Providing Data Transfer with QoS as Agreement-Based Service

Honghai Zhang
Department of Computer Science,
University of Illinois at Urbana, Champaign
hzhang3@uiuc.edu

Kate Keahey, William Allcock
Mathematics and Computer Science Division,
Argonne National Laboratory
{keahey,allcock}@mcs.anl.gov

Abstract

Over the last decade, Grids have become a successful tool for providing distributed environments for secure and coordinated execution of applications. The successful deployment of many realistic applications in such environments on a large scale has motivated their use in experimental science [15, 5] where Grid-based computations are used to assist in ongoing experiments. In such scenarios, quality of service (QoS) guarantees on execution as well as data transfer are desirable.

The recently proposed WS-Agreement model[16,6] provides an infrastructure within which such quality of service can be negotiated and obtained. We have designed and implemented a data transfer service that exposes an interface based on this model and defines agreements which guarantee that, within a certain confidence level, file transfer can be completed under a specified time.

The data transfer service accepts a client's request for data transfer and makes an agreement with the client based on QoS metrics (such as the transfer time and confidence level with which the service can be provided). In our approach we use prediction as a base for formulating an agreement with the client, and we combine prediction and rate limiting to adaptively ensure that the agreement is met.

1. Introduction

Issues of quality of service (QoS) are of increasing importance to the success of Grid-based applications [17]. This need is especially pronounced in experimental science applications such as the National Fusion Collaboratory [5] and NEESgrid [15]. Enabling such interactions on the Grid requires two related efforts: (1) the development of sophisticated resource management strategies and algorithms and (2) the development of protocols enabling structural negotiation for the use of those resources.

Progress in the second area has been facilitated by formulating and refining the abstraction of Grid services [1]. Based on this abstraction, the Global Grid Forum is developing a set of protocols, called WS-Agreement [16,6], for the negotiation, creation, and management of agreements for services in the Grid. An agreement constitutes a provisioning target; a good articulation of such targets allows the provider to better estimate demand and therefore make better provisioning decisions. Further, an agreement can be used as an adaptation target where automatic management allows providers to both leverage and counteract the changing conditions on the Grid.

In this paper we address both issues. Specifically, we illustrate how brokering needs influence the development of agreement terms (in our case, priority levels and confidence levels associated with a request) and how forming agreements based on those terms can improve the client's control over a data transfer and the provider's chances of satisfying a request even in an unpredictable environment such as the Internet. We start by describing an agreement-based data transfer service. The service allows the client to form agreements for quality of data transfer and later accepts and executes data transfer requests based on those agreements. We describe an architecture and protocol for forming and claiming such agreements.

We also present three prototype implementations of such a service. In our first implementation, the "agreement" has a purely informational role and is based on prediction of network bandwidth for a specific future time; the only obligation undertaken is the future transfer of data, not the quality of that transfer. In the second implementation, the provider uses a reservation table and rate limiting to split the bandwidth between several clients requesting overlapping transfers. In other words, the provider offers a stronger guarantee than in the first case, in that if the bandwidth is as predicted (i.e., there is no additional "unexpected" network traffic), the client's requests will be satisfied. In the third implementation, further improvement is provided through the inclusion of adaptation: the client's requests are divided into two priority levels, and the provider uses rate limiting to compensate for encountered Internet traffic for the high-priority requests, at the cost of the low-priority requests.

The paper is structured as follows. In Section 2,we discuss related work. In Section 3, we briefly introduce the data transfer services, factories, and architecture. In Section 4, we present the three data transfer service implementations. In Section 5 we show experimental results on our testbeds. In Section 6, we conclude the paper and discuss future work.

2. Related Work

Two classes of research are related to our work. The first refers to bandwidth prediction and the second to traffic rate control and adaptation.

Numerous researchers have investigated ways to predict network bandwidth [11,20,21,24,25,26,27,28,29, 30]. Most of these use short TCP messages to predict the available end-to-end bandwidth. For example, Carter et al. [24] predicted the available bandwidth based on the dispersion of long packet trains at the receiver, and Lai and Baker [11,25] predicted the bottleneck bandwidth by sending a pair of back-to-back packets. In general, these methods are able to predict only the "current" network bandwidth by sending a few short TCP probing packets. Our application, however, requires predicting the network bandwidth in a "future" time in order to negotiate agreements. Thus, we cannot directly apply these methods. In addition, the predicted available bandwidth may not be achievable by any TCP streams because, in general, TCP's congestion control mechanism prevents TCP from using all available bandwidth [30].

Several efforts have been made to predict the steady-state TCP throughput, the throughput that can be achieved by a long TCP flow. For example, the Network Weather Service (NWS [2,3]) monitors network throughput by sending out short data messages (64 KB). Wolski [2] has investigated different methods of prediction, including mean-based methods, median-based methods, and autoregressive methods; for this work Wolski dynamically chooses the prediction method based on prediction errors in the previous step, and he performs measurements and predictions on a very fine time scale (measuring the data roughly every 30 seconds). Swaney and Wolski [3] use a combination of short probing packets and previously observed long HTTP transfers to predict the "current" long HTTP transfers. In each case, only a one-step prediction is made. Our work aims to provide *multi-step* predictions, and we do not need to generate data at the time of prediction because we use a *history-based data transfer log*.

