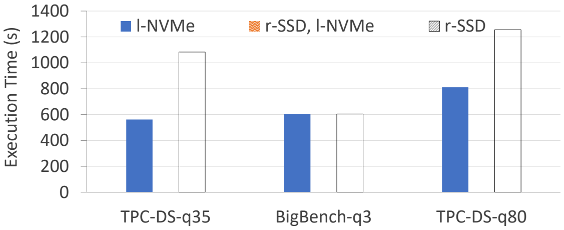


Figure 1: Performance of three applications on eight `i3.xl` instances with different storage configurations.

The choice of storage is often essential, particularly for cloud deployments of data-intensive analytics. Cloud vendors offer a wide variety of storage options including object, file and block storage. Block storage can consist of hard disks (*HDD*), solid-state drives (*SSD*), or high bandwidth, low-latency NVMe Flash devices (*NVMe*). The devices may be local (l) to the cloud instances running the application or remote (r). These options alone lead to storage configuration options that can differ by orders of magnitude in terms of throughput, latency, and cost per bit. The cloud storage landscape is only becoming more diverse as emerging technologies based on 3D X-point become available [35, 16].

Selecting the right cloud storage configuration is critical for both performance and cost. Consider the example of a Spark SQL equijoin query on two 128 GB tables [53]. We find the query takes 8.7× longer when instances in an 8-node EC2 cluster access r-HDD compared to l-NVMe storage. This is in contrast to a recent study, conducted with a prior version of Spark, which found that faster storage can only improve the median job execution time by at most 19% [50]. The performance benefits of l-NVMe lead to 8× lower execution cost for this query, even though NVMe storage has higher cost per unit time. If we also consider a few options for the number of cores and memory per instance, the performance gap between the best and worst performing VM-storage configurations is over 30×.

Determining the right cloud configuration for analytics applications is challenging. Even if we limit ourselves to a single instance type and focus on optimizing performance, the choice of storage configuration for a particular application remains non-trivial. Figure 1 compares the performance of three Spark applications using 8 `i3.xl` AWS instances with *l*-NVMe, *r*-SSD, and a hybrid (*r*-SSD for input/output data, *l*-NVMe for intermediate data). The first application is I/O-bound and benefits from the high throughput of NVMe Flash. The second application has a CPU bottleneck and thus performs the same with all three storage options. The third application is I/O-bound and performs best with the hybrid storage option since it minimizing interference between read and write I/Os, which have asymmetric performance on Flash [40]. This result should not be surprising. Analytics workloads access multiple data streams, including input and output files, logs, and intermediate data (e.g., shuffle and broadcast). Each data stream has distinct characteristics in terms of access frequency, access patterns, and data lifetime, which make different streams more suitable for different types of storage devices. For example, for TPC-DS query 80 in Figure 1, storing input/output data on *r*-SSD and intermediate data on *l*-NVMe Flash outperforms storing all data on *l*-NVMe as it isolates streams and eliminates interference.

We present *Selecta*, a tool that learns near-optimal VM and storage configurations for analytics applications for user-specified performance-cost objectives. Selecta targets analytics jobs that are frequently or periodically rerun on newly arriving data [1, 25, 55]. A configuration is defined by the type of cloud instance (core count and memory capacity) along with the storage type and capacity used for input/output data and for intermediate data. To predict application performance for different configurations, Selecta applies latent-factor collaborative filtering, a machine-learning technique commonly used in recommender systems [10, 57, 11, 22, 23]. Selecta uses sparse performance data for training applications profiled on various cloud configurations, as well as performance measurements for the target application profiled on only two configurations. Selecta leverages the sparse training data to learn significantly faster and more cost-effectively than exhaustive search. The approach also improves on recent systems such as CherryPick and Ernest whose performance prediction models require more information about the target application and hence require more application runs to converge [3, 69]. Moreover, past work does not consider the heterogeneous cloud storage options or the varying preferences of different data streams within each application [71].

We evaluate Selecta with over one hundred Spark SQL and ML workloads, each with two different dataset scaling factors. We show that Selecta chooses a near-optimal performance configuration (within 10% of optimal) with 94% probability and a near-optimal cost configuration with 80% probability. We also analyze Selecta's sensitivity to various parameters such as the amount of information available for training workloads or the target application.

A key contribution of our work is our analysis of cloud storage systems and their use by analytics workloads, which leads to several important insights. We find that in addition to offering the best performance, NVMe-based configurations also offer low execution cost for a wide range of applications. We observe the need for cloud storage options that support fine-grain allocation of capacity and bandwidth, similar to the fine-grain allocation of compute and memory resources offered by serverless cloud services [7]. Disaggregated NVMe Flash can provide the substrate for such a flexible option for cloud storage. Finally, we showcase the need for end-to-end optimization of cloud storage, including application frameworks, operating systems, and cloud services, as several storage configurations fail to meet their potential due to inefficiencies in the storage stack.

2 Motivation and Background

We discuss current approaches for selecting a cloud storage configuration and explain the challenges involved.

2.1 Current Approaches

Conventional configurations: Input/output files for data analytics jobs are traditionally stored in a distributed file system, such as HDFS or object storage systems such as Amazon S3 [62, 6]. Intermediate data is typically read/written to/from a dedicated local block storage volume on each node (i.e., *l*-SSD or *l*-NVMe) and spilled to *r*-HDD if extra capacity is needed. In typical Spark-as-a-service cloud deployments, two remote storage volumes are provisioned by default per instance: one for the instance root volume and one for logs [19].

Existing tools: Recent work focuses on automatically selecting an optimal VM configuration in the cloud [71, 69, 3]. However, these tools tend to ignore the heterogeneity of cloud storage options, at best distinguishing between 'fast' and 'slow'. In the next section, we discuss the extent of the storage configuration space.

2.2 Challenges

Complex configuration space: Cloud storage comes in multiple flavors: object storage (e.g., Amazon S3 [6]), file storage (e.g., Azure Files [45]), and block storage (e.g., Google Compute Engine Persistent Disks [29]). Block and object storage are most commonly used for data analytics. Block storage is further sub-divided into hardware options: cold or throughput-optimized hard drive disk, SAS SSD, or NVMe Flash. Block storage can be local (directly attached) or remote (over the network) to an instance. Local block storage is ephemeral; data persists only as long as the instance is running. Remote volumes persist until explicitly deleted by the user.

Table 1 compares three block storage options available in Amazon Web Services (AWS). Each storage option provides a different performance, cost, and flexibility trade-off. For instance, *l*-NVMe storage offers the highest throughput and lowest latency at higher cost per bit. Currently, cloud providers typically offer NVMe in fixed capacity units directly attached to select instance types, charged per second or hour. AWS currently charges \$0.023 more per hour for an instance with 475 GB of NVMe Flash compared to without NVMe. In contrast, S3 fees are based on capacity (\$0.023 per GB/month) and bandwidth (\$0.004 per 10K GET requests) usage.

In addition to the storage configuration, users must choose from a variety of VM types to determine the right number of CPU cores and memory, the number of VMs, and their network bandwidth. These choices often affect storage and must be considered together. For example, on instances with 1 Gb/s network bandwidth, the network limits the sequential throughput achievable with *r*-HDD and *r*-SSD storage volumes in Table 1.

