A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing

Dengpan Yin, Esma Yildirim,
Sivakumar Kulasekaran, Brandon Ross and Tevfik Kosar (*), *Member, IEEE*

Abstract—In this paper, we present the design and implementation of an application-layer data throughput prediction and optimization service for many-task computing in widely distributed environments. This service uses multiple parallel TCP streams to improve the end-to-end throughput of data transfers. A novel mathematical model is developed to determine the number of parallel streams required to achieve the best network performance. This model can predict the optimal number of parallel streams with as few as three prediction points. We implement this new service in the Stork Data Scheduler, where the prediction points can be obtained using Iperf and GridFTP samplings. Our results show that the prediction cost plus the optimized transfer time is much less than the non-optimized transfer time in most cases. As a result, Stork data transfer jobs with optimization service can be completed much earlier compared to non-optimized data transfer jobs.

Index Terms—Many-Task computing, Modeling, Scheduling, Parallel TCP streams, Optimization, Prediction, Stork

1 Introduction

In a widely distributed many-task computing environment, data communication between participating clusters may become a major performance bottleneck [1]. Today, many regional and national optical networking initiatives such as LONI [2], ESnet [3] and Teragrid [4] provide high speed network connectivity to their users. However, majority of the users fail to obtain even a fraction of the theoretical speeds promised by these networks due to issues such as sub-optimal TCP tuning, disk performance bottleneck on the sending and/or receiving ends, and server processor limitations. This implies that having high speed networks in place is important but not sufficient. Being able to effectively use these high speed interconnects is becoming increasingly important to achieve high performance many-task computing in a widely distributed setting.

The end-to-end performance of a data transfer over the network depends heavily on the underlying network transport protocol used. TCP is the most widely adopted transport protocol, however its AIMD behavior to maintain fairness among streams sharing the network prevents TCP to fully utilize the available network bandwidth. This becomes a major bottleneck especially in wide-area high speed networks, where both bandwidth and delay properties are too large, which in turn results in a large delay before the bandwidth is fully saturated. There has been different implementation techniques both at the transport and application layers to overcome the inefficient network utilization of the TCP protocol. At the transport layer, different variations of TCP have been implemented [5], [6], [7] to more efficiently utilize high-speed networks. At the application layer, other improvements are proposed on top of the regular TCP, such as opening multiple parallel streams or tuning the buffer size.

Parallel TCP streams are able to achieve high network throughput by behaving like a single giant stream which is the combination of n streams, and getting an unfair share of the available bandwidth [8], [9], [10], [11], [12], [13], [14]. However, using too many streams can bring the network to a congestion point very easily especially for low-bandwidth networks, and after that point, it will only cause a drop in the performance. For high-speed networks, use of parallel streams may decrease the time to reach optimal saturation of the network. Not to cause additional processing overhead, we still need to find the optimal parallelism level where the achievable throughput becomes stable. Unfortunately, it is difficult to predict this optimal point and it is variable over some parameters which are unique in both time and domain. Hence, the prediction of the optimal number of streams is very difficult and cannot be done without obtaining some parameters regarding the network environment such as available bandwidth, RTT, packet loss rate, bottleneck link capacity, and the data size.

In this paper, we present the design and implementation of a service that will provide the user with the optimal number of parallel TCP streams as well as a provision of the estimated time and throughput for

(*) D. Yin (First author), E. Yildirim (First author), S. Kulasekaran, B. Ross, and T. Kosar are with the Department of Computer Science and the Center for Computation & Technology, Louisiana State University, LA 70803 USA E-mail: {dyin, esma, sivakumar, bross}@cct.lsu.edu. T. Kosar (Corresponding author) is also with the Department of Computer Science & Engineering, University at Buffalo, NY 14260 USA Email: tkosar@buffalo.edu.

a specific data transfer. The optimal number of TCP streams is calculated using the mathematical models we have developed in our prior work [15]. A user using this service only needs to provide the source and destination addresses and the size of the transfer. To the best of our knowledge, none of the existing models and tools can give as accurate results as ours with a comparable prediction overhead. We believe that our prediction and optimization service is unique in terms of the minimal input requirements, its low overhead, as well as the practical and accurate results it produces. The current version of our prediction and optimization service supports sampling with Iperf [16] and GridFTP [17], however we plan to extend it to be a more generic tool. Also it is integrated with the Stork Data Scheduler [18] as a service that will improve the performance of the data transfer jobs submitted to it.

In Section 2, we present the related work regarding our design goals. In Section 3, we discuss the design of our prediction and optimization service; and in Section 4, we give the implementation details. Section 5 presents the results of the experiments conducted. Finally in Section 6, we discuss the conclusions of our study.

2 Related Work

There is a limited number of studies that try to find the optimal number of TCP streams, and these are mostly based on approximate theoretical models [19], [20], [21], [22], [23]. They all have specific constraints and assumptions; and the correctness of the proposed models are mostly proved with simulation results only.

Hacker et al. claim that multiple number of TCP streams behave like one giant stream equal to the capacity of sum of each streams' achievable throughput [19]. However, this model only works for uncongested networks, and cannot provide a feasable solution in case of congestion. Another study [22] declares the same theory but develops a protocol which at the same time provides fairness. Dinda et al. [20] model the bandwidth of multiple streams as a partial second order polynomial which requires two throughput measurements with different stream numbers to predict all of the others. However, the overall accuracy of this model is very low and it not cannot predict the optimal number of parallel streams necessary to achieve the best transfer throughput. In another model [21], the total throughput always shows the same characteristics depending on the capacity of the link as the number of streams increases and 3 streams are sufficient to get a 90% utilization. A new protocol study [23] that adjusts sending rate according to calculated backlog presents a model to predict the current number of flows which could be useful to predict the future number of flows.

All of the models presented have either poor accuracy or they need a lot of information to be collected from the network or from the user. In most cases, it is impractical to provide the models with all of these necessary information. The users generally prefer to achieve a prediction of their data transfer throughput with least input from them possible. In some cases, mathematical models completely depend on historical data, which can cause two issues: i) historical data may not be available for all transfers; ii) the network traffic characteristics may have significantly changed over time. Especially for individual data transfers, instead of relying on historical information, the transfers should be optimized based on instant feedback. However, an optimization technique not relying on historical data should not bring too much overhead of gathering instant data. The total overhead should not be larger than the speedup gained with multiple streams for a particular data size. Gathering instant information for prediction models can be done by using network performance measurement tools [24], [25], [26], [27], [28] or by performing a miniature version of the transfer.

In our previous study [15], we have presented two mathematical models to predict the optimal number of TCP streams for best end-to-end data throughput. The models were be able to make accurate predictions based on only 3 samplings of different parallelism levels. The development of these models start from the foundations of the Mathis throughput equation:

$$Th <= \frac{MSS}{RTT}\frac{c}{\sqrt{p}} \quad (1)$$

In Equation 1, the achievable throughput (Th) depends on three parameters: round trip time (RTT), packet loss rate (p) and maximum segment size (MSS). The maximum segment size (MSS) is in general the IP maximum transmission unit (MTU) size minus the TCP header size. The round trip time (RTT) is the time it takes for the segment to reach the receiver and for a segment carrying the generated acknowledgment to return to the sender. The packet loss rate (p) is the ratio of missing packets over total number of packets; and c is a constant. Of MSS, RTT, and p variables, packet loss is the most dynamic one while MSS is the most static one.

