Balancing TCP Buffer vs Parallel Streams in Application Level Throughput Optimization

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

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

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

ABSTRACT

The end-to-end performance of TCP over wide-area may be a major bottleneck for large-scale network-based applications. Two practical ways of increasing the TCP performance at the application layer is using multiple parallel streams and tuning the buffer size. Tuning the buffer size can lead to significant increase in the throughput of the application. However using multiple parallel streams generally gives better results than optimized buffer size with a single stream. Parallel streams tend to recover from failures quicker and are more likely to steal bandwidth from the other streams sharing the network. Moreover our experiments show that proper usage of tuned buffer size with parallel streams can even increase the throughput more than the cases where only tuned buffers and only parallel streams are used. In that sense, balancing a tuned buffer size and the number of parallel streams and defining the optimal values for those parameters are very important. In this paper, we analyze the results of different techniques to balance TCP buffer and parallel streams at the same time and present the initial steps to a balanced modeling of throughput based on these optimized parameters.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques, Modeling Techniques, Performance Attributes

General Terms

Design, Measurement, Performance

Keywords

Data transfer, File Transfer, GridFTP, Parallel TCP, Parallelism, Buffer size

DADC'09, June 9–10, 2009, Munich, Germany.

1. INTRODUCTION

Large-scale network-based applications spaning multiple sites over wide area networks heavily depend on the underlying data transfer protocol for their data handling, and their end-to-end performance may suffer significantly if the underlying protocol does not use the available bandwidth effectively. Most of the widely used transfer protocols are based on TCP, which may sacrifice performance for the sake of fairness. There has been considerable research on enhancing TCP as well as tuning its parameters for improved performance [6, 12, 5, 14, 16, 15, 13].

In the application layer, opening parallel streams and tuning the buffer size could improve the bottlenecks of TCP performance. Parallel streams achieve high throughput by mimicking the behavior of individual streams and get an unfair share of the available bandwidth [23, 16, 2, 8, 6, 12, 17]. On the other hand, using too many simultaneous connections reaches the network on a congestion point and after that threshold, the achievable throughput starts to drop down . Unfortunately it is difficult to predict the point of congestion and is variable over some parameters which are unique in both time and domain.

There are a few studies that try to find the optimal number of streams and they are mostly based on approximate theoretical models [7, 18, 1, 14]. They all have specific constraints and assumptions and can not predict the complex behavior of throughput in existence of congestion. 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 [7]. However, this model only works for uncongested networks. Thus, it cannot provide a feasable solution for congested networks. Another study [5] declares the same theory but develops a protocol which at the same time provides fairness. Dinda et al. [18] 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 [1], 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 [14] 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.

In [26], we developed a model that could predict the behavior of parallel stream throughput with as few as 3 data values of throughput of different parallelism levels. The model presented in that study could give very accurate results when the correct data points are selected. Hence we need an algorithm to be able to select the correct data points intelligently that at the same time will maximize the accuracy.

Another important parameter to be tuned for high throughput is the TCP buffer size. It is important in the sense that it affects the maximum number of bits that could be on the fly before an acknowledgement is received. Usually the buffer size is tuned as setting it to twice the value of $bandwidth \times delay$ product (BDP) [11]. However there are also different interpretations of *bandwidth* and *delay* concepts. Most of the current buffer size tuning work either require changes to the kernel stack [4], [22], [24], [25] or are based on estimations made on bandwidth and delay values in the application level [11], [21], [9], [19] .

The techniques that need modifications to the kernel [4], [22], [24], [25] is usually based on dynamic changes to the buffer size during the transfer based on the congestion window or flow control window parameters. The approach presented in [4] requires changes to the Kernel Stack. Based on the current congestion window, RTT and server Read Time, they calculate the following congestion window and set the buffer variables based on the current and next congestion window sizes. Another study [22] is also similar to the previous where the sender buffer size is adjusted based on the congestion window. Also the buffer memory is fairly shared among the connections. The receive buffer is sufficiently large so that it will not limit the throughput. Other two of the competitive techniques which are widely used are Dynamic Right-Sizing [25] and Linux 2.4 Auto-Tuning [24]. DRS is basicly a receiver -based approach where the receiver tries to estimate the $bandwidth \times delay$ product by using TCP packet header information and time stamps. Instead of using static flow control windows the advertised receive window is dynamically changes so that the sender is not limited by the flow control. On the other side Linux auto-tuning is a memory management technique in which it increases or decreases window size continuously based on the available memory and socket buffer space.

The techniques that are applied in the application level without changing the kernel and the protocol [11], [21], [9], [19] are usually static and once the buffer size is set it does not change throughout the transfer. In [9] the estimated throughput of a connection is calculated based on packet loss probability, RTT and Retransmission time out based on the model presented in [20]. Then by using the BDP product the buffer size is determined:

$$Buffersize = Estimated\ Throughput \times RTT \quad (1)$$

In [19], a series of ICMP messages are sent out and the packet length is divided by the time difference between two consecutive packets and by dividing them to the RTT, the available bandwidth of the bottleneck link is calculated. Then by multiplying it with the RTT the buffer size is determined. The study in [11] and [21] separates the buffer of a congested path from a non-congested path. The approach they propose finds the efficient buffer size for especially non-congested paths. In a non-congested path the optimal buffer size is found by considering the existing cross traffic. They propose an application layer tool called SOBAS, where in uncongested paths it limits the buffer size so it does not overflow the buffers of the bottleneck link and in congested paths it does not limit the transfer window so that the buffer size increases and becomes congestion limited. They send periodic UDP probing packets and calculate the throughput based on those probes. When the throughput becomes stable for some period of time they limit the receiver buffer according to that value.

