Performance Tuning of MapReduce Jobs Using Surrogate-Based Modeling

Travis Johnston[1], Mohammad Alsulmi[1], Pietro Cicotti[2] and Michela Taufer[1]

[1] University of Delaware, Newark, DE
travisj@udel.edu, malsulmi@udel.edu, taufer@udel.edu
[2] San Diego Supercomputer Center, San Diego, CA
pcicotti@sdsc.edu

Abstract

Modeling workflow performance is crucial for finding optimal configuration parameters and optimizing execution times. We apply the method of surrogate-based modeling to performance tuning of MapReduce jobs. We build a surrogate model defined by a multivariate polynomial containing a variable for each parameter to be tuned. For illustrative purposes, we focus on just two parameters: the number of parallel mappers and the number of parallel reducers. We demonstrate that an accurate performance model can be built sampling a small set of the parameter space. We compare the accuracy and cost of building the model when using different sampling methods as well as when using different modeling approaches. We conclude that the surrogate-based approach we describe is both less expensive in terms of sampling time and more accurate than other well-known tuning methods.

1 Introduction

This paper presents a general method to tune the parameters in MapReduce (MR) frameworks. The method we propose is based on surrogate modeling and we demonstrate how it can be employed to pursue efficient data analytics across applications and their datasets at runtime. MR is a popular programming model, proposed by Google [1], to process large datasets in parallel. The model has become a dominant methodology for processing large (distributed) datasets for a variety of reasons including: simplicity of programming, automatic load balancing, automatic failure recovery, and ease of scaling. Another feature of MR is its ability to work locally on a dataset by sending map (and reduce) tasks to the data instead of sending data to the tasks. This can substantially reduce the amount of data that must be broadcast across a network. A MapReduce job consists of three stages: map, shuffle (or sort), and finally reduce. A MapReduce job processes a dataset in the following way. First, each element of data begins as a Key-Value (KV) pair. The user defined map function is applied to each KV pair exactly once. The map function produces a list of new KV pairs. The shuffling/sorting task (not implemented by the user) assigns keys (output by the map tasks) to reduce tasks and routes all KV pairs to their assigned reduce task. The user defined reduce function accumulates and combines all the KV pairs into a final value or list of values.

There are several frameworks that implement the MR model, for example MapReduce-MPI [2], Hadoop, and Spark. Every framework includes a number of configuration parameters which individual users may modify in an attempt to maximize performance. Examples of these parameters include the number of parallel map tasks, number of parallel reduce tasks, or the amount of memory given to each map/reduce task (and many others). Every framework also includes default parameter configurations. Default configurations are rarely, if ever, optimal. If a MR job is to be run only a single time then the user may choose to apply any number of rules of thumb defined by experts to optimize performance. For example, experts at IBM suggest that each map/reduce task should be allocated at least 2GB of memory and the total number of parallel map/reduce tasks should be at most the number of CPUs. Experts at AMD suggest that each map/reduce task should be allocated at least 1GB of memory and the total number of parallel map/reduce tasks should be at most three times the number of CPUs. These rules do not define a single set of parameters and we observe that the rules do not always find optimal configurations even across a simple benchmark. Figure 1 shows performance of two Hadoop benchmarks: Wordcount (a single MR job) and PageRank (a chain of MR jobs). The figure illustrates the complexity of finding an optimal set of parameters as well as the significant impact that the parameters may have on performance. Note that in this case, set 1 is optimal for Wordcount but performs the worst for PageRank.

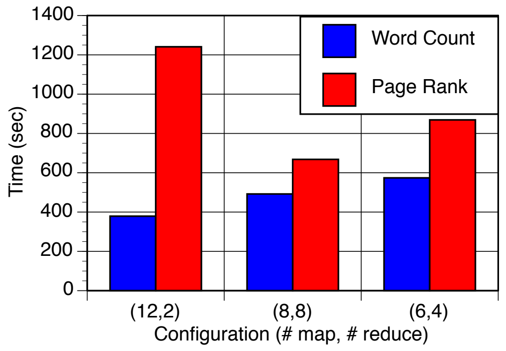


Figure 1: Walltime for two benchmarks, Wordcount and PageRank, with three distinct configurations. The computations were done on a machine with 16 cores.