According to Hacker et al. [19], an application opening n connections can gain n times the throughput of a single connection, assuming all connections experiencing equal packet losses. Also the RTTs of all connections are equivalent since they most likely follow the same path. In that case, Equation 1 is rearranged for n streams as follows:

$$Th_n <= \frac{MSS \times c}{RTT}\left(\frac{n}{\sqrt{p}}\right) \quad (2)$$

However this equation accepts that packet loss is stable and does not increase as the number n increases. At the network congestion point, the packet loss rate starts to increase dramatically and the achievable throughput starts to decrease. Hence, it is important to find that point of knee in packet loss rate.

Dinda et al [20] model the relation between n, RTT and p as a partial second order polynomial by using throughput measurements of two different parallelism levels. This approach fails to predict the optimal number of parallel streams necessary to achieve the best transfer throughput. Instead of modeling the throughput with a partial second order polynomial, we increase the sampling number to three and either use a full second order polynomial or a polynomial where the order is determined dynamically. For the full second order model, we define a variable p'_n as follows:

$$p'_n = p_n \frac{RTT_n^2}{c^2 MSS^2} = a'n^2 + b'n + c' \quad (3)$$

According to Equation 3, we derive:

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'n + c'}} \quad (4)$$

In order to obtain the values of a', b' and c' presented in Equation 4, we need the throughput values of three different parallelism levels ($Th_{n_1}, Th_{n_2}, Th_{n_3}$) which can be obtained through real-time sampling or from past data transfers .

$$Th_{n_1} = \frac{n_1}{\sqrt{a'n_1^2 + b'n + c'}} \quad (5)$$

$$Th_{n_2} = \frac{n_2}{\sqrt{a'n_2^2 + b'n + c'}} \quad (6)$$

$$Th_{n_3} = \frac{n_3}{\sqrt{a'n_3^2 + b'n + c'}} \quad (7)$$

By solving these three equations, we can replace the values of a',b' and c' variables in Equation 4 to calculate the throughput of any parallelism level.

In the second model, we define p'_n as an equation of order c' which is unknown and can be calculated dynamically using *Newton's iteration* based on the values of the samples:

$$p'_n = p_n \frac{RTT_n^2}{c^2 MSS^2} = a'n^{c'} + b' \quad (8)$$

Both of these models are presented in more detail in our previous study [15]. After we calculate the optimal number of parallel streams using one of these models, we can easily calculate the maximum throughput corresponding to that number. The optimization server presented in this paper uses the *Full Second Order* model and needs to get at least three suitable throughput values of different parallelism levels through real-time sampling to apply the model.

There are also other services and tools that try to give an estimate of the available bandwidth and optimize the protocol parameters to improve the throughput. However these services require constant probing, installation privileges as root, or making changes at the protocol level to the TCP AIMD properties. Jin et al. In [29], a service based on Web100 tool is presented which aims to provide an optimal buffer setting by using the information collected by Web100. It divides the buffer size by the number of streams to optimize it, but the authors mention that their results were inconclusive. Also it is stated in the paper that parallel streams should give better performance than buffer tuning. Jin et al. [30] present a service that collects information from every node in the data transfer path with constant probing and make an estimation of the available bandwidth to tune the buffer size. The study in [31] only makes an estimation on the available bandwidth and capacity.

Our service provides an end-to-end optimization throughput via use of parallel streams in the application-level with real-time sampling without constant probing. In our approach, no changes in the transport protocol layer is required. We propose to use Iperf [27] and GridFTP [17] to gather the sampling information to be fed into our mathematical models. Both of these tools are widely adopted by the distributed computing community, and they are convenient for our service since they both support parallel streams. With GridFTP, it is also very convenient to perform third-party transfers. By using our mathematical models and the real-time sampling information, we provide a service that estimates the number of optimal parallel streams with a negligible prediction cost.

3 Design Issues of the Optimization Service

The optimization service presented in this study takes a snapshot of the network throughput for parallel streams through sampling. The sampling data could be generated by using a performance prediction tool or an actual data transfer protocol. Due to the differences in the implementation of different data transfer or prediction tools, the throughput achieved in the same network using different tools could be inconsistent with each other. For this reason, the choice of the tool to perform sampling could result in slight differences in the optimized parameters as well. At the current stage, we have implemented the optimization service based on both Iperf and GridFTP. In the future, it can simply be modified to accommodate other data transfer protocols and prediction tools.

3.1 Sketch of the Optimization Service

Figure 1 demonstrates the structure of our design and presents two scenarios based on both GridFTP and Iperf versions of the service. Site A and site B represent two hosts between which the user wants to transfer data. For the GridFTP version, these hosts would have GridFTP servers and GSI certificates installed. For the Iperf version, these hosts would have Iperf servers running as well as a small remote module ($TranServer$) that will

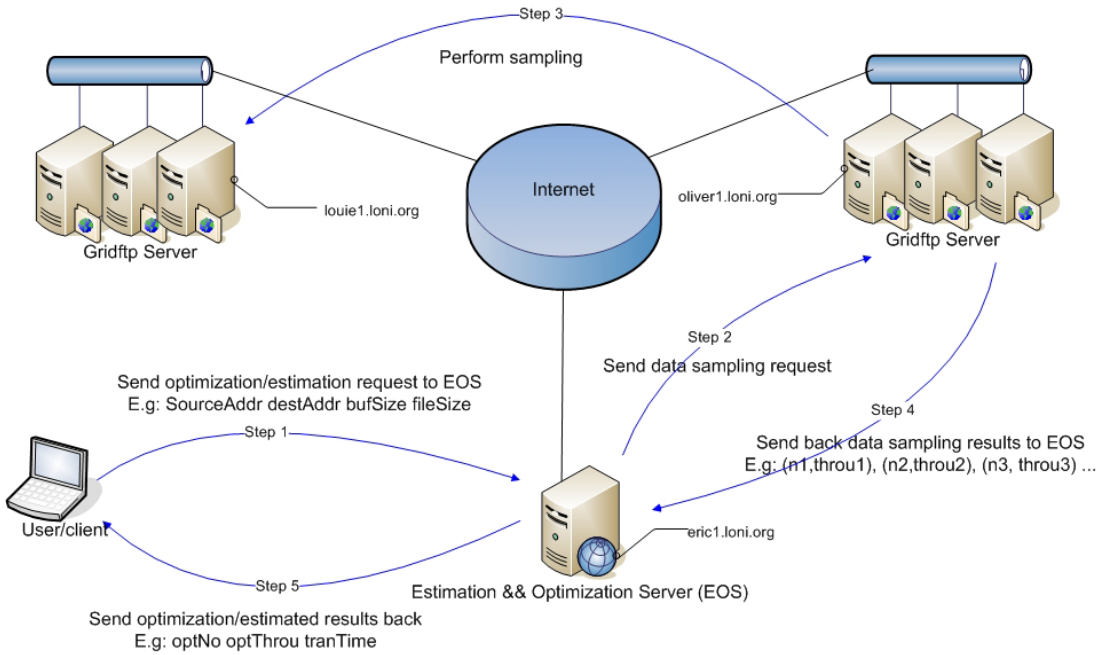

Fig. 1. Overview of the Estimation and Optimization Service (EOS)

make a request to Iperf. Optimization server is the orchestrator host designated to perform the optimization of TCP parameters and store the resultant data. It also has to be recognized by the sites since the third-party sampling of throughput data will be performed by it. User/Client represents the host that sends out the request of optimization to the server. All of these hosts are connected via LAN or WAN.

