Using Bandwidth Data To Make Computation Offloading Decisions

Rich Wolski, Selim Gurun, Chandra Krintz, and Dan Nurmi
Computer Science Dept., Univ. of California, Santa Barbara

***Abstract*— We present a framework for making computation offloading decisions in computational grid settings in which schedulers determine when to move parts of a computation to more capable resources to improve performance. Such schedulers must predict when an offloaded computation will outperform one that is local by forecasting the local cost (execution time for computing locally) and remote cost (execution time for computing remotely and transmission time for the input/output of the computation to/from the remote system). Typically, this decision amounts to predicting the bandwidth between the local and remote systems to estimate these costs. Our framework unifies such decision models by formulating the problem as a statistical decision problem that can either be treated "classically" or using a Bayesian approach. Using an implementation of this framework, we evaluate the efficacy of a number of different decision strategies (several of which have been employed by previous systems). Our results indicate that a Bayesian approach employing automatic change-point detection when estimating the *prior* distribution is the best-performing approach.**

I. Introduction

In computational grid settings [2], [8], when a new computation is initiated, the "scheduler" (either a human user or an automatic system scheduler) must often decide whether to run the computation locally, or to offload it to a more powerful remote resource. The advantage of executing locally is that the computation can be initiated immediately, with all of its input data in place. Alternatively, to gain the performance advantage offered by a faster remote machine, the input and output data must be moved to and from the remote site respectively adding an additional overhead. Thus, the scheduling decision in the simple offloading scenario is based on whether the additional performance offered by the remote system will be overshadowed by the cost of the additional data movement. If the cost is higher, then a local execution will be faster. If it is not, then remote execution yields the faster execution time. This decision is particularly important for systems that implement variants of Grid RPC such as GridSolve [1], [25], Ninf [19], OmniRPC [18], and others [10].

These systems, however, make this decision in some implementation-specific way even though the decision problem is essentially the same. In this paper, we present a methodology for making computation offloading decisions that we believe is general enough to unify the different approaches implemented by most extant grid RPC systems (at least for the instances where the computational resources are accessed interactively). We describe our methodology and then detail its effectiveness using performance measurement traces gathered from "live" execution settings. Our results show that while different individual approaches may work well in specific circumstances, a comprehensive approach based on Bayesian risk evaluation [12] is consistently the most successful.

This work was sponsored, in part, by grants from the National Science Foundation numbered 0627183, 0444412, and 0331645

Specifically, the key observation we make is that the offloading decision is typically based on three predictions:

- a prediction of the time required to execute the computation locally,
- a prediction of the time required to execute the computation remotely (once the data is available at the remote site), and
- a prediction of the time required to move the input data from the local site to the remote site, and to gather the results back to the local site.

Moreover, for many scientific applications and libraries (e.g. LINPACK [7]) high-quality execution time predictions for many machines and architectures are readily available [5], [20]. Predicting the transfer time, however, typically requires a prediction of network bandwidth which many systems make by analyzing on-line network measurements (e.g. by predicting future bandwidth from historically observed network performance) [11], [13], [23].

Typically, the decision algorithm first predicts the bandwidth between the local execution site and the remote site. Using that bandwidth prediction, it then computes the predicted execution time associated with transferring the program data, executing remotely, and gathering the results. This overall time is then compared with the local time prediction (that does not depend on available bandwidth) and whichever is lower indicates the decision to be taken.

In terms of making a "yes/no" (i.e. local or remote execution) decision, if the bandwidth data can be considered to be modeled in some way by a random process (as is typical), this prevalent methodology can be considered a specific instance of a more general Bayesian decision problem. In such a problem formulation, the bandwidth is a phenomenon that is being modeled as a random variable, and each predictor provides a *belief* or *hint* that tempers the conditional distribution on that variable (termed a *posterior* distribution), when computing the expected risk associated with the decision either to offload or to keep the computation local. We compare the usage of various predictors as *ad-hoc* conditional expectation generators to an implementation of the full Bayesian problem formulation. We also examine the value of automatic, on-line, change-point identification in the bandwidth data time series as a method of improving the accuracy of the offloading decision mechanism.

The advantage of this more general approach is that it formally admits the notion of "update" to the state of the prediction system as new data becomes available. On-line prediction techniques like those described in [11], [13], [23] usually incorporate new data (e.g. new bandwidth measurements) as soon as they become available. Every time a new measurement is incorporated into the predictor's state, a new prediction is possible. The weight given

to each new prediction, however, modifies this new prediction in different ways. As a Bayesian decision, however, the computation of the new prediction is a well-defined function of the previously computed prediction. Thus the prediction made by each individual predictor is incorporated into the decision-making process in a uniform way. That is, the methodology we present, and the implementation of it we explore, incorporates the existing predicting methods into a single, on-line prediction framework that produces offloading decisions in this setting.

In summary, the contributions that we make in this paper are:

- a formulation of computation offloading as a statistical decision problem that provides a unified method for incorporating and comparing different network bandwidth predictors,
- a description of an implementation of this methodology that functions on-line and that can be parameterized with different individual prediction techniques,
- an evaluation of the approach using this implementation using a pair of network bandwidth measurement traces,
- a performance evaluation comparing various prediction methods documented in the literature and a number of different decision strategies based on these predictions.

The remainder of the paper is organized as follows. We first describe the offloading problem and its use in modern systems. In addition, we articulate the generality of the statistical decision approach we take. Section III describes the various decision strategies we explore that are based on different techniques for making on-line predictions from bandwidth data. In Section IV we analyze a series of experiments that compare the performance of these strategies. We then present our conclusions in Section V.

