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

Dengpan Yin
Department of Computer Science and CCT
Louisiana State University
Baton Rouge, LA 70803 USA
dyin@cct.lsu.edu

Esma Yildirim
Department of Computer Science and CCT
Louisiana State University
Baton Rouge, LA 70803 USA
esma@cct.lsu.edu

Tevfik Kosar*
Department of Computer Science and CCT
Louisiana State University
Baton Rouge, LA 70803 USA
kosar@cct.lsu.edu

ABSTRACT

In this paper, we present the design and implementation of a network 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 used to decide the number of parallel streams to achieve best 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 unoptimized transfer time in most cases.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Distributed Systems—*distributed applications*

General Terms

Performance

Keywords

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, ESNet and Teragrid 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 bottleneck on the sending and/or receiving ends, and 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 more and more 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 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 problem especially for wide-area high speed networks where both bandwidth and delay properties are too large which also results in a large amount of time to reach up to the point where the bandwidth is fully saturated. There has been different implementation techniques both in the transport and application levels to overcome the poor network utilization of the TCP protocol. In the transport layer, different variations of TCP have been implemented [2, 3, 4] to utilize high-speed networks but there is not a single adopted protocol to replace it. In the application level, other techniques are found just by using the existing underlying protocol. Opening parallel streams is one way of doing that and is highly used in many application areas.

Parallel streams are able to achieve high throughput by behaving as a single large stream that is the combination of n streams and gets an unfair share of the available bandwidth [5, 6, 7, 8, 9, 10, 11]. However, for low-speed networks, using too many streams reaches the network to a congestion point and after that point, it only causes a drop in the performance. For high-speed networks, the parallel streams decrease the time to reach optimal saturation of the network. Yet, 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 data size.

The computational methods used in today's large scale highly parallel and data-dependent applications has lacked the sufficient handling of data. As applications have become more data-intensive, grew in size and the necessity for parallelization to provide both high-performance and high-throughput increased, it has become more important to pay a special attention to the scheduling of data. Although there are few novel methodologies to handle data such as data-aware schedulers or high-level data planners for efficient placement and scheduling of data (e.g. Stork scheduler, SRMs), the applied methods are high-level and prioritize the efficiency and scalability of the whole application rather than low-level single improvement of each data placement task. We believe that a service that will enhance the data transfer speed of each single data placement task will bring a significant improvement over the whole application's performance that consists of many tasks that are both compute and data-dependent.

In this study, we present the design and implementation of a service that will provide the user with the optimal parallel stream number and a provision of the estimated time and throughput information for a specific data transfer. 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 can give as accurate results as ours with a comparable prediction overhead and we believe that our service is unique in terms of the input requirements, and the practical results it produces.

The current version of our prediction and optimization service supports sampling with Iperf and GridFTP, however we plan to extend it to be a more generic tool. Also it is embedded to the Stork data scheduler as a service that will improve the performance of each data transfer job submitted to it. We submitted a large number of jobs with the request for optimized transfers and have seen that the overall finish time of the optimized jobs was far less than the non-optimized version.

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 as well as the problems faced during implementation; and we provide a detailed explanation of the mathematical models we have developed. Section 4 presents the implementation details; and in Section 5 we discuss the results of the experiments conducted. Finally in Section 6, we discuss the conclusions made.

2. RELATED WORK

The studies that try to find the optimal number of streams are so few and they are mostly based on approximate theoretical models [12, 13, 14, 15, 16]. They all have specific constraints and assumptions. Also the correctness of the proposed models are mostly proved with simulation results only. Hacker et al. claim that the total number of streams behaves like one giant stream that transfers in capacity of total of each streams' achievable throughput [12]. However, this model only works for uncongested networks. Thus, it cannot provide a feasable solution for congested networks. Another study [15] declares the same theory but develops a protocol which at the same time provides fairness. Dinda et al. [13] model the bandwidth of multiple streams as a partial second order equation and require two different throughput measurement of different stream numbers to predict the others. However, this model cannot predict the optimal number of parallel streams necessary to achieve best transfer throughput. Yet in another model [14], the total throughput always shows the same characteristics depending on the capacity of the connection as the number of streams increases and 3 streams are sufficient to get a 90% utilization. A new protocol study [16] 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. Unfortunately, users do not want to present this information or have no idea what to supply to a data transfer tool. They need a means to make a projection of their data transfer throughput and must gather the information to optimize their transfer without caring about the characteristics of an environment and the transfer at hand. For individual data transfers, instead of relying on historical information, the transfers should be optimized based on instant feedback. In our case, this optimization is achieving optimal number of parallel streams to get the highest throughput. However, an optimization technique not relying on historical data in this case must not cause overhead of gathering instant data that is larger than the speed up gained with multiple streams for a particular data size. Gathering instant information for prediction models could be done by using network performance measurement tools [17, 18, 19, 20, 21] or doing a miniature version of the transfer.

