Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 5, pp. 910-918

A New Model for Cluster Communications Optimization

A. Rusan, C.-M. Amarandei

Andrei Rusan, Cristian-Mihai Amarandei
"Gheorghe Asachi" Technical University of Iaşi
Department of Computer Science and Engineering,
Adress: Bd. Dimitrie Mangeron, Nr. 53A, 700050, Iaşi, Romania
E-mail: andrei.rusan@tuiasi.ro, camarand@cs.tuiasi.ro

Abstract: Performance losses of cluster applications can arise from various sources in the communications network of computer clusters. Typically, CPU intensive applications generate a small amount of network traffic the overall influence of the network subsystem is minimal. On the other hand, a data-intensive and network aware application generates a large amount of network traffic and the influence of the network subsystem is significantly greater. This paper presents a model that aims to improve the cluster's network performance by reducing the data transfer time, this solution having the advantage that doesn't imply modifications of the original applications or of the kernel.
Keywords: cluster communications optimization, network performance

1 Introduction

The computing performance of a cluster is dependent on the performance of cluster components: computing nodes and communication infrastructure. The cluster communication infrastructure was built using network devices whose performance can be modified only by hardware changes, i.e. replacing 100Mbps Ethernet switches with gigabit switches; therefore this component of the cluster will be neglected. The performance of a computing node is defined by hardware and software performance, where the hardware performance can be considered a constant value and it can be influenced by the hardware changes only. The software can be separated in components: application and operating system. The computing performance of the cluster can be influenced by each of these components.

Cluster performance increasing by performing modifications at the applications level is a goal very hard to achieve, some applications requiring a complete rewrite for this task. Of course, there are exceptions too, like network-aware applications, which are built exactly with this goal in mind and they do not require any optimizations.

The last component that can influence the performance is the operating system through the kernel configuration and at the network layer optimizations.

Taking into account that clusters typical contains a large number of computing nodes, the solution for cluster performance improvements must be done with minimal modifications in the systems. These requirements are necessary to keep the administrative tasks at a decent level.

There is a research work involving the network tuning mechanisms or network-aware applications trying to solve these issues. The projects developed so far, like WAD (Work Around Daemon) [1] or ENABLE [2] don't meet the imposed requirements, as for WAD a modified kernel must be used, and for ENABLE the applications must be rewritten.

The WAD project provides a transparent mechanism to work around a variety of network issues, including TCP buffer size, MTU size, packet reordering, and leaky network loss [1]. The WAD goal is to eliminate the "wizard gap" representing the difference between the network performances achievable by manually handcrafting of the optimal tuning parameters, compared to an untuned application [1]. To attain this goal, WAD requires a modified kernel from the Web100 project. This solution could not be applied in our case, because of the different Linux kernel versions the Web100 project provides a kernel patch starting from the 2.6.12. Through the use of a different kernel version that the provided one by the Linux distribution, CentOS 4.5 in our case, problems in maintaining the operating system across clusters can occur. Therefore our solution works fine no mater the kernel version used.

The ENABLE project, includes monitoring tools, visualization tools, archival tools, problem detection tools, and monitoring data summary and retrieval tools. ENABLE provides an API that makes it very easy for application or middleware developers to determine the optimal network parameters [2]. However, the solution provided by this project was not applicable in our case, because applications could not be rewritten.

This paper presents a model for self optimization of the network communications in order to improve cluster performance by shortening the data transfer time. The model implementation does not require applications and kernel structure modifications or adding new modules to the existing ones. Also, the implementation uses only the tools provided by the operating system for runtime configuration and therefore the automatic operating system and kernel updates can be applied immediately.

In the next section a brief review of the TCP transport protocol issues, Linux kernel network subsystem and the NetPIPE network performance measurement tool are presented. Section 3 describes the network self optimization model and the proposed algorithm. Section 4 presents the test environment and the experimental results. The final section summarizes author's efforts on tuning the cluster communications network and considers future extensions of the work.