Vazhkudai and Schopf [4] use linear regression techniques to predict large data transfers. They start with simple univariate prediction methods, build a linear model between the NWS measured bandwidth, disk load, and the data transfer rate using GridFTP [9,13,19], and then predict the GridFTP performance based on the disk load and measured bandwidth using NWS. This work requires instantaneous data transmission at the time of prediction, whereas we do not. Also, their prediction is used only for one-step prediction, and they predict the network bandwidth only at the current time, whereas we try to predict network bandwidth at a future time.

In summary, most of the existing efforts on prediction focus on predicting current available bandwidth using history data or small-size probing. Our requirement, on the other hand, it to predict the future bandwidth; thus, we perform multistep predictions. In addition, our predictor not only gives a single prediction value but also gives the probability distribution of prediction errors. These values are used to model the confidence level with which the real bandwidth is above or below a given threshold value.

Likewise, many methods [32,33,34,22,23] are also used to control the transmission rate of networking traffic which have the similar functions to the rate limiting presented in this paper. Most of them are implemented in the intermediate routers. Two typical examples are token bucket and leaky bucket [32,33]. In token bucket, tokens are generated with a constant rate until the total number of tokens reaches the capacity of the bucket. When a packet arrives, it is transmitted only if there are tokens in the bucket. When a packet is transmitted, it consumes a token. If a packet arrives and there is no token in the bucket, the packet will be buffered in a queue until a token is generated. A leaky bucket is similar to token bucket except that the capacity of the bucket is 1. In other words, the tokens cannot be accumulated. The rate-limiting and adaptation are similar to token bucket, but they are implemented at the application layer of sender side instead of routers. In addition the rate-limiting and adaptation in this paper control the average transfer rate instead of instantaneous transfer rate.

3. Agreement-Based Architecture

3.1. Overview of WS-Agreement

WS-Agreement [16,6] is a work in progress describing an agreement-based approach to service management. The process of agreement creation starts with a negotiation phase, in which clients represent their requirements to the providers, who respond by defining what capabilities they can provide. This dialogue ends when both sides arrive at a satisfactory description of capabilities and commit to the agreement. We note that an agreement service may be distinct from a group of services or actions that implement the terms of an agreement. The purpose of an agreement service is merely to provide abstractions for the negotiation and management of an agreement.

The service and levels of service that are the object of negotiation are described by agreement terms. The WS-Agreement specification defines a term type for describing agreement terms, but it does not provide a term language for any specific domain. It is assumed that such term languages to describe domain-specific concepts will be developed separately as needed. Specific terms may be grouped using compositors described by WS-Policy [18].

The current focus of the specification is a description of architecture and the negotiation model. The negotiation model allows renegotiating agreements after creation and concludes in a commit stage that can be triggered by either side. The negotiation is fine-grained and proceeds on the level of specific terms that can be annotated as required, optional, observed (agreed on), or ignored.

3.2. Motivating Scenario: Data Transfer

In a simple motivating scenario a client may want to transfer some data (which can be the program to be executed remotely, or the input/output data files) from a source URL to a destination URL. In addition, the client will want to specify the transfer data size D, the transfer start time S, and the transfer duration T. We do not require the client to start to transfer at exact time S. Instead, the client's request should be satisfied as long as he claims his reservation by associating the agreement with a specific transfer instance during an availability time window W. A broker (implemented by the service provider), with the help of prediction of available network bandwidth and the prediction error (detailed in the next section), can estimate the confidence level with which the transfer can be completed within time T.

Based on this motivating example, we find the following information important for negotiating the requests: the data transfer start time S, availability time window W, data size D, transfer time T, and confidence level α. After making a reservation using the 5-tuple (S,W,D,T,α), a client can claim his transfer request. If the client submits his transfer request in any time between S and S+W, and his request data size is less than or equal to D, the transfer will be completed within time T after his request with a confidence level α (which means with probability α, his request will be completed on time).

3.3. Agreement Terms for Data Transfer

Service agreements represent a contract between a service provider and a client. The XML fragment below represents agreement terms for data transfer in our system. In addition to generic terms describing the parties to the agreement, the potential dependencies, and the time window describing the validity period of the agreement, the set of terms contains a service description and information about service-level objectives.