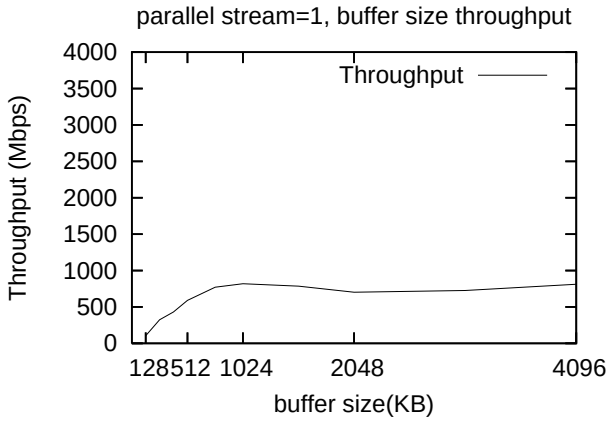

Figure 1: Optimal Buffer Selection

Although the window size parameter is properly tuned, it does not show a better performance than using parallel streams because parallel streams recover from packet loss more quickly rather than a buffer tuned single stream. There are a few studies that tries to derive a mathematical model to find the relationship between buffer size and number of parallel streams [10, 3]. The study in [10] models the throughput of parallel streams as multiple continuous time models of TCP congestion control mechanism. Another study represents the relationship between number of parallel streams, buffer size and round trip time by a single regression equation [3]. However this equation again considers a no-loss network. Mostly the models are derived for no-loss networks and they do not present an optimal value for both parameters.

In this study, we introduce models and algorithms that could calculate the throughput of parallel streams accurately and define and apply techniques that will balance the buffer size parameter with the optimal stream number to gain highest throughput. Our experiments show that a reasonable tuned buffer size parameter based on network samplings combined with our parallel stream optimization models could give higher throughput results than the only buffer size tuning or only parallel stream tuning techniques.

2. BUFFER SIZE OPTIMIZATION

Buffer size parameter affects maximum number of packets that will be on the fly before the sender will wait for an acknowledgement. If a network buffer is undersized then the network cannot be fully utilized. However, if it is oversized

there could also be degradation in the throughput because of packet losses hence window reductions. Usually it is tuned manually by the application users or in the kernel level of the operating system.

A common method to tune the buffer size is to set it to the twice the value of *bandwidth* × *delay* product (BDP). However this assumption of making the buffer size as large as the twice of the BDP only holds when there is no cross traffic along the path which is quite impossible. So it brings the question of whether to use the capacity of the network or the available bandwidth as well as minimum RTT or maximum RTT. Hence there is a quite variety in the understandings of the bandwidth and delay concepts. Below is a list of different meanings of BDP [11]:

- BDP1: $B = C \times RTT_{max}$
- BDP2: $B = C \times RTT_{min}$
- BDP3: $B = A \times RTT_{max}$
- BDP4: $B = A \times RTT_{min}$
- BDP5: $B = BTC \times RTT_{ave}$
- BDP6: $B = B_{\infty}$

In the above equations B represents the buffer size, C represents the capacity of the link, A represents the available bandwidth and RTT is the round trip time. BTC in BDP5 is the average throughput of a bulk congestion-limited transfer and calculated based on the congestion window size. Finally B_{∞} in BDP6 is a large value which is always greater that the congestion window so that the connection will always be congestion-limited.

Most of the tuning methods described in the literature do the tuning in the kernel level and approaches that use application level auto-tuning are so few [11], [21]. The methods that use one of the above equations usually rely on tools to take the measurement of available bandwidth and RTT and do not consider the effect of the cross traffic and congestion created by using large buffer sizes.

A practical way that will predict the buffer size hiding all the details of the cross traffic and congestion factors is to do samplings to measure the point in which the throughput stops increasing and becomes stable. In Figure 1, the buffer size parameter is increased exponentially and around 1MB,it becomes stable. So rather than using a BDP related equation, finding the point that throughput does not increase is a good choice because it will encapsulate all the related issues regarding cross traffic and congestion created.

3. PARALLEL STREAM OPTIMIZATION

In a previous study [26], we have presented mathematical models to predict the throughput of parallel streams. The model called Newton's Method model was able to predict the peek point of throughput and hence the optimal parallel stream number with only three data points known in the throughput curve. If the correct data points were used the accuracy of the model is very high. In this section we introduce another model called Full Second Order model which can increase the throughput accuracy more if the selection of the data points is appropriate. The model derivation is similar to the one presented in [26].

According to [7], the throughput of n streams is equal to :

$$Th_n <= n\frac{MSS}{RTT_n}\frac{c}{\sqrt{p_n}} \tag{2}$$

RTT represents *round trip time*, MSS represents the *maximum segment size*, p represents the *packet loss rate* , n is the number of streams and c is a constant.

We define the relation among RTT, n and p with a new variable p'_n. We assume that p'_n is related to a full second order polynomial and the following equations are derived to be used in this model.

