Statistical Data Reduction for Efficient Application Performance Monitoring

Lingyun Yang[1] Jennifer M. Schopf[2] Catalin L. Dumitrescu[1] Ian Foster[1,2]
[1]*Department of Computer Science, University of Chicago, Chicago, IL 60637*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439*
[lyang, catalind]@cs.uchicago.edu [jms, foster]@mcs.anl.gov

Abstract

There is a growing need for systems that can monitor and analyze application performance data automatically in order to deliver reliable and sustained performance to applications. However, the continuously growing complexity of high performance computer systems and applications makes this process difficult. We introduce a statistical data reduction method that can be used to guide the selection of system metrics that are both necessary and sufficient to describe observed application behavior, thus reducing the instrumentation perturbation and data volume to be managed. To evaluate our strategy, we applied it to one CPU-bound Grid application using cluster machines and GridFTP data transfer in a wide area testbed. A comparative study shows that our strategy produces better results than other techniques. It can reduce the number of system metrics to be managed by about 80%, while still capturing enough information for performance predictions.

1. Introduction

Recent experience in deploying Grid middleware demonstrates the challenges one faces in delivering robust services in distributed and shared environments [12]. Applications often must deliver reliable performance despite the use of distributed and shared resources. The first step toward this kind of fault tolerant and adaptive computing is to monitor the performance of system components such that we can diagnose the reason an anomaly happens. As high-performance computer systems and applications continue to increase in complexity, performance monitoring and analysis grows more difficult.

There are two approaches to understand relationships among performance components and to address application performance problems. The most common one is to build a performance model of the application [15] . However, such models are usually application specific and are themselves difficult and costly to build especially in the distributed and heterogeneous environments.

The second approach is to use instrumentation systems that capture information on a large number of these time-varying system metrics and then analyze the relationship among system components and applications [7]. However, direct instrumentation can influence the performance of target systems and produce tremendous volumes of data [10]. To combat such performance instrumentation consequences and simplify data analysis, we need mechanisms to select only necessary metrics and measurement points.

We focus here on how to use a two-step statistical data reduction strategy that selects only the monitoring metrics that are necessary and sufficient for capturing relevant application behaviors. Our goal is to reduce perturbations and data volume while retaining meaningful event characteristics. We evaluate the effectiveness of our data reduction strategy in two different contexts, namely a CPU-bound astrophysics application and GridFTP data transfer. The rest of this paper is organized as follows. Section 2 describes the problem. Section 3 introduces our data reduction strategy. Section 4 presents our experimental results. Section 5 introduces related work. In Section 6, we briefly summarize our effort.

2. Problem and Approach

Previous studies [9] show that variability in resource characteristics can have a major influence on application performance. To formalize this notion, we consider a distributed system with p resources, each characterized by a set of system metrics. For example, a resource *CPU* might have three system metrics: percentage of CPU utilization at the user level, percentage of CPU utilization at the system level, and percentage of time that the CPU was idle. The characteristics of the system including p resources can be described by the set of all system metrics:

$$M = \begin{bmatrix} m^1_1, m^1_2, \ldots, m^1_{n1}; \\ m^2_1, m^2_2, \ldots, m^2_{n2}; \\ \ldots \\ m^p_1, m^p_2, \ldots, m^p_{np}; \end{bmatrix}$$

where *ni* is the number of metrics for the *i*th resource; and m^i_j is the *j*th metric for the *i*th resource.

We also introduce the notion of a performance metric, a quantitative description of some aspect of application performance. For example, one useful metric for an application that calculates the product of two matrices might be the number of multiplications finished during a time unit. Depending on the system metrics available on a particular system and the performance metrics of interest to the user, we may find that there is a correlation between some function of some subset of the system metrics and a particular performance metric. If such a function and subset exist and can be identified, then we can use this system metric set as predictors for the performance metric.

We now state our problem as follows: Given a system, a set of system metrics, and an application described by a performance metric, identify a minimal set of system metrics that can be used to predict the performance metric with a desirable level of accuracy.

For solving this problem, we exploit the fact that some metrics are redundant. For example, the value of the metric *used memory* is equal to the total memory size minus the value of the metric *unused memory*, and vice versa.