In our service, we propose to use Iperf [20] and GridFTP [22] to gather the sampling information to be fed into our mathematical models. Both of the tools are widely adopted by the Grid community and 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 instant sampling information, we provide a service that will give the optimal parallel stream number 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 throughputs 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

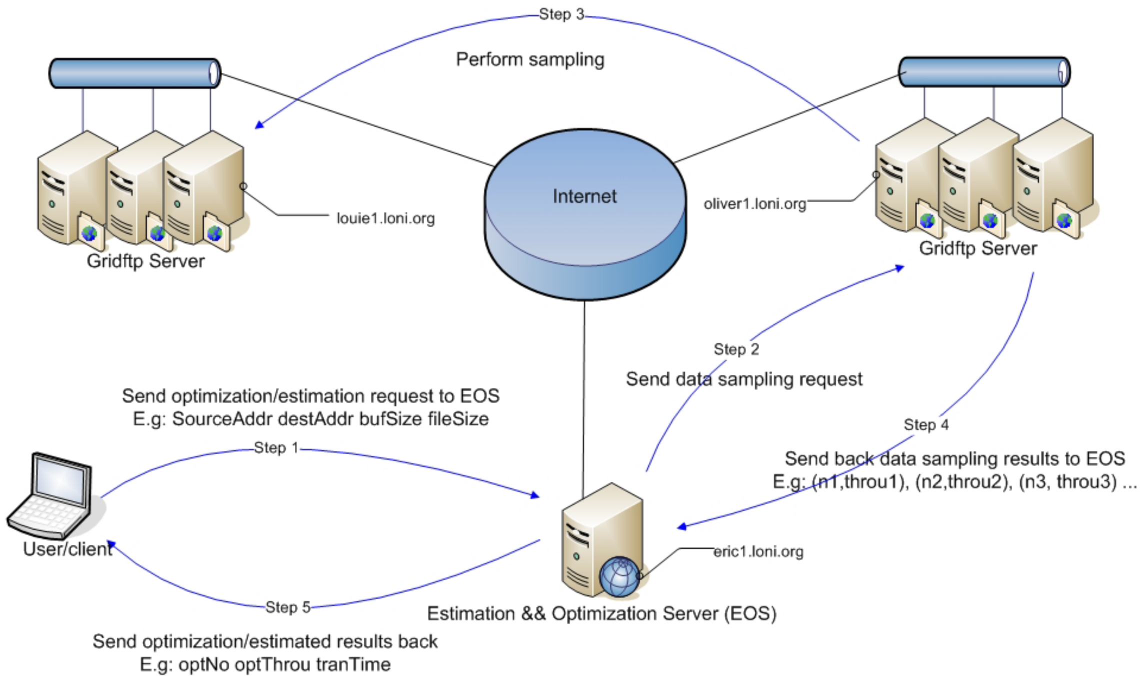

Step 3
Perform sampling
louie1.loni.org
Gridftp Server
Internet
oliver1.loni.org
Gridftp Server
Step 2
Send data sampling request
Send optimization/estimation request to EOS
E.g: SourceAddr destAddr bufSize fileSize
Step 1
Step 4
Send back data sampling results to EOS
E.g: (n1,throu1), (n2,throu2), (n3, throu3) ...
User/client
eric1.loni.org
Step 5
Estimation && Optimization Server (EOS)
Send optimization/estimated results back
E.g: optNo optThrou tranTime

Figure 1: Design sketch

stage, we have implemented the optimization service based on both Iperf and Globus. 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 version of the service. Site A and site B represent two machines between which the user wants to transfer data. For the GridFTP version, those machines should have GridFTP servers and GSI certificates installed. For the Iperf version, those machines should have Iperf servers running as well as a small remote module ($TranServer$) that we have implemented to perform third-party Iperf sampling. Optimization server is the orchestrator machine 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. Client/User represents the terminal that sends out the request of optimization to the optimization server. All of them are connected via WAN or LAN.