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

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 from the predictions of network measurement tools 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. Based on equations 8,9 and 10, the values of a',b' and c' can be calculated easily.

$$a' = \frac{\frac{\frac{n_3^2}{Th_{n_3}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_3 - n_1} - \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1}}{n_3 - n_2} \tag{8}$$

$$b' = \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1} - (n_1 + n_2)a' \tag{9}$$

$$c' = \frac{n_1^2}{Th_{n_1}^2} - n_1^2 a' - n_1 b' \tag{10}$$

As we have mentioned before the selection of correct data points to do the prediction is very important since it could increase the accuracy of the model as well as prevent unlikely behaviors of the throughput prediction curve. We could see in Figure 2 the ineffectiveness of random data point selection strategy clearly. The graphics show the throughput of wide area data transfers for parallel streams and three prediction models are applied with randomly selected points. In the worst case scenario, the prediction curves do not reflect

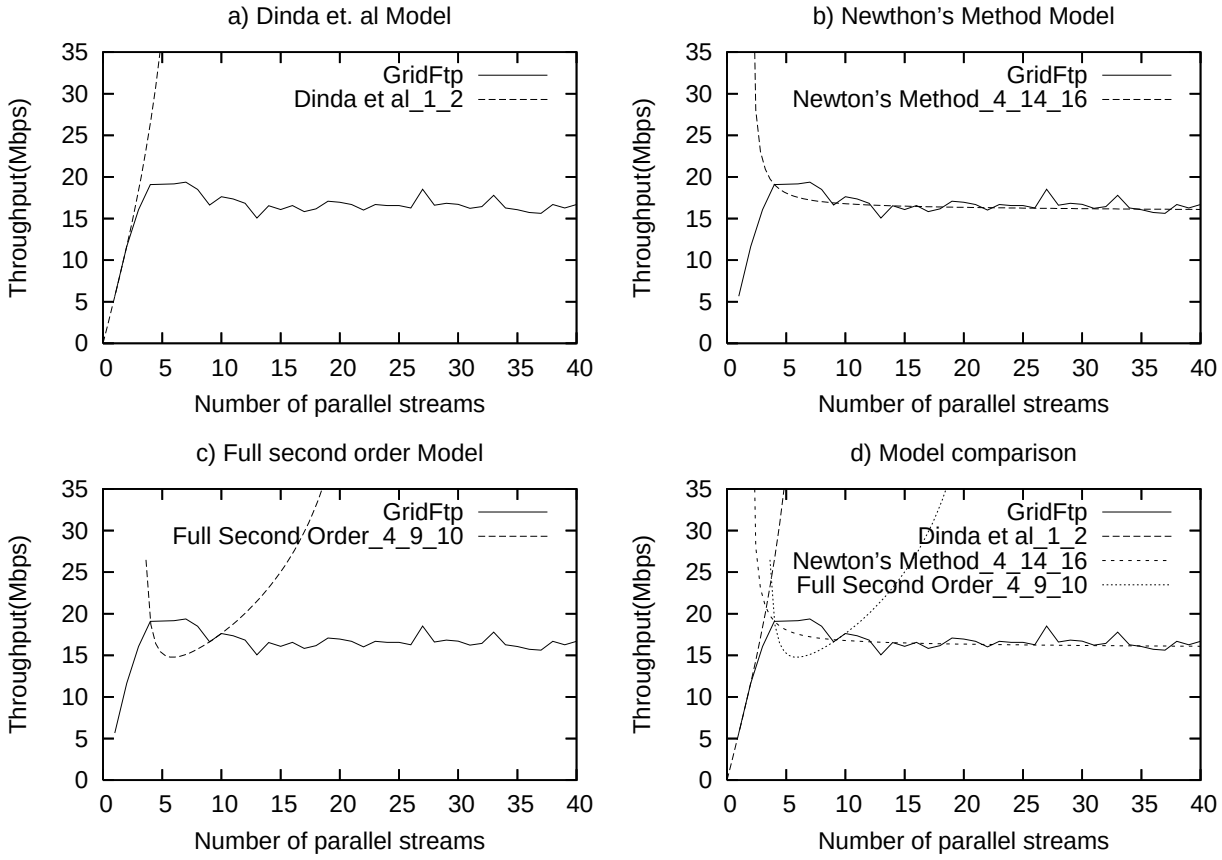

Figure 2: Ineffective combination of Models

the characteristics of the throughput curve at all. Therefore it is important that we should avoid a random selection strategy and make intelligent decisions.

3.1 Delimitation of Coefficients

To figure out whether a certain combination of data points is good or not before we try to use it to calculate the optimum parallel stream numbers, we examine the coefficients a',b' and c' derived from application of the model by using the certain combination of data points. If the coefficients meet some requirements for each model, then we can use this combination to calculate the optimum parallel stream number, otherwise we need to get another combination of data points. But we have to keep in mind that, we may still be far from the combination that will give the most accurate model.

In the following subsections we give out the requirements of the coefficients that should be met so that they can correctly reflect the relationship between the throughput and parallel stream number for the models. Since we have based our model improvements on Dinda et al model [18], we consider this model as a base while comparing our models. Also we present a short proof for the Full Second Order Model.

3.1.1 Statements of the coefficients requirements

i)For Dinda et al model:

- $a' > 0$
- $b' > 0$

ii)For Newton's Method Model:

- $a' > 0$
- $b' > 0$
- $c' \geq 2$
- $\left(\frac{2b'}{a'(c'-2)}\right)^{\frac{1}{c'}} > 1$

iii)For Full second order Model:

- $a' > 0$
- $b' < 0$
- $c' > 0$
- $2c' + b' > 0$

3.1.2 Proof of Full second order Model coefficients requirements

In order to get a curve which increases first and then decreases monotonically, we should guarantee that the throughput function is positive first and when it reaches the peak point, becomes zero, finally decreases into negative value.

$$Th'_{ful} = \begin{cases} \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} > 0 & n < optnum \\ \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} = 0 & n = optnum \\ \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} < 0 & n > optnum \end{cases}$$

$$\Rightarrow \begin{cases} \frac{b'}{2}n + c' > 0 & n < optnum \\ \frac{b'}{2}n + c' = 0 & n = optnum \\ \frac{b'}{2}n + c' < 0 & n > optnum \\ a'n^2 + b'n + c' > 0 & \forall n \in N^+ \end{cases}$$

$$\Rightarrow \begin{cases} a' > 0 \\ b' < 0 \\ c' > 0 \\ optnum = \frac{-2c'}{b'} > 1 \end{cases}$$

$$\Rightarrow \begin{cases} a' > 0 \\ b' < 0 \\ c' > 0 \\ 2c' + b' > 1 \end{cases}$$