Definition 1: Two system metrics m_1, m_2 are *dependent* on each other if and only if the values of the two system metrics are correlated with each other at a level greater than a specified threshold. Otherwise, m_1 and m_2 are *independent*. ■

For example, the value of *used memory* and the value of *unused mem*ory of the same machine are dependent on each other. Only one is necessary; the other is *redundant* and can be eliminated from the set of potential predictors without losing useful information..

We also take into account the fact that not all system metrics are related to the performance metric. For example, in the case of a program that calculates the sum of several integers, CPU utilization is likely to be strongly related with execution speed, while the number of opened files is not. To describe the relationship between system metrics and performance metric, we introduce the following definition:

Definition 2: A performance metric y is *predictable* by system metrics $m_1, m_2, \ldots m_n$ if and only if the value of y can be predicted by a function of variables $m_1, m_2 \ldots m_n$, expressed by $y=F(m_1,m_2,\ldots, m_n)$. We then call y the *response variable* and each system metric m_i (i=1..n) a *predictor* of y. ■

With the above definitions, our problem can be formalized as follow: Given an application characterized by a performance metric y and a system characterized by a set of metrics M, our goal is to find a subset of system metrics $S= (x_1,x_2,\ldots, x_n)$, $S \subseteq M$ such that (a) every pair of system metrics in S, x_i and x_j, i=1..n, j=1..n , $i \neq j$, are independent and (b) every system metric x_i, i=1..n, is a predictor of the performance metric y of the application running on this system, using a given model. The goal of criterion (a) is to remove redundant system metrics. The goal of criterion (b) is to find all metrics that predict application behavior and remove those that do not.

3. Statistical Data Reduction

From the above considerations, we know that two kinds of unnecessary system metrics can be reduced without losing any useful information when predicting a specified performance metric with a desired level of accuracy. We proceed in two steps: (1) eliminate dependent system metrics (Section 3.1) and (2) identify and further eliminate irrelevant system metrics (Section 3.2) —with each step reducing one kind of unnecessary system metrics. The result of our strategy is a subset of system metrics that are necessary for predicting the performance metric and are independent of each other. In Section 3.3 we discuss the criteria used to evaluate the data reduction strategy.

3.1. Redundant System Metrics Reduction

Dependent system metrics (**Definition 1**) are metrics that are strongly correlated with each other. Since a pair of dependent metrics conveys essentially the same information, only one is necessary; the other is redundant and can be eliminated.

We use the *Pearson Product-Moment Correlation Coefficient* (r), or correlation coefficient for short, to obtain a quantitative estimation of the degree of correlation between two system metrics. The correlation coefficient provides a measure of the degree of linear relationship between two variables [16]. A high correlation coefficient value indicates that the two variables can be calculated from each other by some linear model.

To identify system metrics that are dependent on each other, we first construct a correlation matrix by computing the correlation coefficient between every pair of system metrics. Then, we apply a clustering algorithm to this matrix to identify clusters of system metrics such that every metric in a cluster has a correlation coefficient with absolute value above a threshold value to at least one other metric in the cluster. We conclude that the system metrics in one cluster capture essentially the same information and eliminate all but one of those metrics.

The research question we consider is: "How much data do we need to determine whether an observed correlation is significant?" To show why this is important, Figure 1 shows 20 sample correlation coefficients between *the number of transfers issued per second* and *the number of memory pages cached per second* for data collected during the execution of one of our test applications, Cactus [3] (Section 4). Because these two system metrics describe the performance of two independent components in the system, we would not expect them to be strongly correlated. As shown in Figure 1, however, while in most cases (16 cases or 80% of the time) the absolute value of the sample correlation coefficients between these two metrics is small (< 0.2), in a few cases (4 cases or 20% of the time) the absolute value is high (as high as 0.98 in our example). Thus, a correlation coefficient obtained by using a single sample could cause a false positive error and group these two independent system metrics into one cluster.

