Configuring Distributed Computations Using Response Surfaces

Adem Efe Gencer David Bindel Emin Gün Sirer Robbert van Renesse
Computer Science Department, Cornell University
{gencer, bindel, egs, rvr}@cs.cornell.edu

ABSTRACT

Configuring large distributed computations is a challenging task. Efficiently executing distributed computations requires configuration tuning based on careful examination of application and hardware properties. Considering the large number of parameters and impracticality of using trial and error in a production environment, programmers tend to make these decisions based on their experience and rules of thumb. Such configurations can lead to underutilized and costly clusters, and missed deadlines.

In this paper, we present a new methodology for determining desired hardware and software configuration parameters for distributed computations. The key insight behind this methodology is to build a *response surface* that captures how applications perform under different hardware and software configuration. Such a model can be built through iterated experiments using the real system, or, more efficiently, using a simulator. The resulting model can then generate recommendations for configuration parameters that are likely to yield the desired results even if they have not been tried either in simulation or in real-life. The process can be iterated to refine previous predictions and achieve better results.

We have implemented this methodology in a configuration recommendation system for MapReduce 2.0 applications. Performance measurements show that representative applications achieve up to 5× performance improvement when they use the recommended configuration parameters compared to the default ones.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); C.5.1 [**Computer System Implementation**]: Super (very large) computers; H.3.4 [**Systems and Software**]: Distributed systems

1. INTRODUCTION

Configuring distributed systems has become an increasingly difficult task with growing datacenter sizes and simultaneous growth in complexity of distributed applications. Not only do programmers have to decide on the optimal hardware configuration to execute a desired task, but they also need to configure the hundreds of software parameters that control the operation of modern distributed computation frameworks [15, 30, 36, 51, 60].

Naturally, the performance goals for modern distributed applications are quite varied, dependent on context, and subject to both budget and resource constraints. For instance, developers often face challenging questions such as:

- If I want to run this application (or set of applications), what are the best configuration parameters to minimize execution time or cost? Approximately how long will it take to run, and how much will it cost?
- What configuration parameter settings should I use to obtain high performance with my limited budget?
- If I were to spend X more dollars on hardware, how much would the running time be reduced?
- Should I spend money on additional cores, on more memory, higher disk speeds, or on a faster network?
- If I change the network topology from single rack to two-racks, how will that influence the execution time?

Finding a hardware and software configuration that best answers these questions is non-trivial. Typical state of the practice in industry is to employ rules-of-thumb, and to rely on past experience to guess at relevant configuration parameters. More recent academic work has examined techniques based on domain-specific analytical cost models [5, 24, 25, 26, 61, 62], hill climbing algorithms on customized frameworks [31], machine learning techniques [13, 14, 19, 29, 49], and genetic algorithms executed on the real application [32]. While these techniques are promising, they have not been widely deployed due to their inherent limitations. Analytical models are difficult to extract and brittle in practice, subject to becoming outdated any time the underlying hardware or software changes. Similarly, it is difficult to maintain approaches that modify the underlying computation frameworks. Existing machine learning techniques typically require extensive data sets for training and are subject to over-fitting. Configuration steering systems, which use performance data from real runs and perturb configurations using genetic algorithms to evolve the configuration to achieve desired goals, are limited by the running time of applications, which often render the process prohibitively slow.

In this paper, we introduce a novel methodology for determining hardware and software configurations for distributed applications. The key insight behind this methodology is to construct a *response surface methodology* (RSM) model [10, 34, 37, 41] that captures

Middleware '15, December 07-11, 2015, Vancouver, BC, Canada
 ISBN 978-1-4503-3618-5/15/12...$15.00.
DOI: http://dx.doi.org/10.1145/2814576.2814730

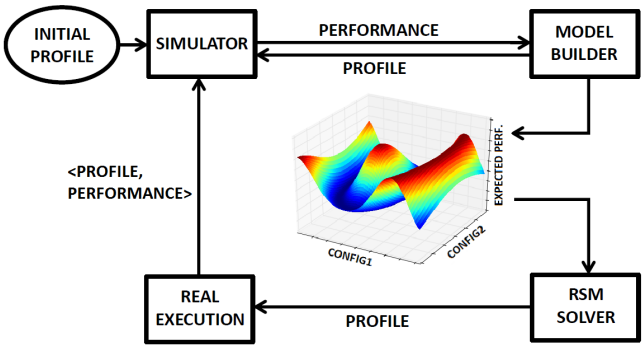


Figure 1: Workflow of the methodology for auto-tuning distributed configurations. Our methodology uses simulations to build a response surface that characterizes the performance response of applications as a function of configuration parameters. It then discovers the optimal combination of configuration parameters that achieve the user's goals. Measurements from the real system can be used to further refine the generated model.

the performance of the system as a function of the hardware and software configuration. Response surfaces are a well-studied technique in statistics for efficiently extracting and exploring the relationship between response variables, in this case, performance, and explanatory variables, such as the hardware resources, software parameters and workload descriptors. Most crucially, there exist efficient techniques for answering the types of resource- and cost-constrained questions of the type described above. A response surface model typically has many dozens of exploratory variables, ranging over a large space. In our case, these parameters include the hardware platform (including the CPU, RAM, disk speeds, and the like), the network (e.g. topology, link speeds), and the software configuration (e.g. block size, number of reduce tasks, sort factors).

A critical challenge is the amount of time required to construct a response surface that sufficiently characterizes the state space. Because of the large number of parameters and their extensive ranges, it would be infeasible to execute the application greedily in every possible configuration to determine the optimal choice. In fact, since modern distributed applications can take many hours, and sometimes days to execute, running the application is typically not feasible. Consequently, our approach relies on parametrizing response surfaces from simulations, which can provide the data required to populate a response surface faster than real executions.

Performance characterization, and finding the optimal configuration parameters, is particularly difficult when a user wants to execute not just a single application, but is faced with a batch of mixed applications. Our proposed methodology lends itself well to characterizing the interdependent performance characteristics of multiple applications executing concurrently on the same cluster.

We have instantiated this methodology in a configuration recommendation tool, called Troika, for MapReduce 2.0 applications [4]. Troika makes application, framework, and hardware-specific configuration recommendations within user-defined budget constraints. Figure 1 illustrates the basic workflow used in Troika. Troika comprises a *simulator* for MapReduce 2.0 applications, a *model builder*, and an *RSM solver*. The simulator is used to quickly estimate the execution time of a *profile*, which is a set of parameters that describe applications on a given hardware platform with a given software configuration. The model builder parses a specification of the profile space, prunes profiles that violate user constraints, and generates profiles for the simulator. Execution times obtained through simulation are used to construct a response surface that captures the various tradeoffs between profiles. Finally, the RSM solver examines the feasible envelope to determine the top few profiles that are estimated to yield the best performance. Note, critically, that these profiles may not have been executed or simulated in the past; in fact, the power of the response surface technique lies in the fact that it can capture and characterize such regions.

Troika is highly effective at discovering profiles that achieve desired user goals, such as optimizing execution time or minimizing cost. In particular, application of Troika to a set of MapReduce 2.0 benchmarks executing on Emulab [18, 56] yields performance improvements up to 5× for the same cost constraint.

Overall, this paper makes three contributions. (1) It introduces a technique based on response surfaces for effectively characterizing performance of applications as a function of their hardware and software configuration. This technique is system-agnostic, lightweight and easy to integrate into existing workflows. (2) It describes the implementation of this methodology for MapReduce 2.0, in a tool called Troika. Troika embodies a simulator that can make performance predictions within 5% to 6% of the actual execution times. The response surface technique enables a relatively small number of such simulated runs to estimate the execution time of applications in a wide range of configurations. (3) It evaluates the performance of Troika on a representative set of benchmarks, and compares it to Starfish, another configuration recommendation system.

2. RELATED WORK

Past work has suggested automating the process of configuring distributed computations. This work can be divided into the following categories based on the underlying methodologies.