The service description tells us *what* will be done under a given agreement: how much data will be moved between what points. The service-level element describes *how* this should happen: the time of data transfer should not exceed a certain bound, the confidence level of providing that bound is given, and the priority of the transfer (which may be linked to some form of payment) is described. In addition, the service-level term may give information about other qualities pertaining to the transfer, for example, describing data integrity (lossy versus non-lossy transfer) or further describing the quality of the guarantee (based on prediction only or on adaptation).

```
<xsd:complexType name="AgreementTermType">
    <xsd:sequence>
        <xsd:element name="parties" type="tns:AgreementPartiesType"/>
        <xsd:element name="serviceInstanceHandle" type="xsd:anyURI"/>
        <xsd:element name="dependency" type="xsd:anyURI"
                minOccurs="0"
                maxOccurs="unbound"/>
        <xsd:element name="availability" type="tns:ScheduleType"/>
        <xsd:element name="expirationTime" type="xsd:dateTime"/>
        <xsd:element name="serviceDescription" type="xsd:serviceDescriptionType"/>
        <xsd:element name="serviceLevel" type="tns:serviceLevelType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AgreementPartiesType">
    <xsd:sequence>
        <xsd:element name="client" type="xsd:anyURI"/>
        <xsd:element name="provider" type="xsd:anyURI"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ScheduleType">
    <xsd:sequence>
        <xsd:element name="startTime" type="xsd:dateTime"/>
        <xsd:element name="endTime" type="xsd:dateTime"/><
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="serviceDescriptionType">
    <xsd:sequence>
        <xsd:element name="source" type="xsd:String"/>
        <xsd:element name="destination" type="xsd:String"/>
        <xsd:element name="size" type="xsd:int"/>
   </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="serviceLevelType">
    <xsd:sequence>
        <xsd:element name="timeBound" type="xsd:duration"/>
        <xsd:element name="confidenceLevel" type="xsd:int"/>
        <xsd:element name="priority" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
```

4. Implementation of Data Transfer Service

The implementation of the data transfer service relies on a combination of prediction, rate limiting, and adaptation. First, prediction of available network bandwidth is based on historical data of similar data transfers collected by service provider. The predictor provides two results: the predicted value and the relative error of the predicted value. We use the error to model the confidence level term because it indicates how much the predicted value is to be trusted. Second, we implement a rate-limiting driver for the Globus XIO (eXtensible Input and Output) system [12] that can control the averaged packet-sending rate and the burst size according to the broker's setting. Third, we perform priority-based adaptation of data transfer by dynamically adjusting the data transfer rate of each simultaneously transmitting flow.

4.1. Prediction-Based Data Transfer Service

The objective of predicting network bandwidth is to obtain necessary information about the network bandwidth at a future time, as well as the prediction error associated with a given prediction. When the service broker receives a request with a certain QoS requirement, it will be able to determine whether it can fulfill the requirement based on the predicted network bandwidth and the prediction errors. If according to the prediction it can accomplish the request with high confidence, performing adaptation may be unnecessary.

4.1.1. Historical Data Generation

Our prediction is based on logs of historic data transfer using GridFTP [9,13,19] and the logs are generated by NetLogger service [8]. The NetLogger service [8] provides a general toolkit for real-time logging, visualization, and diagnosis of system performance data such as data transfer size, time, and duration, and it has been incorporated into the GridFTP. All the experiments reported in this paper are performed on a consistent testbed which consists of three testing sites: Argonne National Laboratory (ANL), the Information Sciences Institute (ISI) at USC, and Lawrence Berkeley National Laboratory (LBNL).

Because such a log does not yet exist, we generate the log by periodic measurement. We send real traffic through GridFTP every time interval D, and we log the bandwidth for each transfer using NetLogger [8]. We now have a time series of bandwidth f(1),...,f(N), where N is the total number of history data. To smooth the measured bandwidth, we calculate the average bandwidth $f_m(1)$,..., $f_m(n)$ (notice that we can still obtain the average bandwidth by averaging the throughput within each time window even if the historic log is aperiodic) by

$$f_m(i) = \frac{\sum_{j=1+(i-1)m}^{im} f(j)}{m} \qquad (1)$$

and then predict the future network bandwidth based on the averaged time series ($f_m(1)$, … , $f_m(n)$) of data samples.

Using GridFTP, we performed experiments to measure the throughput between the three sites in our testbed. Since the number of streams and file sizes may affect the prediction results, we perform measurements using different numbers of streams and different file sizes. In each transfer we randomly generate the sequence of number of streams; and within each number of stream, we randomly generate the sequence of the file size. Then we do the transfer for each combination of number of streams and file size. We measure the throughput once every half hour and use the average throughput of the two transfers in each hour as the throughput during that hour. We then predict the throughput for each number of streams and file size combinations.

4.1.2. Prediction Method

We predict the future bandwidth using the linear minimum mean square error (LMMSE) predictor [7]. Specifically, given the series $\{f_m(k): k = 1,...,n\}$, we express the average bandwidth $f_m(n+1)$ in the next aggregated interval mD as a weighted linear combination of the past n samples. That is, the estimate (written as $\hat{f}_m(n+1)$) of $f_m(n+1)$ is expressed as

$$\hat{f}_m(n+1) = [a_1, a_2, ..., a_n] \begin{bmatrix} f_m(1) \\ f_m(2) \\ \cdots \\ f_m(n) \end{bmatrix}, \qquad (2)$$

where $a_1, a_2, ..., a_n$ are the LMMSE coefficients; a_i's should be chosen to minimize the mean square error of the prediction. In practice, we may not need all the history data to predict the next step data. Instead, we use only the most recent *n* data as the history data; *n* is called the window size of the LMMSE predictor. There are two ways to make the k-step prediction, that is, to predict $f_m(n+k)$: (i) perform one-step prediction k times or (ii) calculate the k-step LMMSE coefficients $a_1,a_2,...,a_n$ [7] and plug the result into Equation (2) to directly calculate the k-step prediction value. In general, the second method is more accurate, but it requires (k+n) history data. We use the second method whenever possible.