2 Background

TCP protocol transmits new data into the network when old data has been received as indicated by acknowledgments from the receiver to the sender. The data rate is determined by the window size and is limited by the application, the buffer space at the sender or the receiver and by the congestion window. TCP adjust the congestion window to find an appropriate share of the network capacity of the path between source and destination. Missing or corrupted data segments are repaired by TCP by retransmitting the data from the sender's buffer. This process requires an entire window of data to fit into both sender and receiver buffers [1].

The largest TCP window can be 216=65KB because the TCP header uses 16 bits to report the receive window size to the sender. The window scale option was introduced defining an implicit scale factor used to multiply the windows size value from TCP header in order to obtain the real TCP window size, as described in [3]. These buffers have default values that may either be changed by the applications using system calls or by using tools provided by the operating system, i.e. `sysctl` tool from Linux/Unix.

The second part of this section is focused on the network subsystem of the Linux kernel. Starting from the version 2.4 of Linux kernel, an auto tuning technique is used to perform memory management. This technique simply increases and decreases buffer sizes depending on available system memory and available socket buffer space. By increasing buffer sizes when they are full of data, TCP connections can increase their window size performance improvements are an intentional side-effect [4]. On the other hand, this is done within the limitation of the available system memory and socket buffer space, and on the busy cluster that are valuable resource.

The network subsystem of the Linux operating system should be tuned in order to obtain an optimal performance of a computing system. In order to do that, changes can be operated at the following levels: network interface and kernel parameters. The kernel parameters allowing to change the network behavior can be tuned by modifying the following files located in

`/proc/sys/net`:

```
/proc/sys/net/core/rmem_max
/proc/sys/net/core/rmem_default
/proc/sys/net/core/wmem_max
/proc/sys/net/core/wmem_default
/proc/sys/net/ipv4/tcp_stack
/proc/sys/net/ipv4/tcp_timestamps
/proc/sys/net/ipv4/tcp_keepalive_time
/proc/sys/net/ipv4/tcp_mem
/proc/sys/net/ipv4/tcp_rmem
/proc/sys/net/ipv4/tcp_wmem
/proc/sys/net/ipv4/tcp_window_scaling
```

The network interface can also be tuned by modifying the speed and duplex settings and the MTU size. Two problems have to be addressed while setting up the cluster:

- the default kernel values don't provide the best performance for the custom environment, and
- the number of communication devices needed to be set.

Since solutions to solve these two problems are missing an optimal value for each system in cluster to get the best possible performance is proposed. By using the right tools, the network settings related changes are available immediately, the optimization algorithm presented in this paper being based on these features. The values of the send/receive buffers (tcp_wmem and tcp_rmem) can be changed by specifying minimum size, initial size, and maximum size as follows:

```
sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"
```

The third value must be the same as or less than the wmem_max and rmem_max values. The first value can be increased on high-speed, high-quality networks so that the TCP window starts out at a sufficiently high value [5]. Also, the TCP window scaling is an option to enlarge the transfer window.

Performance measurements of the cluster network can be done using a wide area of tools like Iperf [6], Netperf [7] or NetPIPE (Network Protocol Independent Performance Evaluator). Because the performance measurements must be done for both TCP and MPI layer and the NetPIPE provides a complete measurement of the communication performance on both of them, the tests were performed using this tool. The NetPIPE utility performs simple ping pong tests, bouncing messages of increasing size between two computers. To provide a complete test, NetPIPE modifies the message size, with a slight perturbation, at regular intervals and measures the point-to-point communications performance between nodes [8]. Because we want to determine of the maximum bandwidth available for different use cases, usage of different message size is a must. From all the performance measurements tools available, only NetPIPE had this feature by default, which defines it as a right tool for this kind of tests. An in depth description of the NetPIPE utility can be found in [8]- [10]. Linux kernel network subsystem information gathered by running NetPIPE on the cluster were used in order to improve the throughput.