Domain-Specific Analytical Cost Models: Starfish [5, 24, 25, 26] proposes a model-based optimization for configuring MapReduce applications. Once such a model exists, this approach requires greedy enumeration (gridding) or recursive random search [59] to find the optimal configuration parameters. Other studies [61, 62] present an approach that assists in the selection of virtual machine instances to run MapReduce applications on Amazon EC2 [1] platform. This approach uses a bounds-based analytical cost model [52] and a simple simulator for evaluating scheduling decisions. However, such analytical models are generally limited in scope, and difficult to produce for complex applications. Any change in the application or underlying framework can invalidate the model and require re-developing an analytical performance model from scratch. It is not always feasible to capture the performance characteristics of an application in an analytical model.

Iterative Improvement Search: Gunther [32] employs a search-based approach for tuning MapReduce configurations. It relies on running the system to test new configurations one at a time. It employs a genetic algorithm to enumerate configurations that will lead towards a good configuration. The number of tests on the real system is limited by the execution time of applications. In contrast, mrOnline [31] employs a gray-box based smart hill climbing algorithm for testing multiple configurations per real run to reduce the

tuning time. It achieves this by modifying the underlying framework for configuring individual tasks of an application. Moreover, it uses a knowledge base to aid the tuning process. Therefore, it faces the same issue as Starfish: changes in the underlying platform requires updates to the system. Another study uses a black-box based version of the smart hill climbing algorithm for auto-tuning web application servers [58]. However, other research [47] has shown that a CMA evolution strategy [23] can outperform this algorithm.

Machine Learning Models: Machine learning approaches [13, 14, 19, 29, 49] employ various algorithms such as KCCA [6], artificial neural networks, decision trees, reinforcement learning, and Bayesian networks [39] to determine a correlation between configuration parameters and performance. The principal obstacle to the widespread adoption of machine learning techniques is the difficulty of the model building process. To yield an accurate model, these techniques typically require a large amount of training data obtained through execution of applications on real platforms.

Simulation: Simulation is another methodology that people use for configuration tuning. We analyzed several MapReduce [15] simulators in the literature. MRPerf [54] is among the first MapReduce simulators. It is particularly good at evaluating different network topologies due to its integration with ns-2 [40]. However, MRPerf omits simulation of task execution details, such as computation-I/O overlap and shuffle phase. As a result, it is difficult to observe the effects of varying parameters for related configurations. HSim [33] focuses on task simulation of MapReduce applications rather than the effect of network topologies on cluster performance. Similar to MRPerf, it assumes that compute requirements of tasks depend only on the input size, but not on the content. Our evaluations demonstrate that this assumption is unrealistic. MRSim [22] and SimMapReduce [50] both work on SimJava [48] and use GridSim [11] for network simulation. MRSim has been evaluated with only one application and it is not clear if the results scale. SimMapReduce suffers from oversimplification because it assumes that all the instructions have an equal cost, and the task computation time is independent of the input content. While simulations are useful on their own, using them for configuration tuning requires a sophisticated logic that can endure approximation errors in performance estimates.

Simulators are also used for testing new features that aim to improve the underlying computation platform itself rather than tuning its parameters. For instance, popularity-based data replication system Scarlett [2] and outlier mitigation scheme Mantri [3] use a trace-driven Dryad [30] simulator for performance evaluations. Mumak [35] is a discrete event simulator for MapReduce that is designed to aid researchers in evaluation of new resource allocation and scheduling algorithms, and scalability analysis. It reproduces the execution time behavior of applications using traces collected through a log processing tool, Rumen [46]. SimMR [53] and Wax-Elephant [45] also simulate MapReduce workloads, which improve Mumak's accuracy by modelling sort and shuffle phases. These simulators focus on workload rather than application-specific simulations.

Resource Management Techniques: Paragon [16] is a greedy online datacenter scheduler that uses collaborative filtering for resource assignment. It classifies incoming applications considering their performance on different servers and contention on shared resources. Quasar [17] performs resource assignment and allocation by extending Paragon's classification technique with the performance impact of scale-out and scale-up. To handle the cold start problem, both systems require offline training by profiling selected applications on different types of servers in the cluster. PACORA [9] performs resource allocation using convex analytic expressions to model the execution times. However, it requires identification of application goals and ignores non-convex execution time behavior. These systems aim to improve the utilization of existing resources, and they are complementary to auto-tuning distributed configurations.

Our response surface–based methodology supports optimization of software as well as hardware configurations. Since it is lightweight and not domain-specific, the methodology is easily applicable to a wide set of modern distributed frameworks. Moreover, it works without requiring an extensive amount of training data.

3. DESIGN

The key components of the methodology are a *model builder*, a *simulator*, and an *RSM Solver*.

3.1 Model Builder

The core technique that our methodology employs for the efficient analysis of a large number of alternative profiles is based on building a response surface model. This model is a combination of statistical and mathematical procedures that is particularly useful for process optimizations. The model builder creates this model for mapping a large set of configuration parameters to the performance of a distributed computation. This process requires a number of samples of the following form:

$$Sample = \langle Profile, Performance \rangle \tag{1}$$

The model builder generates the model in four steps.

1. It preprocesses the profile space to prune the profiles that violate user constraints. This diverse set of constraints may include budget thresholds, SLA, and minimum resource requirements. Identification of important profile parameters is the responsibility of the user.
2. It picks a number of randomly chosen profiles among the alternatives passed through the pruning process. The number of profiles is determined by $\alpha \times \mu$, where α represents the number of parameters in the chosen response surface model, and μ represents *recommendation accuracy*. We define recommendation accuracy to be the number of samples per profile parameter. Exploring different choices of experimental designs is a subject of future work.
3. It obtains the estimated performance of selected profiles from the simulator to create samples.
4. It builds a response surface model using the samples.

The application of response surface methodology requires building an approximate model to map the behavior of the actual response. Several configuration parameters influence this response and the interplay between them can be complex. A response surface needs to be sufficiently detailed to capture this interplay, but higher detail incurs a cost in the amount of additional simulations necessary to yield the polynomial points. Usually, first- or second-order polynomials are used to model this interplay [37].

The general form of a first-order model is as follows.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k \tag{2}$$

A second-order model can capture more detail through additional model variables.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + \beta_{1,1} x_1^2 + \beta_{2,2} x_2^2 + ... + \beta_{k,k} x_k^2 + \beta_{1,2} x_1 x_2 + ... + \beta_{k-1,k} x_{k-1} x_k \quad (3)$$

Notice that α is $k+1$ in the first-order, and $1+2k+k(k-1)/2$ in the second-order model. As a result, for the same value of μ, building a second-order model requires more samples to be collected through simulations. Moreover, the building process of a second-order model requires at least three distinct values per profile parameter in the pruned profile space. Nonetheless, the second-order model is more flexible, and its higher model detail may improve recommendation performance in some cases.

3.2 Simulator

A response surface model can be built using results running a set of profiles on the target cluster. However, this approach is often problematic. Running experiments for real is costly and takes time. Nor does this approach allow running experiments on hypothetical resources. These problems can be overcome by using a high quality simulator. Simulation enables considering a wide set of sample profiles and using any collection of hypothetical resources. Because of this, our methodology uses simulations to create a response surface that represents the performance of distributed computations as a function of configuration parameters.

Simulators are versatile tools in the sense that they can be used for optimization of various performance goals. One typical goal is to minimize the running time, which requires a simulator to deliver estimated execution times. Depending on the desirable performance goals, simulators can also be designed to provide performance data in the form of average task time (e.g. average map time), standard deviation of task times, energy consumption, or resource utilization (e.g. average bandwidth usage).

3.3 RSM Solver

The RSM solver takes a response surface as input, and outputs a profile that is estimated to yield the best performance based on the metric calculated by the simulator.

RSM solver generates this profile in three steps.