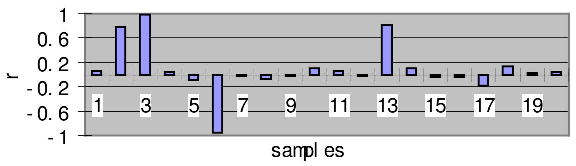

Fig. 1: Sample correlation coefficient between *the number of transfers issued per second* and the *number of memory pages cached per second* for 20 Cactus runs

To reduce the chance of such false positive errors, we use a one-tailed Z-test to determine whether an observed correlation is statistically significant. The Z-test is a statistical method used to test the viability of a hypothesis in the light of sample data with specific confidence. More specifically, in our strategy, the hypothesis to be tested is as follows: The actual correlation coefficient is less than or equal to the threshold value. Given a set of sample data, the Z-test tests the possibility of observing the sample correlation coefficient if the hypothesis is true. If the possibility is small (< 5% in our work), we can reject the hypothesis with more than 95% confidence and say that the real correlation coefficient is statistically larger than the threshold value. We then group the two system metrics involved into one cluster.

Thus, given a set of samples, we proceed as follows. We perform the Z-test for the correlation coefficient between every pair of system metrics, and we group two metrics into one cluster only when the absolute value of their correlation coefficient is larger than the threshold value with 95% confidence. The result of this computation is a set of system metric clusters. System metrics in each cluster are strongly correlated, so only one metric from the cluster can be used as the representatives of the cluster while the others are deleted as redundant. Currently, we pick as the representative the system metric with the highest correlation coefficient value with the application performance metric.

We also find it useful to identify and eliminate system metrics that have no variation in the sample data, which means that no correlation coefficient involving these metrics can be calculated. We group these metrics into a special cluster called "zero variation," in which all metrics have a variation of zero. We consider all metrics with zero variations to be redundant metrics and immediately eliminate them, since (as far as we can determine based on our sample data) they do not carry any useful information for predicting the performance metric.

We also need to decide the value of the threshold used to judge whether the correlation between two system metrics is strong enough to put them into one cluster. In Section 4, we discuss our selection of this value and its influence on data reduction.

3.2. Statistical Variable Selection

After eliminating all dependent system metrics, we obtain a subset of independent system metrics. However some of these system metrics may not relate to our chosen performance metric. Thus, in the second step of our strategy, we identify the subset of all predictors that are necessary to predict the performance metric, further reducing system metrics that either are unrelated to the performance metric, or, given other metrics, are not useful for predicting the performance metric. This data reduction is also known as *variable selection*.

Two basic methods for variable selection have evolved. The first method uses a criterion statistic computed for all possible subsets of predictors. This method is able to find the best solution but is inefficient. The second method, generally called stepwise regression, provides a systematic technique for choosing a path through the possible subsets, first looking at a subset of one size, and then looking only at subsets obtained from preceding ones by deleting one potential predictor. This limiting of the number of subsets of each size that must be considered makes the second method more efficient than the first.

We focus here on the second method. Specifically, we use the Backward Elimination (BE) stepwise regression method [19] to select our predictors. This method is a well-known variable selection technique commonly used in statistics to select a good set of predictors among many potential predictors for a response variable. We use it because of its simplicity and efficiency.

To apply BE in our data reduction problem, we treat every system metric left after the first step as a potential predictor and the application performance metric as the response variable to be predicted. We start with a model that includes all potential predictors, and at each step, delete one metric that is irrelevant or, given other metrics, is not useful to the model, until all metrics left are statistically significant. More specifically, the algorithm can be described as follows:

1) Build a full model by regressing the response variable on all possible predictors using a linear model:

$$Y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots\beta_nx_n \quad (1)$$

where Y is the response variable, or performance metric in our work; and x_i (i=1…n) are all potential predictors, namely., the independent system metrics considered in our work.

2) Pick the least significant predictor in this model by calculating the F value of every predictor in current model. The F value of a predictor captures its contribution to the model. A smaller value indicates a smaller contribution, thus a less significant predictor. The F value of a predictor x is defined as the result of an F test, which assesses the statistical significance of two different models: one with all predictors considered, called the full model; the other with all the other metrics except the predictor x, called the reduced model. The F value indicates how different the two models are: a small F value means there is little difference between the two models, and thus x does not make a large contribution. Hence. we can remove it without reducing the prediction power of the model.

3) If the smallest F value is less than some predefined significant value (in our case, 2 as suggested by [19]), remove the corresponding predictor from the model. Go to step 4; if not, the algorithm stops. The remaining predictors are considered to be significantly related to the response variable and necessary when predicting the response variable.