Further more, since

$$\begin{aligned} \lim_{n\to\infty} \frac{n}{\sqrt{a'n^2+b'n+c'}} &= \lim_{n\to\infty} \frac{2\sqrt{a'n^2+b'n+c'}}{2a'n+b'} \\ &= \lim_{n\to\infty} \frac{\sqrt{a'+\frac{b'}{n}+\frac{c'}{n^2}}}{a'} \\ &= \frac{\sqrt{a'}}{a'} \end{aligned} \quad (11)$$

According to the above equalities and inequalities ,we conclude that *Full second order* Model increases first, then reaches a peak value, later decreases with a lower bound $\frac{\sqrt{a'}}{a'}$.

3.2 Exponential Increasing Steps Solution

In this section we present an exponential increasing algorithm that will intelligently select a sequence of stream numbers $1, 2, 2^2, 2^3, \cdots, 2^k$. Each time we double the number of streams until the throughput of the corresponding stream number begins to drop or increase very slightly compared with the previous one. After $k+1$ steps, we have $k+1$ data pairs that are selected intelligently.

The *ExpSelection* algorithm takes a set T of throughput values for different parallelism values. These values could be gathered from historical transfers or through instant samplings performed at the time of algorithm execution. First the algorithm sets the accuracy to a predefined small value[Line 2], and starts with a single stream and add the stream number and the corresponding throughput values to the output set [Lines 6-7]. The second parallelism level is calculated as twice the value of the previous parallelism value[Line 9]. The slop among the current and the previous throughput values are calculated[Line 11]. If the slop is less than the accuracy value, the loop stops and we gather our selected set of stream number and throughput pairs (O).

ExpSelection(T)
```
▷ Input: T
▷ Output: O[i][j]
1  Begin
2    accuracy ← α
3    i ← 1
4    streamno1 ← 1
5    throughput1 ← T_streamno1
6    O[i][1] ← streamno1
7    O[i][2] ← throughput1
8    do
9      streamno2 ← 2 * streamno1
10     throughput2 ← T_streamno2
11     slop ← (throughput2 − throughput1) / (streamno2 − streamno1)
12     i ← i + 1
13     O[i][1] ← streamno2
14     O[i][2] ← throughput2
15     streamno1 ← streamno2
16     throughput1 ← throughput2
17   while slop > accuracy
18 End
```

Now that we have our selected set O, we apply a comparison algorithm that will select the best combination based on both the coefficient values as well as the error rates of the selected combination. In *BestCmb* algorithm, we give the selected set O, number of items in the set (n) and the model to be applied as input. For every set of combination, the coefficients are calculated and if they are in the effective range that we presented in previous section, the total difference between the actual throughput and the predicted throughput is calculated[Lines 6-8]. The combination with the minimum err is selected and returned.

BestCmb($O, n, model$)
```
▷ Input: O, n
▷ Output: a, b, c, optnum
1  Begin
2    err_m ← init
3    for i ← 1 to (n − 2) do
4      for j ← (i + 1) to (n − 1) do
5        for k ← (j + 1) to n do
6          a', b', c' ← CalCoe(O, i, j, k, model)
7          if a', b', c' are effective then
8            err ← (1/n) Σ_{t=1}^{n} |O[t][2] − Th_pre(O[t][1])|
9            if err_m = init || err < err_m then
10             err_m ← err
11             a ← a'
12             b ← b'
13             c ← c'
14           end if
15         end if
16       end for
17     end for
18   end for
19   optnum ← CalOptStreamNo(a, b, c, model)
20   return optnum
21 End
```

In Figure 3, we give the experimental results of the application of our algorithm. Two different experimental environments are used. In Figure 3.a and 3.b, the experiments are conducted over the LONI network with 1Gbps NICs while