If a user has a job to run multiple times over similar datasets (or datasets that evolve over time) then s/he can greatly improve performance by discarding the rules of thumb and employing a parameter tuning method. Users may employ their observations of performance to make better informed choices regarding these settings. Several tuning algorithms have been proposed and used including a brute force search (over a subset of parameters) [3], an approach using the Grid Hill algorithm on a pruned search space [4], and an application of simulated annealing [5]. All these methods require the user to make many observations or samples of the parameter space. Sampling points is expensive since each sample requires users to run a job as similar to their production job as possible. They are also limited in the sense that they never predict that an optimal setting may be a previously unobserved one.

In this paper, we approach the problem by applying surrogate-based modeling. Surrogate based modeling was originally introduced to reduce the cost of computation in aerodynamic design problems [6] and recently used to model I/O performance [7]. The surrogate-based model that we build models the performance of the entire parameter space by sampling a relatively small portion of it. The more points we sample, the more accurate the model can be. However, the additional accuracy comes with additional sampling time. The goal is to build as accurate a model as possible sampling as few points as possible. We demonstrate that an accurate model can be built using significantly fewer suboptimal observations than previous methods. In addition, we show that the sampling method can also have a significant impact on the cost of building an accurate model.

2 Process to Build a Model

The key steps in building a surrogate model are: sampling a portion of the parameter space, choosing a function to fit the data, and locating an optimal point on the model surface. In the process of building the model, both cost and accuracy should be assessed.

2.1 Sampling the Parameter Space

To build an accurate performance model of a parameter space, a meaningful sample of the space needs to be obtained [8]. We collect multiple performance samples of the parameter space by varying the considered parameters and observing the resulting performance. We consider three different sampling methods incrementally increasing the randomness of the sampled points: an exhaustive sampling applicable only at a small parameter scale and used exclusively for validation purposes in this paper, a grid based sampling, and a random sampling. *Exhaustive sampling* requires us to sample every configuration in the parameter space. The model we can build from this information is the most accurate based on the observations collected but, at the same time, it is also the most expensive. At the large scale, when the number of parameters and their range of values are very large, exhaustive sampling is not feasible. *Grid based sampling* chooses points which lie in a grid. The goal of grid-based sampling is to ensure that points are sampled relatively uniformly from all over the parameter space avoiding any local concentration of samples. To effectuate the sampling, we begin by choosing a *corner* sample (x_0, y_0) and a step size Δ. We sample all points (in a bounded region containing feasible parameters) of the form $(x_0 + a\Delta, y_0 + b\Delta)$ where a and b are non-negative integers. The step size and starting point determine the number of rows and columns of the grid. *Random sampling* is an alternative approach to the highly structured approach of grid based sampling. This sampling method selects n points from the parameter space at random (without replacement). With this method there is no guarantee that all areas of the parameter space are evenly sampled and there may be occasional clusters of samples. Contrary to what we expected, the unstructured nature of random sampling is actually an advantage.

2.2 Building a Model Surface

When building a surrogate model we must decide what type of function we want to apply to fit the data. We chose to fit our data to a polynomial surface built using least squares regression. Polynomial models are advantageous because they are simple to describe and can describe very complex surfaces (especially as the degree of the polynomial increases). Here we present the general process of building the model and the specific adaptations to fit a polynomial surface.

A surrogate model can be built to fit any m-dimensional space of parameters. The vector $\vec{x} = (x_1, x_2, ..., x_m)$ denotes a specific configuration in this space and n will always represent the number of points we have sampled from the space. Assume, for the moment, that we have collected n samples and decided on what function to employ to fit the data. The general surface we build has the form: $z(\vec{x}) = \beta_1 f_1(\vec{x}) + \beta_2 f_2(\vec{x}) + ... + \beta_D f_D(\vec{x})$. When $z(\vec{x})$ is a polynomial each $f_i(\vec{x})$ is a monomial of the form $x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m}$ where $k_1 + k_2 + \cdots + k_m = d_i$ (the degree of the monomial) and $d_i \in \{0, 1, \cdots, d\}$. The surface thus built is a degree d polynomial surface. The number of monomials D depends on the number of parameters, m, and the degree of the polynomial d. Equation 1 gives the number of monomials contained in the polynomial.