1. It uses the response surface to estimate the performance values for the profiles in the processed profile space.
2. It determines the profiles that fall within the feasible envelope. This envelope contains a few profiles that exhibit the best estimated performance.
3. It gets performance estimations for these profiles from the simulator and outputs the profile with the best simulation performance.

Optionally, once the recommended profile is executed in the target environment, its performance outcome can be used to refine the generated response surface by improving the quality of simulations.

3.4 Design Summary

Our methodology employs a statistical approach to approximate the performance behavior of computations. It is applicable to a broad range of distributed frameworks because the approach itself is system-agnostic. It is practical since it works without the need for an extensive number of samples. Moreover, the samples are generated through simulations that are generally much faster than executions. The underlying response surface–based method has a proven track record in areas ranging from analytical chemistry [8] to mechanical engineering [42].

4. TROIKA

We implemented the described methodology in a tool called Troika. This tool comprises a *simulator*, and a *recommendation engine* that embodies the model builder and the RSM solver. Popularity, presence of a representative set of benchmarks, and open-source availability were instrumental in our framework selection.

4.1 MapReduce 2.0 Simulator

Troika achieves its efficiency using a discrete event simulator to estimate the performance of different profiles. It is essential that the simulator provides accurate estimates for the recommendations to be useful.

We made three key design decisions to obtain a highly accurate simulator: (1) In order to capture how resources impact overall performance, our simulator overlaps computation with I/O operations as in the real execution of applications. (2) The simulator considers each input record separately to achieve fine granularity. (3) The simulator considers not only the input size, but also the input content to assess the computational requirements of tasks.

Troika accurately captures the entire MapReduce 2.0 workflow. This includes overlapping of I/O with processing; pipelining of input splits; buffering; contention for disk, CPU, and network; ResourceManager arbitration; scheduling for kernel I/O and NIC; execution time of each record; and the access delay for disk and network I/O operations. This captures the elements such as its slotless design, non-centralized resource management that distinguish MapReduce 2.0 from the classic MapReduce.

To realize such an accurate simulator, we had to pay attention to the following design considerations.

First of all, capturing the many sources of concurrency in MapReduce 2.0 is critical for accuracy. For example, MapReduce applications contain frequent disk accesses as well as data processing. These operations occur concurrently within the same task, among different tasks of the same application, as well as between different applications within the same framework. Similarly, the simulator captures the concurrency in network data transfers. Omitting this concurrency for the sake of a simplified design would lead to a loss of simulation accuracy [55]. Our implementation takes these interactions into account.

Second, simulation granularity plays a critical role in accurate performance estimation. A MapReduce application flow contains successive execution of map and reduce tasks. These tasks process the input data on a record-by-record basis where each record is a key-value pair. This fine granularity impacts several parts of the system, from buffer usage patterns to data processing frequency. An accurate simulation should handle these details in order to realize the impact of changing configuration parameters such as the buffer capacity. Therefore, Troika's simulator performs operations on each input record individually, rather than the whole input data.

Finally, accurate simulation demands that the simulator takes the amount of execution time required per record into account. Our

Parameters	Examples
Application Properties	input size, container resources per task and application master, queue IDs, task intensities, task and final output ratios, ...
Hardware Properties	cores per CPU, disk read and write speed, core and memory capacities, switches
Cluster Topology	placement of cluster components and file splits, link bandwidth
MapReduce 2.0 Configurations	number of reduce tasks, shuffle merge percent, dfs block size, scheduler configurations, minimum and maximum chunk size, task I/O sort factor, sort buffer memory size, sort spill percent, ...

Table 1: A *profile* for Troika contains all parameters that are necessary to do an accurate simulation of a MapReduce application (or set of applications) on a cluster.

simulator captures this with the aid of parameters called *task intensities*. A task intensity specifies the average computational weight of a single byte for each task. Our simulator keeps track of separate task intensities for map, reduce, and combiner functions. Further, the simulator allows modeling non-linear relationship between input size and execution times. For instance, in TeraSort, the correspondence between execution time and input size is non-linear due to the sorting algorithm that it employs.

4.1.1 Profile

The simulator takes as input a profile, which represents a configured cluster with MapReduce 2.0 applications. The profile contains: *application properties* that identify characteristics of target applications, *hardware properties* that identify the cluster components, *cluster topology* that describes how the cluster components are networked, and the *MapReduce 2.0 configuration* that includes properties that affect the behavior of the framework. Table 1 shows some parameters that constitute a profile in Troika. To support rack heterogeneity, the profile enables the properties of each server and network link to be specified independently.

Initial Profile Wizard: Troika uses a tool called *Initial Profile Wizard* (IPW) to facilitate collecting parameters necessary for the simulation. IPW runs on a single server on the target cluster to produce the *initial profile*. Optionally, the simulator can consider the resulting statistics from real runs to improve the profile generated by IPW. The IPW consists of the *Measurer* and the *Parser*.

Measurer: This subcomponent gathers hardware-related properties using a set of command line utilities for Linux. IPW runs some commands on each server locally, collecting the measured values at the IPW server—other commands run directly at the IPW server.

To measure disk read/write speeds, the measurer runs `hdparm` and `dd` on each server. Similarly, it uses `free` to get the available physical memories. The information in `/proc/cpuinfo` contains the number of CPU cores at each server. Bandwidth between the IPW server and the others are measured using `nttcp` in the IPW server.

Parser: Through parsing framework configuration files and input file properties, the parser collects application and framework-specific data. It does not perform profiling, and thus no information regarding the runtime behavior of applications can be collected.

To fetch the framework configurations, the parser reads `mapred-site.xml`, `hdfs-site.xml`, and `yarn-site.xml` configuration files. It executes `hdfs dfs -du` command to measure application input size.

In a MapReduce cluster, configuration values could override each other depending on where they are specified. For instance, the number of reduce tasks could be set within `mapred-site.xml` file. The value in this file can be overridden with a command line argument. Moreover, setting its value within the application code overrides any others. The parser takes this into account just for the `mapreduce.job.reduces` property.

4.1.2 Resource Queues and Scheduling

The simulator models the MapReduce 2.0 queues and corresponding schedulers for resource requests and adjusts available resource capacities accordingly.

For many Linux distributions, CFQ (Complete Fairness Queueing) [12] is the default I/O scheduler. It regulates disk accesses with the aim of providing fair I/O throughput among threads. We implemented a version of the CFQ scheduler to simulate disk accesses. In our implementation, each request accesses the disk either until its time slice expires or it finishes the corresponding I/O operation.

Traffic schedulers determine the packet that will go through the network interface. Pfifo_fast [43] is the commonly used default traffic scheduler in Linux distributions. The simulator employs a version of this scheduler with two bands. Our model gives higher priority to small packets such as requests for file transfers. While our implementation requires knowledge of the network topology, it lets the simulator take network congestion into account.

The simulator does not contain a process scheduler. Instead, it keeps track of the number of active threads and adjusts processing speed of new requests based on this value. The processing speed decreases as the number of active threads increase. However, a new thread does not affect the processing speed of currently active ones.

In general, Troika assumes that all cluster resources are used only by the applications stated in its input. It supports the default scheduler of Apache Hadoop YARN [4], Capacity Scheduler [20], to model concurrently running applications. The simulator introduces resource access delays, and varies the simulated speed of I/O operations between measured maximum and minimum disk performance. These variations in access delays and disk speed are modeled with uniform, constant, or exponential distribution. Such fluctuations between repeated simulations enable Troika to capture the performance impact of diverse execution states, such as collocation of reduce tasks.

Simulation on a Cloud Environment: Simulation of applications deployed on a virtualized cluster involves additional challenges. State-of-the-art public cloud providers [1] do not enforce a quantifiable limit on the variance of resource capacities on shared servers. Unless cluster instances are dedicated, resource queues might grow non-deterministically, which may lead to a loss of predictability. Moreover, the placement of cluster components is not revealed to tenants, which makes it challenging to simulate network congestion. Despite these issues, minimum response guarantees provided by cloud operators may enable the simulator to derive appropriate predictions. Exploring the applicability of the simulator for applications running on cloud platforms is a subject of future work.