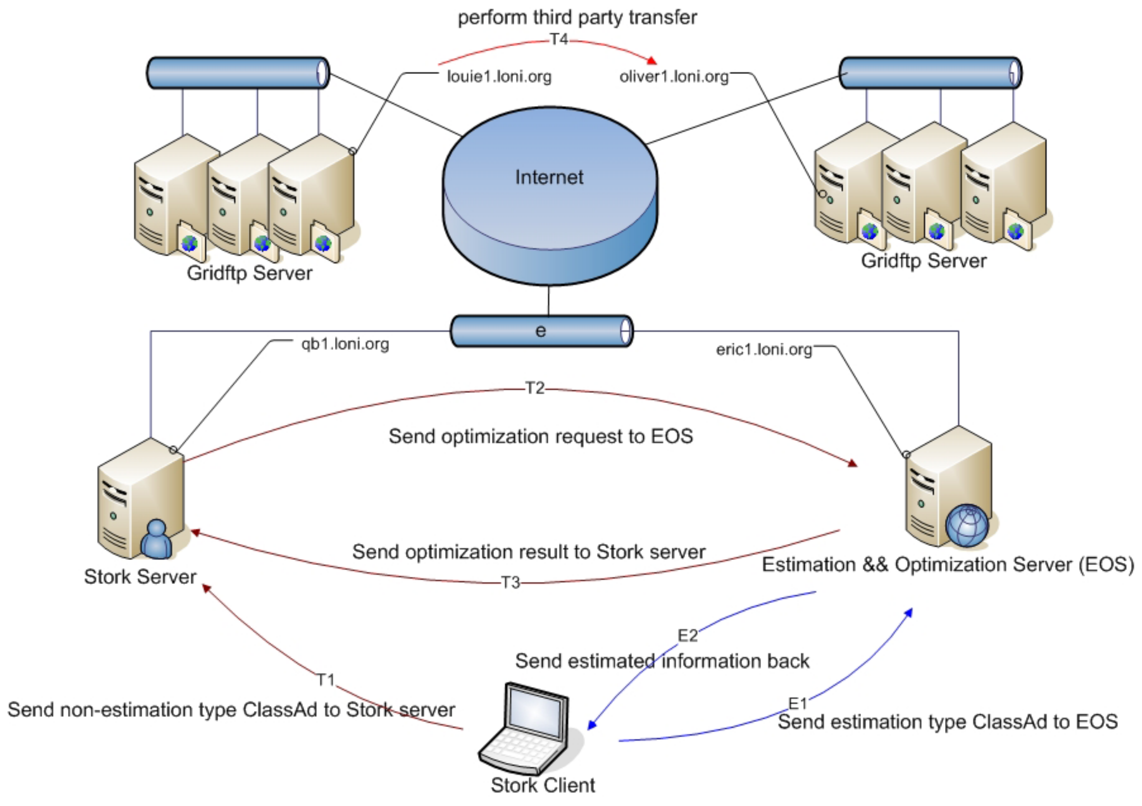
When a user wants to transfer data between site A and site B, the user will first send a request to the optimization server. The optimization server will process the request from the user within a short time and respond to the user with the optimal parallel stream number to do the transfer. At the same 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 also returned back to the user.

3.2 Integration with Stork Scheduler

Stork is a batch scheduler specialized in data placement and movement [23]. Optimization of end-to-end data transfer throughput is an important issue for a scheduler like Stork, especially when moving large size of data across wide-area networks. Figure 2 presents the optimization service integrated with the Stork scheduler.

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 to be done according to the optimized estimation results. Henceforth this service is named as EOS(Estimation and Optimization Service) in short.