$$D = \sum_{i=0}^{d} \binom{m+i-1}{i} = \binom{m+d}{d} \tag{1}$$

We let $\vec{x}_i$ represent the configurations for the i'th observation, and z_i is the observed runtime with that configuration. The sum-of-squares error (SSE) is defined as $\sum(z_i - z(\vec{x}_i))^2$. Independent of both the number of parameters m, and the number of observations n, the values of β_i are chosen in such a way to minimize the SSE. Equation 2 defines the matrices used to determine the coefficients β.

$$X = \begin{bmatrix} f_1(\vec{x}_1) & f_2(\vec{x}_1) & f_3(\vec{x}_1) & ... & f_D(\vec{x}_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_1(\vec{x}_n) & f_2(\vec{x}_n) & f_3(\vec{x}_n) & ... & f_D(\vec{x}_n) \end{bmatrix}, \; Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}, \text{ and } \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}. \qquad (2)$$

We compute the coefficient matrix β by using $\beta = (X^T X)^{-1} X^T Z$.

Both accuracy and cost of our surrogate model depend on the number of samples n and the polynomial degree. The minimum *number of samples* needed to build a surrogate model is the number of functions used in the fitting, D. Equation 1 gives this minimum number for our polynomial surface. Note, however, that merely sampling the minimum number of samples may not be sufficient to actually build the model. Specifically, if $X^T X$ is not invertible, then there is not a unique best fit surface. For $X^T X$ to be invertible, X must contain D linearly independent columns. If X has full column rank then $n \geq D$. In addition, we need to sample a diverse enough set of points so that there are not any hidden dependencies among the samples. In general it is hard to say exactly when one has sampled a diverse enough set of points. For the case study in Section 3 of a polynomial with two variables (x and y), we can give a few examples of what is sufficient, and what is not. First, if there exists a set of d lines which contain all of the sampled data points, then the matrix X corresponding to a degree d polynomial will not have full column rank. If no set of $d + 1$ lines contains all the data points, then the matrix X will have full column rank. The latter condition is sufficient, but not necessary, to guarantee full column rank. Second, if, from among our sampled points, we can choose $d+1$ unique x-values, $\{x_0, x_1, ..., x_d\}$ and $d+1$ distinct y-values, $\{y_0, y_1, ..., y_d\}$ with the property that we sampled at least $D = (d+1)(d+2)/2$ points from the grid $\{x_0, x_1, ..., x_d\} \times \{y_0, y_1, ..., y_d\}$ then the matrix X will have full column rank. When sampling points we keep in mind that we *must* sample at least D points (with enough diversity) and the more points we sample the more accurate our model should become.

Determining the optimal degree of the polynomial surface is critical because if the degree is too low, the polynomial surface may not have enough freedom to reflect what the space actually looks like. Polynomials of larger degree give more flexibility to the model and provide for the opportunity for greater accuracy but if the degree is too large, we are in serious danger of over fitting the data. To search for the polynomial order that is optimal given the observations we have made, we apply k-fold cross validation on surfaces with increasing polynomial orders. The cross validation process proceeds as follows:

Step 1: The data points are partitioned randomly into k equally sized sets; in the event that k does not evenly divide the number of data points we have, we make the sets as evenly sized as possible.

Step 2: We select one set of data points to be the *testing* set; the remaining $k-1$ sets comprise the *learning* set.

Step 3: Using the learning set, we construct a best fit polynomial surface of degree d for each reasonable value of d, i.e. $d \in \{1, 2, ..., M\}$ where M is small enough that the matrix $X^T X$ is invertible.

Step 4: We use the polynomial surface to predict the runtimes of points in the testing set and compute the SSE of the testing set.

Step 5: We repeat the process from Step 2 until every set has been the testing set exactly one time. We record the average SSE from this partition of the dataset on a plot.

Step 6: We repeat the process from Step 1 several times. In our case, we run this process a total of 10 times. In general, this process is repeated until the user is confident of a representative mean SSE.

The k-fold cross validation process yields an average SSE for each polynomial degree considered. The degree with the least average SSE is the ideal polynomial degree. Keep in mind that as the number of samples we have collected grows, the ideal polynomial degree may change.