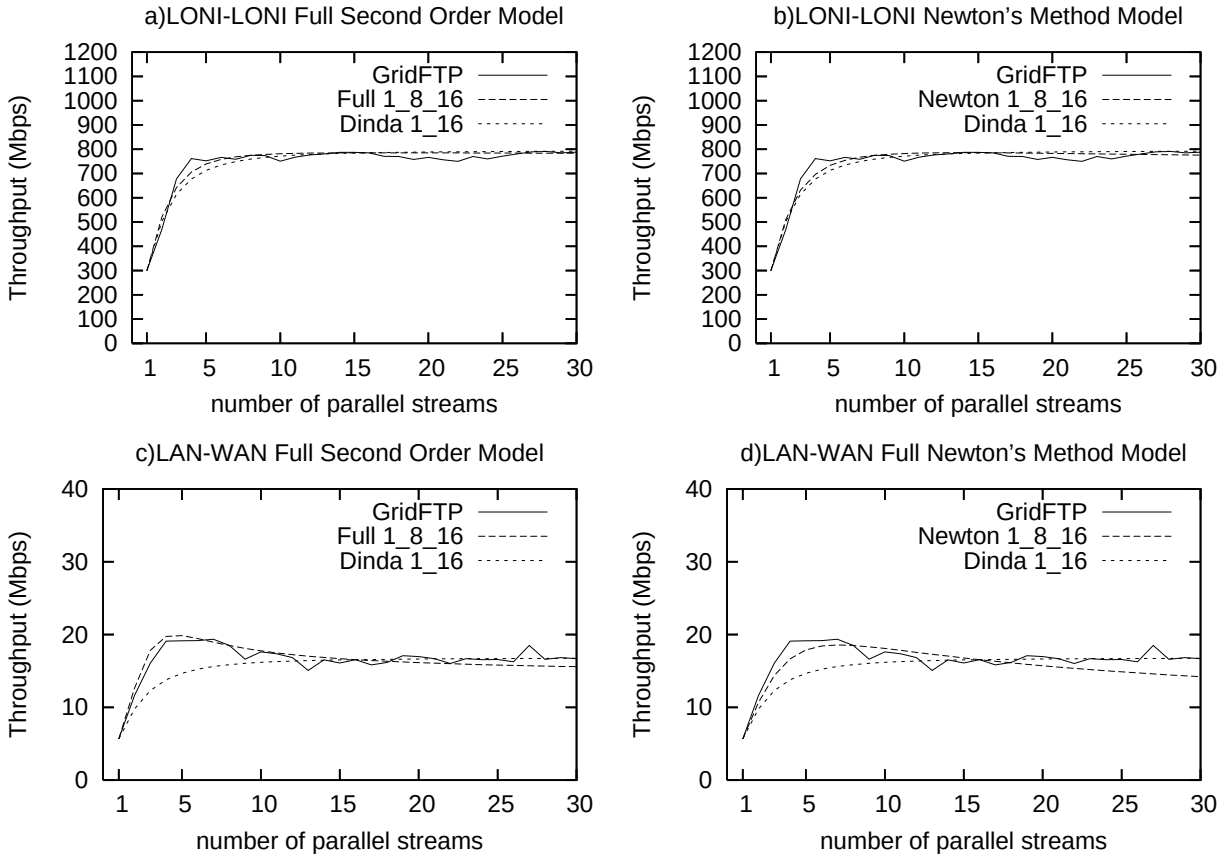

Figure 3: Application of prediction models over past data transfers

in Figure 3.c and 3.d, wide area transfers are conducted over 100Mbps NICs. Our models are compared with the Dinda et al Model [18]. We have seen that $ExpSelection$ algorithm combined with $BestCmp$ algorithm gives very accurate predictions. With these algorithms we minimize the amount of data we need to calculate an accurate prediction while maximizing the accuracy at the same time. The cost of the algorithm is even suitable for making instant samplings to get the relevant data points to do an accurate optimization rather than selecting them from past historical transfers.

4. BALANCING THE BUFFERSIZE AND PARALLEL STREAM NUMBER

There has been a large number of studies in the area of buffer optimization and our results in parallel stream optimization are very promising. However, a good combination of tuned buffer size and parallel streams could even give more effective results than the single applications of these two techniques. Unfortunately there are not any practical work to balance the buffer size and parallel stream number to achieve the optimal throughput. In this section, we present the results and discussion of simulations performed on NS-2 and a set of experiments in real network environment combined with the application of our models to describe the way a balance must be set. We conducted those experiments in LONI network with 10G NICs with a 6 ms delay. The network traffic of the LONI network varies a lot hence we could see the effect of the application of both techniques better.

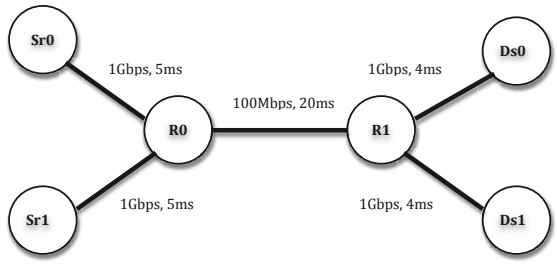

Figure 7: Topology

4.1 Simulation Results and Discussion

In our simulations we have tried different scenarios by changing the buffer size and parallel streams. The network topology we used is represented in Figure 7. The bottleneck link bandwidth is 100Mbps with a 20ms delay. The sources($Sr0$,$Sr1$) for our transfers and the cross traffic are connected to the bottleneck link router $R0$ with 1Gbps bandwidth and 5ms delay, while the destination nodes ($Ds0$,$Ds1$)