Performance-cost objectives: While configurations with the most CPU cores, the most memory, and fastest storage generally provide the highest performance, optimizing for runtime cost is much more difficult. Systems designed to optimize a specific objective (e.g., predict the configuration that maximizes performance or minimizes cost) are generally not sufficient to make recommendations for more complex objectives (e.g., predict the configuration that minimizes execution time within a specific budget). By predicting application execution time on candidate configurations, our approach remains general. Unless otherwise specified, we refer to cost as the cost of executing an application.

Heterogeneous application data: We classify data managed by distributed data analytics frameworks (e.g., Spark [74]) into two main categories: *input/output data* which is typically stored long-term and *intermediate data* which lives for the duration of job execution. Examples of intermediate data include shuffle data exchanged between mappers and reducers, broadcast variables, and cached dataset partitions spilled from memory. These streams typically have distinct access frequency, data lifetime, access type (random vs. sequential), and I/O size. For example, input/output data is generally long-lived and sequentially accessed, whereas intermediate data is short-lived and most accesses are random.

Storage	Seq	Seq	Rand	Rand	Rand
	Read	Write	Read	Write	Rd/Wr
	MB/s	MB/s	IOPS	IOPS	IOPS
r-HDD	135	135	132	132	132
r-SSD	165	165	3,068	3,068	3,068
l-NVMe	490	196	103,400	35,175	70,088

Table 1: Block storage performance for 500GB volumes. Sequential IOs are 128 KB, random IOs are 4 KB.

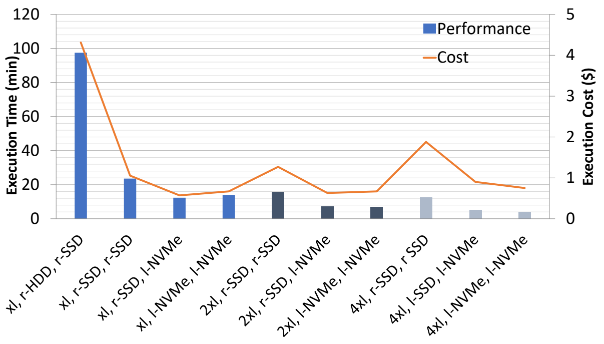


Figure 2: Comparison of execution time and cost for TPC-DS query 64 on various VM and storage configurations, defined as <VM size, storage for input/output data, storage for intermediate data>.

Storage decisions are complex: Selecting the right configuration for a job significantly reduces execution time and cost, as shown in Figure 2, which compares a Spark SQL query (TPC-DS query 64) on various VM and storage configurations in an 8-node cluster. We consider 3 `i3` VM instance sizes in EC2 (`xl, 2xl, and 4xl`) and heterogeneous storage options for input/output and intermediate data. The lowest performing configuration has 24× the execution time of the best performing configuration. Storing input/output data on *r*-SSD and intermediate data on *l*-NVMe (the lowest cost configuration) has 7.5× lower cost than storing input/output data on *r*-HDD and intermediate data on *r*-SSD.

3 Selecta Design

3.1 Overview

Selecta is a tool that automatically predicts the performance of a target application on a set of candidate configurations. As shown in Figure 3, Selecta takes as input: i) execution time for a set of training applications on several configurations, ii) execution time for the target application on two reference configurations, and iii) a performance-cost objective for the target application. A configuration is defined by the number of nodes (VM instances), the CPU cores and memory per node, as well as the storage type and capacity used for input/output data and for intermediate data. Selecta uses latent factor collaborative filtering (see §3.2) to predict the performance

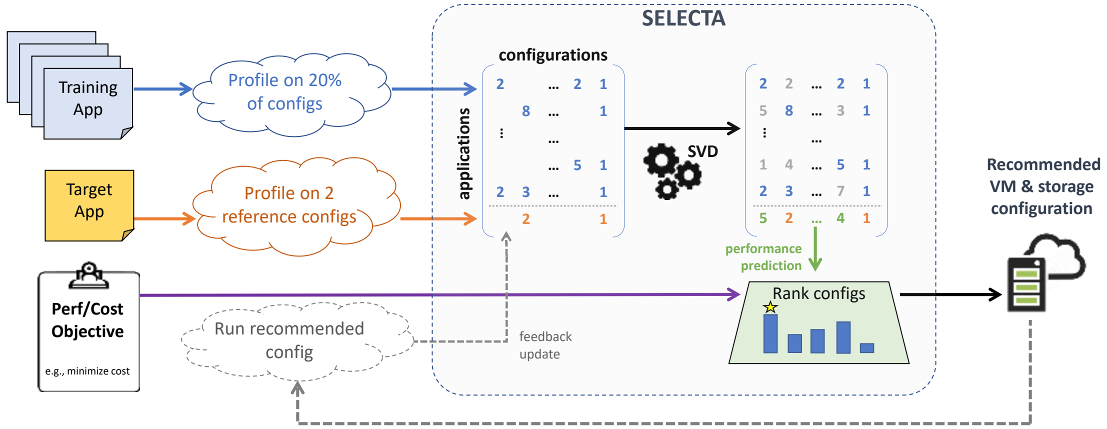


Figure 3: An overview of performance prediction and configuration recommendation with Selecta.

of the target application on the remaining (non-reference) candidate configurations. With these performance predictions and the per unit time cost of various VM instances and storage options, Selecta can recommend the right configuration for the user's performance-cost objective. For example, Selecta can recommend configurations that minimize execution time, minimize cost, or minimize execution time within a specific budget.

As new applications are launched over time, these performance measurements become part of Selecta's growing training set and accuracy improves (see § 4.4). We also feed back performance measurements after running a target application on a configuration recommended by Selecta — this helps reduce measurement noise and improve accuracy. Since Selecta takes ∼1 minute to generate a new set of predictions (the exact runtime depends on the training matrix size), a user can re-run Select when re-launching the target application with a new dataset to get a more accurate recommendation. In our experiments, the recommendations for each target application converge after two feedback iterations. The ability to grow the training set over time also provides Selecta with a mechanism for expanding the set of configurations it considers. Initially, the configuration space evaluated by Selecta is the set of configurations that appear in the original training set. When a new configuration becomes available and Selecta receives profiling data for applications on this configuration, the tool will start predicting performance for all applications on this configuration.

3.2 Predicting Performance

Prediction approach: Selecta uses collaborative filtering to predict the performance of a target application on candidate configurations. We choose collaborative filtering as it is agnostic to the details of the data analytics framework used (e.g., Spark vs. Storm) and it allows us to leverage *sparse* training data collected *across applications* and configurations [56]. While systems such as CherryPick [3] and Ernest [69] build performance models based solely on training data for the target application, Selecta's goal is to leverage training data available from multiple applications to converge to accurate recommendations with only two profiling runs of a target application. We discuss alternatives to collaborative filtering to explain our choice.

Content-based approaches, such as as linear regression, random forests, and neural network models, build a model from features such as application characteristics (e.g., GB of shuffle data read/written) and configuration characteristics (e.g., I/O bandwidth or the number of cores per VM). We find that unless inputs features such as the average CPU utilization of the target application *on the target configuration* are used in the model, content-based predictors do not have enough information to learn the compute and I/O requirements of applications and achieve low accuracy. Approaches that require running target applications on all candidate configurations to collect feature data are impractical.