4) Re-regress the response variable on all left potential predictors as function 1. Go to step 2.

We note that while the BE regression method is usually employed with a linear model as defined in function 1, it need not be limited to a linear function. For example, we can add a quadratic item for every potential predictor in the model. For each system metric *x*, we not only consider *x* itself, but also treat x^2 as a potential predictor to predict the performance of application. The new regression model is:

$$Y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots\beta_nx_n+\beta_{n+1}x_1^2+\beta_{n+2}x_2^2+\ldots\beta_{2n}x_n^2 \quad (2)$$

Using this model, the BE method will select those system metrics that are either linearly *or* quadratically related to the performance metric. Quadratic terms turn out to be important in the GridFTP data transfer that we consider in Section 4.4.

3.3. Evaluating Data Reduction Strategies

Recall that the general goal of our strategy is to reduce the number of system metrics to be monitored while still capturing enough important information. We use two criteria to evaluate this data reduction strategy.

The *reduction degree* criterion is the total percentage of system metrics eliminated. This criterion measures how many system metrics are reduced by the strategy, so the larger the better. This criterion is used to ensure that we do not leave many redundant or unnecessary metrics in the results.

The *coefficient of determination* [19] criterion, denoted as R^2, uses a statistical measurement that indicates the fraction of the total variability in the response variable (performance of application in our case) that can be explained by the predictors (the system metrics in our work) in a given model. In another words, R^2 measures whether the predictors used in this model are sufficient to predict the response variable. R^2 is a scale-free number, ranging from 0 to 1, the larger the better. A small value of R^2 may indicate that we lost useful information.

R^2 can be calculated by the following formula:

$$R^2 = 1 - \frac{SSE}{SSyy} \quad (3)$$

where, SSE is the error sum of squares = $\Sigma((Yi - EstYi)^2)$, Yi is the actual value of Y for the ith case and EstYi is the regression prediction for the ith case; and SS_{yy} is the total sum of squares = $\Sigma((Yi - MeanY)^2)$.

4. Experimental Evaluation

To evaluate the validity of our data reduction strategy, we ran a series of comparative experiments involving one parallel program running in a shared local area network environment and GridFTP data transfers in a shared wide area network environment.

4.1. Test Applications and Data Collection

We tested our strategy on data collected on one CPU-bound Grid application, Cactus, and on GridFTP data transfers. Cactus [3] is a numerical simulation of a 3D scalar field produced by two orbiting astrophysical sources. The application can run on multiple processors; each processor updates each iteration its local point and then uses MPI to synchronize the boundary. This performance metric is defined as the elapsed time per iteration.

GridFTP [2] is part of the Globus Toolkit [6] and is widely used as a secure, high-performance data transfer protocol. It extends the standard FTP implementation with several features needed in Grid environments, such as security, partial file transfers and third party transfers. The application metric of the GridFTP transfer is the rate with which the data transferred, in megabits per second.

We collected system metrics and application metrics at a frequency of 0.033 Hz. Each data point included one performance metric and roughly 100 system metrics on each machine. All machines were shared with other users during the data collection

We collected system metrics on each machine using three utilities: (1) the *sar* command of the SYSSTAT tool set [1], (2) network weather service (NWS) sensors [20], and (3) the Unix tool *ping*. A detailed description of the system metrics used for each application is available online [21].

4.2. Experimental Methodology

We divided the data collected into two disjoint sets: the training data and the verification data. The first step of our experiment involved using our data reduction strategy on the training data to select a subset of system metrics that are both necessary and sufficient to capture the application behavior. In this step, we also evaluated the influence of the threshold parameter (Section 3.1) on our data reduction strategy, by exhaustively searching the space of feasible selections. We present our results in Sections 4.3 and 4.4.

In the second step of our experiment, we evaluate the efficiency of our strategy using the verification data. We compared the result of our statistical data reduction strategy (**SDR**) to two other strategies. The first method, **RAND**, randomly picks a subset of system metrics equal in number to those selected by our strategy. The second method, **MAIN**, uses a subset of system metrics that are commonly used to model the performance of applications [15]. More specifically, the MAIN metrics for Cactus application included (1) network measurements, including *bandwidth*, *latency* and *the time required to establish a TCP connection* between every pair of the machines; (2) CPU measurements, including the *fraction of CPU available to a process that is already running* and *to a newly started process*, and *the system load average for the last minute* on every machine; (3) *the amount of space unused in memory* on every machine; and (4) *the amount of space unused on the disk* of every machine. For GridFTP data transfer, besides the cited MAIN metrics, disk I/O behavior plays a large role in data transfer time [17]. Therefore, we added disk I/O measurements when applying MAIN strategy to the GridFTP transfer data: (5) *amount of data read from/write to the physical disk per sec* on every machine.