2.3 Locating a Minimal Point

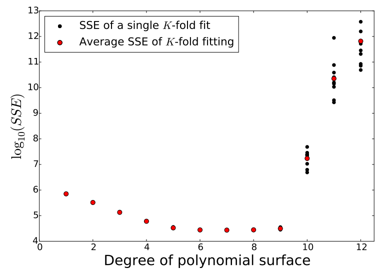
Having built the model, we search for the minimal point on the surface. Some areas on the surface are very uncertain because we have not sampled nearby. To strengthen our model, and avoid predictions that are highly uncertain, we apply a 99% confidence interval to the model. At each point z in the model, we can with 99% accuracy say that the expected runtime is in $z \pm \epsilon_z$. We replace each z on the surface with $z + \epsilon_z$. Areas that we've sampled more thoroughly will have a much smaller ϵ_z. To locate a minimum value on the model we apply a limited memory quasi-Newton [9] optimization algorithm. The location of the minimum corresponds to a *predicted* set of optimal parameters and their expected walltime. The optimal point on the surface may or may not have been sampled directly.

2.4 Validating the Model

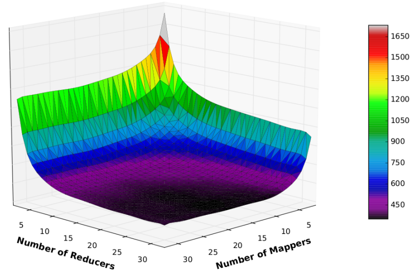
There are two components to validating a model that need to be assessed: the accuracy of the predicted walltime and the additional cost associated to running the MR jobs with parameter values that are suboptimal. The goal of any model is to obtain sufficient accuracy while minimizing the additional cost.

To assess the *accuracy* we use $A(\vec{x})$ to denote the actual walltime of the MR jobs with parameter settings $\vec{x}$. An optimal set of parameters is denoted by $\vec{x}_0$. $M(\vec{x})$ denotes modeled walltime using parameters $\vec{x}$ and $\vec{x}_1$ minimizes M. In order to validate the model on our small sample space, we first performed an exhaustive sampling. With these measurements in hand, we define the error of the model M to be $|A(\vec{x}_1) - A(\vec{x}_0)|$ and report this value as a percentage of $A(\vec{x}_0)$. This represents how close (in runtime) our predicted configuration is to the optimal one. This method of measuring error requires us to know the optimal running time. In practice researchers do not know this ahead of time. In that case, they may prefer to compare $A(\vec{x}_1)$ to a runtime with a default parameter setting and measure the speed-up they obtain.

When building a model there is a cost for each sample. If we sample several points in the space, inevitably many of them will be suboptimal and we incur *additional cost* for which we pay the price. However, after we have made our samples, if our performance is significantly improved we can recoup the cost with future savings. In other words, we amortize the cost of building the model with future performance gains. We refer to the total number of samples needed to recoup the entire additional cost of sampling as a break even point. Note that a break even point is never reached if the end result one uses is worse than the *default* setting.



(a) Average SSE by degree of surface



(b) The full model surface

Figure 2: K-fold fitting applied to the full data set. We observe that a polynomial of degree 9 is the best fit for our sample data (a). The associated surface is in (b).

3 From Theory to Practice: The PageRank Case Study

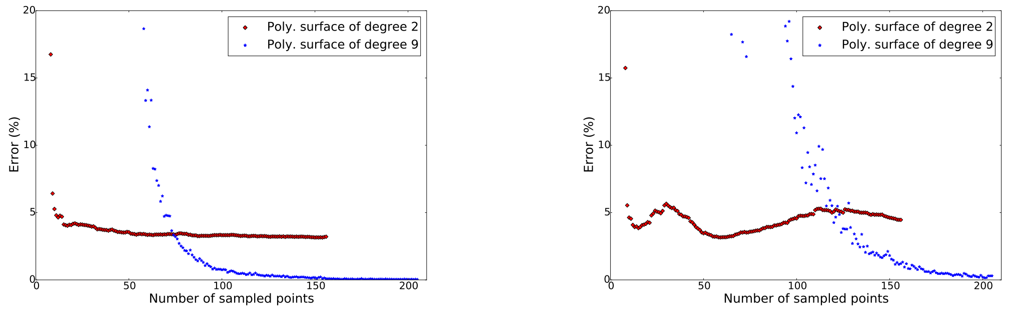
As a case study, we show how to apply the surrogate-based approach for tuning the number of parallel map and reduce tasks in a workflow consisting of PageRank computations. We chose to use the Hadoop framework and PageRank from the HiBench benchmark suite. Hadoop is an open source, Java-based framework sponsored by Apache, and is used for processing large scale datasets in a distributed computing environment. We chose to use Hadoop because it has been widely used and is easy to configure. Hadoop is flexible and can run on a small cluster containing only a few individual machines or in a large cluster containing thousands of nodes. The HiBench benchmarking suite is a comprehensive suite of MapReduce applications stemming from multiple application domains. PageRank is an ideal choice because it is the type of job that is run repeatedly on an evolving dataset. In our study, we chose a representative data size of 5M pages (i.e., 1GB) because it is sufficiently large that varying the parameters will yield discernable results and is sufficiently small that we are able to make an exhaustive search for validating our models. We performed all of our testing on the Stampede supercomputer housed at the Texas Advanced Computing Center (TACC). We use a single large memory compute node containing 32 cores.