History Data Fixing: Occasionally we observe a single spike in the history data. In general the occurrence of the spike is unpredictable. Such a spike can adversely affect the prediction of the normal data. Therefore, we replace any spikes with the mean of the history data (but we keep them in the calculation of the prediction error of the spike data). In order to repair such spike data, we first calculate the mean and estimated standard deviation of the data used for prediction. Then we replace any data in the history that is out of mean ± 2 * standard deviation. Such methods are commonly used in statistical estimation to eliminate abnormal samples [31].

4.1.3. Prediction Results

We first make a step-k prediction of the network bandwidth at time s using only the data before and at s (hour). Then at time k+s, we again perform a real measurement using GridFTP. We calculate the relative error as follows:

$$e_r = \frac{predicted\_throughput - measured\_throughput}{predicted\_throughput} \quad (3)$$

The absolute relative error is the absolute value of the relative error. We performed experiments and predictions for each step k and for different window sizes, (i.e., different n). We summarize our four main results below.

First, in general, using larger window size may have a smaller prediction error. However, once the window size is greater than 24 (hours), the prediction error does not reduce significantly. Second, short-term prediction errors are usually smaller than long-term prediction errors. Most of the prediction errors for up to 48-step predictions are less than 15%, and most one-step prediction errors are less than 10%, except for single-stream transfers (the transfers that only use one tcp connection). Third, prediction errors for multistream are usually smaller than those for single stream. Fourth, using the history data fixing technique reduces the prediction error, making the prediction results more stable.

One additional feature of our predictor is that it gives the prediction error distribution based on historic information. This is important for providing a confidence level of a given prediction result. With such information, we can obtain a confidence level with which the real measured bandwidth is at least the predicted value. For example, for the single stream and 500 MB file transfer, using the error distribution graph, we find that, with a confidence level 90%, the relative error is at least -12%. Now if at some time the predicted available bandwidth is 20 Mbps, then based on Equation (3)(the relative prediction error), we can conclude that in the next hour, the real measured bandwidth will be at least 17.6 Mbps (=20MBps * (1-12%)) with a confidence level 90%.

4.2. Rate Limiting

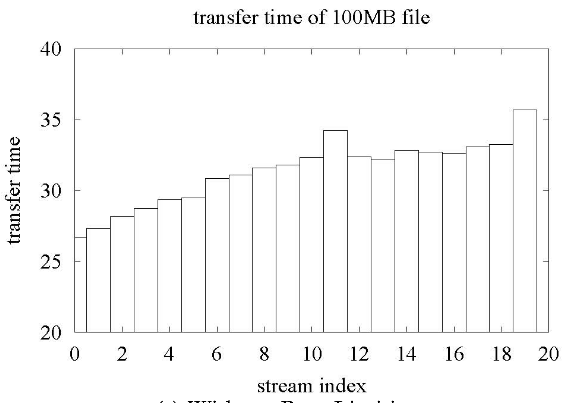
Prediction allows us to build a data transfer service capable of predicting QoS of data transfer at a given time for one client, and then performing that data transfer. However, if several clients want to share the same connection with various QoS requirements, a mechanism of splitting the bandwidth between those competing demands is necessary. We therefore augmented our data transfer service to implement rate limiting, thereby enabling the service to share available bandwidth between different clients. Specifically, we implemented a bandwidth limiter as Globus XIO [12] (eXtensible Input and Output) driver.

The XIO system used by GridFTP [9,13,19] presents a Read/Write/Open/Close interface. A stack of drivers is written to support all of these functions. Drivers are implemented for common operations, such as TCP and UDP network transport, Posix [14] file access, and GSI security. When a handle is opened, an XIO stack is built. The standard GridFTP stack consists of TCP and GSI, providing a GSI-authenticated TCP network connection. A rate-limiting driver could also be pushed onto this stack.

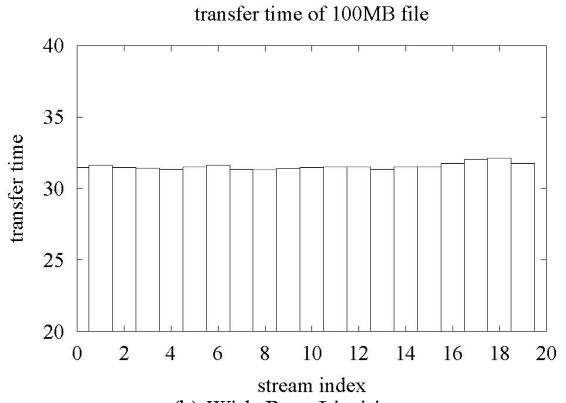
In our implementation, the rate-limiting driver can specify the rate bandwidth, maximum "catchup" rate (which is the maximum transmission rate when a transmission is behind its schedule), burst size, and maximum random interval. When a buffer is passed in for network transmission, the size of the buffer is divided by the burst size. A periodic callback can then be scheduled every interval (seconds) that writes the data with maximum burst size. This interval is calculated as the burst size divided by the rate and then minus a random interval that is uniformly distributed from 0 to the maximum value of a random interval. Thus, what we have is a burst of best-effort transmission, followed by no transmission. Over the aggregate of the transfer, however, we have a limited "smooth" transfer rate. The random interval is added to avoid possible synchronization of different data transfers. The next transmission must happen after the current transmission completes. If for some reason, such as network congestion, a transmission takes longer time than the scheduled interval, the next transmission will start right after its previous transmission completes. If the network condition later becomes better and the transmissions are still lagging behind the scheduled time according to the specified rate, the driver will try to catch up the lost rate by using a shorter interval, which is calculated by the burst size divided by the "catchup" rate.