II. Computation Offloading

Computation offloading is a technique for improving the execution efficiency of a distributed system by moving computation from a less-capable system to one that is more capable. This *capability* can take the form of compute power, memory, system load, as well as battery lifetime. Offloading is also referred to in the literature as remote execution and is employed in computational grid systems to improve the performance of an application typically through the use of Grid RPC [1], [10], [18], [19], to speed execution through parallelization and improved task execution performance. Offloading has also been shown to be effective for mobile, resource constrained devices, to extend battery life and computational capacity [6], [9], [11], [15], [17], [26].

Figure 1 shows the conceptual design of a simple computation offloading system with one client (local computer) and one server (remote computer). The scheduler must consider the expected cost of both local and remote execution, requiring accurate predictions of the demand and supply of several parameters both at the local and the remote device during the lifetime of the task. There are three fundamental parameters that the decision maker has to predict: time needed to execute computation locally, time needed to execute computation remotely (excluding the cost of data transfer), and time needed to move input data and results back and forth between the local and remote computers.

In this work, we investigate a framework for decision making strategies that attempt to maximize expected offloading performance by using network bandwidth predictions the most effective way. In particular, we assume that the local and remote task

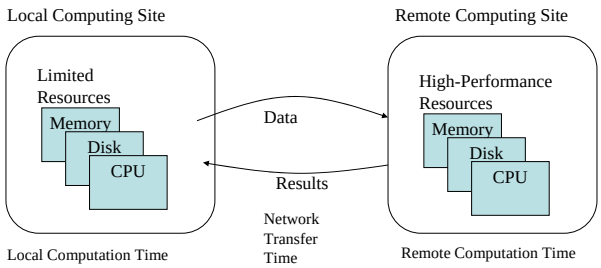


Fig. 1. Components of a Typical Offloading System. A decision process forecasts network bandwidth to decide whether it is beneficial or not to offload the computation to the remote system.

execution time is known by the scheduler. This assumption is often true in scientific computing settings, particularly for calls to highly tuned numerical libraries such as LAPACK [5]. We refer to task execution time (in seconds) on the local machine as C_l.

We define the ratio of local and remote execution, excluding the network transfer cost, to be α. α is a unit-less metric that reflects the fraction of local execution time that remote execution takes if the task data is readily available at the remote system. Consequently, $1/\alpha$ is the speed-up (i.e., how fast the remote device is), relative to the local device (excluding transfer delay). We assume that the total amount of data that has to be transferred between client and server, s (measured in bits), is also known. If the network bandwidth between the client and server during the computation offloading is β, then, we can compute the expected cost of remote execution $C_r(\beta)$ as:

$$C_r(\beta) = C_l\alpha + \frac{s}{\beta} \tag{1}$$

A computation offloading is beneficial only when $C_l > C_r(\beta)$. Thus, a decision manager should offload if;

$$C_l > C_l\alpha + \frac{s}{\beta} \tag{2}$$

which we can rewrite as:

$$\beta > \frac{s}{C_l - \alpha C_l} \tag{3}$$

Note that our cost formulas 1 and 2 are only approximations as in reality other factors such as communication and computation overlap impact the cost of remote execution. In this paper we ignore such factors since our primary goal is to compare decision making strategies and not to estimate the actual cost.

We define the smallest β that makes the inequality above true the *critical network bandwidth* and we denote it as the threshold value T. For a given local execution cost C_l, transfer size s, and speed ratio α, if the network bandwidth is larger than the critical network bandwidth T, the cost of execution is minimized by offloading the task. Otherwise, it is minimized by local execution of the task. We use this fundamental cost model to explore different decision making strategies presented in the rest of this work. In some sense, each strategy represents a different method for estimating the relevant parameters in this model so that the *expected* execution cost is minimized.

III. Computation Offloading As a Statistical Decision Problem

In this section, we discuss several decision strategies that are based on the cost model described in the previous section. While many of these strategies have been implemented previously in

different settings [6], [9], [10], [17]–[19], [26], they are all instances of either classical or Bayesian statistical decision-making procedures [12]. As such, they can be represented and analyzed using a common nomenclature, mathematical framework, and software tool base. Thus it is possible to unify the various approaches to minimizing execution time through computation offloading. We begin by formulating the computation offloading process as a statistical decision problem. We then discuss each strategy in detail.

A. Offloading as A Classical Statistical Decision Problem

The simplest form of the decision problem requires that the decision maker compute the expected local and remote costs at each point where offloading is possible, and to choose the decision corresponding to the lowest expected cost. With C_l known, the problem is to compute the expected value of $C_r(\beta)$ as a function of the available network bandwidth as:

$$E[C_r] = \int C_r(\beta) * f(\beta) d\beta \tag{4}$$

where $f(\beta)$ is the probability distribution function (PDF) describing the available bandwidth β. Since C_l is constant, $E[C_l] = C_l$. Thus an offloading decision is indicated when $E[C_r] < C_l$.

Determining $f(\beta)$, however, is difficult. A large body of research has focused on determining good probabilistic models for network bandwidth [3], [16], [21] with little agreement on the best method, and some indication that purely analytical model may not be possible. Moreover, in an on-line execution setting – one where the decisions are being made "instantaneously" while the application is executing – new information becomes available as time progresses. The strictly classical formulation of the problem assumes that this new information does not indicate change in the underlying dynamics of the system. That is, new bandwidth measurements are assumed to be samples from the distribution $f(\beta)$ and that $f(\beta)$ does not change over time. Additionally, the bandwidth distribution that is required here is for *end-to-end* throughput which is subject to interactions with the operating system scheduling discipline on both ends, buffer availability, interrupt scheduling, etc.