3.1 Predictions with Exhaustive Sampling

The two parameters we tune are the number of parallel map tasks (number of mappers, x) and the number of parallel reduce tasks (number of reducers, y). In order to be able to validate the models we build, we begin by exhaustively sampling the parameter space considering all integer points with $2 \leq x, y \leq 32$, a total of 961 points. We apply a k-fold fitting to the complete data set and determine that a degree 9 polynomial is the most appropriate, see Figure 2(a). We use this surface as the benchmark for accuracy when building models from much smaller data set; borrowing notation from Section 2.4 we refer to this model as $A(x, y)$.

3.2 Building an Accurate Model from a Sparse Sampling

Exhaustive sampling and a relatively high polynomial degree generate the most accurate surrogate model, this model also comes with a higher cost. While the small parameter space in our



(a) Accuracy evaluation of random sampling (b) Accuracy evaluation of grid sampling

Figure 3: Comparison of the accuracy of models built from random sampling and grid sampling.

case study allows us to build this model, this is not the case for larger parameter space. Surrogate based modeling produces similar results even though we only sample a limited number of points and use a lower polynomial degree.

As described in Section 2.2, to build a surrogate model we require a certain number of points depending on the number of parameters and the polynomial degree. In our case study we consider two variables (the number of map and reduce tasks) and thus to build a degree d surface we must sample at least $\binom{d+2}{2}$ points. We refer to this number as MIN_d and recall that simply sampling this many points does not guarantee that X^TX is invertible, and if it is invertible, does not guarantee that our model is accurate. In our experiments, we start out by sampling MIN_d points and build the degree d model with d ranging from 2 to 9. In an effort to emulate the generation of the model in a real scenario, we add sample points to our set of previously sampled points and rebuild the model after each additional sampling. Note that when building the model the expensive operation is sampling a new point, rebuilding the surface is significantly cheaper. At each step we record the accuracy of the model by comparing to $A(x, y)$. For each polynomial degree, we build 100 different surfaces, starting from 100 different sets of MIN_d points and stop when the individual datasets have size $MIN_d + 150$ points.

Figure 3 shows the average accuracy of these models for both random sampling, Figure 3(a), and grid-based sampling, Figure 3(b), for polynomial degrees equal to 2 and 9 (the reasonable extremes observed in Figure 2(a)). We observe in Figure 3(a) that by sampling $MIN_d + 50$ points randomly the degree d models appear to have converged. While additional sampling yields severely diminishing returns in terms of improved accuracy, the standard deviation of the error does decrease. One might achieve better accuracy by using the larger collection of samples to build a higher degree surface, but this is not necessarily the case. Contrary to what we expected, the grid based sampling method exhibits several undesirable traits. First, particularly in the degree 2 model, we observe that sampling more points does not give us, on average, a more accurate model. For example we see local minima near 16 samples and 64 samples. This behavior is particularly disturbing for the reason that one is led to believe that the model is mistakenly accurate when relatively few samples are taken. Second, the degree 9 model converges to zero as expected, but requires at least 50 more samples to achieve comparable accuracy with the random method and exhibits a higher degree of variation. Finally, grid sampling poses a problem for actually fitting a surface. Specifically, all the data can be covered by relatively few lines, limiting the degree of the surface one can build without sampling many points. We conclude that the random sampling method outperforms the grid sampling method and focus our attention on the former for the remainder of this paper.

From Figure 3 we select three sample sizes, 10, 60, and 120, to analyze in detail. Key statistics from these points are recorded in Table 1. We record the average error of a surface built from 10, 60, or 120 points, the standard deviation of the error, and the number of jobs required to amortize the cost of building the model. The number of samples required to break even is calculated assuming a *fixed sampling* strategy (or default) with 16 Mappers and 16 Reducers. This is a reasonable first approximation with 15% error.