In our experiments, we empirically set the burst size to be 100 KB and maximum random interval to be 0. In our experimental setting, the random interval does not seem to affect the bandwidth significantly, but we still keep this function for future purposes. We set the maximum catchup rate to be 4 times the specified rate based on experiments.

We performed experiments using the rate-limiting driver and compare the results with non-rate-limited transfer. We assume that the data transfer service is splitting the bandwidth between 20 clients, each assigned one stream for data transfer. Each client transfers a 100 MB file. First we perform data transfer under no (or little) background traffic. Figure 1 compares the data transfer time (a) without rate limiting driver, and (b) with rate limiting driver. We can see clearly that although both achieve approximately the same aggregate bandwidth, not all of the clients can get the requested bandwidth without using rate-limiting driver.



(a) Without Rate Limiting



(b) With Rate Limiting

Figure 1: File transfer time with and without a rate-limiting driver and with no (or little) background traffic

In addition, using rate limiting allows us to assign different transfer rates to different cliets, so that the available bandwidth is split unevenly between clients based on agreement terms requested by individual clients. As an example, Figure 2 shows the data transfer time of all clients where we apply 4.0 Mbps to the first 10 clients and 2.2 Mbps to the next 10 clients.

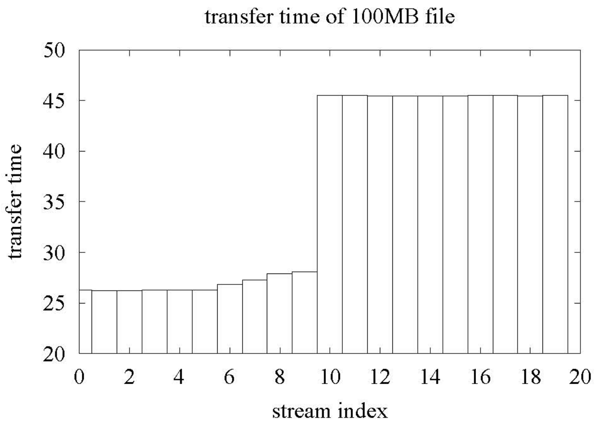


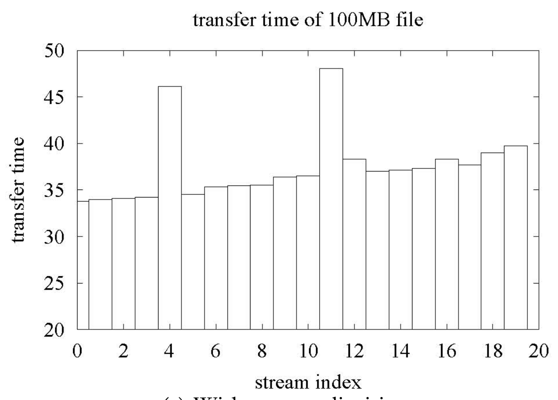
Figure 2: File transfer time with a rate-limiting driver: the first 10 streams are assigned a rate of 4.0 Mbps, and the second 10 streams are assigned a rate of 2.2 Mbps

If bursty background traffic comes in during the transmission, the data transfer rate even with the rate-limiting driver could be affected. As an example in Figure 3, we inject four background TCP streams during the data transfer (the rate of each steams used in the rate limiting driver is still assigned based on the predicted value). We can see that both rates of the data transfers with and without rate limiting driver are seriously affected. Because of the chaotic nature of Internet [10,11], avoiding such background traffic is impossible, and the data transfer service may not be able to fulfill its obligations to the individual clients. However, we can still see that with rate-limiting, flows are affected more evenly rather than just a few of them are heavily affected and others remain almost intact (as in the case of without rate-limiting).

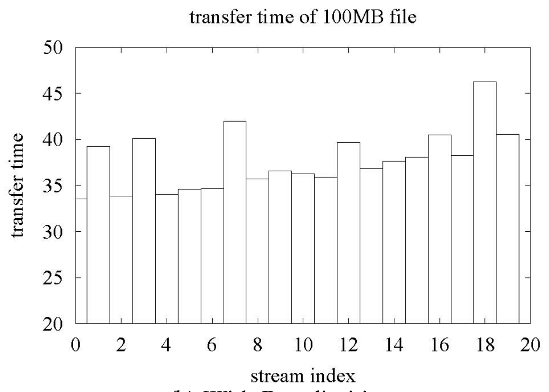
4.3. Adaptation

To deal with the uncontrolled background traffic, we augmented the implementation of the data transfer service to include adaptation and thus allow quality data streams to recover from the effects of burstiness.