4.2 Recommendation Engine

The recommendation engine comprises of a model builder and an RSM solver. It aims to find a near-optimal profile by means of response surfaces. It targets a good tradeoff between recommendation optimality and the time spent in the recommendation process.

Model Builder: This component takes profile space and user constraints as its input. User constraints include the cost of each profile in the profile space, resource constraints, and a budget limit. Profile cost indicates the price of server and network components.

The model builder parses the profile space and prunes the profiles that violate user constraints based on two criteria. First, using the provided network topology information, it calculates the number of servers and network links and eliminates profiles that are over the user budget. Second, it eliminates profiles that do not have sufficient resources to execute the target applications. Currently, the resource pruning process considers only memory-based constraints for this.

To obtain performance estimates, the model builder sends a number of profiles that passed through the pruning process to the simulator. Troika's model builder provides a user-configurable parameter, recommendation accuracy, to control the number of such calls. To ensure an accurate recommendation, our implementation enforces a threshold for this parameter. Once a sufficient number of samples are collected, Troika creates a response surface that maps profiles to the overall execution times.

RSM Model: Troika supports both a first- and a second-order RSM model to map configuration parameters to estimated performance. Eq. 2 and Eq. 3 describe the correlation between k configuration parameters and performance. Troika uses the least squares method to compute the coefficients of these models.

The model in Eq. 2 is also called the *main effects model*, because it assumes independence between different parameters by considering only the linear main effects. A response surface with the model in Eq. 3 can capture curvatures and parameter interactions. In some cases a linear model might be sufficient for modeling performance of certain MapReduce 2.0 applications.

RSM Solver: This component uses the response surface and the simulator to generate the profile with the fastest estimated execution time. The recommendation engine configuration enables customization of the feasible envelope size, which determines the number of profiles to be sent to the simulator for performance estimation. The process details are described in Sec. 3.3

Upon determination of a profile, the recommendation engine creates a report containing the estimated overall execution time and the expense of building a cluster with this profile. For profiles with a single application, the report presents details such as average task completion, shuffle, and the total map time (the elapsed time from the initiation of the first map task to the completion of the last one). Although Troika's performance metric is to minimize the overall execution time, it is easy to configure it to provide optimizations for any of the reported values.

5. EVALUATION

In this section, we evaluate the accuracy of Troika's simulator for common workloads and understand the impact of fine-grained simulation compared to the simplifying assumptions of the simulators presented in Sec. 2. We then evaluate recommendation quality, and analyze the tradeoffs between running time and quality. Next we compare the efficacy of Troika with Starfish [5, 24, 25, 26], a recommendation system that employs a domain-specific model-based methodology for configuring MapReduce applications. Then, we evaluate the quality of hardware recommendations and examine the impact of budget constraints in determining what hardware to invest in. Finally, we analyze the fit for first- and second- order models with different MapReduce applications.

Topology	Double Rack (2x3 servers)
CPU/server	Xeon E5530 (4x2.4 GHz cores)
Disk/server	500GB SATA
RAM/server	12GB DDR2 at 1066MHz
Bandwidth/link	1 Gbps

Table 2: Experimental cluster configuration. Each rack is connected to a top-of-rack switch and single aggregation switch connects the racks.

We created a MapReduce 2.0 cluster with six servers on the Emulab testbed [18]. Emulab provides full control over the network. We arranged the cluster components using the common ToR architecture. Table 2 summarizes the configuration details of our test environment that we used for evaluation of simulation accuracy, efficacy of recommendations, comparison with Starfish, and analysis of the fit for response surfaces. We executed Troika itself on a 64-bit local machine with 16GB DDR2 RAM and Intel Core i5-3320M @2.60GHz processor.

5.1 Accuracy of Simulations

In our experiments, we used TeraSort [21], WordCount, PI[1], and Grep to validate performance predictions of Troika's simulator. We chose these applications because they cover the usage from data- to compute-intensive MapReduce applications.

To examine accuracy of the simulator for a particular application, we chose a set of profile parameters such as the size of application inputs, the number of reduce tasks, and task intensities. We then specified for each parameter a set of values that yielded an execution time no more than an hour in actual runs. By choosing randomly from those values, we created 15 experiments for TeraSort, 20 for WordCount and Grep, and 5 for PI.

We define the *simulation accuracy* as follows:

$$\frac{\min(EstimatedTime, RealTime)}{\max(EstimatedTime, RealTime)} \quad (4)$$

Simulation accuracy is a number in the range $[0-1]$, with 0 meaning completely inaccurate and 1 meaning completely accurate. *EstimatedTime* represents the execution time that our simulator generates and *RealTime* represent the execution time on the target cluster. We repeated each experiment five times on both the real cluster and on our simulator and used the average execution times to calculate the simulation accuracy of each experiment.

[1]PI approximates π using a Quasi-Monte Carlo method. The user provides a parameter that controls accuracy—the parameter also impacts task intensities for the simulator.

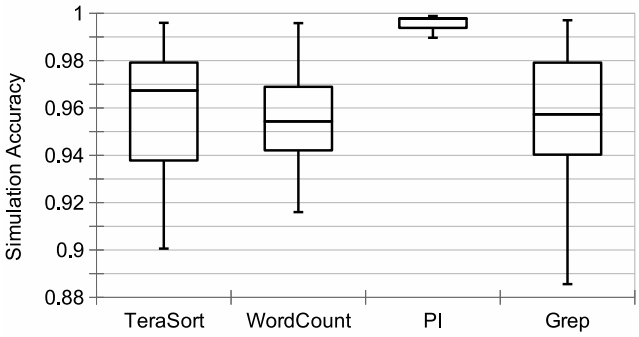


Figure 2: The accuracy of Troika's simulator. Error bars represent the maximum and the minimum simulation accuracies.

Figure 2 presents the simulation accuracy of Troika's simulator for different applications. The results show that Troika's simulator can accurately capture the performance impact of different profile parameters. Examining in more detail where time was spent, we believe that the slight inaccuracies we observed stem mainly from the variation in network latencies.

Figure 3 shows the amount of time it took to run a single simulation compared to running the actual application. The results show that simulations provide performance data orders of magnitude faster than real executions. Variance in execution time of the simulator is mainly due to the difference in the number of records caused by the changes in the application input size.

Troika's simulator reflects variations in the compute requirements through task intensities. These intensities may change due to varying input size and content. Figure 4 shows the impact of reflecting this change on the simulation accuracy of TeraSort. In calibrated simulations, we experimentally determined task intensities to match real execution times. In fixed simulations, we set task intensity values for a 1GB input file and preserved their values for varying input sizes. The results show that adjusting task intensities significantly improves the ability of the simulator to estimate execution times.

Troika provides an "intensity optimizer tool" to determine task intensities automatically from measurements of real runs of applications.

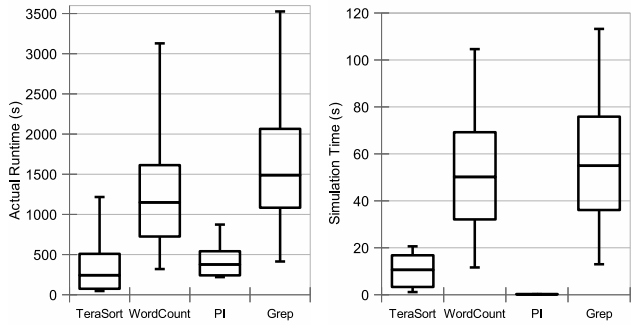


Figure 3: Comparison of actual execution and simulation times. The error bars represent the maximum and the minimum times.