Stork inherits ClassAds structure from Condor [24] batch scheduler which are used for submission of jobs. We extend ClassAds with more fields and classify them as estimation or transfer 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. 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 done by using the optimized parameters acquired from EOS, otherwise, it will use the default value. Another important field added to ClassAds is *use_history*. This option enforces EOS to search from



perform third party transfer
T4
louie1.loni.org
oliver1.loni.org
Internet
Gridftp Server
Gridftp Server
e
qb1.loni.org
eric1.loni.org
T2
Send optimization request to EOS
Send optimization result to Stork server
T3
Stork Server
Estimation && Optimization Server (EOS)
E2
Send estimated information back
T1
Send non-estimation type ClassAd to Stork server
E1
Send estimation type ClassAd to EOS
Stork Client

Figure 2: EOS adopted by Stork

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 into the database, then provide Stork with the optimized parameters. Below is an example submission file to Stork server for a 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/work/default/scratch/dest.dat";
optimization = "YES";
arguments = "-b 128K -s 10M";
output = "tran.out";
err = "tran.err";
log = "tran.log";
x509proxy =  "default";
]
```

3.3 Prediction Scheme

We have developed two mathematical models to predict the aggregated throughput of parallel streams that could 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:

$$Th <= \frac{MSS}{RTT} \frac{c}{\sqrt{p}} \tag{1}$$

In this equation, 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 is in general IP maximum transmission unit (MTU) size - TCP header. Round trip time 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 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 the study in [12], an application opening n connections actually gains 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:

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

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

Dinda et al [13] model the relation between n, RTT and p as a partial second order equation by using two throughput measurements of 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 order two equation, we increase the sampling number to three and either use a full second order equation or an equation where the order is determined dynamically. For the full second order model, we define a variable p'_n :

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

According to Equation 3, we derive:

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

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 sampling or past data transfers .

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

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

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

By solving the following three equations we could place the a',b' and c' variables to 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 based on the values of the samples:

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

After we calculate the optimal number of parallel streams, we can calculate the maximum throughput corresponding to that number. The optimization server needs to get at least three suitable throughput values of different parallelism levels through sampling to be able to apply the models. When the requests from the users come, the optimization server will initiate data transfers between the expected source and destination supplied by the user. This procedure terminates when the optimization server determines that it has obtained sufficient sampling data for an accurate prediction.

4. IMPLEMENTATION TECHNIQUE

In this section, we present the implementation details of our service design. Depending on whether we choose to use Globus GridFTP or Iperf, the implementation slightly differs because while GridFTP supports third-party transfers, Iperf works as a Client/Server model. Considering the differences of the two categories of data transfer tools, we will discuss the implementation of optimization 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 optimization service 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, the implementation of optimization service 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 ClassAdds. The server module on the other hand is independent of the Stork server and able 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 process 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 does 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.

Algorithm 1 The optimization server implementation

```
 1: create a socket to be connected by the client
 2: bind the socket to an empty port
 3: listen to this port
 4: while TRUE do
 5:   if a new connection request arrives then
 6:     the optimization server accepts the connection from
        the client program
 7:     $processId \leftarrow fork()$
 8:     if $processId = parent\ processId$ then
 9:       back to listening to the designated port
10:     else {in the child process}
11:       Child : close the listening port
12:       Child : receive the request information from the
          client
13:       Child : perform sampling transfers
14:       Child : build a mathematical model and process
          the sampling results
15:       Child : send back the optimized parameters to
          the clients
16:       Child : close the connection
17:       Child : terminate
18:     end if
19:   else {no new connection request comes}
20:     block until a new connection comes
21:   end if
22: end while
```

4.2 Quantity Control of Sampling Data Transfers

The time interval between the arrival of a request from the client and an optimized decision 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 models 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 slop 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 2 Sampling data transfers