By default, the rate-limiting driver will automatically catch up the "lost rate" when the network conditions become better. But this is not sufficient when the networks experience bad conditions for a time longer than the data transmission time or near the end of a data transmission. Therefore, we divided clients into two classes: high-priority connections and low-priority (presumably "cheaper") connections. The data transfer service acts like a broker to manage all connections. At the time of network congestion, the broker adjusts the transmission rates so that high-priority connections catch up first, at the cost of the low-priority connections.



(a) Without rate limiting



(b) With Rate limiting

Figure 3: File transfer time with and without a rate-limiting driver when four background TCP streams are injected

Specifically, we created a notification and control mechanism between the broker and rate-limiting driver. The broker provides a callback function to the rate-limiting driver when invoking it for the high priority connections. When the rate-limiting driver observes that it is transmitting slower than the set rate, it invokes a callback function to notify the broker. The broker then looks for connections with low priority and sends them a control message to reduce their transmission rate. In this way, connections with higher priority get more bandwidth at the time of network congestion.

In Figure 2, we observe that the first ten clients don't get the exact bandwidth they request, namely, that the file transfer be completed within 25 seconds. The reason rests with congestions and competitions from low-priority flows. We now perform adaptations on the transfer in which the first ten streams are still assigned the 4.0 rate but are now categorized as high-priority clients, and the second ten streams are still assigned with the 2.2 rate but as categorized as low-priority clients.

Figure 4 shows the file transfer time for each client under these new conditions. The first ten clients now meet their requirements very well, while some of the low priority clients no longer get the requested bandwidth.

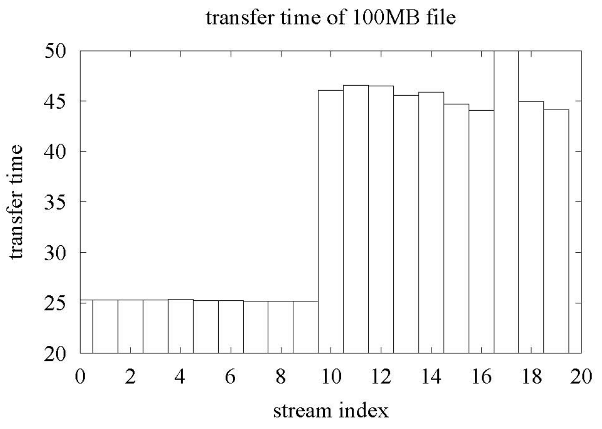


Figure 4: Data transfer time with a rate-limiting driver: the first 10 streams are assigned a rate of 4.0 Mbps, and the last 10 streams are assigned a rate of 2.2 Mbps. We perform adaptation in which the first ten streams are assigned high priority and the second ten streams low priority.

5. Experimental Results

Using our new prediction results and resource management methods, we perform three sets of experiments. In the first set of experiments, we use prediction alone without rate limiting. In the second set of experiments, we use prediction combined with rate limiting. In the third set of experiments, we use prediction combined with rate limiting and adaptation. For the method of rate limiting, we observe that if we set the rate to be exactly the predicted network bandwidth, we still get quite a large violation. Hence, our goal is to investigate what rate (the percentage of the predicted rate) should be set in order to obtain a small violation. For the method of adaptation, our goal is to investigate how many flows should be set as high priority and how many should be set as low priority, in order to obtain small violation for the high-priority flows.

In each set of experiments, we assume there are ten clients. Each wants to transmit a data file with 50M bytes. We are interested in whether each client's request is met. In all the experiments, we consider predictions over 6 hours (i.e., we make predictions at time t and do another real experiments at t+6 hour). In each experiment, we measure the real transmission time for each client and compare it with the agreed transmission time (which is specified based on the predicted available rate and prediction error bounds; see below). We calculate the violation as

$$violation = \frac{real\_transfer\_time - agreed\_transfer\_time}{agreed\_transfer\_time} \quad (4)$$

A negative or zero violation indicates the real transmission time is less than equal to the agreed transmission time. In each set, we perform 10 experiments, and totally we have 100 values of violation. We then plot the distribution function of the violation for each set of experiments.

In most of the experiments below, we see that the real transmission time may exceed the agreed transmission time based on the rate we set in the rate-limiting driver. If we allow the violation to be less than a certain tolerance level, the transmission is considered as meeting the request of timely transmission (we can either make this as part of the agreement or set the rate in the rate limiting driver slightly higher than the required rate based on the agreement). In the following experiments we assume a tolerance level of 5% (the value may be adjusted based on network conditions and/or the agreement between the client and server).

5.1. Prediction Alone vs. Rate Limiting

We first compare two sets of experiments. In the first set, we use the prediction alone (i.e., no rate limiting or adaptations). In the second set, we use the rate-limiting technique and set the rate as the predicted available bandwidth. In both sets, we set the agreed transmission time as the data size divided by the predicted available bandwidth. Figure 5 shows the violation distribution of the two sets of experiments.

We can see in Figure 5 that without rate limiting, some clients can get a much shorter transmission time than the agreed one, which may create traffic congestion over the Internet.