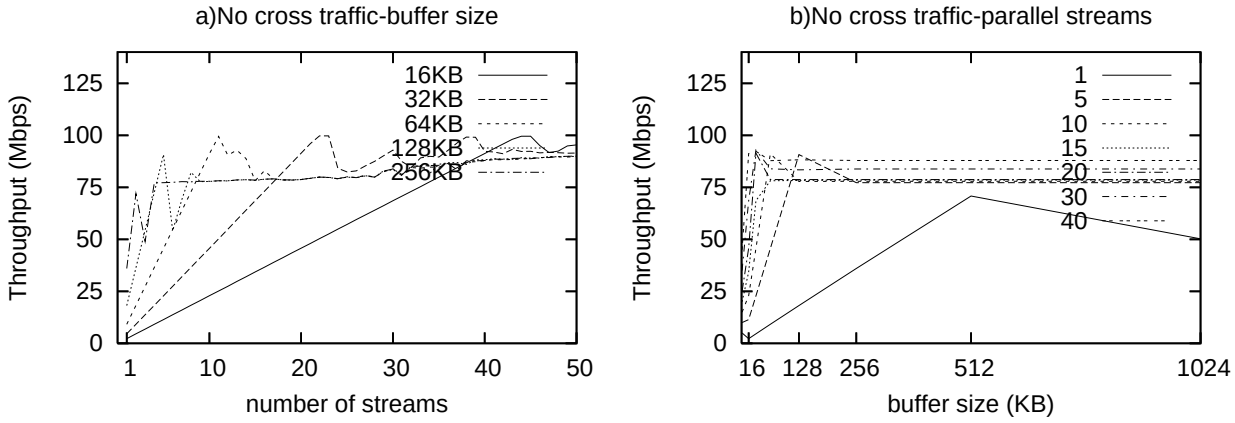

Figure 4: Effect of parallel streams and buffer size under no cross traffic

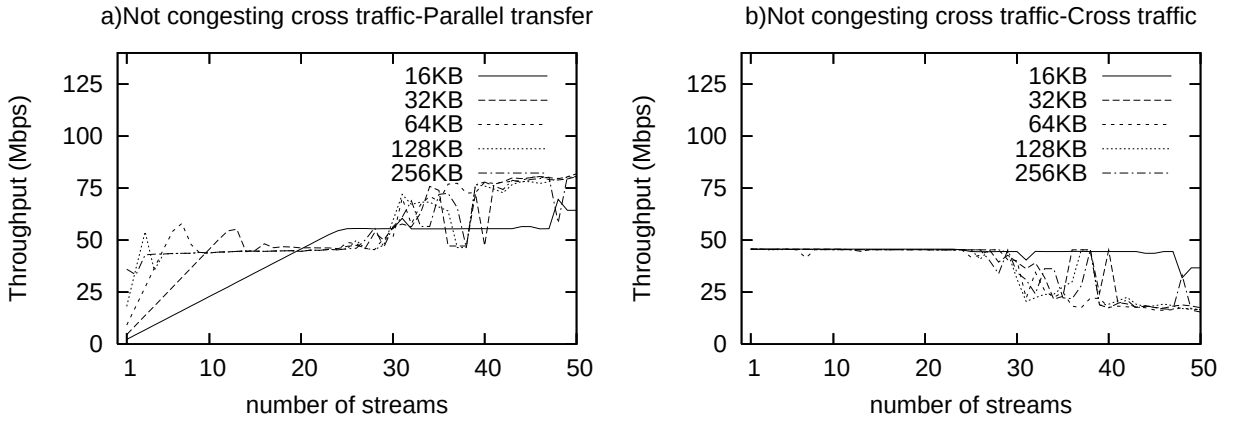

Figure 5: Effect of parallel streams and buffer size with a cross traffic of 5 streams with 64KB buffer size

are connected to $R1$ with 1Gbps bandwidth and 4ms delay. The cross traffic flows from $Sr0$ to $Ds0$ while the actual transfer occurs between $Sr1$ and $Ds1$.

In the first set of experiments we did not use any cross traffic and changed the buffer size with parallel streams. With a very small buffer size such as 16KB, the full utilization of the network can be achieved only with a very large number of streams (Figure 4.a). After reaching its peak point, the throughput starts to fall down around 45 streams. Increasing the buffer size into 32KB, peak throughput point is pulled back to 22 streams. Further increasing the buffer size to 64KB pulls the stream number further to 10 streams. However for larger buffer sizes than 64KB the throughput never can reach into its maximum point. The reasonable selection in this case is to use a buffer size around 16KB-64KB and parallel streams around 45-10 respectively to be able to get the highest throughput. In this case further increasing the buffer size does not help but causes a decrease in the throughput achieved. The maximum throughput values can be gained with a smaller buffer size than BDP and usage of parallel streams. The figure shows us a wave behavior of throughput, when we increase the buffer size and decrease parallel streams, it eventually decreases in its peak point. In Figure 4.b, we compare the parallel streams in vice versa. We could see that larger stream numbers could gain more throughput in smaller buffer sizes.

In the second set of experiments, we have done the simulations in the existence of a non-congesting cross traffic of 5 streams of 64KB buffer sizes. The results were quite interesting. With a very small buffer size of 16KB, the throughput increases linearly up to the congestion point with the cross traffic as we increase the parallel streams(Figure 5.a). The throughput is shared between the two traffics. Further increasing the buffer size will pull the peak throughput point into smaller stream numbers. This kind of behavior is similar to the previous case where there is no cross traffic except that the throughput is shared. In the mean time there is no effect on the cross traffic as the parallel stream number increases(Figure 5.b). However, further increase of parallel stream number in our traffic, results in the cross traffic to loose the fight and as the total throughput of our traffic starts to increase, the throughput of the cross traffic starts to decrease. The best results of throughput without affecting the cross traffic in this case is 32KB-64KB buffer size with a parallel stream range of 6-13 streams.

In the third case, there is congesting cross traffic of 12

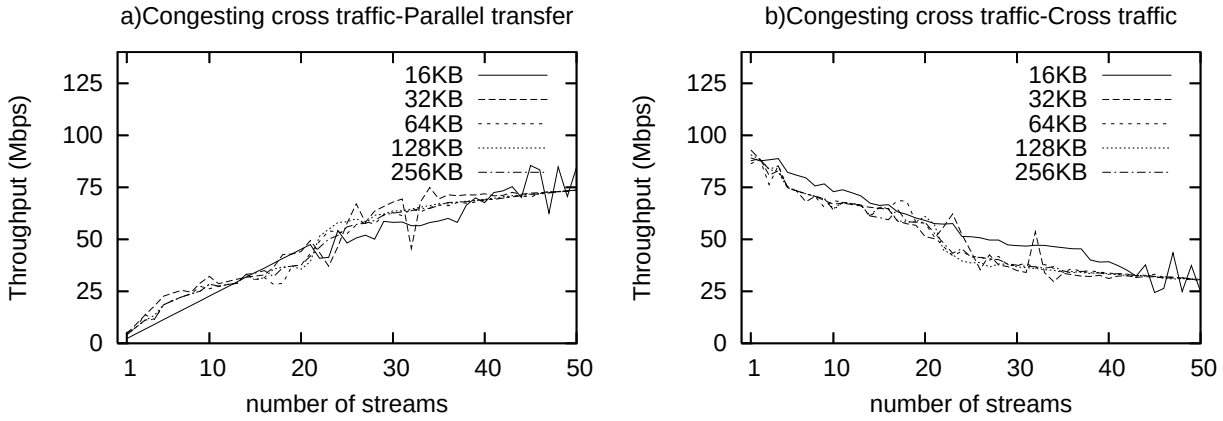

Figure 6: Effect of parallel streams and buffer size with a cross traffic of 12 streams with 64KB buffer size

streams with 64KB buffer sizes. Interestingly tuning the buffer size has no effect on the cross traffic since all of curves show the similar characteristics . As the number of streams increases, the throughput increases as well(Figure 6.a). In the mean time it starts to steal from the throughput of cross traffic as it gradually decreases(Figure 6.b). The only way to gain more throughput in this case is to open parallel streams.