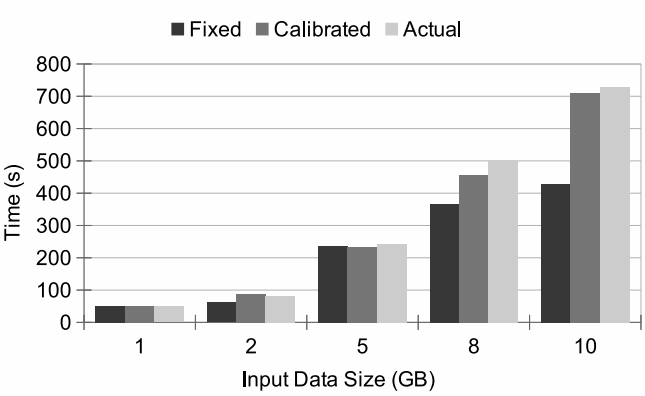


Figure 4: TeraSort simulation with and without calibrated task intensities (reduce task count: 2).

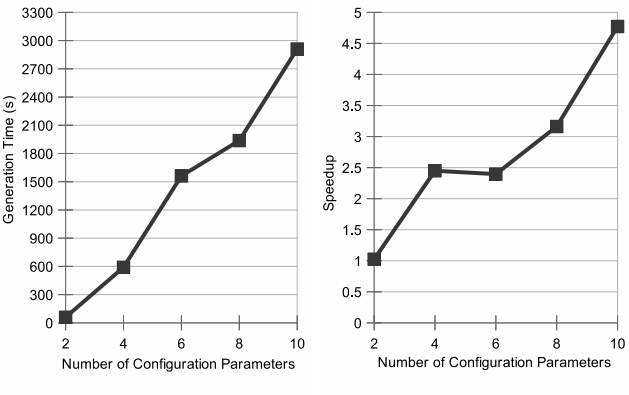
Profile Parameter	Default	Troika
mapreduce.job.reduces	1	3
yarn.nodemanager.resource.memory-mb	8,192	11,264
dfs.blocksize	128MB	64MB
mapreduce.input.fileinputformat.split.minsize	0	64MB
mapreduce.job.reduce.slowstart.completedmaps	0.05	0.20
mapreduce.map.sort.spill.percent	0.80	0.75
mapreduce.task.io.sort.factor	10	11
mapreduce.task.io.sort.mb	100	108
mapreduce.reduce.shuffle.merge.percent	0.66	0.68
mapreduce.reduce.shuffle.input.buffer.percent	0.70	0.72

Table 3: Default vs. Troika's recommended profile parameters with the first-order response surface model for TeraSort.

5.2 Efficacy of Troika Recommendations

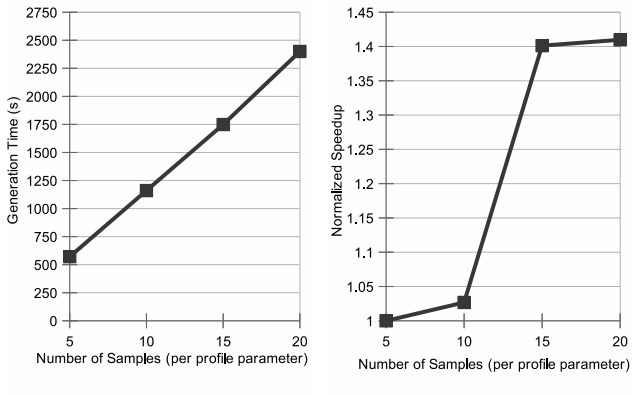
To evaluate the quality of Troika's recommendations, we used kMeans and Bayesian from the HiBench Benchmark Suite 3.0 [27, 28] in addition to the four applications listed in Sec. 5.1 and Word Co-occurrence. We executed each application five times using the default framework parameters of MapReduce 2.0 and with the ones that Troika recommended with a first- and a second-order model. Due to space constraints, we focus on the experimental results and the analysis of TeraSort using Troika with a first-order response surface model. In both executions, TeraSort used a 10 GB input file generated by TeraGen application. Table 3 lists the considered profile parameters together with the corresponding default and recommended values.

When the application runs with the default configuration values, on average it takes 1217 seconds to complete. Troika's recommended parameters lower the execution time to 255 seconds, or 4.8× faster. The response surface–based methodology lets Troika boost the performance by modifying different aspects of the execution flow that would be hard to identify without expertise on MapReduce 2.0. The recommended parameters in Table 3 increase the number of reduce tasks and the amount of memory allocated for containers while halving the filesplit size. As a result, the number of concurrently running tasks in the cluster increases. Moreover, the recommended configuration parameters delay the scheduling of reduce tasks. This helps in steering more resources to map tasks, and maintaining a low network traffic by delaying the shuffle phase. These parameters also modify the flow of sort, shuffle, and merge operations to speed up the overall execution.



(a) (b)

Figure 5: The impact of configuration space dimensionality on recommendation generation time (a), and the speedup obtained over using the default profile parameters (b).



(a) (b)

Figure 6: The impact of recommendation accuracy on recommendation generation time (a), and the speedup obtained over using the default profile parameters, normalized to the speedup obtained with five samples (b).

5.2.1 Impact of High Dimensionality

Due to the *curse of dimensionality* [7], the number of candidate profiles for recommendation tends to grow rapidly for each additional profile parameter listed in Table 1. This increases the number of samples needed to build an accurate model (see Sec. 3.1). Since sample generation is a time-consuming operation, the overall recommendation process can potentially take more time to complete.

To examine the impact of high dimensionality on recommendation generation time and obtained speedup, we ran an experiment starting with the profile parameters in Table 3 for optimizing TeraSort using Troika with the first-order response surface model. At each step, we measured the recommendation generation time, and the corresponding speedup over using the default parameter values in the configuration of the application. Before beginning the next step, we randomly eliminated two parameters and measured the performance of recommendations for the remaining ones. Figure 5(a) shows that increase in dimensionality slows down recommendation generation, but even with 10 parameters the recommendation generation time is acceptable. Although the model is first-order, the correlation between number of parameters and recommendation generation time is non-linear because the load of simulating the changes in one profile parameter differs from simulating the changes in another. Figure 5(b) shows that being able to analyze such high dimensionality enables Troika to make better recommendations. Since the expected performance contribution due to each parameter is unknown beforehand, the rate of increase in speedup is subject to uncertainty.

5.2.2 Impact of Recommendation Accuracy

A higher recommendation accuracy requires more samples to produce a model (see Sec. 3.1), and obviously it takes more time for model generation. Model-based methodologies require a certain number of samples per profile parameter to avoid over-fitting. In general, a higher number of samples used in developing a model increases the accuracy of predictions [38].

For our experiments, we selected four configuration parameters[2]: (1) `mapreduce.map.sort.spill.percent`: the ratio of the map task output buffer at which the buffer is flushed to disk (2) `yarn.node manager.resource.memory-mb:` the amount of physical memory per container. (3) `mapreduce.job.reduce.slowstart.completedmaps`: required completion ratio of map tasks before launching reduce tasks. (4) `mapreduce.job.reduces`: the number of reduce tasks.

We evaluated the effect of different recommendation accuracies on Troika's execution time with first-order response surface model and analyzed the recommendation performance on the specified cluster in Table 2. We executed each experiment five times. Figure 6(a) shows that the execution time of Troika is linearly proportional to the number of samples. This is expected because the recommendation engine generates each additional sample using the simulator, and each simulation takes approximately the same time for the selected configuration parameters.

Higher accuracy tends to yield better configuration recommendations. Figure 6(b) shows the speedup as a function of recommendation accuracy normalized for the speedup we obtain with 5 samples. Clearly, the gain in application performance is not proportional to the recommendation generation time. Contrary to increasing the number of configuration parameters to broaden the scope of optimization space, recommendation accuracy targets fine-tuning of small set of configuration parameters.