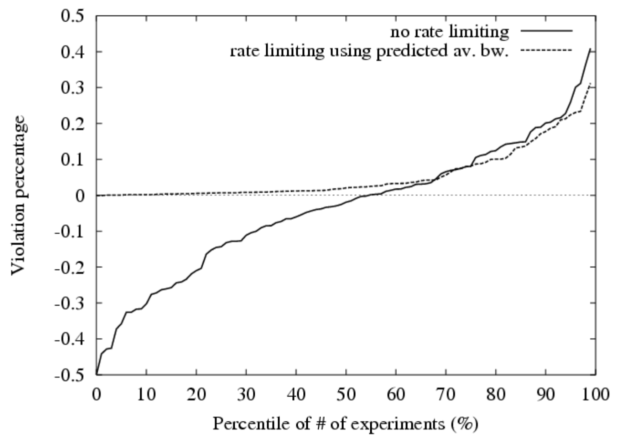


Figure 5: Prediction alone vs. rate limiting using the predicted available bandwidth

In addition, other clients may get a higher violation. Moreover, we may still get quite a large violation if we set the agreed transmission rate using the predicted available bandwidth. The reason is that using the predicted rate may still cause instantaneous network congestion.

5.2. Rate Limiting

Next we perform three sets of experiments to compare different rates in the rate-limiting driver (see Figure 6). In the first set of experiments, we set the rate exactly to the value we predicted. If we allow a 5% tolerance level, nearly 70% of the experiments meet the request of timely transmission. In the second and third sets of experiments, we set the rate such that the real available bandwidth is above that value with 90% and 95% probability, which we call 90% confidence level rate and 95% confidence level rate. If we again allow a 5% tolerance level, approximately 90% of the experiments meet the request of transmission time in the second set and approximately 95% of the experiments do in the third set.

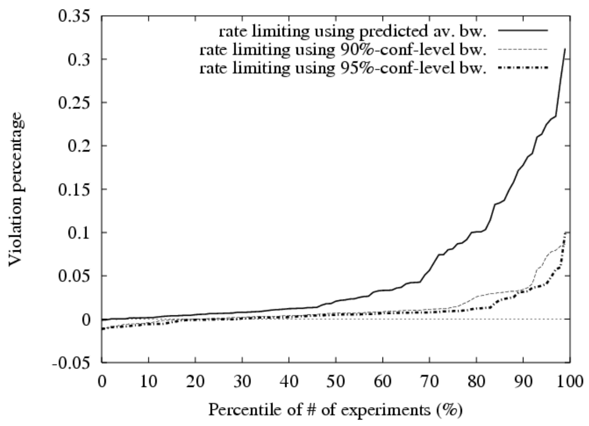


Figure 6: Rate-limiting using the predicted available rate, 90% confidence rate, and 95% confidence rate

5.3. Rate Limiting with Adaptation

We evaluate rate limiting with adaptation by comparing two sets of experiments (see Figure 7). In the first set, we use rate limiting with the predicted rate. In the second set, we use the rate limiting combined with adaptation, and we let two streams among the 10 streams have low priorities, whose rates will be adjusted at the time of network congestions. We see that with rate adaptation, if we set 80% of the flows to be high-priority flows and the rest low-priority flows, most of the high-priority streams have lower violations, at the cost that some of the lower-priority streams get much larger violations.

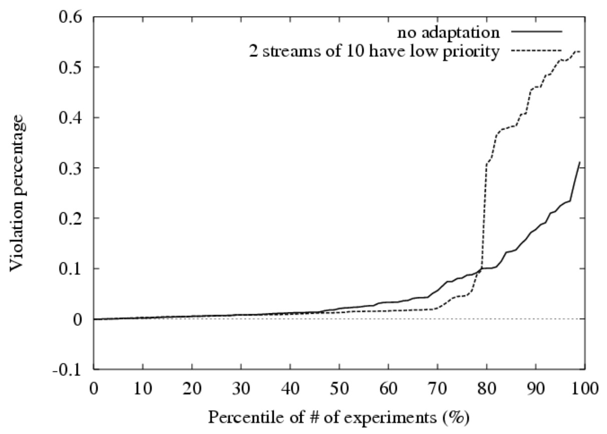


Figure 7: Rate-limiting using the predicted available rate, with and without adaptation

6. Conclusion

Our experiments illustrate how data transfer brokering can use the concept of agreements to improve a client's information and control over the quality of data transfer over the Internet without resorting to low-level bandwidth reservations. In the first of the presented implementations, the agreement has merely an informational role. At the same time, the information provided is closely coupled with the service that will eventually carry out the request, and the prediction is made for a specific period of time in the future, which is of value to the client. The second implementation improves on the first by managing quality of service between clients of that particular transfer service. The third widens its scope to account for third-party traffic and other conditions outside the control of the service.

We find that if we reserve 80% for high-priority transfers within our testbed, the prediction combined with rate limiting and adaptation almost always guarantees that the violation of data transfer agreements will be small (≤5%).

Overall, we find that from the client's viewpoint, the ability to enter into agreements for data transfer significantly increases their tractability, while from the provider's perspective the development of specific terms of service (such as differences in priority) improves the chances of delivering requested QoS even in an unpredictable environment.