In the second step, we used the coefficient of determination (R^2) to determine whether the system metrics selected on the training data are sufficient to capture the application behavior. As noted above, R^2 indicates the fraction of the total variability in the application performance that can be explained by the system metrics considered. A small value may indicate that the system metrics selected are not sufficient and that the strategy is less effective.

4.3. Cactus Results

We ran Cactus on six shared Linux machines at the University of California, San Diego over a one-day period. Data was partitioned into 12 roughly equal-sized chunks. Every data point comprised the values of a total of 628 system metrics for six machines and one application metric. We used the first chunk of data as the training data to select the necessary and sufficient system metrics, while varying the threshold value to evaluate its influence on the data reduction result. The threshold value was evaluated at intervals of 0.05 between 0 and 1. Two criteria, the coefficient of determination (R^2) and reduction degree (RD), were calculated for every selection, as shown in Figure 2.

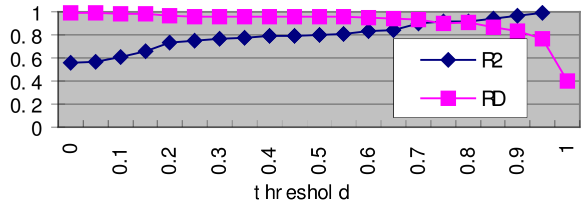

Fig. 2: R^2 and RD as function of threshold value on Cactus data

As expected, when the threshold increases, fewer system metrics are grouped into one cluster and removed as redundant. At the same time, R^2 increases since more information is available to model the application performance. However, when the threshold value reaches 1, dependent system metrics are left as potential predictors in the multiple regression model and we obtain confusing and unreasonable results. The regression fails, and no unrelated system metrics are reduced. This problem is called multicollinearity [19]. The reduction degree decreases dramatically to as low as 40%, and no R^2 value is calculated. Thus, before we begin the BE stepwise to delete unrelated predictors, we must identify and reduce the dependent predictors to avoid the multicollinearity problem.

When the threshold value is equal to 0.95, our strategy produces a reduction degree equal to 0.78 and R^2 as high as 0.98. In other words, 78% of the 628 metrics has been eliminated and a total of 141 system metrics that remain on the six machines (i.e., about 24 per machine on average) can explain 98% of the variation in the application performance metric. Each of these metrics is related linearly to Cactus performance. An example of the system metrics selected on one machine is provided online [21]. We have observed that in addition to CPU, network and memory capability measurements, which are commonly used to model application performance (as selected by MAIN strategy), cache, system page, and signal measurements are also important for modeling Cactus performance.

Although we choose the subset of 141 system metrics for further verification in the next step, we could further increase the reduction degree of our strategy by decreasing the threshold value. For example, the total system metrics can be reduced to 44 on the six machines, with about 7 system metrics per machine on average, when the threshold value is equal to 0.70. The R^2 value decreases to 0.90 in this case, but it is still acceptable. It indicates that 90% of the variation in the application performance metric can be explained by the 44 system metrics selected. We choose the result set that achieves the highest R^2 value because we prefer to keep as many as possible the meaningful event characteristics for our anomaly detection purpose in the future. In the case that the number of system metrics selected is the main concern, we could increase the reduction degree by sacrificing the R^2 value.