5.2.3 Recommendations with Multiple Applications

In many cases MapReduce clusters need to run multiple applications simultaneously. Troika can use simulations of concurrently running applications to make recommendations for these cases. To evaluate the quality of such recommendations, we performed three experiments with Troika using the first-order model. In our setup, we used a workload consisting of four TeraSort and four PI applications. Each TeraSort application used a separate 10GB input generated by TeraGen application. PI applications are configured to run with 1000 map tasks and 400,000,000 samples. We repeated each experiment five times.

[2]The selection is among the most frequently tuned configuration parameters of MapReduce 2.0.

Profile Parameter	One Pool	Two Pools
mapreduce.job.reduces	3	3
yarn.nodemanager.resource.memory-mb	8,192	11,264
dfs.blocksize	256MB	256MB
mapreduce.input.fileinputformat.split.minsize	64MB	32MB
mapreduce.job.reduce.slowstart.completedmaps	0.25	0.15
mapreduce.map.sort.spill.percent	0.75	0.75
mapreduce.task.io.sort.factor	11	11
mapreduce.task.io.sort.mb	104	106
mapreduce.reduce.shuffle.merge.percent	0.70	0.68
mapreduce.reduce.shuffle.input.buffer.percent	0.72	0.73
capacity of each resource pool	100%	50%

Table 4: Recommended profile parameters with a mix of concurrently running applications for single and two resource pools.

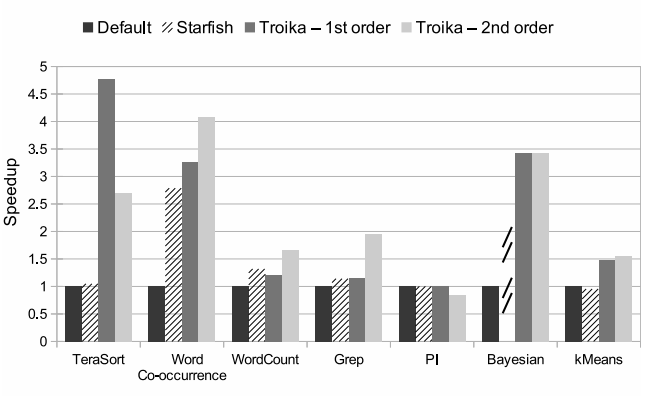


Figure 7: Speedup achieved by Starfish and Troika using first- and second-order models for various data- and compute-intensive applications.

The first experiment measured the accuracy of simulations with concurrently running applications using the default profile parameters with a single *resource pool*. Resources among applications are shared through pools. Our evaluation shows that Troika's simulator can predict the behavior of concurrent applications with a high accuracy. The simulation accuracy is within 98%.

In the second experiment we kept the scheduler properties the same and executed Troika with a profile space containing the parameters listed in Table 3. The second column in Table 4 presents the recommended parameters that yield a 2.4% speedup over the default parameters.

In the final experiment, we partitioned the cluster into two resource pools and measured the performance under the default and recommended configuration parameters shown in Table 4. The recommended parameters yield a 22.2% performance improvement in this case.

5.2.4 Efficacy Analysis

As our evaluations show, in some cases Troika's recommendation generation time exceeds the real execution time on the cluster. This is partially due to the sequential implementation of Troika. However, there is sufficient room for parallel simulation of the sample profiles. Exploring the performance impact of parallelization is a subject of the future work. Nonetheless, using Troika comes with the following benefits:

- MapReduce 2.0 applications are not generally designed as one-time-use applications. Assuming that the fluctuations in task intensities are small, the benefits of performance gain become increasingly significant in the long term.
- Testing hardware-related configuration changes with trial and error would generally be too costly. Troika performs such tests on hypothetical hardware. Therefore, it is capable of making optimizations that may not be practical otherwise.
- Contrary to prior systems [26, 31, 32, 58, 61, 62], Troika does not require using cluster resources. It can generate recommendations on a local machine. As Figure 3 shows, simulation is orders of magnitude faster than running the real application. Therefore, it is capable of testing multiple profiles without the need for running each one in a real execution environment

5.3 Comparison with Starfish

We compared the efficacy of Troika with Starfish [5, 24, 25, 26], a system that employs a domain-specific model-based methodology for configuring MapReduce applications. The evaluation process analyzed the speedup achieved over using the default configuration parameters for a set of applications, consisting of TeraSort [21], Word Co-occurrence, WordCount, Grep, PI, and kMeans and Bayesian from the HiBench Benchmark Suite [28].

For each application, we executed Troika with first- and second-order response surfaces, and used Starfish version 0.3.0 for the comparison. Since Starfish does not support MapReduce 2.0, we performed its evaluations using a classic MapReduce [57] deployment on the cluster presented in Table 2. TeraSort processed a 10 GB input file generated by TeraGen application. Word Co-occurrence, WordCount, and Grep ran with 2.5, 10, and 45GB input files respectively, composed of e-books collected from Project Gutenberg [44]. PI application was configured to run with 1000 map tasks and 400,000,000 samples, and Bayesian and kMeans with the benchmark defaults.

Figure 7 shows the speedup obtained for each MapReduce application using the configuration parameters recommended by Troika and Starfish. There is no bar for the Starfish for Bayesian, since it was one of the benchmarks that did not terminate successfully. The results demonstrate that Troika outperforms Starfish for optimization of various MapReduce applications regardless of the chosen RSM model.

5.4 Efficacy of Hardware Recommendations

Depending on the choice of server hardware, one may observe high variability in performance of applications [17]. However, selecting the right hardware in a cluster is non-trivial for various reasons. (1) The performance bottleneck of the cluster might differ depending on the application requirements. While compute-intensive applications would benefit from a faster CPU, better hard disks would be preferred in other cases to boost performance. Usually, it is hard to manually pinpoint the bottleneck. (2) The cost of chosen hardware must fit within a predetermined user budget. (3) It is hard to estimate the performance impact of replacing multiple cluster components at the same time. Troika helps cluster operators by providing hardware recommendations while considering the applications

Type-A	Type-B	Type-C
Xeon E5530 (4x2.4GHz)	Xeon E5530 (2x2.4GHz)	Xeon(TM) 3GHz
500GB SATA	500GB SATA	146GB SCSI
12GB DDR2 (at 1066MHz)	12GB DDR2 (at 1066MHz)	2GB DDR2 (at 400MHz)

Table 5: Processor, disk and physical memory properties of server types used to evaluate hardware recommendations for executing various applications.

to be executed on the cluster, the budget threshold, as well as the performance impact achieved by replacing or adding hardware.

We considered the three server types listed in Table 5 using the topology presented in Table 2. Up to now, we only used the server Type-A. A Type-B server has the same hardware configuration as Type-A but with half of its cores disabled.

We evaluated if, under a limited budget, it makes sense to replace all servers by Type-B or Type-C by considering the four applications presented in Sec. 5.1. For each application, Troika recommended a server type that yields the best expected performance among the three alternatives. We tested all applications under the default profile parameters after adjusting the memory related constraints for Type-C. Figure 8 presents the performance measurements on these systems, normalized for the performance on a cluster with Type-C nodes. When there is no budget limit, Troika picks Type-A. If using Type-A is unaffordable, Troika recommends Type-B as the replacement.

The evaluations demonstrate that Troika can consider the performance impact of multiple resource modifications and select a system that provides good performance within a specified budget constraint.