```
 1: $threshold \leftarrow \alpha$
 2: $streamNo1 \leftarrow 1$
 3: $throughput1$ is the throughput corresponding to
    $streamNo1$
 4: repeat
 5:   $streamNo2 \leftarrow 2 * streamNo1$
 6:   $throughput2$ is the throughput corresponding to
      $streamNo2$
 7:   $slop \leftarrow \frac{throughput2 - throughput1}{streamNo2 - streamNo1}$
 8:   $streamNo1 \leftarrow streamNo2$
 9:   $throughput1 \leftarrow throughput2$
10: until $slop < threshold$
```

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 has two categories. In the first category, we measure the accuracy of the optimization service as a stand-alone application and tested in various environments by changing parameters such as the sampling and file size. 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.

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. Our testbed consists of 256-processor clusters in the LONI network with 1Gbps interface and workstations on the DSL lab at LSU with 100Mbps interfaces. We range the sampling size and file size parameters and conducted tests for 1Gbps and 100Mbps interfaces for GridFTP version and for 1Gbps interface with Iperf version of our service. 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 3 shows the averaged results of tests run on LONI machines with 1Gbps interface. The file size is ranged in

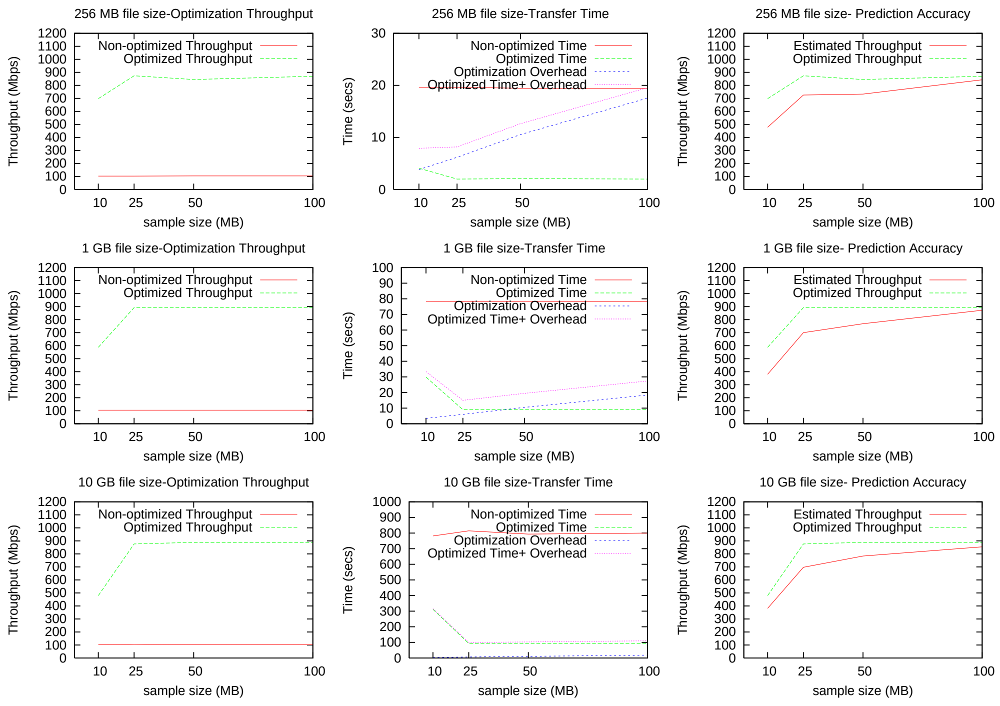


Figure 3: Optimization results over LONI network with 1Gbps network interfaces based on GridFTP

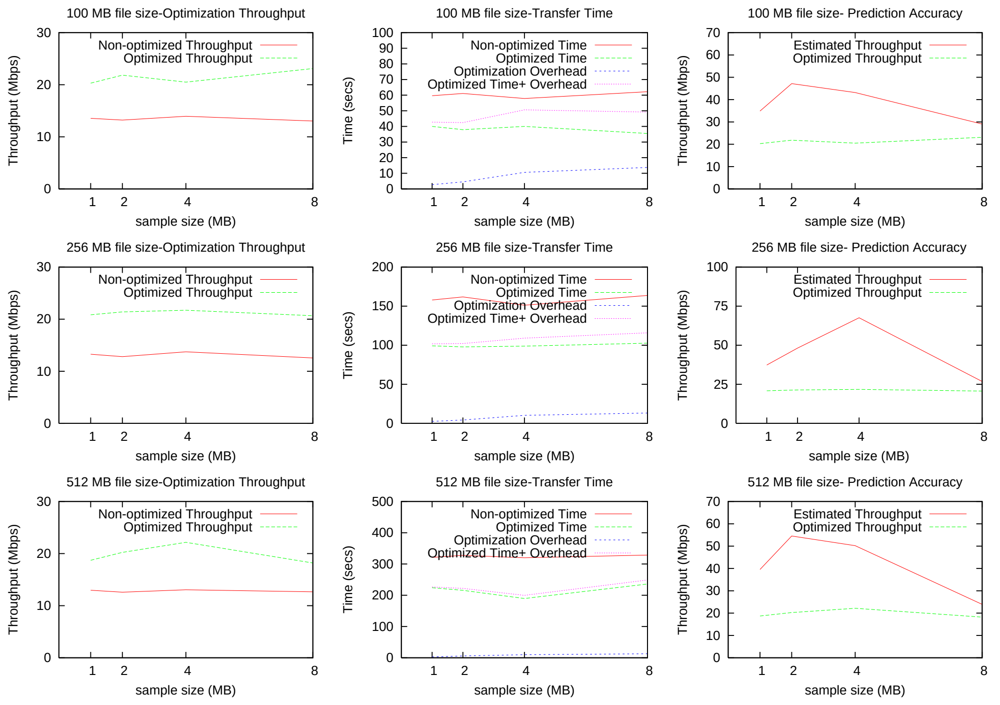


Figure 4: Optimization results between LAN 100Mbps and LONI 1Gbps network interfaces based on GridFTP

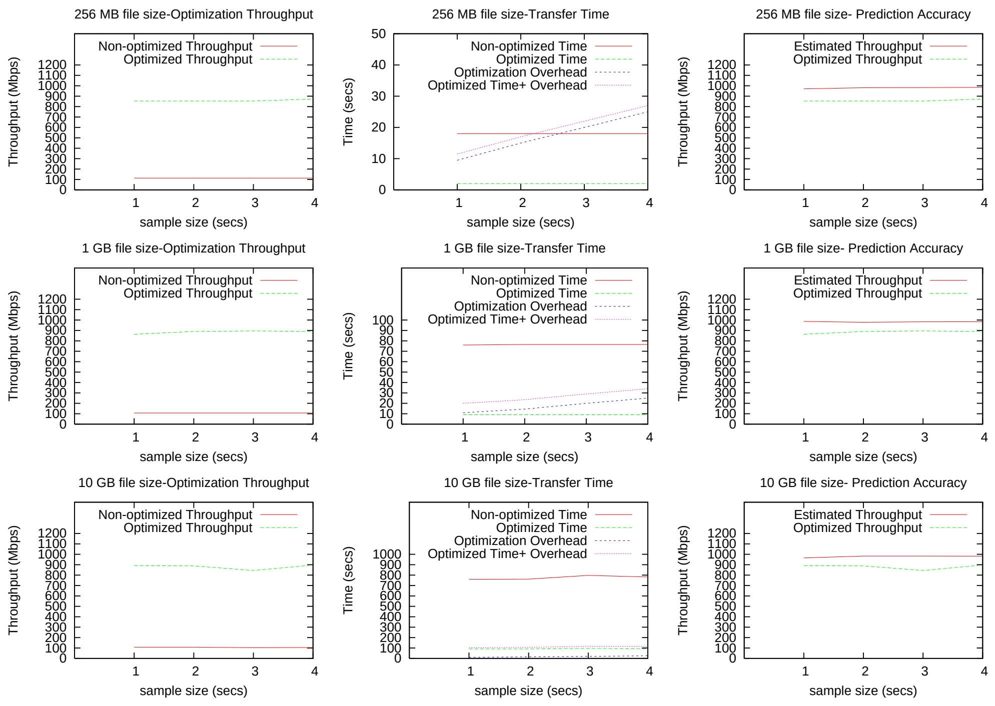


Figure 5: Optimization results over LONI network with 1Gbps network interfaces based on Iperf

[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 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 affect the optimized throughput. 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 conducted transfers between a Linux workstation in DSL lab with a 100Mbps interface and a cluster in LONI network. Figure 4 presents the results obtained from GridFTP optimization service. The file size and sample size ranges are decreased to [100MB-256MB-512MB] and [1M-8M] respectively. Increasing the sample size did not cause a significant change in the optimized throughput which could reach up to 23 Mbps at most. In all of the cases the total transfer time including the optimization overhead did not surpass the non-optimized time and as the file size is increased the optimization cost became negligible. The estimated throughput gave promising accuracy for larger sample sizes.