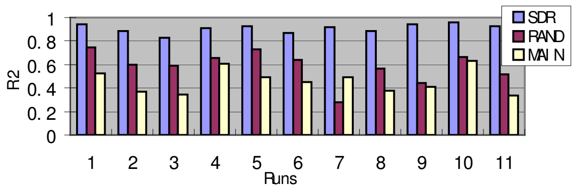

Fig. 3: R^2 values when regressing Cactus performance on the system metrics selected by three strategies

In the second step of our experiment, we validated our strategy by comparing its result to those of other two strategies, MAIN and RAND, using the verification data. The coefficient of determination (R^2) was calculated for every strategy, as shown in Figure 3. We see from Figure 3 that over the 11 chunks of data, our statistical data reduction strategy exhibited an average R^2 value of 0.907, with a range of 0.831 to 0.957. This result is 55.0% and 98.5% higher than those of RAND and MAIN, which have an average R^2 value of 0.585 and 0.457, respectively. We conclude that the system metrics selected by our strategy are significantly more efficient than the alternatives for predicting Cactus performance. We also observed that in many cases, the RAND strategy outperforms MAIN. One possible explanation is that the MAIN metric commonly used to model the performance of Cactus application is far from complete. Thus, it achieves even worse results than randomly picked metrics when to predict the performance of Cactus in a dynamic environment during a relatively long period.

4.4. GridFTP Data Transfer

We ran GridFTP experiments on PlanetLab [4], transferring files ranging from 10MB to 200MB. We ran the server on a node at Harvard University and ran the 25 clients on nodes located in 18 different countries. All nodes are connected via a 100Mb/s network and have processor speeds exceeding 1.0 GHz IA32 PIII class processor and at least 1 GB RAM. Each data point includes values for 217 system metrics on one pair of server and client machines and one transfer rate. All transfers were made with TCP buffers size of 1MB and one TCP socket. A list of the name of the machines used is available on line [21].

We first ran GridFTP using one client at North Carolina State University for one-day period and partitioned the data into 12 roughly equal-sized chunks. We used the first two chunks of data as the training data to select the subset of system metrics, while changing the threshold value from 0 to 1 as we did for Cactus. However, the highest R^2 achieved was only 0.77. This low R^2 value indicates that we lacked some information when modeling GridFTP performance. Thus, we extended the linear model to include quadratic items, as described in Section 3.2.

We redid the experiment using the new model including quadratic items for the GridFTP data. The results are shown in Figure 4. We now obtain far better results. As with Cactus, the reduction degree decreases and the R^2 value increases as the threshold value increases, except that when the threshold value reaches 1, the BE regression fails because of multicollinearity.

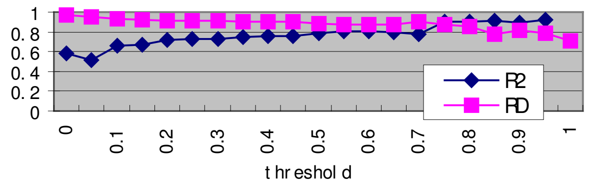

Fig.4: R^2 and RD as function of threshold on GridFTP data

When the threshold value is equal to 0.95, our data reduction strategy achieved a data reduction degree of 0.783 and an R^2 value as high as 0.920, which indicates that 92.0% of the variation in GridFTP performance is explained by the system metrics selected on linear and/or quadratic relations. Of the 217 metrics, 78.3% has been reduced by our strategy and only 47 system metrics in total are left on server and client machines, about 23 system metrics per machine on average. An example of the system metrics selected on server machine is presented online [21]. We have observed in the GridFTP server case that in addition to CPU, network, and disk capability measurements (system metrics selected by MAIN strategy), the memory page, buffer, and cache measurements are important predictors of GridFTP transfer rate. In addition, we note that the memory related system metrics (kbmemfree, kbbuffers, kbcached, frmpg/s, etc.) are quadratic in the regression model. This result may indicate that memory capability is quadratically related to GridFTP transfer rate.

Although we choose the subset of 46 system metrics for further study, we could further reduce the number of system metric selected by decreasing the threshold value. As seen in Figure 4, when the threshold value is equal to 0.75, our data reduction strategy can achieve a reduction degree as high as 0.88 and a R^2 value still equal to 0.898. This result indicates that about 90% of the variation in GridFTP performance can be explained by only 26 system metrics selected.

We then verified the stability of this result in two steps. First, we compared the results of three strategies using the remaining 10 chunks of verification data collected from one client. Then we ran GridFTP using 24 different clients distributed all over the world, and collected two hours of data for each client. At any time, only one client was transferring data. We also verified that the results were consistent for different clients. The coefficient of determination (R^2) of three data reduction strategies were calculated, in both steps, as shown in Figure 5 and Figure 6, respectively.