Practically, one approach to estimating the potentially time-varying distributional properties of end-to-end bandwidth is to use a periodic series of measurements from which the distribution is estimated empirically. Further, as new measurements are gathered, the estimation is updated. In this paper, we will assume such a methodology is in place. Namely, we assume that historical measurement data (either gathered from a periodic probe or via the logging of previous activity) is available. Moreover, as time progresses and new data becomes available, that data is incorporated as soon as possible.

Often, $E[C_r(\beta)]$ (c.f. Equation 4) is computed as $C_r(E[\beta])$. That is, the expected cost is computed as the execution cost derived from the expected value of the bandwidth. However, because $C_r(\beta)$ is linear in the inverse of the bandwidth value (and not the bandwidth value itself) this substitution is not valid for the cost formulation we have chosen. Thus, we maintain a histogram of previously observed bandwidths to compute an estimate of the integral in Equation 4 using $\sum[C_r(\beta_i)P_i]$. Here, P_i is the relative frequency of the bandwidth observations that are in bin i of the histogram, and β_i is the corresponding bandwidth (of that bin). We update the histogram as new bandwidth values are observed.

Strictly speaking, because new data is incorporated in the histogram that we use to approximate $f(\beta)$, the problem might not be viewed as a "classical" statistical decision problem since there is a data update phase in the procedure. Each time a decision is made, however, the formulation simply uses its "best estimate" at that moment for $f(\beta)$ to compute the expected cost. That is, each decision is "classical" even though the series of decisions is essentially based on a time series of conditional distributions. Since this methodology is often used in practice (i.e. the average bandwidth is commonly, and perhaps incorrectly, used in the cost calculation) we feel it is important to classify. In terms of nomenclature it is truly a "best effort" attempt to implement a statistical decision procedure that is based on an estimate of the distributional properties associated with a single random variable. In this sense, the methodology is most properly categorized as "classical." Perhaps to avoid such ambiguity, this approach is also often termed a *Data* decision strategy since it uses the data directly to compute expectations of risk. We will henceforth refer to the classical strategy as the *Data* strategy to follow suit.

B. Offloading as a Bayesian Decision Problem

Because approximating $f(\beta)$ can be so challenging, a number of previous approaches have employed prediction techniques (often statistical) to forecast future bandwidth values. The cost functions are then computed from the forecasts in a decision strategy. Again, because the dynamics of end-to-end network bandwidth have yet to be fully understood, these forecasting techniques are usually evaluated empirically and not analytically [6], [9], [11], [17], [23], [24]. As such, however practically successful they may be, in a formal setting they can be viewed as heuristics that temper or weight the perceived probability associated with future bandwidth values. From this perspective, then, "black-box" predictors generate belief that weights the probabilities described by $f(\beta)$ leading to a Bayesian formulation of the decision problem.

The Predictor Strategy: The most straightforward approach is simply to accept the output of any predictor as having probability 1.0 of being correct. That is, to "believe" that the predictor is correct and to ignore the previously observed bandwidth values. We term this strategy the Predictor strategy.

The Predictor strategy uses the network forecast directly in decision making. For a certain task, if the network bandwidth forecast is larger than the critical network bandwidth T, it offloads the task to a remote device, otherwise, it chooses local execution. That is, in Predictor strategy the decision algorithm is "yes" if the predicted bandwidth is $>$ T and "no" otherwise.

When using the Predictor strategy, the critical question concerns the degree to which a given predictor improves overall execution performance and whether there is a significant difference in efficacy from one predictor to another. A popular way of comparing network predictors is to use them *post facto* using a trace of network observations that are collected from real and/or simulated networks. Let x be the observed network bandwidth, y be the corresponding prediction and n be the number of observations. To compare network predictors, one can compute the *mean square error* over the observed and predicted values. Mean square error (MSE) is the average of the square prediction error; i.e. $MSE = (\sum_i (x_i - y_i)^2)/n$. The squaring of the prediction error, $(x - y)$, allows MSE to be particularly sensitive

for large incorrect predictions. It is generally assumed that the lower the MSE, the better is the predictor.

In computation offloading, however, MSE is not necessarily the best metric to measure predictor performance. Referring back to Equation 3, the goal is to estimate when the network bandwidth is less than, or over the critical network bandwidth, T. A predictor that has a larger MSE could be better than (for the purpose of offloading) another predictor that has a lower MSE, as long as the one with larger MSE estimates when network bandwidth is less or more than the critical network bandwidth more accurately. In other words, for computation offloading, a less accurate prediction that none-the-less indicates a correct decision is preferable over a more accurate prediction that indicates an incorrect decision. In contrast, MSE is a symmetric metric that measures the distance between the predicted and the actual value quadratically, and therefore, it cannot capture the impact of predictor performance in an offloading setting.

In the next subsection, we evaluate a wide range of predictors in terms of their offloading efficacy and explore the differences between sophisticated and adaptive predictors and simple, parameterized predictors in a computation offloading setting.

Bayesian Risk and Bayesian Strategies: The computation offloading systems that rely solely on network predictors suffer from an important problem. The Predictor strategy does not take into account how much a risk one takes by believing the network predictor (i.e. it assumes a network predictor is correct 100% of time). A Bayesian approach allows us to compute the expected risk we take by believing the predictor, which we can then use to give our decisions.

Assume that $C_r(\beta)$ and C_l are defined as in the previous section, and $f(\beta|\beta_p > T)$ is the probability of observing a network bandwidth value β, given the network forecast β_p is larger than T. The Bayesian risk is the expected cost as a function of the *posterior* distribution associated with offloading:

$$R_r = \int C_r(\beta) f(\beta|\beta_p > T) d\beta \tag{5}$$

Similarly, the risk that we take by locally executing the computation is:

$$R_l = \int C_l f(\beta|\beta_p < T) d\beta \tag{6}$$

Note that, it may not be immediately apperant why we compute a risk factor R_l for local execution. In computation offloading, the goal is to choose the execution site (local vs. remote) that minimizes execution cost. If we choose remote execution, and if the network bandwidth turns out to be smaller than T, local execution becomes cheaper, and we pay a penalty for remote execution decision. In the same way, if we choose local execution, and if the network bandwidth turns out to be larger than T, remote execution becomes cheaper, and we pay a penalty for local execution decision. Since both decisions include a potential penalty, we have to compute the risk for both before giving a decision.