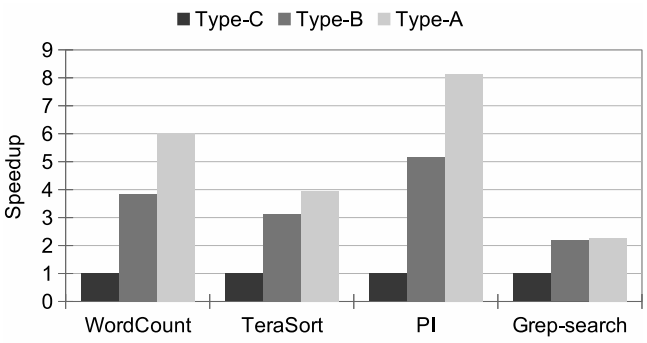
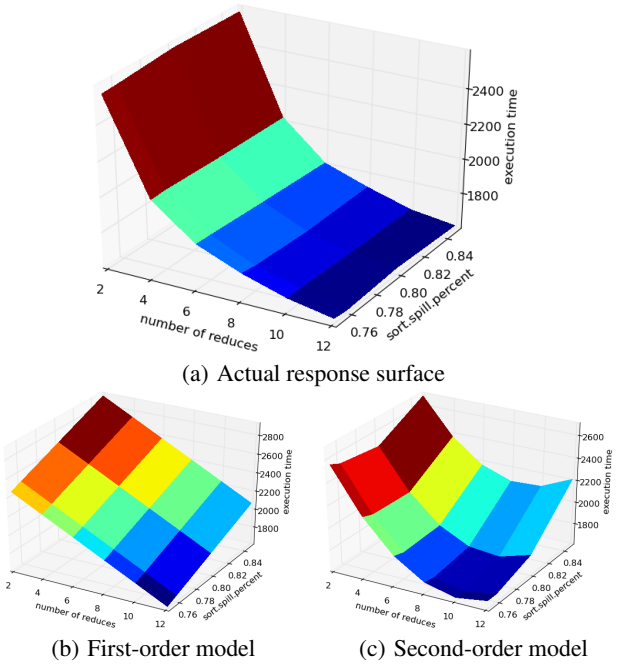


Figure 8: Application performance using different types of servers, normalized to the performance obtained using Type-C servers.

5.5 Modeled vs. Real Response Surfaces

To compare the real and modeled response surfaces, we used the configuration parameters listed in Sec. 5.2.2 and ran experiments with Word Co-occurrence and TeraSort using corresponding input files described in Sec. 5.3. Due to space constraints, we present only Figure 9, which illustrates 3-dimensional projections of the actual 5-dimensional response surface and how it is modeled using first- and second-order polynomials for Word Co-occurrence. As



(a) Actual response surface

(b) First-order model

(c) Second-order model

Figure 9: Actual vs. modeled projections of response surfaces for Word Co-occurrence. (`yarn.nodemanager.resource.memory-mb`: 9216, `mapreduce.job.reduce.slowstart.completedmaps`: 0.2).

the results show, the performance behavior of the application can be captured in both models.

To verify the fit for each model, we calculated accuracy (see Eq. 4) using the real measurement data and the corresponding modeled response of profiles. For Word Co-occurrence, the mean accuracies for the first- and the second-order models are 0.778 and 0.781, respectively. For TeraSort, the mean accuracies are 0.727 and 0.731 respectively. For these cases, the accuracy of both models are comparable in modeling the actual response.

6. CONCLUSION

This paper introduces a new approach for configuring distributed computations. This approach employs a response surface–based technique to determine software and hardware configuration parameters with minimal user assistance. Response surface models already have proven track record as an optimization tool in other engineering disciplines. We have combined this methodology with an accurate simulator and have shown that this approach can achieve desired performance goals while maintaining user constraints. Troika, a full implementation of this approach, shows that performance gains up to 5× could be achieved for MapReduce 2.0 applications. We believe this methodology constitutes a valuable addition to the emerging toolkit of the distributed systems programmer.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments and suggestions. This work was supported in part by NSF grants CCF-0424422, CNS-1111698, CNS-1518779, and CCF-1116298, AFOSR grant FA95550-11-1-0137, DARPA grant FA8750-11-2-0256, DOE grant DE-SC0006791, as well as gifts from Intel and Facebook.

7. REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] ANANTHANARAYANAN, G., AGARWAL, S., KANDULA, S., GREENBERG, A., STOICA, I., HARLAN, D., AND HARRIS, E. Scarlett: Coping with skewed content popularity in MapReduce clusters. In *Proceedings of the Conference on Computer Systems* (Salzburg, Austria, 2011), EuroSys '11, ACM, pp. 287–300.

[3] ANANTHANARAYANAN, G., KANDULA, S., GREENBERG, A., STOICA, I., LU, Y., SAHA, B., AND HARRIS, E. Reining in the outliers in Map-Reduce clusters using Mantri. In *Proceedings of the Conference on Operating Systems Design and Implementation* (Vancouver, BC, Canada, 2010), USENIX Association, pp. 265–278.

[4] Apache Hadoop YARN. http://hadoop.apache.org/docs/r2.3.0/.

[5] BABU, S. Towards automatic optimization of MapReduce programs. In *Proceedings of the ACM Symposium on Cloud Computing* (Indianapolis, IN, USA, 2010), SoCC '10, ACM, pp. 137–142.

[6] BACH, F. R., AND JORDAN, M. I. Kernel independent component analysis. In *International Conference on Acoustics, Speech, and Signal Processing* (Hong Kong, 2003), vol. 4, IEEE, pp. 876–879.

[7] BELLMAN, R. *Dynamic programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[8] BEZERRA, M. A., SANTELLI, R. E., OLIVEIRA, E. P., VILLAR, L. S., AND ESCALEIRA, L. A. Response Surface Methodology (RSM) as a tool for optimization in analytical chemistry. *Talanta 76*, 5 (2008), 965–977.

[9] BIRD, S. L., AND SMITH, B. J. Pacora: Performance aware convex optimization for resource allocation. In *Proceedings of the USENIX Workshop on Hot Topics in Parallelism (HotPar: Posters)* (Berkeley, CA, USA, 2011).

[10] BOX, G. E., AND WILSON, K. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B 13*, 1 (1951), 1–45.

[11] BUYYA, R., AND MURSHED, M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience 14*, 13 (2002), 1175–1220.

[12] CFQ scheduler. https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt.

[13] CHEN, H., JIANG, G., ZHANG, H., AND YOSHIHIRA, K. Boosting the performance of computing systems through adaptive configuration tuning. In *Proceedings of the Symposium on Applied Computing* (Honolulu, HI, USA, 2009), ACM, pp. 1045–1049.

[14] CHEN, H., ZHANG, W., AND JIANG, G. Experience transfer for the configuration tuning in large scale computing systems. *SIGMETRICS Performance Evaluation Review 37*, 2 (2009), 51–52.

[15] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Conference on Operating Systems Design and Implementation* (San Francisco, CA, USA, 2004), USENIX Association, pp. 137–150.

[16] DELIMITROU, C., AND KOZYRAKIS, C. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, TX, USA, 2013), ACM, pp. 77–88.

[17] 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* (Salt Lake City, UT, USA, 2014), ACM, pp. 127–144.

[18] Emulab: Network emulation testbed. https://www.emulab.net/.

[19] GANAPATHI, A., DATTA, K., FOX, A., AND PATTERSON, D. A case for machine learning to optimize multicore performance. In *Proceedings of the USENIX Conference on Hot Topics in Parallelism* (Berkeley, CA, 2009), USENIX Association, pp. 1–6.

[20] Hadoop capacity scheduler. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.

[21] Hadoop TeraSort example. http://hadoop.apache.org/docs/current2/api/org/apache/hadoop/examples/terasort/package-summary.html.

[22] HAMMOUD, S., LI, M., LIU, Y., ALHAM, N. K., AND LIU, Z. MRSim: A discrete event based MapReduce simulator. In *Fuzzy Systems and Knowledge Discovery* (Yantai, Shandong, China, 2010), M. Li, Q. Liang, L. Wang, and Y. Song, Eds., IEEE, pp. 2993–2997.

[23] HANSEN, N., AND OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*, 2 (2001), 159–195.

[24] HERODOTOU, H. Hadoop performance models. Tech. rep., Duke University, 2011.