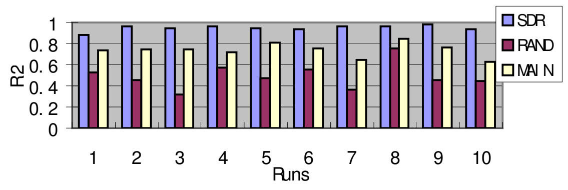

Fig. 5: R^2 value when regressing the transfer rate of GridFTP on the system metrics selected by three strategies on one client data

The linear and quadratic items of every system metric selected were treated as a potential predictor when modeling the GridFTP transfer rate. The results in Figures 5 and 6 show that our statistical data reduction strategy achieves better results than do the other strategies considered for GridFTP transfer data. Over 10 chunks of data collected on one client, our statistical data reduction strategy achieved a mean R^2 value of 0.947 (from 0.884 to 0.982). This result is 92.5% and 28.1% higher than those of the RAND and MAIN strategies, which have average R^2 values of 0.492 and 0.739, respectively. Over 24 chucks of data collected from 24 different clients, our statistic data reduction strategy achieves a mean R^2 value of 0.935. The result is 93.2% and 35.1% higher than those of the RAND and MAIN strategies, which have average R2 value of 0.484 and 0.692 respectively. This consistent result with different clients indicates that the set of system metrics selected by our data reduction strategy is fairly stable with the machines if all machines have the same configurations.

5. Related Work

Data management and reduction have been widely studied in many areas. In the area of application performance monitoring and analysis, event throttling [14] can replace event tracing with less invasive measures like counts. Although throttling prevents generation of large data volumes, it sacrifices the consistent behavior view.

Duesterwald et. al studied the time-varying behavior of programs using metrics derived from hardware counters [5]. However, they focused on several system metrics and assume all system metrics studied were related to the targeted applications. Our work, instead, aims at selecting a subset of related system metrics among a larger set.

Zhang et. al showed how to predict compliance with service-level objectives in a dynamic environment by managing an ensemble of Bayesian network models [22]. Their strategy includes a process called feature selection, for selecting the subset of metrics that are most relevant to modeling the relations in the data.

Dynamic clustering [13] and statistical sampling [11] allow the analysis to focus on subsets of processors or metrics, thus reducing the data to be collected. But their usage is limited to simple cases, such as finding free nodes and estimating average load.

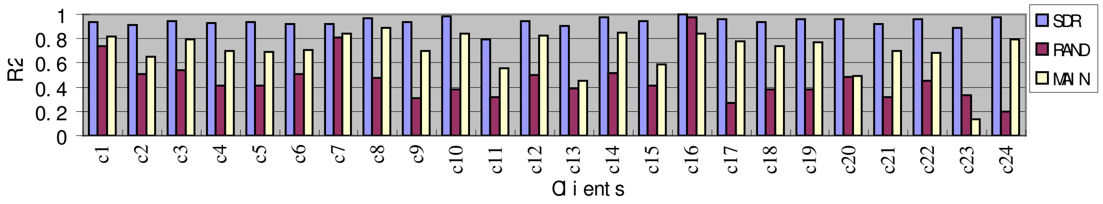

Fig. 6: R^2 value when regressing the transfer rate of GridFTP on the system metrics selected by three strategies on data collected from 24 different clients

Two approaches that are similar to the work described in this paper are correlation elimination [8] and projection pursuit [18], both of which identify a relevant, statistically interesting subset of system metrics. Correlation elimination [8] diminishes the volume of performance data by grouping metrics with high correlation coefficient into clusters and picking only one as a representative. However, that work assumes that all metrics collected are related to the performance metrics. Thus, only redundant metrics are reduced by using the correlation elimination technique, as in the first step of our data reduction strategy. Our strategy further improves the correlation elimination technique by using a statistical Z-test instead of a pure mathematical comparison when trying to group two metrics into one cluster.

Projection pursuit [18] focuses performance analysis on interesting performance metrics. However, projection pursuit selects those metrics from all smoothed input data at some discrete point in time, and thus captures only transient relationships between data. The cited work presented only data reduction results using a total of 12 system metrics, while our strategy tries to capture inherent relationships between data and the system metrics selected by our strategy are able to capture the application performance variation over a longer time and for a much larger metrics set.