Thus, while computing R_r, we need only consider the integral from 0 to T computationally, because, when β is larger than T, there is no penalty involved in offloading the computation (i.e. remote execution is the lower cost decision). In the same way, while computing R_l, we take the integral starting from T to infinity, because, when β is smaller than T, there is no penalty involved in locally executing the computation. Also notice that in this case, R_l is not constant, even though C_l is.

We can compute $f(\beta|\beta_p > T)$ and $f(\beta|\beta_p < T)$ using Bayes Theorem:

$$f(\beta|\beta_p > T) = \frac{f(\beta) f(\beta_p > T|\beta)}{f(\beta > T)} \tag{7}$$

$$f(\beta|\beta_p < T) = \frac{f(\beta) f(\beta_p < T|\beta)}{f(\beta < T)} \tag{8}$$

In the equation above, $f(\beta)$ is the PDF associated with network bandwidth (as before), $f(\beta_p > T)$ is the PDF associated with the predictor predicting a value above threshold T, and $f(\beta_p > T|\beta)$ is the conditional PDF for a prediction above T given an observed bandwidth value β. Thus, the conditional PDF for bandwidth given a prediction β_p is greater than T is the *posterior PDF* based on the *prior* PDF for bandwidth ($f(\beta)$), the conditional PDF on the predictor given a bandwidth value ($f(\beta_p > T|\beta)$) and the absolute PDF on the predictor itself ($f(\beta > T)$). The same is true for a prediction β_p being less that T except that the inequalities are reversed.

In an implementation of this Bayesian formulation, we maintain a histogram for the *prior* PDF $f(\beta)$ as described previously. We also implement an array of histograms (one per range of possible bandwidth values) for the conditional PDF $f(\beta_p > T|\beta)$ and a single histogram for the absolute PDF $f(\beta_p > T)$. That is, whenever a bandwidth value β is observed, we determine what the predictor indicated just before the bandwidth value was recorded with respect to T. The fraction of correct indications is maintained in a bin indexed by β to implement the conditional PDF for bandwidth. Similarly, a single fraction capturing the proportion of correct indications by the predictor is maintained as the absolute PDF. Finally, to estimate the Bayesian risk values described in Equations 5 and 6, we first compute the empirical *posterior* PDFs shown in Equations 7 and 8, and then compute expectations using these *posterior* PDFs as weighted sums.

To better explain this, assume that we use a histogram that has 10 bins and our maximum network bandwidth is 100Mb/s. In this case, the first bin of the histogram that is associated with $f(\beta)$ gives the prior probability of observing a network bandwidth that is less than 10Mb/s, the second bin gives the prior probability of observing a network bandwidth that is larger than 10Mb/s but less than 20Mb/s, and so forth. For computing $f(\beta_p > T|\beta)$, we use one histogram with only two bins (one for condition $\beta_p \leq T$ and the other for condition $\beta_p > T$) per each of the 10Mb/s range. As an example, assume that T is larger than 20 Mb/s. If a network observation is between 10Mb/s and 20Mb/s, and the corresponding network prediction is less than T, we update the bin that counts the condition $\beta_p \leq T$ of the 2^{nd} histogram. Similarly, the histogram for the $f(\beta > T)$ also has two bins, each of which giving the probability of observed value being less than or more than T. We update this histogram the same way after each observation.

Notice that the "machinery" necessary to compute expectations based on empirically determined *posterior* PDFs is somewhat involved. If the data is correlated (and it is, in this case) and slowly changing, it might be reasonable to assume that the *posterior* PDFs required for the next offloading decision are similar to the ones observed after the fact for the previous decisions. To investigate this possibility, we also implement an Observed Bayes strategy in which we simply record (again as empirical PDFs) $f(\beta|\beta_p > T)$ and $f(\beta|\beta_p < T)$. That is, rather than computing the next *posterior* PDF based on current observations of the bandwidth and the predictor, we use the results of the previous

outcomes to record previously observed conditional distributions of bandwidth based on predictor indication. If the "look ahead" that the full *posterior* PDF computation is not important, the Observed Bayes strategy should be as cost efficient (or more cost efficient since fewer PDFs are being estimated) than the computed Bayes Strategy with a significantly less complex implementation.

Bayes Strategy Incorporating Change-point Detection: Notice that in all strategies discussed so far except for the Predictor strategy, the estimation of the bandwidth PDF $f(\beta)$ is critical since it is necessary to compute expected risk. However, the end-to-end bandwidth that is available between two machines may change over time in a way that is better described by a series of PDFs, each estimating $f(\beta)$ over a given time period. In previous work [4], [14] we describe a method for detecting change-points in highly autocorrelated, highly variable time series. This methodology is based on a non-parametric quantile estimation technique that uses empirically computed Binomial distributions to identify unlikely sequences of values. When a sequence that would have a low probability of occurring given the quantile estimates occurs, the system indicates that the series has crossed a change point. All data before the change point is then discarded and new data is used to generate future estimates.