Another alternative is to build performance prediction models based on the structure of an analytics framework, such as the specifics of the map, shuffle, and reduce stages in Spark [36, 75]. This leads to framework-specific models and may require re-tuning or even remodeling as framework implementations evolve (e.g., as the CPU efficiency of serialization operations improves).

Latent factor collaborative filtering: Selecta's collaborative filtering model transforms applications and configurations to a latent factor space [10]. This space characterizes applications and configurations in terms of latent (i.e., 'hidden') features. These features are *automatically inferred* from performance measurements of training applications [56]. We use a matrix factorization technique known as Singular Value Decomposition (SVD) for the latent factor model. SVD decomposes an input matrix P, with rows representing applications and columns representing configurations, into the product of three matrices, U, λ, and V. Each element p_{ij} of P represents the normalized performance of application i on configuration j. The latent features are represented by singular values in the diagonal matrix λ, ordered by decreasing magnitude. The matrix U captures the strength of the correlation between a row in P and a latent feature in λ. The matrix V captures the strength of the correlation between a column in P and a latent feature in λ. Although the model does not tell us what the latent features physically represent, a hypothetical example of a latent feature is random I/O throughput. For instance, Selecta could infer how strongly an application's performance depends on random I/O throughput and how much random I/O throughput a configuration provides.

One challenge for running SVD is the input matrix P is sparse, since we only have the performance measurements of applications on certain configurations. In particular, we only have two entries in the target application row and filling in the missing entries corresponds to predicting performance on the other candidate configurations. Since performing SVD matrix factorization requires a fully populated input matrix P, we start by randomly initializing the missing entries and then run Stochastic Gradient Descent (SGD) to update these unknown entries using an objective function that minimizes the mean squared error on the *known* entries of the matrix [13]. The intuition is that by iteratively decomposing and updating the matrix in a way that minimizes the error for known entries, the technique also updates unknown entries with accurate predictions. Selecta uses the Python sci-kit `surprise` library for SVD [33].

3.3 Using Selecta

New target application: The first time an application is presented to Selecta, it is profiled on two reference configurations which, preferably, are far apart in their compute and storage resource attributes. Selecta requires that reference configurations remain fixed across all applications, since performance measurements are normalized to a reference configuration before running SVD. Profiling application performance involves running the application to completion and recording execution time and CPU utilization (including iowait) over time.

Defining performance-cost objectives: After predicting application performance across all configurations, Selecta recommends a configuration based on a user-defined ranking function. For instance, to minimize runtime cost, the ranking function is min(runtime × cost/hour). While choosing a storage technology (e.g., SSD vs. NVMe Flash), Selecta must also consider the application's storage capacity requirements. Selecta leverages statistics from profiling runs available in Spark monitoring logs to determine the intermediate (shuffle) data and and input/output data capacity [63].

Adapting to changes: Recurring jobs and their input datasets are likely to evolve. To detect changes in application characteristics that may impact the choice of optimal configuration, Selecta relies on CPU utilization information from both initial application profiling and subsequent executions rounds. When an application is first introduced to the system, Selecta assigns a unique ID to store application specific information such as iowait CPU utilization. Whenever an application is re-executed, Selecta compares the current iowait time to the stored configuration. Depending on the difference in iowait time, Selecta will either compute a refined prediction based on available measurements or treat the workload as new application, starting a new profiling run.

Dealing with noise in the cloud: An additional challenge for recommending optimal configurations is noise on public cloud platforms, which arises due to interference with other tenants, hardware heterogeneity, or other sources [59]. To account for noise, Selecta relies on the feedback of performance and CPU utilization measurements. Initially, with few profiling runs, Selecta's performance predictions are affected by noise. As more measurements are fed into the system, Selecta averages performance and CPU utilization and uses reservoir sampling to avoid high skew from outliers [70]. Selecta keeps a configurable number of sample points for each entry in the application-configuration matrix (e.g., three) to detect changes in applications as described above. If a particular run is heavily impacted by noise such that the compute and I/O bottlenecks differ significantly from previous runs, Selecta's mechanism for detecting changes in applications identifies the outlier.

4 Selecta Evaluation

Selecta's collaborative filtering approach is agnostic to the choice of applications and configurations. We evaluate Selecta for data analytics workloads on a subset of the cloud configuration space with the goal of understanding how to provision cloud storage for data analytics.

4.1 Methodology

Cloud configurations: We deploy Selecta on Amazon EC2 and consider configurations with the instance and storage options shown in Tables 2 and 3. Among the possible VM and storage combinations, we consider seventeen candidate configurations. We trim the space to stay within our research budget and to focus on experiments that are most likely to uncover interesting insights about cloud storage for analytics. We choose EC2 instance families that are also supported by Databricks, a popular Spark-as-a-service provider [18]. `i3` is currently the only instance family available with NVMe Flash and

Instance	CPU cores	RAM (GB)	NVMe
`i3.xlarge`	4	30	1 x 950 GB
`r4.xlarge`	4	30	-
`i3.2xlarge`	8	60	1 x 1.9 TB
`r4.2xlarge`	8	60	-
`i3.4xlarge`	16	120	2 x 1.9 TB
`r4.4xlarge`	16	120	-

Table 2: AWS instance properties

Storage	Type	Locality	Use for Input/Output Data?	Use for Intermediate Data?
r-HDD	Block	Remote	✓	-
r-SSD	Block	Remote	✓	✓
l-NVMe	Block	Local	✓	✓
S3	Object	Remote	✓	-

Table 3: AWS storage options considered

`r4` instances allow for a fair comparison of storage options as they have the same memory to compute ratio. We only consider configurations where the intermediate data storage IOPS are equal to or greater than the input/output storage IOPS, as intermediate data has more random accesses. Since we find that most applications are I/O-bound with *r*-HDD, we only consider *r*-HDD for the instance size with the least amount of cores. We limit our analysis to *r*-HDD because our application datasets are up to 1 TB whereas instances with *l*-HDD on AWS come with a minimum of 6 TB disk storage, which would not be an efficient use of capacity. We do not consider local SAS/SATA SSDs as their storage capacity to CPU cores ratio is too low for most Spark workloads. We use Elastic Block Store (EBS) for remote block storage [5].

We use a cluster of 9 nodes for our evaluation. The cluster consists of one master node and eight executor nodes. The master node runs the Spark driver and YARN Resource Manager. Unless input/output data is stored in S3, we run a HDFS namenode on the master server as well. We configure framework parameters, such as the JVM heap size and number of executors, according to Spark tuning guidelines and match the number of executor tasks to the VM's CPU cores [15, 14].

Applications: We consider Spark [74] as a representative data analytics framework, similar to previous studies [50, 68, 3]. We use Spark v2.1.0 and Hadoop v2.7.3 for HDFS. We evaluate Selecta with over one hundred Spark SQL and ML applications, each with two different dataset scales, for a total of 204 workloads. Our application set includes 92 queries of the TPC-DS benchmark with scale factors of 300 and 1000 GB [67]. We use the same scale factors for Spark SQL and ML queries from the TPC-BB (BigBench) benchmark which has of structured, unstructured and semi-structured data modeled after the retail industry domain [27]. Since most BigBench queries are CPU-bound, we focus on eight queries which have more substantial I/O requirements: queries 3, 8, 14, 16, 21, 26, 28, 29. We also run 100 and 400 GB sort jobs [52]. Finally, we run a SQL equijoin query on two tables with 16M and 32M rows each and 4KB entries [53]. For all input and output files, we use the uncompressed Parquet data format [26].