When a user wants to transfer data between site A and site B, the user will first send a request that consists of source and destination addresses, file size and an optional buffer size parameter to the optimization server, which process the request and respond to the user with the optimal parallel stream number to do the transfer. The buffer size parameter is an optional parameter that is given to the GridFTP protocol to set the buffer size to a different value than the system set buffer size. At the same time, the optimization server will estimate the optimal throughput that can be achieved and the time needed to finish the specified transfer between sites A and B. This information is then returned back to the User/Client making the request.

3.2 Integration with Stork Data Scheduler

Stork is a batch scheduler specialized in data placement and movement [18]. Optimization of end-to-end data transfer throughput is an important problem for schedulers like Stork, especially when moving large-scale datasets across wide-area networks.

In this implementation, Stork is extended to support both estimation and optimization tasks. A task is categorized as an $estimation$ task, if only estimated information regarding to the specific data movement is reported without the actual transfer. On the other hand, a task is categorized as $optimization$ if the specific data movement is performed according to the optimized estimation results.

Stork inherits ClassAds from Condor [32] batch scheduler which are used for submission of data placement jobs. We extend ClassAds with more fields and classify them as *transfer* or *estimation* by specifying the dap_type field. If it is an *estimation* type, it will be submitted directly to EOS, otherwise it will be submitted to the Stork server, which in turn performs the data transfer with or without optimization. Since an estimation task takes much shorter time than an optimization task, distinguishing the submission path by different task types enables an immediate response to the estimation tasks. $Optimization$ field is added to ClassAds in order to determine if the specified transfer will adopt the optimization strategy supplied by EOS. If $optimization$ is specified as YES, then the transfer is performed by using the optimized parameters acquired from EOS, otherwise, it will use the default values.

Another important field added to ClassAds is $use\_history$. This option enforces EOS to search from the database which keeps the optimized parameters for the previous transfers of one specified source and destination pair. If there is such a record, then Stork will use the history information to perform transfers, otherwise, EOS should first perform optimization and store the information in the database, then provide Stork with the optimized parameters. The user is able to define the tcp buffer size as well as sampling size parameters. If

not defined by the user, Stork will use a default sampling size of 20MB and the default system buffer size. The dynamic decision of sampling size is a future issue to be studied. Below is an example submission file to the Stork server for a data transfer with optimization:

```
[
dap_type   = "transfer";
stork_server  =  "oliver1.loni.org";
opt_server = "oliver1.loni.org";
src_url = "gsiftp://eric1.loni.org/default/scratch/test.dat";
dest_url = "gsiftp://qb1.loni.org/default/scratch/dest.dat";
optimization = "YES";
arguments = "-b 128K -s 10M";
output = "tran.out";
err = "tran.err";
log = "tran.log";
x509proxy =  "default";
]
```

While *-s* represents the sampling size *-b* represents the TCP buffer size. Both are optional parameters.

4 Implementation Technique

In this section, we present the implementation details of our Estimation and Optimization Service (EOS). Depending on whether we choose to use GridFTP or Iperf, the implementation slightly differs, because GridFTP supports third-party transfers whereas Iperf works as a client/server model. Considering the differences of the two categories of data transfer tools, we will discuss the implementation of EOS server based on both GridFTP and Iperf. The implementation technique used for these two data transfer tools can be applied to other data transfer tools no matter it supports third-party transfers or not.

The implementation of EOS based on tools supporting third-party transfers is simply a typical client/server model. We have a client module running on the user site and an optimization server module running on one of the machines that is part of the Grid. On the other hand, for the implementation of EOS for data transfer tools not supporting third-party transfers such as Iperf, we need an extra module running on the remote source and destination sites to invoke the tool. The client module of the service is embedded into Stork client application and the requests are done by using ClassAds. The server module on the other hand is independent of the Stork server and able to handle requests coming both from Stork client and Stork server.

4.1 Optimization Server Module

The implementation of the optimization server module is more complicated than that of the client side module. The server should support multiple connections from thousands of clients simultaneously. The processing time for each client should be less than a threshold. Otherwise the user would prefer to perform the data transfer using the default configurations since the time saved by using optimized parameters cannot compensate the time waiting for the response from the optimization server.

There is a slight difference on the implementation based on tools supporting third-party transfers and those do not. In common, the optimization server keeps listening to the request from clients at a designated port. When a new request arrives, it accepts the connection and forks a child process to execute that request. Then the parent process continues to listen to new connections leaving the child process to respond to the client's request.

The child process is responsible for sampling data transfers between the remote sites and get the data pairs (throughput and number of parallel streams) from them. Then it will analyze the data and generate an aggregate throughput function with respect to the number of parallel streams. Finally, it will calculate the maximum aggregate throughput with respect to the optimal number of parallel streams and send back the information to the client. Algorithm 1 presents the outline of the optimization server.

At step 13 in Algorithm 1, the performing of sampling transfers is different on data transfer tools that support third-party transfers and tools that do not support third-party transfers. For the implementation based on GridFTP, the child process is able to invoke *globus-url-copy* command to control the data transfers between the remote sites. However, for the implementation based on Iperf, the child process belonging to the optimization server has no privilege to control the data transfers between the remote sites. We need an extra module running on the remote sites that can be connected by the optimization server. So the optimization server plays dual roles. When a request comes from the client it acts as a server and when it asks the remote module to start Iperf transfers, it acts as a client.

4.2 Quantity Control of Sampling Data Transfers

The time interval between the arrival of a request from the client until an optimized decision is made for the corresponding request mainly depends on the time consumed on the sampling data transfers. The cost of application of the mathematical model on the sampling data and derivation of optimal parameters is negligible, around several milliseconds on a $2.4Ghz$ CPU. However, each sampling data transfer takes nearly 1 second based on the sampling size. At least 3 sampling data transfers are required because of the property of the mathematical model we propose. However relying only on 3 measurements makes the model susceptible to the correct selection of the three parallelism levels.

We propose to find a solution to satisfy both the time limitation and the accuracy requirements. Our approach doubles the number of parallel streams for every iteration of sampling, and observe the corresponding throughput. While the throughput increases, if the slope of the curve is below a threshold between successive iterations, the sampling stops. Another stopping condition is if the throughput decreases compared to the previous iteration before reaching that threshold. Algorithm 2 presents the outline of the sampling method.