4.2 LONI Experiments

Although simulation results gave us a good idea about the behavior of tuning buffer size and parallel streams, we would like to see the effects of both in real network environment settings. We applied two different techniques to decide which one is more effective in gaining maximum throughput.

4.2.1 1st Settings

In the first technique, we would like to measure the effect of a buffer size tuning over a transfer that is already tuned with optimal parallel stream number with our model. In Figure 8.a, we conducted transfers with an untuned buffer size of 256K and applied Newton's Method Model. According to the model the optimal stream number that gives the peak point is 14 and around 1.7 Gbps throughput value in average is gained. By using this optimal stream number, we ranged the buffer size from 128K to 4M(Figure 8.b). We have seen that the highest throughput results were gained again with default buffer size of 256K that we used to optimize the stream number. There was not an improvement in average throughput values.

4.2.2 2nd Settings

In the second technique, we decided to apply the procedure in the reverse order. We conducted some sampling transfers by increasing the buffer size exponentially using only single stream. In Figure 9.a, the throughput curve started to become stable around 1M buffer size. So we picked up this value as the optimal buffer size value and conducted parallel transfers with this buffer size value. The results could be seen in Figure 9.b. We applied the Full Second Order model this time. The optimal stream number with the Full Second Order was 4 and we were able to get a throughput value of more than 2 Gbps. With this technique we were able to get a higher throughput value and opened less number of streams for optimization and hence placed less burden on the end system.

To better compare the techniques we present, we prepared a comparison graphic that presents both the average and maximum throughput results of the techniques applied. In Figure 10, the first column represents the settings where only single stream was used with an untuned buffer size. In the second column, the result of an optimized throughput of parallel streams with our models and an untuned buffer size is presented. In the last column, the result of the second technique is presented. Clearly the last technique was able to get more throughput for both average and maximum values.

5. CONCLUSION

Tuning buffer size and parallel stream number are two ways of optimizing the application level throughput. When the correct balance is found, these techniques applied together could improve the end-to-end throughput comparing to the single application of each technique. In this study, we presented a mathematical model and an algorithm that could predict the throughput of parallel streams very accurately. Also we presented the steps for constructing a balance among tuned buffer sizes and parallel streams. Our experiments showed that the usage of parallel streams on fairly tuned buffers could give higher throughput values and less number of streams are needed for optimization.

Acknowledgment

This project is in part sponsored by the National Science Foundation under award numbers CNS-0846052 (CAREER), CNS-0619843 (PetaShare) and EPS-0701491 (CyberTools), 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.

6. REFERENCES

[1] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel tcp sockets: Simple model, throughput and validation. In *Proc. IEEE Conference on Computer*