Experiment methodology: We run each application on all candidate configurations to obtain the ground truth performance and optimal configuration choices for each application. To account for noise in the cloud we run each experiment (i.e., each application on each candidate configuration) three times and use the average across runs in our evaluation. Two runs are consecutive and one run is during a different time of day. We also validate our results by using data from one run as input to Selecta and the average performance across runs as the ground truth. To train and test Selecta, we use leave-one-out cross validation [58], meaning one workload at a time serves as the target application while the remaining workloads are used for training. We assume training applications are profiled on all candidate configurations, except for the sensitivity analysis in §4.4 where we investigate training matrix density requirements for accurate predictions.

Metrics: We measure the quality of Selecta's predictions using two metrics. First, we report the relative root mean squared error (RMSE), a common metric for recommender systems. The second and more relevant metric for Selecta is the probability of making an accurate configuration recommendation. We consider a recommendation accurate if the configuration meets the user's cost-performance objective within a threshold T of the true optimal configuration for that application. For example, for a minimum cost objective with T = 10%, the probability of an accurate prediction is the percentage of Selecta's recommendations (across all tested applications) whose true cost is within 10% of the true optimal cost configuration. Using a threshold is more robust to noise and allows us to make more meaningful conclusions about Selecta's accuracy, since a second-best configuration may have similar or significantly worse performance than the best configuration. Our performance metric is execution time and cost is in US dollars.

4.2 Prediction Accuracy

We provide a matrix with 204 rows as input to Selecta, where one row (application) is designated as the target application in each test round. We run Selecta 204 times, each time considering a different application as the target. For now, we assume all remaining rows of training data in the matrix are dense, implying the user has profiled training applications on all candidate configurations. The single target application row is sparse, containing only two entries, one for each of the profiling runs on reference configurations.

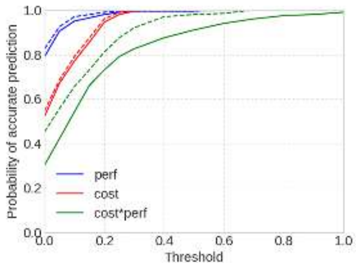


Figure 4: Probability of accurate recommendations within a threshold from optimal. Dotted lines are after one feedback iteration.

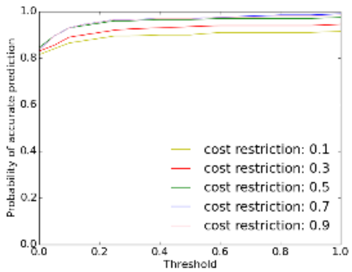


Figure 5: Probability of accurate configuration recommendation for performance within threshold, given strict cost restrictions.

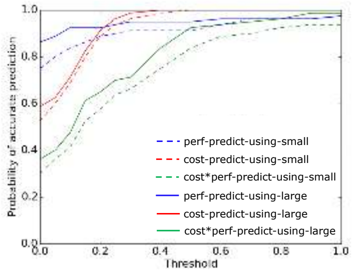


Figure 6: Accuracy with large datasets using predictions from small dataset vs. re-computing prediction with large dataset.

Selecta predicts performance with a relative RMSE of 36%, on average across applications. To understand how Selecta's performance predictions translate into recommendations, we plot accuracy in Figure 4 for performance, cost and cost*performance objectives. The plot shows the probability of near-optimal recommendations as a function of the threshold T defining what percentage from optimal is considered close enough. When searching for the best performing configuration, Selecta has a 94% probability of recommending a configuration within 10% of optimal. For a minimum cost objective, Selecta has a 80% probability of recommending a configuration within 10% of optimal. Predicting cost*performance is more challenging since errors in Selecta's relative execution time predictions for an application across candidate configurations are squared: cost*performance = (execution_time)^2 * config_cost_per_hour.

The dotted lines in Figure 4 show how accuracy improves after a single feedback round. Here, we assume the target application has the same dataset in the feedback round. This provides additional training input for the target application row (either a new entry if the recommended configuration was not a reference configuration, or a new sample to average to existing data if the recommended configuration was a reference configuration). The probability of near-optimal recommendations increases most noticeably for the cost*performance objective, from 52% to 65% after feedback, with T=10%.

Figure 5 shows the probability of accurate recommendations for objectives of the form "select the best performing configuration given a fixed cost restriction C." For this objective, we consider Selecta's recommendation accurate if its cost is less than or equal to the budget and if its performance is within the threshold of the true best configuration for the objective. Selecta achieves between 83% and 94% accuracy for the cost restrictions in Figure 5 assuming T=10%. The long tail is due to performance prediction errors that lead Selecta to underestimate the execution cost for a small percentage of configurations (i.e., cases where Selecta recommends a configuration that is actually over budget).

In Figure 7, we compare Selecta's accuracy against four baselines. The first baseline is a random forest predictor, similar to the approach used by PARIS [71]. We use the following features: the number of CPU cores, disk IOPS and disk MB/s the configuration provides, the intermediate and input/output data capacity of the application, and the CPU utilization, performance, and total disk throughput measured when running the application on each of the two reference configurations. Although the random forest predictor leverages more features than Selecta, it has lower accuracy. Collaborative filtering is a better fit for the sparse nature of the training data. We find the most important features in the random forest model are all related to I/O (e.g., the I/O throughput measured when running the application on the reference configurations and the read/write IOPS supported by the storage used for intermediate data), which emphasizes the importance of selecting the right storage.

The second baseline (labeled 'default') in Figure 7 uses the recommended default configurations documented in Databricks engineering blog posts: *l*-NVMe for intermediate data and S3 for input/output data [19, 21, 20]. The 'max cost per time' baseline uses the simple heuristic of always picking the most expensive instance per unit time. The 'min cost per time' baseline chooses the least expensive instance per unit time. Selecta outperforms all of these heuristic strategies, confirming the need for a tool to automate configuration selection.

4.3 Evolving Datasets

We study the impact of dataset size on application performance and Selecta's predictions using the small and large dataset scales described in §4.1. We train Selecta using all 102 workloads with small datasets, then evaluate Selecta's prediction accuracy for the same workloads with large datasets. The dotted lines in Figure 6 plots Selecta's accuracy when recommending configurations for applications with large datasets solely based on profiling runs of the application with a smaller dataset. The solid lines show accuracy when Selecta re-profiles applications with large datasets to make predictions. For approximately 8% of applications, profiling runs with small datasets are not sufficient indicators of performance with large datasets.

We find that in cases where the performance with a small dataset is not indicative of performance with a large dataset, the relationship between compute and I/O intensity of the application is affected by the dataset size. As described in §3.3, Selecta detects these situations by comparing CPU utilization statistics for the small and large dataset runs. Figure 8 shows an example of a workload for which small dataset performance is not indicative of performance with a larger dataset. We use the Intel Performance Analysis Tool to record and plot CPU utilization [34]. When the average iowait percentage for the duration of the run changes significantly between the large and small profiling runs on the reference configuration, it is generally best to profile the application on the reference configurations and treat it as a new application.