In this work, we apply our previously developed methodology to the problem of estimating $f(\beta)$ as a function of time. Specifically, we compute quantile estimates for the cumulative distribution function (CDF) on bandwidth using the Binomial method with change-point detection enabled. We then numerically differentiate the CDF to generate an instantaneous approximation of the PDF $f(\beta)$. The result is a PDF representation that only takes into account bandwidth data occurring since the last change point, and which has been estimated using the Binomial quantile estimator (which we have observed to be robust with respect to high levels of autocorrelation). We then use this time-sensitive $f(\beta)$ to compute the *posterior* PDFs required to implement a full Bayes strategy.

A No-Data Strategy: Finally, it is important to compare strategies (classical or Bayesian) that are based on risk expectations derived from bandwidth observations, to a strategy that does not rely on a direct probabilistic characterization of bandwidth. We term such a strategy a *No Data* strategy since it does not rely on estimating a PDF on the "data" (which is bandwidth data in this case).

At the time of an offloading decision, let E_l be the average risk associated with *always* making a local decision before each previously observed bandwidth measurement occurred. Similarly, let E_r be the average risk associated with choosing a remote execution are each point in time that has occurred previously. The No-Data strategy chooses local execution, if $E_l < E_r$ and remote execution if $E_l > E_r$. That is, if "on the average" it has been better to choose local execution over remote execution in the past, choose local, otherwise, choose remote. To compute the expected cost of local and remote execution decisions, the No-Data strategy simply keeps track of cost of wrong decisions for both local and and remote execution decisions each time a new bandwidth value is observed.

IV. Evaluation

In this section, we evaluate the efficacy of computation offloading strategies that we described in the previous section. We couple each strategy with a wide range of popular prediction algorithms, and use trace based simulation to compare their performance over two large network traces that we collected from real networks. Our preliminary results show that *Bayes+Change Point* strategy (c.f. Subsection III-B) is significantly more powerful than its competitors, regardless of the specific predictor that it uses.

TABLE I

The network predictors that we evaluate.

Name	Description
Last	Last value
Avg	Running mean filter
ODY	Exponential smoothing with gain 0.875
FlipFlop	Adaptive filter combining two predictors [11]
NWS	Network weather service forecaster [22]
NwsLite	Extension to NWS for resource-restricted devices [9]

TABLE II

The bandwidth traces that we consider.

	Trace Size	Avg Bw	Network Configuration
Trace1	59101	48.9	100 Mb/s
Trace2	11316	553.9	1 Gb/s

A. Experimental Methodology

Table I presents the predictors that we employ within our framework as part of this evaluation. *Last* is a Last Value predictor, it uses the last observation as a forecast for next measurement. Last, thus, is very responsive to sudden changes but very susceptible to noise in the measurements. *Ody* is an exponential smoothing predictor. We employ a gain factor of 0.875 which enables Ody to filter noise while responding quickly to the changes over time. *FlipFlop* is an extension of the Ody predictor that consists of two exponential smoothing filters, one agile and one stable, and a statistical control component. Both filters run concurrently and the control component selects the best-performing filter for each prediction. *Avg* is a running mean predictor. NWS [23] is the forecasting component of Network Weather Service [24]. NWS prediction uses a mixture-of-experts approach to prediction. It implements a large set of time-series forecasters, each having its own parameterization. NWS keeps histories of observed measurements of different windows and runs all models simultaneously for each window size. A control component selects the forecaster with the lowest error for each prediction it makes. NWSLite [9] is a scaled down version of the NWS that trades off prediction overhead (execution time cost) for predictor accuracy.

We evaluate each strategy and predictor using two real network traces. These traces include network bandwidth measurements that we collected using NWS sensor probes. Each probe is 512K bytes in length and measures TCP/IP performance between two computers. We show the basic characteristics of data traces in Table II. The first trace is for a 100Mb/s network and includes 59101 network observations. The second trace if for a 1Gb/s network and includes 11316 network observations.

In our simulations, we emulate the transfer of a task's input/output data over the network. We assume all task parameters except network bandwidth is known; we choose the task parameters according to the following: α, the speed-up ratio is equal to 0.25, i.e., the remote computer processes computation four times faster than the local device. C_l is local execution cost is 100 seconds. We fix these parameters and vary task size to evaluate

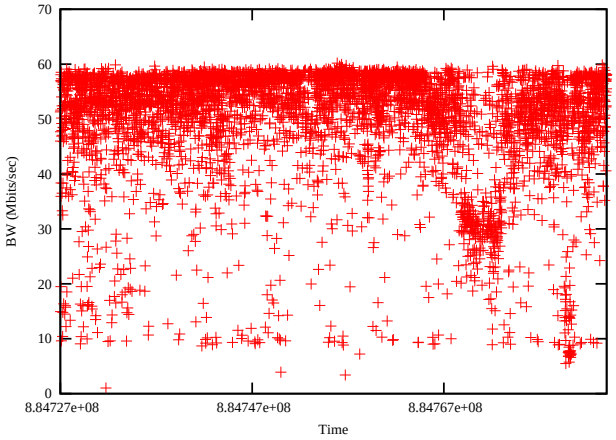


Fig. 2. Network bandwidth observations for 100Mb/s trace for first 5000 observations.

the performance of offloading strategies. We use the first 5000 network observations as a warm-up period for our strategies to construct the histograms from which we compute the probability density functions (PDFs), and use the remaining trace to evaluate our predictors.

There are two implementation specific parameters in our evaluation. The first is the number of bins for the histogram to approximate each PDF; the second is the number of quantiles that our framework uses in the CDF for the change-point strategy. We empirically identified the value of 20 for both as the value at which the improvement in accuracy levels off.

We compare the performance of predictors using a *regret* metric. Regret is the penalty that an offloading system pays for a wrong decision. Total regret is the sum of the regret across all decisions made by a decision strategy across a network trace. Total regret is zero when there are only correct decisions and greater than zero when there are one or more incorrect decisions. We compute regret as the absolute value of the difference between the cost of local and remote execution.