Degree of Model	Samples (#)	Avg. Error Random (%)	Std. Dev Random	Break Even	Avg. Error Grid (%)	Std. Dev Grid	Break Even
2	10	5.27	3.82	38	4.63	2.82	36
2	60	3.35	1.13	192	3.17	0.93	188
9	60	14.10	17.32	2211	42.05	76.57	–
9	120	0.36	0.65	311	4.26	4.23	415

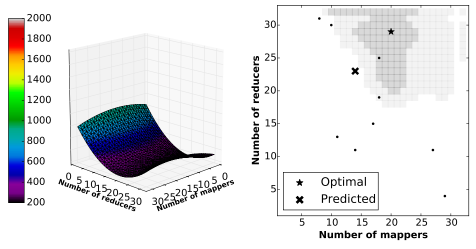
Table 1: Summary of statistics related to surrogate-based model construction and accuracy.

Observe that, simply having enough points to build a model does not guarantee that the *higher degree* model will be more accurate than a lower degree model. In particular, building a degree 2 model from 60 points has an average error of only 3.35% with a standard deviation of 1.13; whereas a degree 9 model on the same number of points has an average error over 14% and a standard deviation of more than 17. The average error represents a *typical* error one might expect. The standard deviation, in a sense, measures how far off one's expectations might be. Specifically, we can be certain that 80% of the surfaces we build will have error at most the average plus two standard deviations, 90% of them have error less than the average plus three standard deviations. As we sample more and more points, the standard deviation of the error tends to decrease, however, it typically does not decrease fast enough to yield a smaller break even point. Figure 4 depicts several representative surfaces built in our computations for Figure 3(a) and Table 1. While the degree 2 surfaces do not look like the *real* surface in Figure 2(b) they do predict parameters with near optimal runtime. Note that the first degree 9 surface, Figure 4(c), is erratic. This is largely due to the fact that we have barely sampled enough points to even build the surface. Once we have sampled more points, Figure 4(d), the model no longer contains significant differences from the full model, Figure 2(b). For further illustration, we have shown the sampled points and shaded the regions corresponding to $< 3.5\%$ error (darker shade) and error $\leq 7\%$ (lighter shade).

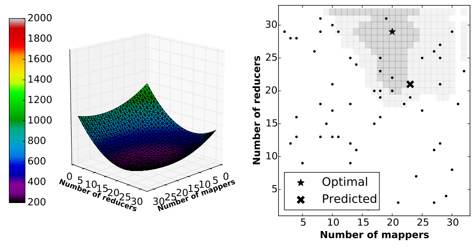
3.3 Comparison to Grid Hill Search

Grid Hill searching is a commonly used optimization method and has been employed for parameter tuning [4]. The method works by partitioning the search space into a grid, the fineness of the grid is chosen by the user. A random starting point is chosen in each grid; each of these becomes a *current point*. Neighboring points are sampled around each *current point*. As neighbors with better runtimes are found, the current position of each point is updated to the better configuration. The process of sampling neighbors continues until each *current point* is a local minimum. Among all the local minima found the one with shortest runtime is selected–predicted to be the optimal.

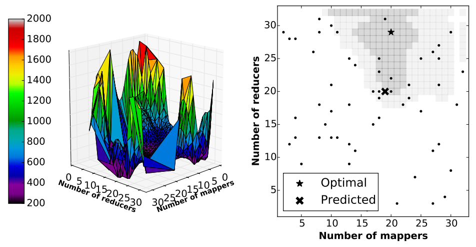
We experiment with grid hill searching and compare the results with those of our surrogate-based modeling. The results are recorded in Table 2. For each grid size, we ran the grid hill



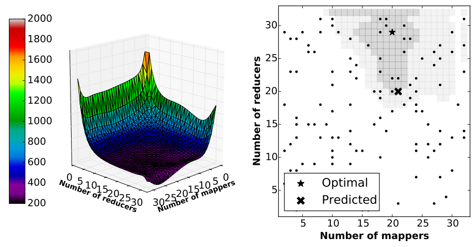
(a) Degree 2 surface with 10 points



(b) Degree 2 surface with 60 points



(c) Degree 9 surface with 60 points



(d) Degree 9 surface with 120 points

Figure 4: Example of surfaces built from a growing set of sample points.