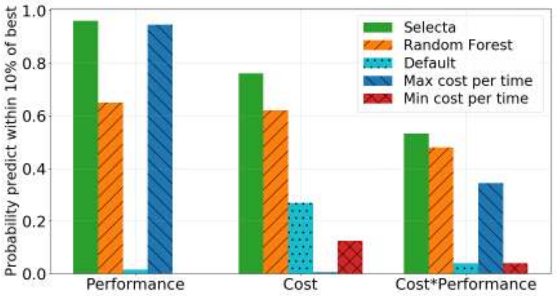
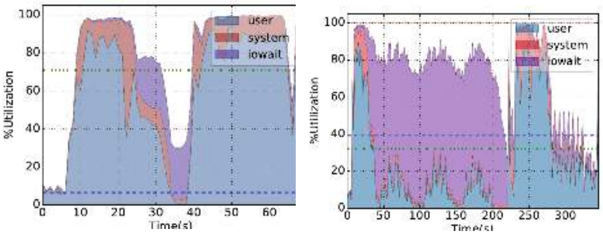


Figure 7: Selecta's accuracy compared to baselines.



(a) Query on 300GB is CPU-bound. (b) Query on 1TB is IO-bound.

Figure 8: CPU utilization over time for TPC-DS query 89 on `r4.xlarge` cluster with *r*-SSD. For this query, performance with a small dataset is not indicative of performance with a larger dataset. Selecta detects difference in average iowait percentage (blue dotted line).

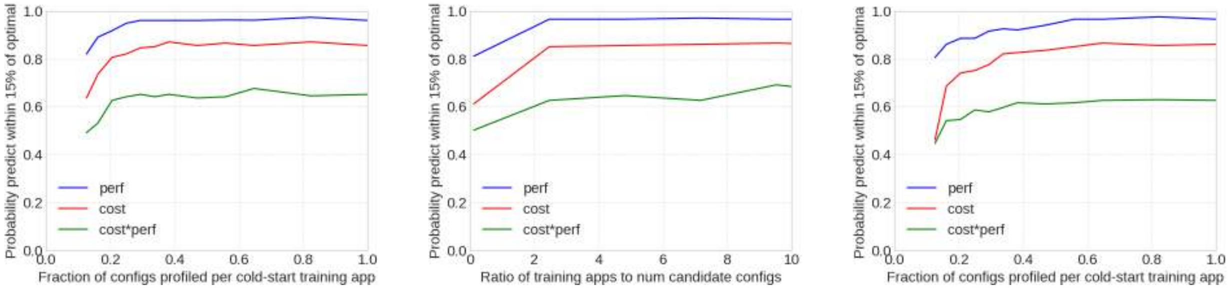
4.4 Sensitivity Analysis

We perform a sensitivity analysis to determine input matrix density requirements for accurate predictions. We look at both the density of matrix rows (i.e., the percentage of candidate configurations that training applications are profiled on) and the density of matrix columns (i.e., the number of training applications used). We also discuss sensitivity to the choice of reference configurations.

Figure 9a shows how Selecta's accuracy for performance, cost and cost*performance objectives varies as a function of input matrix density. Assuming 203 training applications have accumulated in the system over time, we show that, on average across target applications, rows only need to be approximately 20 to 30% dense for Selecta to achieve sufficient accuracy. This means that at steady state, users should profile training applications on about 20-30% of the candidate configurations (including reference configurations). Profiling additional configurations has diminishing returns.

Next, we consider a cold start situation in which a user wants to jump start the system by profiling a limited set of training applications across all candidate configurations. Figure 9b shows the number of training applications required to achieve desired accuracy. Here, for each target application testing round, we take the 203 training applications we have and randomly remove a fraction of the rows (training applications). We ensure to drop the row corresponding to the different dataset scale factor run of the target application, to ensure Selecta's accuracy does not depend on a training application directly related to the target application. Since the number of training applications required to achieve desirable accuracy depends on the size of the configuration space a user wishes to explore, the *x*-axis in Figure 9b represents the ratio of the number of training applications to the number of candidate configurations, *R*. We find that to jump start Selecta with dense training data from a cold start, users should provide 2.5× more training applications than the number of candidate configurations to achieve desirable accuracy. In our case, jump starting Selecta with more than $43 = \lceil 2.5 \times 17 \rceil$ training applications profiled on all 17 configurations reaches a point of diminishing returns.

Finally, we investigate whether, a cold start requires profile training applications on all configurations. We use *R*=2.5, which for 17 candidate configurations corresponds to using 43 training applications. Figure 9c plots accuracy as we vary the percentage of candidate configurations on which the training applications are profiled (including reference configurations, which we assume are always profiled). The figure shows that for a cold start, it is sufficient for users to profile the initial training applications on 40% to 60% of candidate configurations. As Selecta continues running and accumulates more training applications, the percentage of configura-



(a) Sensitivity to input matrix density in *steady state*: 20% density per row suffices for accurate predictions.

(b) Sensitivity to number of training applications, profiled on all configurations: 2.5× the number of configs suffices.

(c) Sensitivity to input matrix density for *cold start*: ∼50% density per row (training application) required.

Figure 9: Sensitivity analysis: accuracy as a function of input matrix density

tions users need to profile for training applications drops to 20-30% (this is the steady state result from Figure 9a).

We experimented with different reference configurations for Selecta. We find that accuracy is not very sensitive to the choice of references. We saw a slight benefit using references that have different VM and storage types. Although one reference configuration must remain fixed across all application runs since it is used to normalize performance, we found that the reference configuration used for the second profiling run could vary without significant impact on Selecta's accuracy.

5 Cloud Storage Insights

Our analysis of cloud configurations for data analytics reveals several insights for cloud storage configurations. We discuss key takeaways and their implications for future research on storage systems.

NVMe storage is performance and cost efficient for data analytics: We find that configurations with NVMe Flash tend to offer not only the best performance, but also, more surprisingly, the lowest cost. Although NVMe Flash is the most expensive type of storage per GB/hr, its high bandwidth allows applications to run significantly faster, reducing the overall job execution cost.

On average across applications, we observe that *l*-NVMe Flash reduces job completion time of applications by 27% compared to *r*-SSD and 75% compared to *r*-HDD. Although we did not consider *l*-SSD or *l*-HDD configurations in our evaluation, we validate that local versus remote access to HDD and SDD achieves similar performance since our instances have sufficient network bandwidth (up to 10 Gb/s) and modern networking adds little overhead on top of HDD and SSD access latency [8]. In contrast, a previous study of Spark applications by Ousterhout et al. concluded that optimizing or eliminating disk accesses can only reduce job completion time by a median of at most 19% [50]. We believe the main reason for the increased impact of storage on end-to-end application performance is due to the newer version of Spark we use in our study (v2.1.0 versus v1.2.1). Spark has evolved with numerous optimizations targeting CPU efficiency, such as cache-aware computations, code generation for expression evaluation, and serialization [17]. With ongoing work in optimizing the CPU cycles spent on data analytics computations, for example by optimizing the I/O processing path [66], we expect the choice of storage to be of even greater importance.