B. 100 Mb/s Network Experiments

We first present the performance characteristics of the 100Mb/s network trace. Figure 2 shows the first 5000 observations which are representative of the entire trace. The y-axis is network bandwidth and the x-axis is time. Most of the observations range from 50-60 Mb/s with occasional drops (i.e. one such drop to 30 Mb/s is visible to the right of the figure). The bandwidth measurements are almost always higher than 10Mb/s, although there are several observations that are much lower; i.e. closer to 1Mb/s range.

We next evaluate the performance of various decision strategies using our *regret* metric. Figure 3 presents the regret for the different offloading strategies when we vary task size. The y-axis is regret, in seconds, and the x-axis is task size. For each graph, we present the Bayes strategy alone, Bayes with change-points (Bayes+CP), the predictor alone (Predictor), and the no-data (No Data) and data (Data) decision strategies. We present two graphs, for two different predictors: NWS prediction in (a) and Odyssey (Ody) prediction in (b). We omit the graphs for the other predictors due to space constraints; however, these predictors are representative of the others. Note that the Data and No Data strategies do not use predictors; we include them here for comparison. We include all predictors in our evaluation of the different Bayes strategies in the next section.

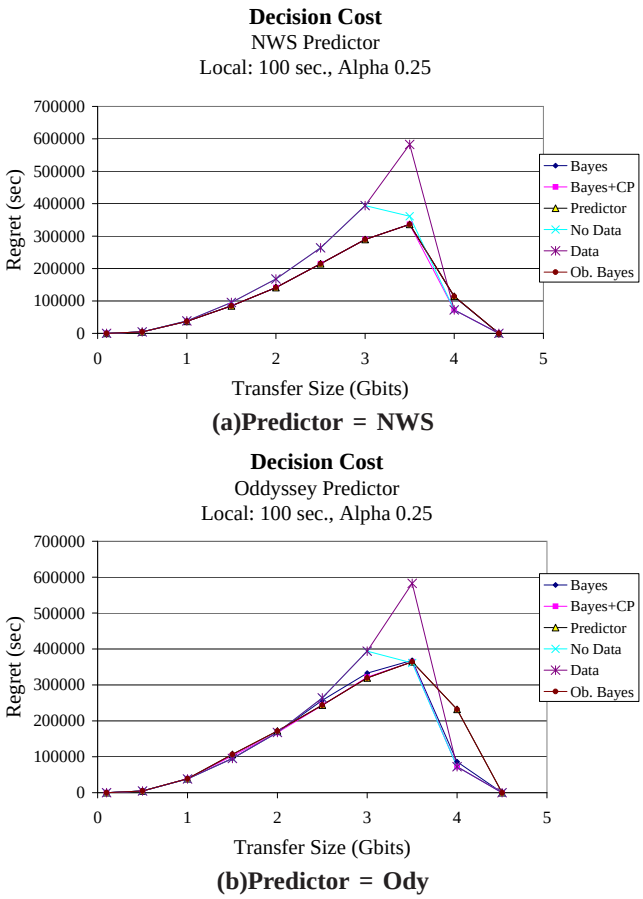


Fig. 3. Predictor performance (regret) for different task sizes and decision strategies for the 100Mb/s trace for predictors NWS (a) and Ody (b).

We observe that the task size has a significant impact on offloading strategy efficacy. As the task size increases from 0.1 Gb (i.e. gigabits) to 4 Gb, regret increases to a peak at approximately 3.5 Gb, then decreases. That is, the various strategies perform relatively worse for task sizes that range from 3 to 4 Gb. This is because, these task sizes correspond to a critical network bandwidth values in the range of 45 to 55Mb/s.

In terms of relative performance, the *Bayes+CP* strategy matches or exceeds all other strategies in all cases. For smaller transfer sizes (size $\leq$ 3.5 Gb) *Bayes* and *Predictor* strategies match the performance of *Bayes+CP* strategy. For larger sizes, they perform worse. The *No Data* strategy performs worse than *Bayes+CP* for smaller tasks, and matches it for larger sizes. NWS prediction strategies perform significantly better than Ody. The *Data* strategy is the worst strategy overall as its use results in the most regret for both predictors, for all task sizes.

We next zoom in on the critical bandwidth range to analyze the prediction-based strategies in greater detail. In particular, we consider a transfer size of 4 Gb which equates to the critical bandwidth 53.3Mb/s, in Figure 4. We present the regret for the strategies (Predictor, Bayes, Observed Bayes (Ob. Bayes), and Bayes with change-points (Bayes+CP) using all of the different predictors: NWS, NWSLite, Ody, FlipFlop, Last, and Avg. We omit Data and No Data strategies since they do not make use of network predictors for their decision.

The non-adaptive predictors, Last and Ody, perform significantly worse than adaptive predictors, NWS, NWSLite and Flip-

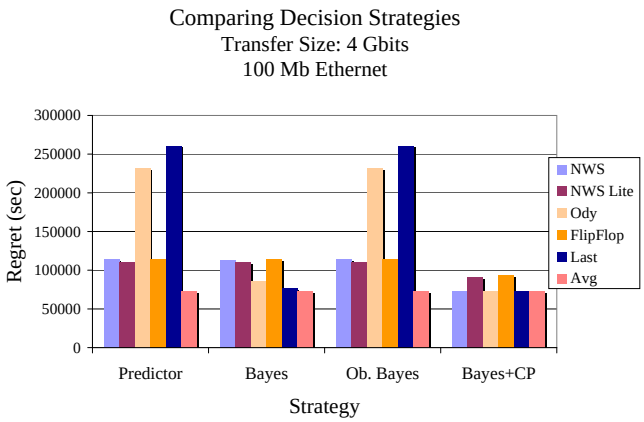


Fig. 4. Predictor performance (regret) for all prediction-based strategies for the critical bandwidth of 53Mb/s using the 100Mb/s network.