In the final test case, we tested our optimization service based on Iperf with 1Gbps interface over the LONI network. Iperf accepts the transfer duration as a parameter, hence we decided to range the sampling size in the rage [1-4] seconds. Increasing the sample size did not have a significant effect on the optimized throughput and we could get up to 900Mbps throughput with optimized parameters while the default configurations reached up to 100 Mbps. The tool overhead is negligible for the file sizes of 1GB-10GB and since 1 second is enough to get the highest throughput, the optimization service gives promising results for the 256MB file size case as well. The estimated throughput is very accurate comparing to actual optimized throughput. The results obtained with Iperf is more stable comparing to GridFTP and does not saturates much.

The optimization service gives good results and the overhead is negligible for most of the cases 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. The estimated throughput is very accurate with larger sample sizes.

5.2 Optimization Service as part of the Stork 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. Two LONI clusters with 1Gbps are used for this experiment. The number of jobs are ranged between [100-500]. For each job a random file size is picked from the list of [100M, 256M, 512M, 1G, 5G]. To better measure the over-

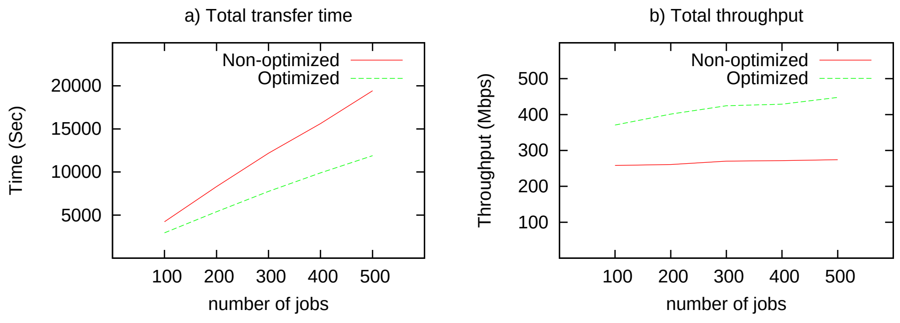


Figure 6: Total time and throughput of jobs submitted to Stork scheduler

head of the service and prevention of overlapping the jobs, we submit the jobs sequentially and wait for the previous job to finish.

In Figure 6.a the total transfer time is presented based on the number of jobs submitted. This time includes overhead of the optimization as well as of the scheduler and includes the submission time of the first job and the finish time of the last job. 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. In Figure 6.b, the total throughput is calculated based on the total time and total amount of data sent. While a non-optimized reach up to 250Mbps throughput an optimized throughput achieve 450 Mbps and the optimized throughput increases as the number of jobs increases.