Algorithm 1 The optimization server implementation

```
create a socket to be connected by the client
bind the socket to an empty port
listen to this port
while TRUE do
  if a new connection request arrives then
    the optimization server accepts the connection from the client program
    processId ← fork()
    if processId = parent processId then
      back to listening to the designated port
    else {in the child process}
      Child : close the listening port
      Child : receive the request information from the client
      Child : perform sampling transfers
      Child : build a mathematical model and process the sampling results
      Child : send back the optimized parameters to the clients
      Child : close the connection
      Child : terminate
    end if
  else {no new connection request comes}
    block until a new connection comes
  end if
end while
```

Algorithm 2 Sampling data transfers

```
threshold ← α
streamNo1 ← 1
throughput1 is the throughput corresponding to streamNo1
repeat
  streamNo2 ← 2 * streamNo1
  throughput2 is the throughput corresponding to streamNo2
  slope ← (throughput2 − throughput1) / (streamNo2 − streamNo1)
  streamNo1 ← streamNo2
  throughput1 ← throughput2
until slope < threshold
```

Let n be the number of parallel streams with respect to the maximum aggregated throughput of the underlying network. According to our exponentially increasing scheme,the total sampling time s is equal to the logarithm of n, i.e, $s = \log n$. For example, if the optimal parallel number of streams is less than 32, we only need less than 5 sampling iterations.

5 Experimental Results

Our experiments have two categories: i) in the first category, we measure the accuracy of the optimization service as a stand-alone application and test in various environments by changing parameters such as the sampling and file sizes; ii) in the second category, we decided to measure the scalability of our approach in terms of many-task-computing and conducted the experiments in the form of job submissions to the Stork data scheduler. Table 1 presents the system information of the test environment we have used in the experiments. We have used clusters that are part of the LONI and Teragrid network The end-system characteristics as well as the network characteristics affect the maximum throughput we could achieve in the transfers conducted.

5.1 Optimization Service as a Stand-alone Application

In these experiments, requests are sent to the optimization service and the optimized results based on the prediction of the service are compared to actual data transfers performed with GridFTP.

We evaluate our results based on three metrics. First, we compare the throughput of a default transfer for a specific file size and the throughput obtained with an optimized transfer. Second, we add the overhead of the optimization cost to the time of the optimized transfer and compare it to the time of the non-optimized transfer. We do this to see if the optimization cost does not surpass the time gained by optimizing the transfer. Finally, we compare the throughput of an actual optimized transfer and the estimated throughput given by our optimization service.

Figure 2 shows the average results of test runs on LONI hosts with 1Gbps interface. The file size is ranged in [256MB-1GB-10GB] and the sampling size takes values in [10MB-100MB]. For all of the cases, the optimized throughput reaches up to 900Mbps while the transfers done with default configurations with single stream and system default buffer size stays in 100Mbps at most. As the sample size is increased the optimized throughput also increases until 25 MB, after that point increasing the sample size does not increase the optimized throughput although a more accurate throughput estimation is made . In all of the cases the total transfer time including the overhead of the optimization service does not surpass the time of the non-optimized transfers and for large file sizes it is even negligible. For 256MB file transfer using a 100MB sample size is not wise and we also know that 20MB sample size is enough to get maximum throughput optimization. Since our service tries to estimate the instant throughput it can predict more accurate results comparing to actual optimized data transfer throughput as the sample size increases.

In the second test case, we have used two clusters with 10G network interfaces over Teragrid with 30ms RTT(3). Due to the long RTT and auto-tuning used in the end-systems, we have used large sampling and file sizes to fill the network pipe. In these tests we used samplings from the file to be transferred. In all of the cases the optimized throughput was better than the non-optimized version as much as 2-4 times especially for large sampling sizes. The total time including the overhead of the service

TABLE 1
Test environment

Resource	System	Disk system	Interface	Buffer Size
painter1.loni.org(LONI)	Redhat Linux	Lustre	1G	128K
louie1.loni.org(LONI)	Redhat Linux	Lustre	1G	128K
eric1.loni.org(LONI)	Redhat Linux	Lustre	10G	Linux-Autotuning
oliver1.loni.org(LONI)	Redhat Linux	Lustre	10G	Linux-Autotuning
neptune.loni.org (LONI)	IBM AIX	Local Disk	10G	4M
zeke.loni.org (LONI)	IBM AIX	Local Disk	10G	4M
Ranger(TACC-Teragrid)	Redhat Linux	Lustre	10G	Linux-Autotuning
Abe(NCSA-Teragrid)	Linux (CentOS)	Lustre	10G	Linux-Autotuning

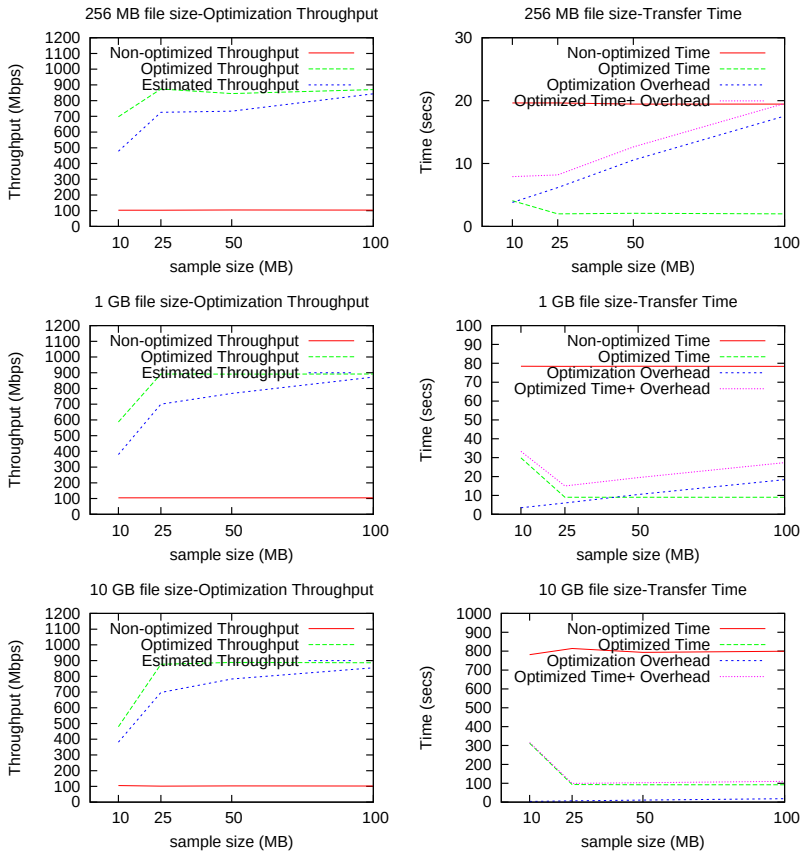

Fig. 2. Optimization results over LONI network with 1Gbps network interfaces based on GridFTP

was always less than the non-optimized time. In our previous study [33], a more detailed experiment set is given including Iperf and local area experiments.