The need for flexible capacity and bandwidth allocation: Provisioning storage involves selecting the right capacity, bandwidth, and latency. Selecta uses statistics from Spark logs to determine capacity requirements and applies collaborative filtering to explore performance-cost trade-offs. However, the cost-efficiency of the storage configuration selected is limited by numerous constraints imposed by cloud providers. For example, for remote block storage volumes, the cloud provider imposes minimum capacity limits (e.g., 500 GB for *r*-HDD on AWS) and decides how data in the volume is mapped to physical devices, which directly affects storage throughput (e.g., HDD throughput is proportional to the number of spindles). A more important restriction is for local storage, such as *l*-NVMe, which is only available in fixed capacities attached to particular instance types. The fixed ratio between compute, memory and storage resources imposed by cloud vendors does not provide the right balance of resources for many of the applications we studied. For example the SQL equijoin query on two 64 GB tables saturates the IOPS of the 500 GB NVMe device on a `i3.xl` instance, but leaves half the capacity underutilized. Furthermore, local storage is ephemeral, meaning instances must be kept on to retain data on local devices. Thus, although we showed it is cost-efficient to store input/output and intermediate data on *l*-NVMe for the duration of a job, storing input/output files longer term on *l*-NVMe would dramatically increase cost compared to using remote storage volumes or an object storage system such as S3.

We make the case for a fast and flexible storage option in the cloud. Emerging trends in cloud computing, such as serverless computing offerings like AWS Lambda, Google Cloud Functions and Azure Functions, provide fine-grain, pay-per-use access to compute and memory resources [31, 7, 28, 46]. Currently, there is no option that allows for fine-grain capacity and bandwidth allocation of cloud storage with low latency and high bandwidth characteristics [41]. Although S3 provides pay-per-use storage with high scalability, high availability and relatively high bandwidth, we show that data analytics applications benefit from even higher throughput (i.e., NVMe Flash). S3 also incurs high latency, which we observed to be a major bottleneck for short-running SQL queries that read only a few megabytes of data.

Disaggregated NVMe is a promising option for fast and flexible cloud storage: Disaggregating NVMe Flash by enabling efficient access to the resource over the network is a promising option for fast and flexible cloud storage. Recent developments in hardware-assisted [49, 44] and software-only [40] techniques enable access to remote NVMe devices with low latency overheads over a wide range of network options, including commodity Ethernet networking with TCP/IP protocols. These techniques allow us to build disaggregated Flash storage that allows fine-grain capacity and IOPS allocation for analytics workloads and independent scaling of storage vs. compute resources. Applications would allocate capacity and bandwidth on demand from a large array of remotely accessible NVMe devices. In this setting, Selecta can help predict the right capacity and throughput requirements for each data stream in an analytics workload to guide the allocation of resources from a disaggregated Flash system.

There are several challenges in implementing flexible cloud storage based on disaggregated Flash. First, networking requirements can be high. Current NVMe devices on AWS achieve 500 MB/s to 4 GB/s sequential read bandwidth, depending on the capacity. Write throughput and random access bandwidth is also high. The networking infrastructure of cloud systems must be able to support a large number of instances accessing NVMe Flash remotely with the ability to burst to the maximum throughput of the storage devices. An additional challenge with sharing remote Flash devices is interference between read and write requests from different tenants [40, 61]. We observed several cases where separating input/output data and intermediate data on *r*-SSD (or S3) and *l*-NVMe, respectively, led to higher performance (and lower cost) than storing all data on *l*-NVMe. This occurred for jobs where large input data reads overlapped with large shuffle writes, such as for TPC-DS query 80 shown in Figure 1. A disaggregated Flash storage system must address interference using either scheduling approaches [40, 47, 61, 51, 60] or device-level isolation mechanisms [12, 54, 38]. Finally, the are interesting trade-offs in the interfaces used to expose disaggregated Flash (e.g., block storage, key-value storage, distributed file system, or other).

The need for end-to-end optimization: In our experiments, remote HDD storage performed poorly, despite its cost effectiveness for long-living input/output data and its ability to match the sequential bandwidth offered by SSD. Using the Linux `blktrace` tool [37] to analyze I/O requests at the block device layer, we found that although each Spark task reads/writes input/output data sequentially, streams from multiple tasks running on different cores interleave at the block device layer. Thus, the access stream seen by a remote HDD volume consists of approximately 60% random I/O operations, dramatically reducing performance compared to fully sequential I/O. This makes solutions with higher throughput for random accesses (e.g., using multiple HDDs devices or Flash storage) more appropriate for achieving high performance in data analytics. Increasing random I/O performance comes at a higher cost per unit time. In addition to building faster storage systems, we should attempt to optimize throughout the stack for sequential accesses when these accesses are available at the application level. Of course, there will always be workloads with intrinsically random access patterns that will not benefit from such optimizations.

6 Discussion

Our work focused on selecting storage configurations based on their performance and cost. Other important considerations include durability, availability, and consistency, particularly for long-term input/output data storage [42]. Developers may also prefer a particular storage API (e.g., POSIX files vs. object interface). Users can use these qualitative constraints to limit the storage space Selecta considers. Users may also choose different storage systems for high performance processing versus long term storage of important data.

Our study showed that separating input/output data and intermediate data uncovers a richer configuration space and allows for better customization of storage resources to the application requirements. We can further divide intermediate data into finer-grained streams such as shuffle data, broadcast data, and cached RDDs spilled from memory. Understanding the characteristics of these finer grain streams and how they should be mapped to storage options in the cloud may reveal further benefits.

Compression schemes offer an interesting trade-off between processing, networking, and storage requirements. In addition to compressing input/output files, systems like Spark allow compressing individual intermediate data streams using a variety of compression algorithms (lz4, lzf, and snappy) [64]. In future work, we plan to extend Selecta to consider compression options in addition to storage and instance configuration.

We used Selecta to optimize data analytics applications as they represent a common class of cloud workloads. Selecta's approach should be applicable to other data-intensive workloads too, as collaborative filtering does not make any specific assumptions about the application structure. In addition to considering other types of workloads, in future work, we will consider scenarios in which multiple workloads share cloud infrastructure. Delimitrou et al. have shown that collaborative filtering can classify application interference sensitivity (i.e., how much interference an application will cause to co-scheduled applications and how much interference it can tolerate itself) [22, 23]. We also believe Selecta's collaborative filtering approach can be extended to help configure isolation mechanisms that limit interference between workloads, particularly on shared storage devices like NVMe which exhibit dramatically different behavior as the read-write access patterns vary [40].

7 Related Work

Selecting cloud configurations: Several recent systems unearth near-optimal cloud configurations for target workloads. CherryPick uses Bayesian Optimization to build a performance model that is just accurate enough to distinguish near-optimal configurations [3]. Model input comes solely from profiling the target application across carefully selected configurations. Ernest predicts performance for different VM and cluster sizes, targeting machine learning analytics applications [69]. PARIS takes a hybrid online/offline approach, using random forests to predict application performance on various VM configurations based on features such as CPU utilization obtained from profiling [71]. These systems do not consider the vast storage configuration options in the cloud nor the heterogeneous data streams of analytics applications which can dramatically impact performance.