We have also made a detailed analysis of the job transfer time and queue waiting time to see the overhead of the optimization service. Figure 7.a presents the average transfer time of the job which 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 the figure, the optimized time is twice as faster than the non-optimized time and the transfer time decreases as the number of jobs increases. 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 7.b shows that the queue waiting time of the non-optimized version is less however the difference is negligible considering the time gain in the optimized transfer in Figure 6.a. The last figure (Figure 7.c) presents the average throughput excluding the optimization overhead. The optimized throughput could reach up to 700 Mbps while the non-optimized throughput stays at 300Mbps. Notice that the optimized throughput increases as the number of jobs increases. This suggests that the optimization brings more gain in terms of throughput and time for large number of jobs.

Another important parameter to be analyzed is the file size. For a total of 500 jobs, Figure 8.a shows that the gap between the non-optimized and optimized average transfer times increases as the file size increases. The throughput of the non-optimized transfers increases as the file size increases but it stabilizes after 512MB at 300Mbps. On the other hand, the throughput of the optimized transfers is highest at 1GB file size and able to achieve around 800Mbps throughput. For larger file sizes the throughput decreases to 650Mbps for optimized transfers. Overall, our optimization service improves the time and throughput of data transfer jobs significantly.

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 improves the total finish time of a large number of data transfer jobs submitted to the scheduler significantly comparing to the non-optimized Stork transfers.

7. 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.

8. REFERENCES

[1] I. Raicu, I. T. Foster, and Y. Zhao. Many-Task Computing for Grids and Supercomputers. *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*, 2008.