5.1.1 Analysis of Number of Streams

The average number of streams is an important comparison metric to understand the characteristics and behavior of the service depending on the different network and machine configurations. In Figure 4, we present the average number of streams used by the service over the LONI and TeraGrid networks for transfers conducted between machines with 1Gbps and 10Gbps network interfaces. In the first figure, where the system set buffer size is 128K, and the RTT is around 10ms, the number of streams gets stable after 25MB sampling size. The parallel stream number is very high for 10MB sampling size, while for larger sampling sizes it is around 8-9.

On the other hand, the TeraGrid network results are quite different. As the sampling size increased the average number of parallel streams to be used increased as well due to long RTT and underlying buffer auto-tuning. The MTU size is set to use jumbo frames. The average number of streams gets stable around 7-8 streams.

The optimization service gives good results and the overhead is negligible for most of the cases especially for large file sizes while the total overhead does not surpass the non-optimized time in almost all of the cases. A small sample size is enough to reach the maximum throughput that can be obtained and further increasing the sample size does not have an effect on the optimized throughput

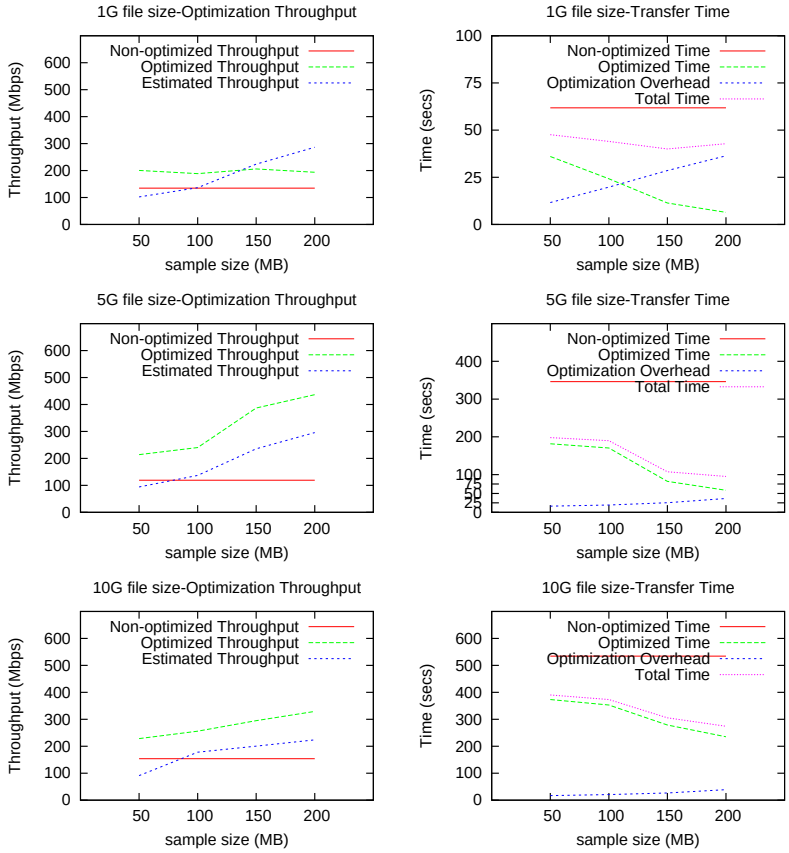

Fig. 3. Optimization results over Teragrid with 10 Gbps network interfaces based on GridFTP

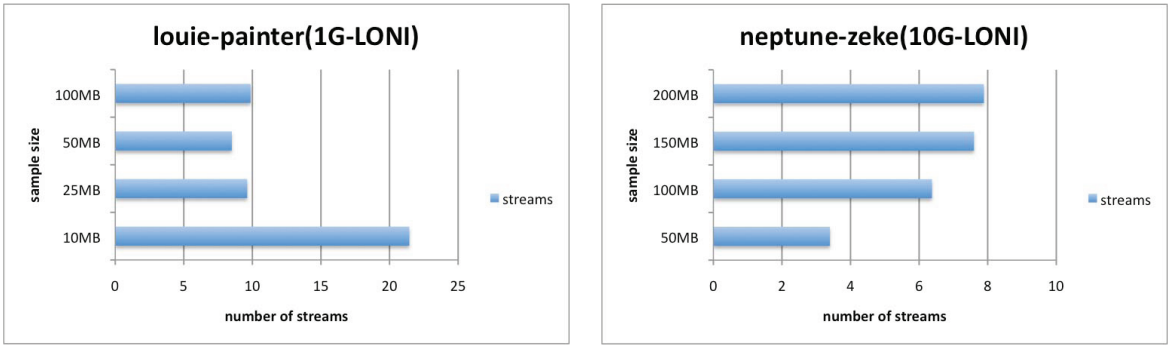

Fig. 4. Average number of streams used per transfer

for short RTTs however it becomes a dominating factor for long RTTs. The estimated throughput is very accurate with larger sample sizes for short RTTs.

5.2 Optimization Service as part of the Stork Data Scheduler

In this section, we designed our experiments to measure the efficiency of the optimization service when it is embedded to a data-aware scheduler and the requests are done by the jobs submitted and the actual transfers are done by the scheduler itself. Four LONI clusters with 1Gbps and 10Gbps interfaces are used for this experiment. The optimization service is able to make prediction by either doing immediate sampling or by using the history information over past transfers. That option is left to the user to be specified in the job submission file. We compare the optimization service with immediate sampling or history information to non-optimized transfers as well as transfers with a fixed parallel stream number of 4. We used 4 streams because it is believed that 3-4 is a good number to fully utilize the network.

In the first test case, we measure the scalability of the optimization service when it is embedded to the Stork Data Scheduler. The number of jobs is the main

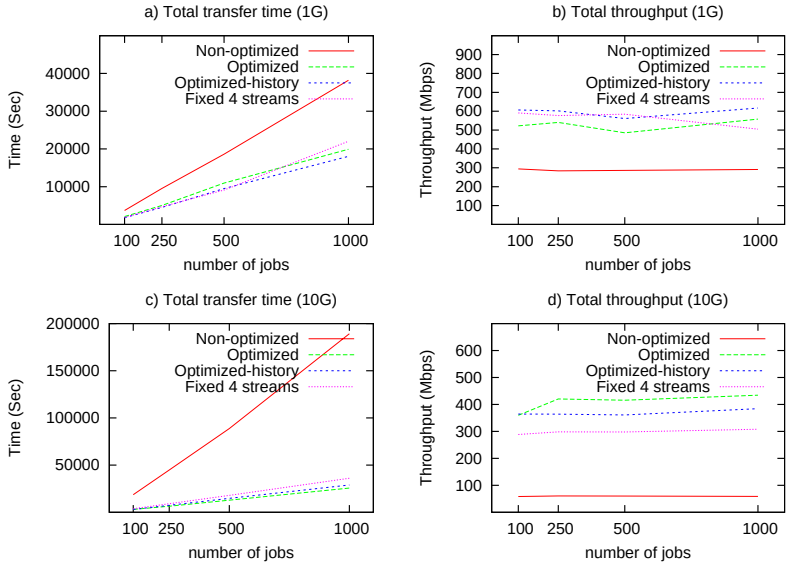

Fig. 5. Total time and throughput of jobs submitted to Stork scheduler

parameter of which effects over the transfer time and throughput is measured. It is ranged between [100-1000]. For each job a random file size is picked from the list of [100M, 256M, 512M, 1G, 5G]. To better measure the overhead of the service and prevention of overlapping the jobs, we submit the jobs all together however configured the Stork server to execute one job at a time.