Further work on improving the adaptation strategies and therefore the provider's ability to satisfy the client's QoS would include employing rate limiting to favor under-performing requests whose transfer time is about to end and employing multipath routing in which the broker can dynamically adjust the flow rate on different route to satisfying the QoS requirements.

References

[1] I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22,2002

[2] R. Wolski. Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. *Proc. of 6th IEEE Symp. on High Performance Distributed Computing*, Portland, Oregon, 1997

[3] Martin Swany and Rich Wolski. Multivariate resource performance forecasting in the network weather service. *In Supercomputing, 2002*

[4] S. Vazhkudai and J. Schopf. Using Regression Techniques to Predict Large Data Transfers, *submitted to the Internal Journal of High Performance Computing Applications, 2002*

[5] Keahey, K., M.E. Papka, Q. Peng, D. Schissel, G. Abla, T. Araki, J. Burruss, S. Feibush, P. Lane, S. Klasky, T. Leggett, D. McCune, and L. Randerson. Grids for Experimental Science: the Virtual Control Room. *In Challenges of Large Applications in Distributed Environments (CLADE)*. 2004.

[6] Keahey, K., T. Araki, and P. Lane. Agreement-Based Interactions for Experimental Science. *In Europar*. 2004.

[7] G. Grimmett and D. Stirzaker. Probability and Random Processes. *Oxford University Press*, 2001

[8] B. Tierney, W. Johnson, B. Crowley, G. Hoo, C. Brooks and D. Gunter. The NetLogger Methodology for High Performance Distributed Systems Performance Analysis. *Proc. of 7th IEEE Symp. on High Performance Distributed Computing,* 1998

[9] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke.Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, Vol. 28 (5), May 2002, pp. 749-771

[10] V. Paxson. Measurements and Analysis of End-to-End Internet Dynamics. *Ph.D thesis of University of California, Berkeley.* April, 1997

[11] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. Proceedings of the USENIX Symposium on Internet Technologies and Systems. March, 2001

[12] The eXtensible Input Output library for the Globus Toolkit. http://www-unix.globus.org/developer/xio/

[13] GFD.020 GridFTP: Protocol extensions to FTP for the Grid. http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf

[14] J. Barkley, L. Carnahan, K. Olsen and J. Wack. Improving Security in a Network Environment. http://csrc.nist.gov/publications/nistpubs/800-7/node136.html

[15] Pearlman, L., C. Kesselman, S. Gullapalli, B.F. Spencer, J. Futrelle, K. Ricker, I. Foster, P. Hubbard, and C. Severance, Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-Shaking Grid Application. *accepted to the 13th International Symposium on High Performance Distributed Computing (HPDC-13)*, 2004

[16] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke and M. Xu. Agreement-based Grid Service Management (OGSI-Agreement) Version 0. https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-agreement_Specification/en/1/Draft_ OGSI-Agreement_Specification.doc.

[17] I. Foster. What is the Grid? A Three Point Checklist. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf , 2002

[18] M. Hondo, and C. Kaler. Web Services Policy Framework (WS-Policy). http://www-106.ibm.com/developerworks/ webservices/library/ws-polfram/, 2003

[19] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. *IEEE Mass Storage Conference*, 2001[20] C. Dovrolis, P. Ramanathan and D. Moor. What do packet dispersion techniques measure? *IEEE INFOCOM*, April 2001.

[21] M. Mathis and J. Madhavi. Diagnosing Internet congestion with a transport layer performance tool. *In Proceedings of INET'96*, 1996.

[22] S. Jamaloddin Golestani, A self-clocked fair queuing scheme for broadband applications, *In Proc. IEEE INFOCOM'94*, pages 636--646, IEEE, 1994.

[23] P. Goyal, H. M. Vin, and H. Cheng, Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks, *ACM SIGCOMM'96*, pages 157--168, ACM press, August, 1996

[24] R. L. Carter and M.E. Crovella. Measuring bottleneck link speed in packet-switched networks. *In Proc. of IEEE INFOCOM*, Apr. 2001, pp. 905-914

[25] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. *In Proc. of IEEE INFOCOM*, Apr. 1999, pp 235-245.

[26] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on networking.* Vol 7. pp277-292, June 1999.

[27] G. Jin, G. Yang, B. Crowley and D. Agarwal. Network characterization service (NCS). *In Proc. of 10th IEEE Symp. High Performance Distributed Computing*, Aug. 2001, pp 289-299.

[28] B. Melander, M. Bjorkman and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. *In IEEE Globecom 2000*, pp 415-420.

[29] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks and R. Baraniuk. Multifractal cross-traffic estimation. *In Proc. ITC Specialist Seminar IP traffic Measurement Modeling and Management*, Sep. 2000

[30] M. Jain and C. Dovrolis, End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *In Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[31] R. O. Kuehl. Statistical Principles of Research Design and Analysis. *Duxbury press* 1994. pp116-117

[32] D. Bertsekas and R. Gallager. Data Networks. *Prentice* Hall. 1992

[33] R. Jain. http://www.cse.ohio-state.edu/~jain/talks/ftp/ mplste/sld032.htm

[34] L. Zhang, VirtualClock: a new traffic control algorithm for packet-switched networks. *ACM Trans. on Computer Systems*, Vol. 9, No. 2, May 1991