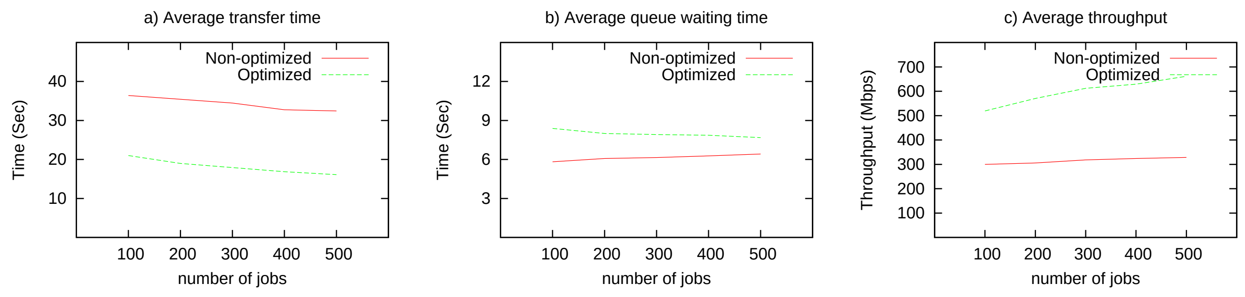


Figure 7: Average transfer time, queue waiting time and throughput of jobs submitted to Stork scheduler

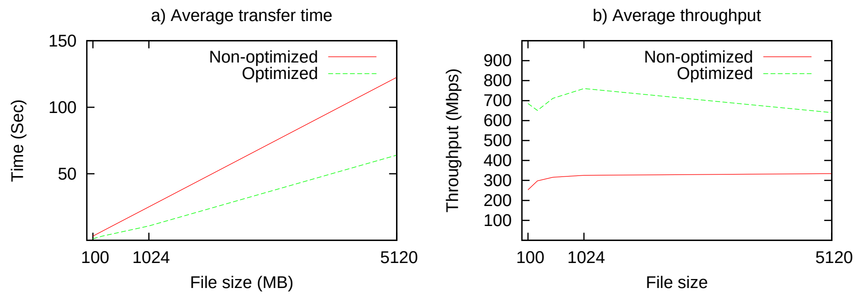


Figure 8: Average transfer time and throughput of jobs based on the file size

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

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

[4] 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.

[5] 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," Texas, USA, Nov. 2000, pp. 63–63.

[6] 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," Beijing, China, Sept. 2001.

[7] H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. K. M. Stemm, "Tcp behavior of a busy internet server: Analysis and improvements," California, USA, Mar. 1998, pp. 252–262.

[8] T. J. Hacker, B. D. Noble, and B. D. Atley, "Adaptive data block scheduling for parallel streams," July 2005, pp. 265–275.

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

[10] R. P. Karrer, J. Park, and J. Kim, "Adaptive data block scheduling for parallel streams," Deutsche Telekom Laboratories, Tech. Rep., 2006.

[11] D. Lu, Y. Qiao, and P. A. Dinda, "Characterizing and predicting tcp throughput on the wide area network," June 2005, pp. 414–424.

[12] 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," 2002, pp. 434–443.

[13] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Modeling and taming parallel tcp on the wide area network," Apr. 2005, p. 68b.

[14] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel tcp sockets: Simple model, throughput and validation," Apr. 2006, pp. 1–12.

[15] 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.

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

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

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

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

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

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

[22] 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.

[23] T. Kosar and M. Livny, "Stork: Making data placement a first class citizen in the grid," in *ICDCS*, 2004, pp. 342–349.

[24] 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.