[25] HERODOTOU, H., AND BABU, S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. In *PVLDB: Proceedings of the VLDB Endowment* (Seattle, WA, USA, 2011), vol. 4, pp. 1111–1122.

[26] 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 *Proceedings of the Conference on Innovative Data Systems Research* (Asilomar, CA, USA, 2011), pp. 261–272.

[27] HiBench benchmark suite. https://github.com/intel-hadoop/HiBench.

[28] HUANG, S., HUANG, J., DAI, J., XIE, T., AND HUANG, B. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDE Workshops* (Long Beach, CA, USA, 2010), IEEE, pp. 41–51.

[29] IPEK, E., DE SUPINSKI, B. R., SCHULZ, M., AND MCKEE, S. A. An approach to performance prediction for parallel applications. In *Proceedings of the International Euro-Par Conference on Parallel Processing* (Lisbon, Portugal, 2005), Springer-Verlag, pp. 196–205.

[30] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems* (Lisbon, Portugal, 2007), ACM, pp. 59–72.

[31] LI, M., ZENG, L., MENG, S., TAN, J., ZHANG, L., BUTT, A. R., AND FULLER, N. MrOnline: MapReduce online performance tuning. In *Proceedings of the International Symposium on High-performance Parallel and Distributed Computing* (Vancouver, BC, Canada, 2014), ACM, pp. 165–176.

[32] LIAO, G., DATTA, K., AND WILLKE, T. L. Gunther:

Search-based auto-tuning of MapReduce. In *Proceedings of the International Conference on Parallel Processing* (Aachen, Germany, 2013), Euro-Par'13, Springer-Verlag, pp. 406–419.

[33] LIU, Y., LI, M., ALHAM, N. K., AND HAMMOUD, S. HSim: A MapReduce simulator in enabling cloud computing. *Future Generation Computer Systems 29* (2013), 300–308.

[34] MONTGOMERY, D. *Design and Analysis of Experiments: Response Surface Method and Designs*. John Wiley & Sons, Inc., 2005.

[35] Mumak: Map-Reduce simulator. https://issues.apache.org/jira/browse/MAPREDUCE-728.

[36] MURRAY, D. G., MCSHERRY, F., ISAACS, R., ISARD, M., BARHAM, P., AND ABADI, M. Naiad: A timely dataflow system. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Farmington, PA, USA, 2013), ACM, pp. 439–455.

[37] MYERS, R. H., MONTGOMERY, D. C., AND ANDERSON-COOK, C. M. *Response Surface Methodology: Process and Product in Optimization Using Designed Experiments*, 3rd ed. John Wiley & Sons, Inc., New Jersey, USA, 2009.

[38] NAES, T., ISAKSSON, T., FEARN, T., AND DAVIES, T. *A user friendly guide to multivariate calibration and classification*. NIR publications, Chichester, UK, 2002.

[39] NEAPOLITAN, R. E. *Learning Bayesian Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[40] The network simulator - ns2. http://www.isi.edu/nsnam/ns/.

[41] OEHLERT, G. W. *Design and analysis of experiments: Response Surface design*. W.H. Freeman and Company, 2000.

[42] ÖKTEM, H., ERZURUMLU, T., AND KURTARAN, H. Application of Response Surface Methodology in the optimization of cutting conditions for surface roughness. *Journal of Materials Processing Technology 170*, 1 (2005), 11–16.

[43] pfifo_fast queuing discipline. http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html#qs-pfifo_fast.

[44] Project Gutenberg. http://www.gutenberg.org/.

[45] REN, Z., LIU, Z., XU, X., WAN, J., SHI, W., AND ZHOU, M. WaxElephant: A realistic Hadoop simulator for parameters tuning and scalability analysis. In *ChinaGrid Annual Conference* (Beijing, China, 2012), IEEE, pp. 9–16.

[46] Rumen: a tool to extract job characterization data from job tracker logs. https://issues.apache.org/jira/browse/MAPREDUCE-751.

[47] SABOORI, A., JIANG, G., AND CHEN, H. Autotuning configurations in distributed systems for performance improvements using evolutionary strategies. In *Proceedings of the International Conference on Distributed Computing Systems* (Beijing, China, 2008), IEEE Computer Society, pp. 769–776.

[48] Simjava. http://www.dcs.ed.ac.uk/home/hase/simjava/.

[49] SINGER, J., KOVOOR, G., BROWN, G., AND LUJÁN, M. Garbage collection auto-tuning for Java MapReduce on multi-cores. In *Proceedings of the International Symposium on Memory Management* (San Jose, CA, USA, 2011), ACM, pp. 109–118.

[50] TENG, F., YU, L., AND MAGOULES, F. SimMapReduce: A simulator for modeling MapReduce framework. In *International Conference on Multimedia and Ubiquitous Engineering* (Los Alamitos, CA, USA, 2011), IEEE, pp. 277–282.

[51] VAVILAPALLI, V. K., MURTHY, A. C., DOUGLAS, C., AGARWAL, S., KONAR, M., EVANS, R., GRAVES, T., LOWE, J., SHAH, H., SETH, S., SAHA, B., CURINO, C., O'MALLEY, O., RADIA, S., REED, B., AND BALDESCHWIELER, E. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the Annual Symposium on Cloud Computing* (Santa Clara, CA, USA, 2013), ACM, pp. 5:1–5:16.

[52] VERMA, A., CHERKASOVA, L., AND CAMPBELL, R. H. ARIA: automatic resource inference and allocation for MapReduce environments. In *Proceedings of the ACM International Conference on Autonomic Computing* (Karlsruhe, Germany, 2011), ACM, pp. 235–244.

[53] VERMA, A., CHERKASOVA, L., AND CAMPBELL, R. H. Play it again, SimMR! In *Proceedings of the IEEE International Conference on Cluster Computing* (Washington, DC, USA, 2011), IEEE, pp. 253–261.

[54] WANG, G., BUTT, A. R., PANDEY, P., AND GUPTA, K. A simulation approach to evaluating design decisions in MapReduce setups. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (London, UK, 2009), IEEE/ACM, pp. 1–11.

[55] WANG, G., BUTT, A. R., PANDEY, P., AND GUPTA, K. Using realistic simulation for performance analysis of MapReduce setups. In *Proceedings of the Workshop on Large-Scale System and Application Performance* (Munich, Germany, 2009), ACM, pp. 19–26.

[56] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. In *Proceedings of the Conference on Operating Systems Design and Implementation* (Boston, MA, USA, 2002), vol. 36, USENIX Association, pp. 255–270.

[57] WHITE, T. *Hadoop: The Definitive Guide*, 3rd ed. O'Reilly, USA, 2012.

[58] XI, B., LIU, Z., RAGHAVACHARI, M., XIA, C. H., AND ZHANG, L. A smart hill-climbing algorithm for application server configuration. In *Proceedings of the International Conference on World Wide Web* (New York, NY, 2004), ACM, pp. 287–296.

[59] YE, T., AND KALYANARAMAN, S. A recursive random search algorithm for large-scale network parameter configuration. In *Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2003), ACM, pp. 196–205.

[60] YU, Y., ISARD, M., FETTERLY, D., BUDIU, M., ERLINGSSON, U., GUNDA, P. K., AND CURREY, J. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the Conference on Operating Systems Design and Implementation* (San Diego, CA, USA, 2008), USENIX Association, pp. 1–14.

[61] ZHANG, Z., CHERKASOVA, L., AND LOO, B. T. Exploiting cloud heterogeneity for optimized cost/performance MapReduce processing. In *Proceedings of the International Workshop on Cloud Data and Platforms* (Amsterdam, The Netherlands, 2014), ACM, pp. 1:1–1:6.

[62] ZHANG, Z., CHERKASOVA, L., AND LOO, B. T. Optimizing cost and performance trade-offs for MapReduce job processing in the cloud. In *Network Operations and Management Symposium* (Krakow, Poland, 2014), IEEE, pp. 1–8.