Figure 5 shows the effect of number of jobs over total transfer time and total throughput for 1Gbps and 10Gbps network configurations. The total transfer time, includes overhead of the optimization as well as of the scheduler and is the difference between the submission time of the first job and the finish time of the last job. The total throughput is calculated by dividing the total data size transferred by the total transfer time.

In Figure 5.a, the total transfer time is presented based on the number of jobs submitted for 1Gbps interface clusters. A range of random size files are transferred and the optimized time is much less than the non-optimized time and this gap between them gets larger as the number of jobs increases. When we use the history information option the time is even less. The transfer time of the fixed-4 streams is better than the optimized with immediate sampling however it is worse than the optimized version with history information. The best results are taken with optimized-history transfers. If the fixed 4 stream is close to the optimal number, the optimized transfer may perform worse due to the overhead of sampling and calculation. However in history transfers this overhead is eliminated. Hence it performs better than the fixed and optimized transfers. In Figure 5.b, the total throughput is presented and the distinction among them is more clear. While a non-optimized transfer reach up to 300Mbps throughput an optimized throughput achieve around 500 Mbps. The optimized-history throughput reaches up to 600Mbps while the fixed 4 stream shows its best results only for 500 jobs. For a 1000 jobs both optimized and optimized-history throughput outperform the fixed-4 streams. Figure 5.c and d shows the transfer time and throughput for 10Gbps clusters. The network throughput of these transfers varies a lot due to large system buffer sizes and disk subsystem used and hence the results are different comparing to 1Gbps interfaces. The optimized transfer time and throughput outperforms all others and reaches up to 425Mbps while the non-optimized throughput is under 100Mbps. The optimized history transfers follows it and reaches up to 390Mbps while the fixed 4 stream transfers stays around 300Mbps only. In this case, because the network throughput varies a lot, it is better to use immediate sampling rather than history information. The number of jobs does not seems to have a significant effect over the total throughput although the optimization gain is higher for high number of jobs.

We have also made a detailed analysis of the job transfer time and queue waiting time to see the overhead of the optimization service. The average transfer time of the job is the difference between the time the job is picked up from the queue for transfer and the time it is removed from the queue. According to Figure 6.a, the optimized time is three times faster than the non-optimized time. The optimized-history and fixed 4 streams time follows it. The queue waiting time is the difference between the time the job is submitted and the time it is picked up from the queue. This also includes the optimization cost as well. Figure 6.b shows that the queue waiting time of the non-optimized version is greater than the optimized version. This is due to the fact that all the jobs are submitted at the same time hence their waiting time depends on the finish times of the previous jobs. The waiting times of the optimized-history and the fixed 4 streams are less than the opti-

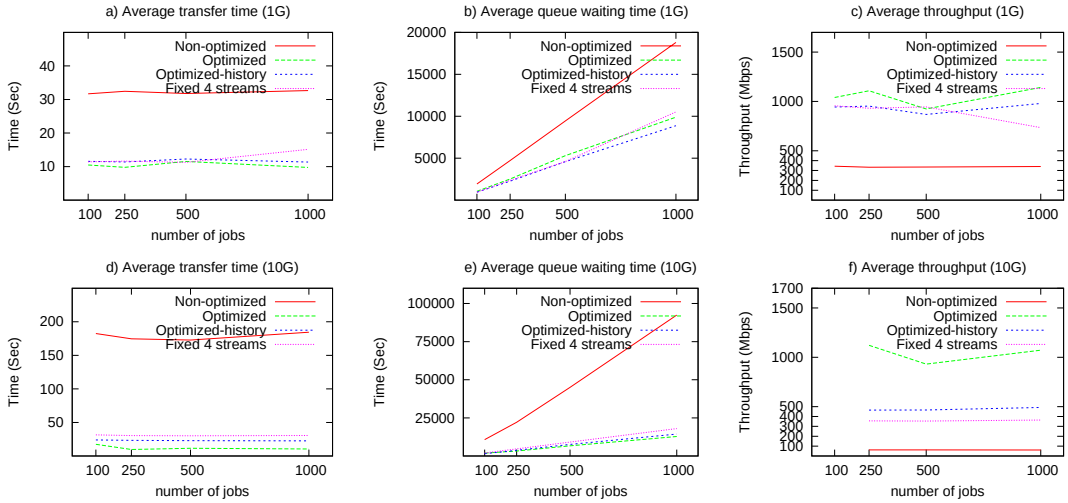

Fig. 6. Average transfer time, queue waiting time and throughput of jobs submitted to Stork Data Scheduler

mized version because there is no sampling overhead. Figure 6.c presents the average throughput excluding the optimization overhead. The optimized throughput could reach up to over 1Gbps while the non-optimized throughput stays around 300Mbps. The throughput of the optimized version is better than the fixed 4 streams and optimized history transfer because the immediate sampling gives a better prediction information regarding the current network conditions when we exclude the sampling overhead.

The results for the 10Gbps interface is different from the 1Gbps interfaces in terms of the gap between the optimized and non-optimized throughput (Figure 6.d,e,f). The optimized throughput outperforms all others. It is around 1Gbps while the optimized history throughput is around 500 Mbps(Figure 6.f). The fixed 4 streams throughput reaches up to only 400 Mpbs. One interesting point is that the queue waiting time of the optimized version is less than the queue waiting time of all others. This could be due to the large gap in average throughput. In short, it is wiser to use optimized version when the network throughput varies a lot but using history information is better when the network is mostly stable.

Another important parameter to be analyzed is the file size. We measure the effect of optimization for various file sizes which are categorized as small and large. Small file sizes range between 50-250 MB while large file sizes range between 0.5-2.5 GB. For a total of 200 jobs, random file sizes are selected from the range and the average throughput is compared for non-optimized, optimized, optimized with history data and fixed 4 streams cases. Figure 7.a shows the compared average throughput results with small file sizes for transfers with 1GigE interface. The optimized throughput is better than the non-optimized throughput. Both increase as the file size increases until 200MB. The optimized results compete with the fixed 4 streams throughput results and the optimized throughput with history information follows them but outperforms the non-optimized throughput except for the 250MB file size. For large file sizes (Figure 7.b), the optimized throughput outperforms all others. The optimized throughput with history information follows it and the worst performance is presented with non-optimized results. For transfers with 10Gig interface (7.c and 7.d), the gap between the non-optimized and optimized average throughput increases upto 1G file size. The optimized throughput outperforms all others while optimized transfers with history information follows it.