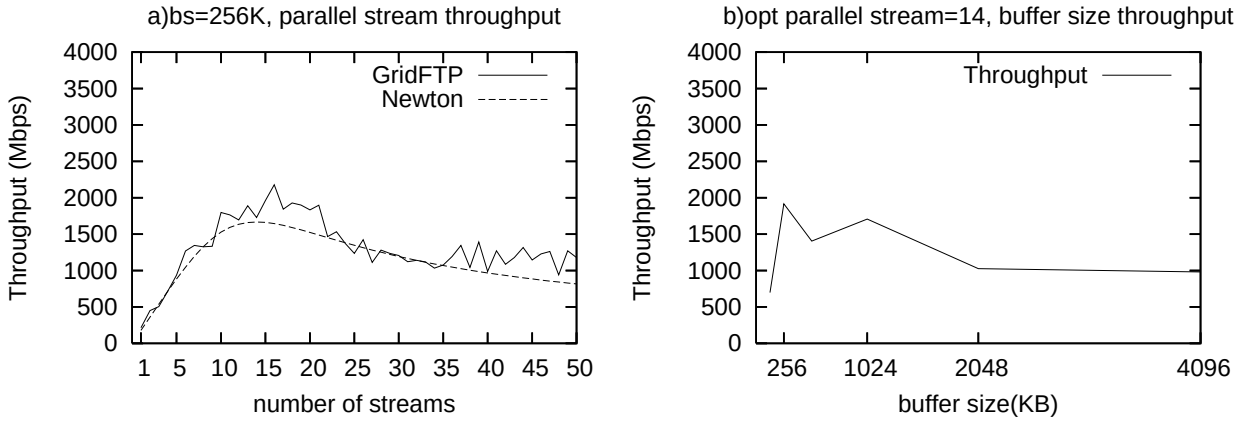

Figure 8: First Technique

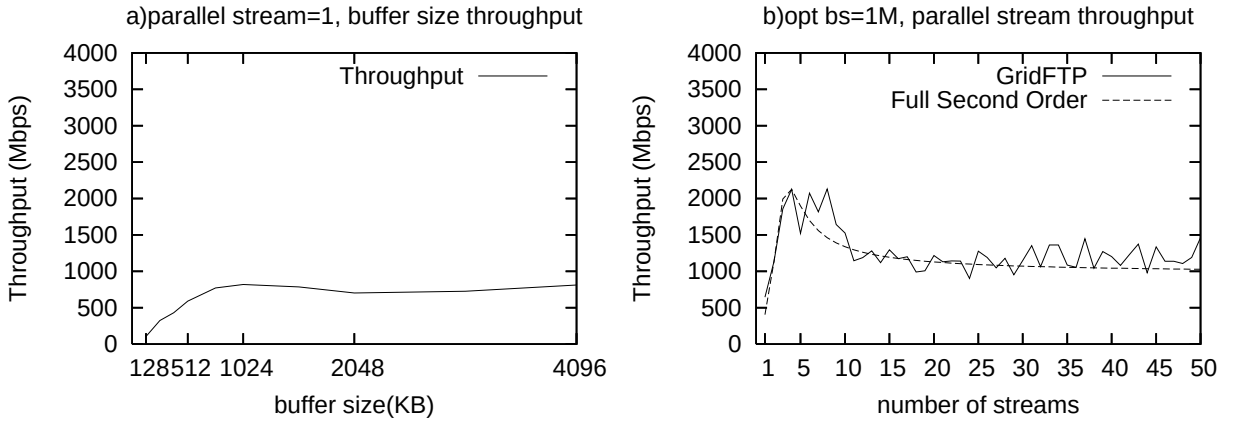

Figure 9: Second Technique

Communications (INFOCOM'06), pages 1–12, Apr. 2006.

[2] 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 (INFOCOM'98)*, pages 252–262, California, USA, Mar. 1998.

[3] K. M. Choi, E. Huh, and H. Choo. Efficient resource management scheme of tcp buffer tuned parallel stream to optimize system performance. In *Proc. Embedded and ubiquitous computing*, Nagasaki, Japan, Dec. 2005.

[4] A. Cohen and R. Cohen. A dynamic approach for efficient tcp buffer allocation. *IEEE Transactions on Computers*, 51(3):303–312, Mar. 2002.

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

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

[7] 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 (IPDPS'02)*, pages 434–443, 2002.

[8] 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 (HPDC'05)*, pages 265–275, July 2005.

[9] G. Hasegawa, T. Terai, T. Okamoto, and M. M. Scalable socket buffer tuning for high-performance web servers. In *International Conference on Network Protocols(ICNP01)*, page 281, 2001.

[10] T. Ito, H. Ohsaki, and M. Imase. On parameter tuning of data transfer protocol gridftp for wide-area networks. *International Journal of Computer Science and Engineering*, 2(4):177–183, Sept. 2008.

[11] M. Jain, R. S. Prasad, and C. Davrolis. The tcp bandwidth-delay product revisited: network buffering,

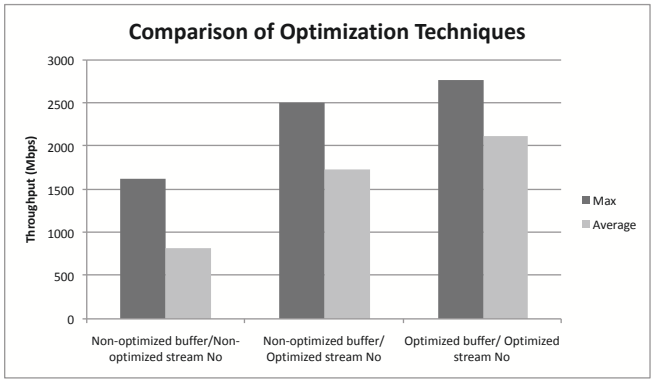

Figure 10: Balanced Optimization

cross traffic, and socket buffer auto-sizing. Technical report, Georgia Institute of Technology, 2003.

[12] R. P. Karrer, J. Park, and J. Kim. Adaptive data block scheduling for parallel streams. Technical report, Deutsche Telekom Laboratories, 2006.

[13] G. Kola, T. Kosar, and M. Livny. Run-time adaptation of grid data-placement jobs. *Scalable Computing: Practice and Experience*, 6(3):33–43, 2005.

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

[15] 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 (ICDCS'04)*, pages 342–349, 2004.

[16] 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 (CHEP'01)*, Beijing, China, Sept. 2001.

[17] 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 (ICDCS'05)*, pages 414–424, June 2005.

[18] 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 (IPDPS'05)*, page 68b, Apr. 2005.

[19] A. Morajko. *Dynamic Tuning of Parallel/Distributed Applications*. PhD thesis, Universitat Autonoma de Barcelona, 2004.

[20] J. Padhye, V. Firoiu, T. Towsley, and J. Kurose. Modeling tcp throughput: a simple model and its empirical validation. *ACM SIGCOMM'98*, 28(4):303–314, Oct. 1998.

[21] R. S. Prasad, M. Jain, and C. Davrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of Grid Computing*, 1(4):361–376, Aug. 2004.

[22] J. Semke, J. Madhavi, and M. Mathis. Automatic tcp buffer tuning. *ACM SIGCOMM'98*, 28(4):315–323, Oct. 1998.

[23] 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 (SCï£¡00)*, pages 63–63, Texas, USA, Nov. 2000.

[24] L. Torvalds and T. F. S. Company. The linux kernel. http://www.kernel.org.

[25] E. Weigle and W. Feng. Dynamic right-sizing: A simulation study. In *Proc. IEEE International Conference on Computer Communications and Networks (ICCCN'01)*, 2001.

[26] E. Yildirim, M. Balman, and T. Kosar. Dynamically tuning level of paralllelism in wide area data transfers. In *Proc. IEEE Data-aware Distributed Computing Workshop (DADC'08)*, June 2008.