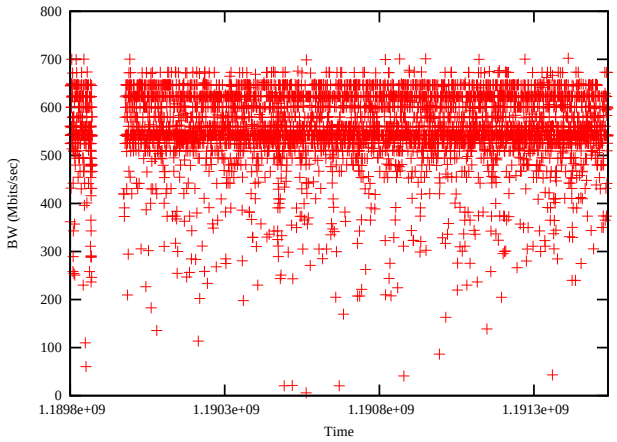


Fig. 5. Network bandwidth observations for 1Gb/s trace for first 5000 observations.

Flop, in Predictor and Ob. Bayes strategies. The Avg predictor, which is also a non-adaptive predictor, performs remarkably well for all decision strategies. Predictor, Ob. Bayes, and Bayes strategies introduce similar regret for all predictors, except for Ody and Flip-Flop. For these two predictors, Bayes strategy performs significantly better than Predictor and Ob. Bayes strategies. Overall, the Bayes+CP strategy outperforms all others, and it performs well regardless of the predictor that it employs.

C. 1 Gb/s Network Experiments

We next present our experimental data and analysis for the 1Gb/s network trace. Figure 5 plots the first 5000 network observations in this trace. The y-axis is network bandwidth and the x-axis is time.

There are important similarities and differences between this trace and the 100Mb/s trace. In this trace, most observations are within 500-700Mb/s, whereas in the 100Mb/s trace, most measurements are within 50-60Mb/s. In this trace, there are two dominant performance modes at approximately 550Mb/s and 650Mb/s. Moreover, this trace is quite noisy and there are many observations between 100 and 500 Mb/s. The gap in the data on the left side of the plot shows that there is a certain period that no observations are collected, i.e., the network was unavailable.

Figure 6 presents the regret for the different offloading strategies when we vary task size using the NWS and Ody predictors (note that the Data and No Data strategies do not use the predictors at all; we include them here for comparison). The y-axis is regret, in seconds, and the x-axis is task size. We employ the same strategies as we do above in Figure 3. In simulations of Gigabit trace, we investigate task sizes that are an order of magnitude larger than the tasks for the previous trace, since this network is significantly faster.

As for the 100Mb/s experiments, total regret increases as we increase the task size. The regret peaks for tasks approximately 40 Gb in size, and then decreases. For this network, NWS prediction shows no significant differences across strategies for tasks smaller than 35 Gb. For larger tasks, the Predictor strategy introduces the most regret. As in the prior set of experiments, there is a critical range of task sizes (between 30 and 50) that separates the strategies in term of performance (regret). Ody (a) and NWS (b) prediction perform similarly, however, the Predictor strategy performs poorly for a much larger range of task sizes.

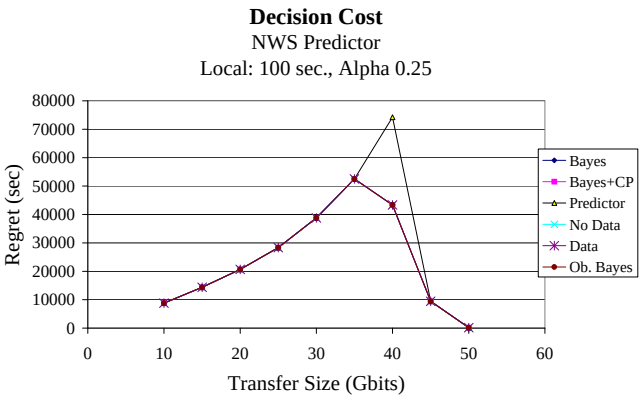
We next zoom in on a critical task size, in this case 40 Gb, in Figure 4. This task size equates to a bandwidth value of 533.3Mb/s. We again investigate strategies that employ prediction and include all predictors. The Predictor strategy alone introduces the most regret over all strategies. The different Bayes techniques perform similarly for most predictors. Bayes+CP performs well regardless of the predictor it integrates.

Result Summary. Our results show that there can be a significant difference in computation offloading performance depending on the decision strategy that we use. The efficacy of a Predictor strategy, which uses a network forecast to make a “yes/no” decision without considering the risk associated in doing so, is strictly dependent on how well a predictor can forecast the future network bandwidth. However, the relationship between predictor forecasting quality and offloading efficacy is a complex one. Consequently, even adaptive and sophisticated predictors may not perform well on certain datasets at certain points. A computation offloading decision, thus, must to take into account the potential risks and benefits of a decision before taking an action. The Bayes strategies provide a mechanism to compute such risks and benefits. Bayes strategies use network forecasts as a way to compute the expected state of network bandwidth for the next offloading decision, which our results show, can avoid bad decisions when in a critical bandwidth range. Moreover, our results indicate that it is important for Bayes strategies to account for change points in the data. By doing so, the predictors are able to avoid considering history data that occur prior to a recent change in network behavior. By coupling predictors, change-point identification, into a Bayes decision formulation, we are able to extract the best computational offloading performance, regardless of the prediction technology available.