6. Conclusions

The work described in this paper comprises two steps. First, we show how to reduce redundant system metrics using correlation elimination and a Z-test. The result of this step is a set of independent system metrics. Then, we show how to identify system metrics that are related to an application performance metric by a stepwise regression-based technique. We have applied our data reduction strategy to data collected from two applications. We find that our strategy reduces about 80% of total system metrics and that the remaining system metrics can explain application performance variation as high as 90%, 35% to 98% more than metrics selected by other strategies.

Acknowledgements

This work was supported in part by the U.S. Dept. of Energy under Contract W-31-109-Eng-38.

References

[1] SYSSTAT utilities home page http://perso.wanadoo.fr/sebastien.godard/.
[2] Allcock, B., Foster, I., Nefedova, V., et al., High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. *SC'01*, 2001.
[3] Allen, G., Benger, W., Dramlitsch, T., et al., Cactus Tools for Grid Applications, *Cluster Computing*, 4 (2001) 179-188.
[4] Bavier, A., Bowman, M., Chun, B., et al., Operating System Support for Planetary-Scale Network Services. *1st Symp. on Network System Design and Implementation*, 2004.
[5] Duesterwald, E., Cascaval, C. and Dwarkads, S., Characterizing and Predicting Program Behavior and its Variability. *12th International Conference on Parallel Architecture and Compilation Techniques*, 2003.
[6] Foster, I. and Kesselman, C., The Globus Project: A Status Report. *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998.
[7] Jain, A.K., A Guideline to statistical approaches in computer performance evaluation studies, *ACM SIGMETRICS Performance Evaluation Review*, 7 (1978) 18-32.
[8] Knop, M.W., Schopf, J.M. and Dinda, P.A., Windows Performance Monitoring and Data Reduction Using WatchTower. *SHAMAN*, 2002.
[9] Liu, C., Yang, L., Foster, I., et al., Design and Evaluation of a Resource Selection Framework for Grid Applications. *HPDC 11*, 2002.
[10] Malony, A.D., Reed, D.A. and Wijshoff, H.A.G., Performance Measurement Intrusion and Perturbation Analysis, *IEEE Trans. Parallel and Distributed System*, 3 (1992) 433-450.
[11] Mendes, C.L. and Reed, D.A., Monitoring Larger Systems Via Statistical Sampling. *LACSI Symposium*, 2002.
[12] Nabrzyski, J., Schopf, J.M. and Weglarz, J., *Grid Resource Management:State of the Art and Future Trends*, Kluwer Publishing, 2003.
[13] Nickolayev, O.Y., Roth, P.C. and Reed, D.A., Real-Time Statistical Clustering For Event Trace Reduction, *The International Journal of Supercomputer Applications and High Performance Computing*, 11 (1997) 144-159.
[14] Reed, D.A., Aydt, R.A., Noe, R.J., et al., Scalable Performance Analysis: The pablo Performance Analysis Environment. *Scalable Parallel Libraries Conference*, 1993.
[15] Ripeanu, M., Iamnitchi, A. and Foster, I., Performance Predictions for a Numerical Relativity Package in Grid Environments, *International Journal of High Performance Computing Applications*, 15 (2001).
[16] Stockburger, D.W., *Introductory Statistics: Concepts, Models, and Applications*, 1996.
[17] Vazhkudai, S. and Schopf, J.M., Using Disk Throughput Data in Predictions of End-to-End Grid Data Transfers. *Grids2002*, 2002.
[18] Vetter, J.S. and Reed, D.A., Managing Performance Analysis with Dynamic Statistical Projection Pursuit. *SC'99*, Portland, OR, 1999.
[19] Weisberg, S., *Applied Linear Regression*, 2 edn., John Wiley &Sons, 1985.
[20] Wolski, R., Dynamically Forecasting Network Performance Using the Network Weather Service, *Journal of Cluster Computing* (1998).
[21] Yang, L., Document: http://www.cs.uchicago.edu/~lyang/work/MetricsList.doc
[22] Zhang, S., Cohen, I., Goldszmidt, M., et al., Ensembles of Models for Automated Diagnosis of System Performance Problems. *IEEE Conference on Dependable Systems and Networks (DSN)*, 2005.