National Conference on Advances in Computer Science and Applications with International Journal of Computer Applications (NCACSA 2012)
Proceedings published in International Journal of Computer Applications® (IJCA)

Data Scheduler Throughput Prediction using Estimation and Optimization Server for Mtc

CH. Mohana Bindu (M.E)
CSE dept, PITAM

N. Sri Priya
Lecturer / Dept. of Computer Science, PITAM

ABSTRACT

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. We propose Stork data transfer jobs with optimization service can be completed much earlier compared to non-optimized data transfer jobs.

Keywords

Many task Computing, Scheduling.

1. INTRODUCTION

Many-Task Computing can play a significant role in making such predictions feasible. In a widely distributed many-task computing environment, data communication between participating clusters may become a major performance bottleneck. Provide high speed network connectivity to their users. Majority of the users fail to obtain even a fraction of the theoretical speeds promised by these networks due to issues, such as suboptimal 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 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 have 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 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. When applications use the obvious round robin scheduling algorithm for multiplexing data blocks, differences in transmission rate between individual TCP streams can lead to significant data block reordering. This forces the de-multiplexing receiver to buffer out-of-order data blocks, consuming memory and potentially causing the receiving application to stall.

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. 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.

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 a specific data transfer. The optimal number of TCP streams is calculated, using the mathematical models. A user of this service only needs to provide the source and destination addresses and the size of the transfer. None of the existing models and tools can give as accurate results as ours with a comparable prediction overhead.

2. RELATED WORK

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 . However, this model only works for uncongested networks, and cannot provide a feasible solution in case of congestion. Another study declares the same theory, but develops a protocol, which, at the same time, provides fairness. Lu et al. [2] 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 cannot predict the optimal number of parallel streams necessary to achieve the best transfer throughput. In another model [6], the total throughput always shows the same characteristics depending on the capacity of the link as the number of streams increases, and three streams are sufficient to get 90 percent utilization.

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: 1) historical data may not be available for all transfers; 2) 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.

According to Hacker et al. [7], 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.

Lu 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 uses a full second order polynomial or a polynomial where the order is determined dynamically.

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.

In Dunigan et al. [7], 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 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 realtime sampling without constant probing. In our approach, no changes in the transport protocol layer are required. We propose to use Iperf and GridFTP 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 realtime 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.

3.1 Sketch of the Optimization Service

Fig. 1 demonstrates the structure of our design and presents two scenarios based on both, GridFTP and Iperf versions of the service. Sites A and 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. When a user wants to transfer data between sites A and 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, which 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.

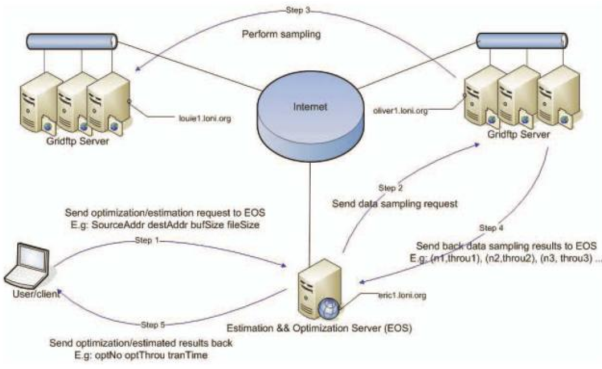
Step 3
Perform sampling
Internet
louie1.loni.org
oliver1.loni.org
Gridftp Server
Gridftp Server
Step 2
Send data sampling request
Send optimization/estimation request to EOS
E.g: SourceAddr destAddr bufSize fileSize
Step 1
Step 4
Send back data sampling results to EOS
E.g: (n1,throu1), (n2,throu2), (n3,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

Fig.1 Overview of the EOS.

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 data sets 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 batch schedulers 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 data type field. If it is an estimation type, it will be submitted directly to an estimation and optimization service (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 whether 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.

4. IMPLEMENTATION TECHNIQUES

We present the implementation details of our 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.

4.1 Optimization Server 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.

4.2 Quantity Control of Sampling Data Transfers

The time interval between the arrivals of requests 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 2.4 Ghz CPU. However, each sampling data transfer takes nearly 1 second based on the sampling size. At least three sampling data transfers are required because of the property of the mathematical model we propose. However, relying only on three measurements makes the model susceptible to the correct selection of the three parallelism levels.

5. EXPERIMENTAL RESULTS

Our experiments have two categories: 1) in the first category, we measured 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; 2) 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.

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.

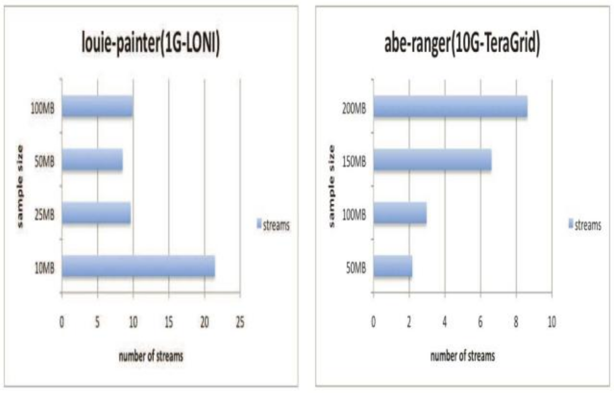

Fig 2. Average number of streams used per transfer.

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 1 and 10 Gbps 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.

6. CONCLUSION

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. 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. 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. REFERENCES

[1] Allcock .W, Bresnahan .J, Kettimuthu .R, Link .M, Dumitrescu .C, Raicu .I, and Foster .I, “The Globus Striped Gridftp Framework and Server,” Proc. 2005 ACM/IEEE Conf. Supercomputing (SC’05), p. 54, 2005.

[2] Altman .E, Barman .D, Tuffin .B, and Vojnovic .M, “Parallel TCP Sockets: Simple Model, Throughput and Validation,” Proc. IEEE INFOCOM ’06, pp. 1-12, Apr. 2006.

[3] Karrer .R.P, Park .J, and Kim .J, “TCP-ROME: Performance and Fairness in Parallel Downloads for Web and Real Time Multimedia treaming Applications,” technical report, Deutsche Telekom Labs,2006.

[4] Kosar .T and Livny .M, “Stork: Making Data Placement a First Class Citizen in the Grid,” Proc. IEEE Int’l Conf. Distributed Computing Systems (ICDCS ’04), pp. 342-349, 2004.

[5] Raicu .I, Foster .I , and Zhao .Y, “Many-Task Computing for Grids and Supercomputers,” Proc. IEEE Workshop Many-Task Computing on Grids and Supercomputers (MTAGS), 2008.

[6] Timothy G. Armstrong, ”Scheduling Many-Task Workloads on Supercomputers: Dealing with Trailing Tasks,” Zhao Zhang Department of Computer Science University of Chicago.

[7] Yildirim .E, Yin .D, and Kosar .T, “Prediction of Optimal Parallelism Level in Wide Area Data Transfers,” to be published in IEEE Trans. Parallel and Distributed Systems, 2010

.