The majority of the test results presented in this paper assumes that the scheduler executes one job at a time. However, to understand the effect of jobs executing concurrently, we have designed a test case that will increase the jobs executed by Stork to 2 and conducted transfers for 2 different files of 2GB size between the same source and destination. For this experiment we have used the 4 Linux clusters in our testbed with 1G and 10G cards. While the former had a fixed 128K buffer size the latter had Linux autotuning. We have submitted 500 jobs and calculated the average throughput per job and the total throughput to see the effect of job parallelism better. In Figure 8, the average throughput decreases 100Mbps when the job parallelism is increased for the 10G machines while it stays the same for 1G machines. There is also a 2 stream decrease and a 1 stream increase in the average number of streams respectively. The major difference is seen on the total throughput results, because the throughput is increased almost 2 times when the job parallelism is set to 2. The total throughput values are much lower than the average throughput values for no job parallelism. The reason for the increase in the throughput could depend on many reasons. The first reason for the 10G is that the network bandwidth is not utilized due to end-system capacities and settings(e.g.disk access). Concurrent access improves the throughput in this case. For both of the settings the sampling sizes may not be enough to fill the

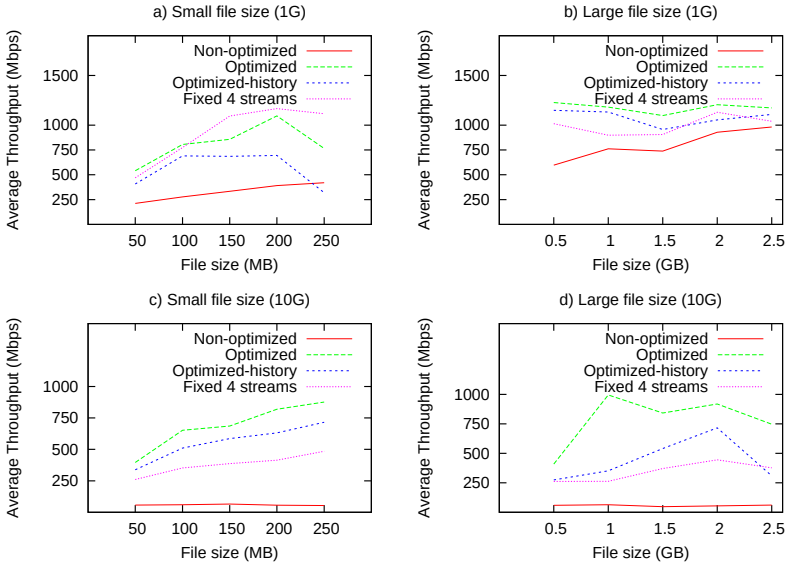

Fig. 7. Effect of filesize over optimization throughput

NIC capacity hence the network pipe. A more detailed study for analyzing and finding the correct setting of the job parallelism is a future work to be implemented to the Stork scheduler.

Overall, the optimization service improves the throughput for all file size ranges in different nework settings. The fixed 4 streams throughput performs worse and that disproves the claim that a fixed number of streams is able to get the same throughput with an optimized stream number.

6 Conclusion

This study describes the design and implementation of a network throughput prediction and optimization service for many-task computing in widely distributed environments. This involves the selection of prediction models, the quantity control of sampling and the algorithms applied using the mathematical models. We have improved an existing prediction model by using three prediction points and adapting a full second order equation or an equation where the order is determined dynamically. We have designed an exponentially increasing sampling strategy to get the data pairs for prediction. The algorithm to instantiate the throughput function with respect to the number of parallel streams can avoid the ineffectiveness of the prediction models due to some unexpected sampling data pairs.

We implement this new service in the Stork Data Scheduler, where the prediction points can be obtained using Iperf and GridFTP samplings. The experimental results justify our improved models as well as the algorithms applied to the implementation. When used within the Stork Data Scheduler, the optimization service decreases the total transfer time for a large number of data transfer jobs submitted to the scheduler significantly compared to the non-optimized Stork transfers.

7 Future Work

The optimization service we provide could be improved in many ways such as fairness among multiple data transfers, decision of sampling sizes for the data transfers and adaptation to the network variations that could occur during a transfer. We are currently working on a version where the samplings are repeated during the transfer by using the transfer file itself and change the optimal stream number based on these variations so that the optimization scheme will be adaptive to the transfer speed changes during the transfer. We believe this feature will be helpful especially for large file transfers. As a future work, we plan to develop sampling size algorithms based on the network as well as the size of the file to be transferred. Currently, we are using the service for each data transfer job submitted separately; however, we plan to develop a sharing mechanism for parallel transfer jobs having the same source and destination sites. The ultimate goal for the service would be the addition of multiple CPU and disk parameters to remove the end-system bottlenecks.

8 Acknowledgments

This project is in part sponsored by the National Science Foundation under award numbers CNS-0846052 (CAREER), CNS-0619843 (PetaShare), OCI-0926701 (Stork) and EPS-0701491 (CyberTools), by the U.S. Department of Energy under Award Number DE-FG02-04ER46136 (UCoMS), and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03.

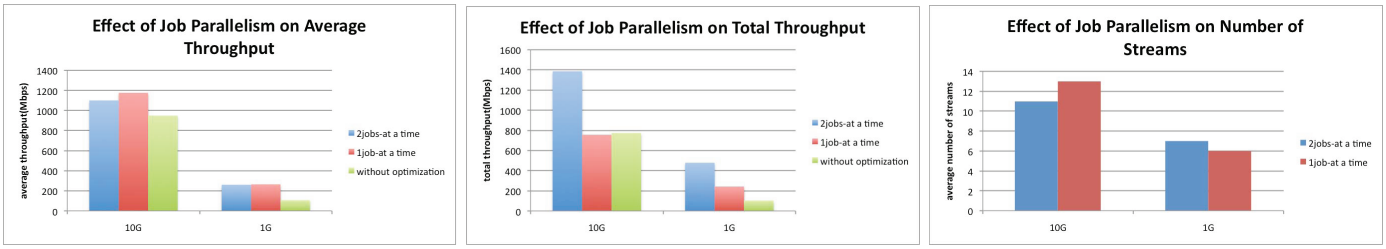

Fig. 8. Effect of job parallelism

REFERENCES

[1] I. Raicu, I. Foster, and Y. Zhao, "Many-task computing for grids and supercomputers," in *IEEE Workshop on Many-Task Computing on Grids and Supercomputers(MTAGS08)*, 2008.

[2] "Louisiana optical network initiative (LONI)," http://www.loni.org/.

[3] "Energy sciences network (ESNet)," http://www.es.net/.

[4] "TeraGrid," http://www.teragrid.org/.

[5] S. Floyd, "Rfc3649: Highspeed tcp for large congestion windows."

[6] R. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *Computer Communication Review*, vol. 32(2), 2003.