Resource allocation with collaborative filtering: Our approach for predicting performance is most similar to Quasar [23] and Paragon [22], which apply collaborative filtering to schedule incoming applications on shared clusters. ProteusTM [24] applies collaborative filtering to auto-tune a transactional memory system. While these systems consider resource heterogeneity, they focus on CPU and memory. While Selecta applies a similar modeling approach, our exploration of the cloud storage configuration space is novel and reveals important insights.

Automating storage configurations: Many previous systems provide storage configuration recommendations [9, 65, 2, 48, 4, 30, 39]. Our work analyzes the trade-offs between traditional block storage and object storage available in the cloud. We also considering how heterogeneous streams in data analytics applications should be mapped to heterogeneous storage options.

Analyzing performance of analytics frameworks: While previous studies analyze how CPU, memory, network and storage resources affect Spark performance [50, 68, 66, 43], our work is the first to evaluate the impact of new cloud storage options (e.g., NVMe Flash) and provide a tool to navigate the diverse storage configuration space.

Tuning application parameters: Previous work auto-tunes data analytics framework parameters such as the number of executors, JVM heap size, and compression schemes [32, 73, 72]. Our work is complementary. Users set application parameters and then run Selecta to obtain a near-optimal hardware configuration.

8 Conclusion

The large and increasing number of storage and compute options on cloud services makes configuring data analytics clusters for high performance and cost efficiency difficult. We presented Selecta, a tool that learns near-optimal configurations of compute and storage resources based on sparse training data collected across applications and candidate configurations. Requiring only two profiling runs of the target application, Selecta predicts near-optimal performance configurations with 94% probability and near-optimal cost configurations with 80% probability. Moreover, Selecta allowed us to analyze cloud storage options for data analytics and reveal important insights, including the cost benefits of NVMe Flash storage, the need for fine-gain allocation of storage capacity and bandwidth in the cloud, and the need for cross-layer storage optimizations. We believe that, as data-intensive workloads grow in complexity and cloud options for compute and storage increase, tools like Selecta will become increasingly useful for end users, systems researchers, and even cloud providers (e.g., for scheduling 'serverless' application code).

Acknowledgements

We thank our anonymous reviewers as well as Christina Delimitrou, Francisco Romero, and Neeraja Yadwadkar for their feedback. This work is supported by the Stanford Platform Lab, Samsung, Huawei and NSF grant CNS-1422088. Ana Klimovic is supported by a Stanford Graduate Fellowship and Microsoft Research PhD Fellowship.

References

[1] Agarwal, S., Kandula, S., Bruno, N., Wu, M.-C., Stoica, I., and Zhou, J. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (2012), NSDI'12, pp. 21–21.

[2] Albrecht, C., Merchant, A., Stokely, M., Waliji, M., Labelle, F., Coehlo, N., Shi, X., and Schrock, C. E. Janus: Optimal flash provisioning for cloud storage workloads. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (2013), USENIX ATC'13, pp. 91–102.

[3] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. CherryPick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), pp. 469–482.

[4] Alvarez, G. A., Borowsky, E., Go, S., Romer, T. H., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., and Wilkes, J. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. Comput. Syst. 19*, 4 (Nov. 2001), 483–518.

[5] Amazon. Amazon elastic block store (EBS). `https://aws.amazon.com/ebs`, 2017.

[6] Amazon. Amazon simple storage service. `https://aws.amazon.com/s3`, 2017.

[7] Amazon. AWS lambda. `https://aws.amazon.com/lambda`, 2017.

[8] Ananthanarayanan, G., Ghodsi, A., Shenker, S., and Stoica, I. Disk-locality in datacenter computing considered irrelevant. In *Proc. of USENIX Hot Topics in Operating Systems* (2011), HotOS'13, pp. 12–12.

[9] Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., and Veitch, A. Hippodrome: Running circles around storage administration. In *Proc. of the 1st USENIX Conference on File and Storage Technologies* (2002), FAST '02, USENIX Association.

[10] Bell, R., Koren, Y., and Volinsky, C. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2007), KDD '07, pp. 95–104.

[11] Bell, R. M., Koren, Y., and Volinsky, C. The BellKor 2008 Solution to the Netflix Prize. Tech. rep., 2008.

[12] Bjørling, M., Gonzalez, J., and Bonnet, P. Lightnvm: The linux open-channel SSD subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (2017), pp. 359–374.

[13] Bottou, L. *Large-Scale Machine Learning with Stochastic Gradient Descent*. Physica-Verlag HD, 2010, pp. 177–186.

[14] Cloudera. How-to: Tune your apache spark jobs. `https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/`, 2015.

[15] Cloudera. Tuning spark applications. `https://www.cloudera.com/documentation/enterprise/5-9-x/topics/admin_spark_tuning.html`, 2017.

[16] Corporation, I. Intel Optane SSD DC P4800X Available Now on IBM Cloud. `https://www.ibm.com/blogs/bluemix/2017/08/intel-optane-ssd-dc-p4800x-available-now-ibm-cloud`, 2017.

[17] Databricks. Project tungsten: Bringing apache spark closer to bare metal. `https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html`, 2015.

[18] Databricks. AWS configurations for Spark. `https://docs.databricks.com/user-guide/clusters/aws-config.html#ebs-volumes`, 2016.

[19] Databricks. Supported instance types. `https://databricks.com/product/pricing/instance-types`, 2016.

[20] Databricks. Accelerating workflows on databricks. `https://databricks.com/blog/2017/10/06/accelerating-r-workflows-on-databricks.html`, 2017.

[21] Databricks. Benchmarking big data sql platforms in the cloud. `https://databricks.com/blog/2017/07/12/benchmarking-big-data-sql-platforms-in-the-cloud.html`, 2017.

[22] Delimitrou, C., and Kozyrakis, C. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (2013), ASPLOS '13, pp. 77–88.

[23] Delimitrou, C., and Kozyrakis, C. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (2014), ASPLOS '14, pp. 127–144.

[24] Didona, D., Diegues, N., Kermarrec, A.-M., Guerraoui, R., Neves, R., and Romano, P. Proteustm: Abstraction meets performance in transactional memory. In *Proc. of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems* (2016), ASPLOS '16, pp. 757–771.

[25] Ferguson, A. D., Bodik, P., Kandula, S., Boutin, E., and Fonseca, R. Jockey: Guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems* (2012), EuroSys '12, pp. 99–112.

[26] Foundation, A. S. Apache parquet. `https://parquet.apache.org/`, 2014.

[27] Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., and Jacobsen, H.-A. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), SIGMOD '13, pp. 1197–1208.

[28] Google. Cloud functions. `https://cloud.google.com/functions`, 2017.

[29] Google. Google compute engine persistent disk. `https://cloud.google.com/persistent-disk`, 2017.

[30] Gulati, A., Shanmuganathan, G., Ahmad, I., Waldspurger, C., and Uysal, M. Pesto: Online storage performance management in virtualized datacenters. In *Proc. of the 2Nd ACM Symposium on Cloud Computing* (2011), SOCC '11, pp. 19:1–19:14.

[31] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)* (2016).

[32] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. Starfish: A self-tuning system for big data analytics. In *In CIDR* (2011), pp. 261–272.

[33] Hug, N. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

[34] Intel. Performance analysis tool (PAT). `https://github.com/intel-hadoop/PAT`, 2016.