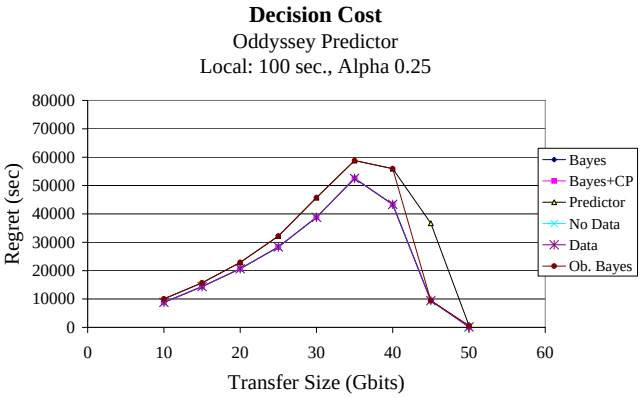
V. Conclusion

In this paper, we investigate how best to formulate decision problems for computational offloading systems for computational grid settings. We compare a classical approach to several variations of a Bayes decision model and a “no data” approach. We also describe how all of these approaches (including some used in previous systems) can be represented in a common intellectual framework for which we have developed an effective implementation.

We find that a Bayesian approach which incorporates change-point detection in its formulation of the *prior* distribution is



(a) Predictor=NWS



(b) Predictor=Odyssey

Fig. 6. Predictor performance (regret) for different task sizes and decision strategies for the 1Gb/s trace for predictors NWS (a) and Ody (b).

the most efficacious of those we investigated. In addition, by comparing a widely disparate set of techniques using a single implementation of our framework, we demonstrate how the heretofore separate approaches to making offloading decisions can be unified.

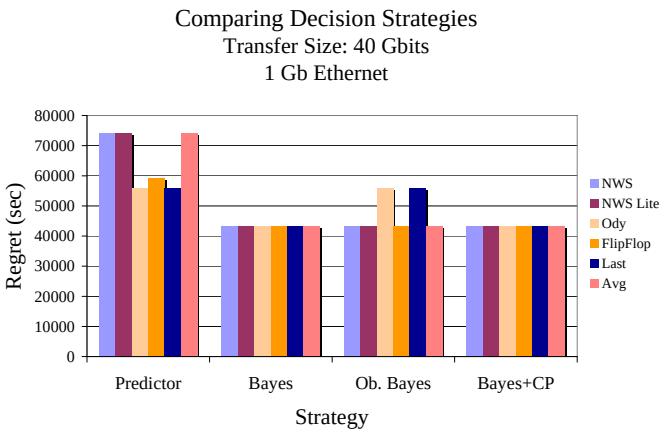


Fig. 7. Predictor performance (regret) for all prediction-based strategies for the critical bandwidth of 533Mb/s using the 1Gb/s network.

REFERENCES

[1] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[2] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.

[3] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, X. Ya, and Y. Haobo. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[4] J. Brevik, D. Nurmi, and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of PPoPP 2006*, March 2006.

[5] E. Caron and G. Utard. On the performance of parallel factorization of out-of-core matricies. *Parallel Computing*, 30(3):357–375, march 2004.

[6] D.Narayanan, J. Flinn, and M. Satyanarayanan. Using history to improve mobile application adaptation. In *Third Workshop on Mobile Computing Systems and Applications*, December 2000.

[7] J. Dongarra, J. Bunch, C. Moler, and G.W. Stewart. *LINPACK User's Guide*. SIAM publications, Philadelphia, PA, 1979.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[9] S. Gurun, C. Krintz, and R. Wolski. Nwslite: a light-weight prediction utility for mobile devices. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 2–11. ACM Press, 2004.

[10] T. Hiroyasu H. Shimosaka, M. Miki, and J. Dongarra. Optimization Problem Solving System Using Grid RPC. *IEEE Transactions on Parallel and Distributed Systems*, 1(1), October 2005.

[11] M. Kim and B. Noble. Mobile network estimation. In *Mobile Computing and Networking*, pages 298–309, 2001.

[12] B.W. Lindgren. *Statistical Theory*. Macmillan, 1976.

[13] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile application-aware adaptation for mobility. In *sixteenth ACM symposium on Operating systems principles*, pages 276–287, 1997.

[14] D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue bounds estimation from time series. In *Proceedings of 13th Workshop on Job Scheduling Strategies for Parallel Processing (with ICS07)*, June 2007.

[15] S. Ou, K. Yang, and J. Zhang. An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362–385, August 2007.

[16] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, 1995.

[17] A. Rudenko, P. Reiher, G.Popek, and G.Kuenning. The remote processing framework for portable computer power saving. In *ACM Symp. Appl. Comp.*, San Antonio,TX, February 1999.

[18] M. Sato, T. Boku, and D. Takahasi. Omnirpc: a grid rpc system for parallel programming in cluster and grid environment. In *Proceedings of CCGrid2003*, pages 206–213, May 2003.

[19] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A network based information library for global world-wide computing infrastructure. In *HPCN Europe '97: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, pages 491–502, 1997.

[20] Top500 – http://www.top500.org.

[21] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. Netw.*, 5(1):71–86, 1997.

[22] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, January 1998.

[23] R. Wolski. Experiences with predicting resource performance online in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, March 2003.

[24] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5,6):757–768, 1999.

[25] A. Yarkhan, J. Dongara, K. Seymour, D. Fike, E. Meek, and Z. Shi. Gridsolve: Evolution of a network enabled solver. In *Proceedings of IFIP Working Conference on Grid-based Problem Solving Environments (WoCo9)*, July 2006.

[26] Z.Li, C.Wang, and R.Xu. Computation offloading to save energy on handheld devices:a partition scheme. In *Proc. of International Conference on Compilers,Architectures and Synthesis for Embedded Systems (CASES)*, pages 238–246, 2001.