algorithm 100 times. One challenge with grid hill sampling is that the user has little control over how long the process takes to converge. We record the average number of samples required to converge (along with the standard deviation). Note that the simplest case, a single grid, requires on average 25 iterations (samples) to converge, but (because the standard deviation is almost 16) it is not uncommon to require 41 or more samples to converge. If the user interrupts the sampling, the may finish with a significantly higher degree of error. We also note that the average error and standard deviation from grid hill is higher than that of the surrogate modeling. Because we don't have control over precisely how many grid hill samples we take it is hard to make a one-to-one comparison. But, note that sampling 90 times (grid hill) yields an average error of 3.17% with standard deviation 2.18, whereas sampling 2/3 that many points, building a degree 2 surface, gives us a comparable (3.35%) error rate with a lower standard deviation. Keep in mind that the grid hill method it is not uncommon to sample 112 points still with about 3% error. One other major drawback with grid hill is that it will not suggest a parameter set that has not been sampled; the surrogate model will make such a prediction.

Size of Grid	Samples (Avg. #)	Std. Dev (Samples)	Error (%)	Std. Dev (Error)	Break even (#)
1×1	25	15.8	10.96	8.33	225
2×2	90	22.1	3.17	2.18	283
3×3	190	22.2	1.70	1.15	531
4×4	289	25.3	1.15	0.69	777

Table 2: Summary of performance statistics related to grid-hill sampling.

4 Conclusion

We proposed a surrogate based model to accurately predict optimal MR configurations. We demonstrated that such a model can be easily built sampling a relatively small portion of the parameter space. Specifically, we achieved an average error rate of 3.35% by building a degree 2 surface from 60 sample points (selected at random from a space of 961). Building a model in this way is particularly useful when running the same job (i.e., PageRank) on similar or evolving datasets. The strength of the modeling is that we can build an accurate model from only a few samples whose cost can be quickly amortized–with as little as 38 total jobs.

One major question we will address in future work is to determine the number of samples needed (beyond MIN_d) to build an accurate model. We observed that 50 points was sufficient in our case study. However, it is unclear how that number scales. Is it a constant 50? Probably not. Is it bounded by a constant multiple of MIN_d, e.g. $\leq 10MIN_d$? Perhaps. Or, is 5% of the total space? Possible. If we could bound this number by $cMIN_d$ (for some constant c) then the method would decisively dominate other methods (including grid hill searching) in terms of scalability. Finally, we plan to explore hybrid sampling methods. For example, begin by sampling a sparse grid in such a way to guarantee that X^TX is invertible, then attempt to improve accuracy by sampling randomly.

Acknowledgments

The authors are supported by NSF grants CCF-1318445/1318417. They gratefully acknowledge the use of XSEDE resources supported by NSF grant ACI-1053575. Code samples and timing data are available on github: https://github.com/TauferLab.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[2] S. Plimpton and K. Devine, "MapReduce in MPI for Large-scale Graph Algorithms," *Parallel Comput.*, vol. 37, pp. 610–632, September 2011.

[3] P. K. Lakkimsetti, "A Framework for Automatic Optimization of MapReduce Programs Based on Job Parameter Configurations," Master's thesis, Kansas State University, August 2011.

[4] K. Wang, X. Lin, and W. Tang, "Predator- An Experience Guided Configuration Optimizer for Hadoop MapReduce," in *Proc. of 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 419–426, December 2012.

[5] D. Wu, "A Profiling and Performance Analysis based Self-tuning System for Optimization of Hadoop MapReduce Cluster Configuration," Master's thesis, Vanderbilt University, May 2013.

[6] Y. Jin, "Surrogate-assisted Evolutionary Computation: Recent Advances and Future Challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61 – 70, 2011.

[7] M. Matheny, S. Herbein, N. Podhorszki, S. Klasky, and M. Taufer, "Using Surrogate-based Modeling to Predict Optimal I/O Parameters of Applications at the Extreme Scale," in *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, December 2014.

[8] S. Koziel, D. Ciaurri, and L. Leifsson, "Surrogate-Based Methods," in *Computational Optimization, Methods and Algorithms*, vol. 356 of *Studies in Computational Intelligence*, pp. 33–59, 2011.

[9] R. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM J. Sci. Comput.*, vol. 16, pp. 1190–1208, September 1995.