[35] Intel. Intel optane technology. `https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html`, 2017.

[36] Jalaparti, V., Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Bridging the tenant-provider gap in cloud services. In *Proceedings of the Third ACM Symposium on Cloud Computing* (2012), SoCC '12, pp. 10:1–10:14.

[37] Jens Axboe, A. D. B., and Scott, N. blktrace man page. `https://linux.die.net/man/8/blktrace`, 2006.

[38] Kang, J.-U., Hyun, J., Maeng, H., and Cho, S. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems* (2014), HotStorage'14, pp. 13–13.

[39] Keeton, K., Santos, C., Beyer, D., Chase, J., and Wilkes, J. Designing for disasters. In *Proc. of the 3rd USENIX Conference on File and Storage Technologies* (2004), FAST '04, pp. 59–62.

[40] Klimovic, A., Litz, H., and Kozyrakis, C. ReFlex: Remote flash == local flash. *SIGPLAN Not. 52*, 4 (Apr. 2017), 345–359.

[41] Klimovic, A., Wang, Y., Kozyrakis, C., Stuedi, P., Pfefferle, J., and Trivedi, A. Understanding ephemeral storage for serverless analytics. In *Proc. of the USENIX Annual Technical Conference* (2018), ATC'18.

[42] Kovacs, G. EBS, EFS, or Amazon S3: which is the best cloud storage system for you? `https://cloud.netapp.com/blog/ebs-efs-amazons3-best-cloud-storage-system`, 2017.

[43] LI, H., GHODSI, A., ZAHARIA, M., SHENKER, S., AND STOICA, I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing* (2014), SOCC '14, pp. 6:1–6:15.

[44] METZ, J., HUFFMAN, A., SARDELLA, S., AND MINTRUN, D. The performance impact of NVM Express and NVM Express over Fabrics. `http://www.nvmexpress.org/wp-content/uploads/NVMe-Webcast-Slides-20141111-Final.pdf`, 2015.

[45] MICROSOFT. Azure files. `https://azure.microsoft.com/en-us/services/storage/files`, 2017.

[46] MICROSOFT. Azure functions. `https://azure.microsoft.com/en-us/services/functions`, 2018.

[47] NANAVATI, M., WIRES, J., AND WARFIELD, A. Decibel: Isolation and sharing in disaggregated rack-scale storage. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 17–33.

[48] NARAYANAN, D., THERESKA, E., DONNELLY, A., ELNIKETY, S., AND ROWSTRON, A. Migrating server storage to ssds: Analysis of tradeoffs. In *Proceedings of the 4th ACM European Conference on Computer Systems* (2009), EuroSys '09, pp. 145–158.

[49] NVM EXPRESS INC. NVM Express over Fabrics Revision 1.0 . `http://www.nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1_0_Gold_20160605.pdf`, 2016.

[50] OUSTERHOUT, K., RASTI, R., RATNASAMY, S., SHENKER, S., AND CHUN, B.-G. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (2015), NSDI'15, pp. 293–307.

[51] PARK, S., AND SHEN, K. FIOS: a fair, efficient flash I/O scheduler. In *Proc. of USENIX File and Storage Technologies* (2012), FAST'12, p. 13.

[52] PERFORMANCE IO RESEARCH GROUP AT IBM RESEARCH ZURICH, H. Example terasort program. `https://github.com/zrlio/crail-spark-terasort`, 2017.

[53] PERFORMANCE IO RESEARCH GROUP AT IBM RESEARCH ZURICH, H. Spark sql benchmarks. `https://github.com/zrlio/sql-benchmarks`, 2017.

[54] PETERSEN, C., AND HUFFMAN, A. Solving latency challenges with NVM express SSDs at scale. `https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_SIT6_Petersen.pdf`, 2017.

[55] POPESCU, A. D., ERCEGOVAC, V., BALMIN, A., BRANCO, M., AND AILAMAKI, A. Same Queries, Different Data: Can we Predict Query Performance? In *Proceedings of the 7th International Workshop on Self Managing Database Systems* (2012).

[56] RICCI, F., ROKACH, L., SHAPIRA, B., AND KANTOR, P. B. *Recommender Systems Handbook*, 1st ed. Springer-Verlag New York, Inc., 2010.

[57] SALAKHUTDINOV, R., AND MNIH, A. Probabilistic matrix factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems* (2007), NIPS'07, pp. 1257–1264.

[58] SAMMUT, C., AND WEBB, G. I., Eds. *Leave-One-Out Cross-Validation*. Springer US, 2010, pp. 600–601.

[59] SCHAD, J., DITTRICH, J., AND QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 460–471.

[60] SHEN, K., AND PARK, S. FlashFQ: A fair queueing I/O scheduler for flash-based SSDs. In *Proc. of USENIX Annual Technical Conference* (2013), ATC'13, USENIX, pp. 67–78.

[61] SHUE, D., AND FREEDMAN, M. J. From application requests to virtual IOPs: provisioned key-value storage with Libra. In *Proc. of European Conference on Computer Systems* (2014), EuroSys'14, pp. 17:1–17:14.

[62] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The Hadoop distributed file system. In *Proc. of IEEE Mass Storage Systems and Technologies* (2010), MSST '10, IEEE Computer Society, pp. 1–10.

[63] SPARK, A. Monitoring and instrumentation. `https://spark.apache.org/docs/latest/monitoring.html`, 2017.

[64] SPARK, A. Spark configuration. `https://spark.apache.org/docs/latest/configuration.html`, 2017.

[65] Strunk, J. D., Thereska, E., Faloutsos, C., and Ganger, G. R. Using utility to provision storage systems. In *6th USENIX Conference on File and Storage Technologies, FAST 2008, February 26-29, 2008, San Jose, CA, USA* (2008), pp. 313–328.

[66] Stuedi, P., Trivedi, A., Pfefferle, J., Stoica, R., Metzler, B., Ioannou, N., and Koltsidas, I. Crail: A high-performance i/o architecture for distributed data processing. *IEEE Data Eng. Bull. 40*, 1 (2017), 38–49.

[67] TPC, T. P. P. C. TPC-DS is a Decision Support Benchmark. `http://www.tpc.org/tpcds/`, 2017.

[68] Trivedi, A., Stuedi, P., Pfefferle, J., Stoica, R., Metzler, B., Koltsidas, I., and Ioannou, N. On the [ir]relevance of network performance for data processing. In *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing* (2016), HotCloud'16, pp. 126–131.

[69] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, 2016), pp. 363–378.

[70] Vitter, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1 (Mar. 1985), 37–57.

[71] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), SOCC'17, pp. 452–465.

[72] Yeh, C. C., Zhou, J., Chang, S. A., Lin, X. Y., Sun, Y., and Huang, S. K. Bigexplorer: A configuration recommendation system for big data platform. In *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)* (Nov 2016), pp. 228–234.

[73] Yigitbasi, N., Willke, T. L., Liao, G., and Epema, D. Towards machine learning-based autotuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems* (Aug 2013), pp. 11–20.

[74] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (2010), HotCloud'10, pp. 10–10.

[75] Zhou, P., Ruan, Z., Fang, Z., Shand, M., Roazen, D., and Cong, J. Doppio: I/o-aware performance analysis, modeling and optimization for in-memory computing framework.