[7] C. Jin, D. X. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "Fast tcp: from theory to experiments," *IEEE Network*, vol. 19(1), pp. 4–11, Feb. 2005.

[8] H. Sivakumar, S. Bailey, and R. L. Grossman, "Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proc. IEEE Super Computing Conference (SC00)*, Texas, USA, Nov. 2000, pp. 63–63.

[9] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke, "Applied techniques for high bandwidth data transfers across wide area networks," in *Proc. International Conference on Computing in High Energy and Nuclear Physics (CHEP01)*, Beijing, China, Sept. 2001.

[10] H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. K. M. Stemm, "Tcp behavior of a busy internet server: Analysis and improvements," in *Proc. IEEE Conference on Computer Communications (INFOCOM98)*, California, USA, Mar. 1998, pp. 252–262.

[11] T. J. Hacker, B. D. Noble, and B. D. Atley, "Adaptive data block scheduling for parallel streams," in *Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC05)*, July 2005, pp. 265–275.

[12] L. Eggert, J. Heideman, and J. Touch, "Effects of ensemble tcp," *ACM Computer Communication Review*, vol. 30(1), pp. 15–29, 2000.

[13] R. P. Karrer, J. Park, and J. Kim, "Tcp-rome:performance and fairness in parallel downloads for web and real time multimedia streaming applications," Deutsche Telekom Labs, Tech. Rep., 2006.

[14] D. Lu, Y. Qiao, and P. A. Dinda, "Characterizing and predicting tcp throughput on the wide area network," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS05)*, 2005, pp. 414–424.

[15] E. Yildirim, D. Yin, and T. Kosar, "Prediction of optimal parallelism level in wide area data transfers," *To appear in IEEE Transactions on Parallel and Distributed Systems*, 2010.

[16] A. Tirumala, L. Cottrell, and T. Dunnigan, "'measuring end-to-end bandwidth with iperf using web100," in *Proc. Passive and Active Measurement Workshop (PAM03)*, 2003.

[17] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridftp framework and server," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 54.

[18] T. Kosar and M. Livny, "Stork: Making data placement a first class citizen in the grid," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS04)*, 2004, pp. 342–349.

[19] T. J. Hacker, B. D. Noble, and B. D. Atley, "The end-to-end performance effects of parallel tcp sockets on a lossy wide area network," in *Proc. IEEE International Symposium on Parallel and Distributed Processing(IPDPS02)*, 2002, pp. 434–443.

[20] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Modeling and taming parallel tcp on the wide area network," in *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS05)*, Apr. 2005, p. 68b.

[21] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel tcp sockets: Simple model, throughput and validation," in *Proc. IEEE Conference on Computer Communications (INFOCOM06)*, Apr. 2006, pp. 1–12.

[22] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing tcp," *ACM SIGCOMM Computer Communication Review*, vol. 28(3), pp. 53–69, July 1998.

[23] G. Kola and M. K. Vernon, "Target bandwidth sharing using endhost measures," *Performance Evaluation*, vol. 64(9-12), pp. 948–964, Oct. 2007.

[24] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *In Proc. of Passive and Active Measurements (PAM) Workshop*, 2002, pp. 14–25.

[25] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network," in *In Passive and Active Measurement Workshop*, 2003.

[26] P. Primet, R. Harakaly, and F. Bonnassieux, "Experiments of network throughput measurement and forecasting using the network weather," in *CCGRID '02: Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2002, p. 413.

[27] E. Yildirim, I. H. Suslu, and T. Kosar, "Which network measurement tool is right for you? a multidimensional comparison study," in *Proc. of the 2008 9th IEEE/ACM International Conference on Grid Computing(GRID'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 266–275.

[28] J. Strauss, D. Katabi, and M. F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Internet Measurement Conference*, 2003, pp. 39–44.

[29] T. Dunigan, M. Mathis, and B. Tierney, "A tcp tuning daemon," in *Proc. IEEE International Conference of Supercomputing (SC02)*, Nov. 2002, pp. 1–16.

[30] G. Jin, G. Yang, B. R. Crowley, and D. A. Agarwal, "Network characterization service (ncs)," in *Proc. IEEE International Symposium on High Performance Distributed Computing(HPDC01)*, 2001, p. 289.

[31] LBNL, "Netest," http://www-didc.lbl.gov/NCS/netest/.

[32] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 2-4, pp. 323–356, 2005.

[33] D. Yin, E. Yildirim, and T. Kosar, "A data throughput prediction and optimization service for widely distributed many-task computing," in *Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers(MTAGS'09)*, nov 2009.

Dengpan Yin has received his Master's degree of computer science from Southern University in May 2008. Currently, he is pursuing his Phd of computer science at Louisiana State University. Meanwhile, he is working as a graduate research assistant at CCT of LSU. His research interests include distributed computing, Grid computing, parallel computing and high performance network.

Esma Yildirim has received her BS degree from Fatih university and MS degree from Marmara University Computer Engineering Departments in Istanbul, Turkey. She worked for one year in Avrupa Software Company for the Development of ERP Software. She also worked as a Lecturer in Fatih University Vocational School until 2006. She is currently in Louisiana State University as a Phd student in Computer Science since August 2006. She is working for CCT as a graduate research asistant. Her research interests are Distributed Computing, Web technologies and Sofware Engineering.

Sivakumar Kulasekaran is an Information Analyst in the Center for Computation & Technology (CCT) at LSU from Spring 2010. He holds a BE degree in Computer Science and Engineering from Periyar University, India and a M.S. degree in Computer Science and Engineering from Mississippi State University. Dr. Sivakumar received his PhD degree in Computer Science from Mississippi State University in Fall 2009. Dr. Sivakumars research interest includes Distributed Computing, Networks and Security.

Brandon Ross is an undergraduate student in the Computer Science Department at LSU. He is a member of the Distributed Systems Lab team at LSU and works under Dr. Tevfik Kosar as a developer on the Stork project. His areas of interest within the field of computer science include distributed and high-performance computing, computer security, and networking.

Tevfik Kosar is an Associate Professor in the Department of Computer Science & Engineering, University at Buffalo. Prior to joining UB, Kosar was with the Center for Computation & Technology (CCT) and the Department of Computer Science at Louisiana State University. He holds a B.S. degree in Computer Engineering from Bogazici University, Istanbul, Turkey and an M.S. degree in Computer Science from Rensselaer Polytechnic Institute, Troy, NY. Dr. Kosar has received his Ph.D. degree in Computer Science from University of Wisconsin-Madison. Dr. Kosar's main research interests lie in the cross-section of petascale distributed systems, eScience, Grids, Clouds, and collaborative computing with a focus on large-scale data-intensive distributed applications. He is the primary designer and developer of the Stork distributed data scheduling system, and the lead investigator of the state-wide PetaShare distributed storage network in Louisiana. Some of the awards received by Dr. Kosar include NSF CAREER Award, LSU Rainmaker Award, LSU Flagship Faculty Award, Baton Rouge Business Reports Top 40 Under 40 Award, and 1012 Corridors Young Scientist Award.