3 Proposed model

The proposed model implies the computation of a best possible set of values for a given set of parameters. Figure 1 presents the model schematics composed from three parts: "Control logic",

”Parameter computation” and ”Network test tool”. The first component is responsible for sending starting set of values to ”Parameter computation” and keeps the running flux under control. The second component computes the sets of values based on previously known sets of data and on the results of the current run received from the ”Network test tool” and send it to the kernel in order to set the network subsystem. The ”Network test tool” is responsible to run set of tests and provide the results to ”Parameter computation” component. The optimization process is started by ”Control logic”, which sends starting values to ”Parameter computation” that are set in the kernel network subsystem and starts the first set of tests through the ”Network test tool”. After the tests are finished, the results are sent to "Parameter computation" which computes a new set of values and the process continues until meeting the end condition.

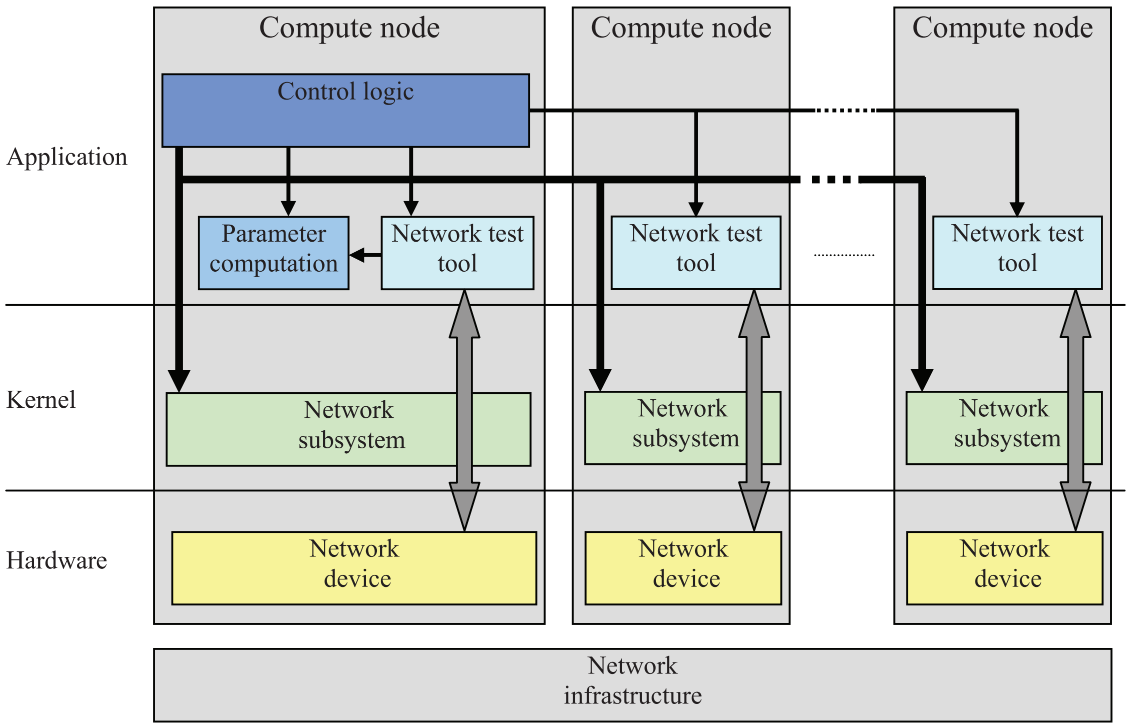


Figure 1: The cluster communication optimization model

The process provides self optimization of the kernel network subsystem, the only necessary interaction with the application being the configuration file that contains necessary values for the application startup and for the components behavior. These values can be set by administrators to meet the specific needs.

An algorithm implementing model functionality is proposed. This algorithm performs the bandwidth measurements and adjusts the sets of parameters to obtain the highest bandwidth usage for each case and running tests.

Given l, the number of tests to be performed, let it be $N = \{n_1, n_2, \ldots, n_k\}$ the set of nodes in cluster, $T = \{t_1, t_1, \ldots, t_l\}$ the set of test variables (i.e. `tcp_rmem`, `tcp_wmem`, `tcp_window_scalling`), $I = \{i_1, i_2, \ldots, i_l\}$ the kernel network subsystem parameters start values and $E = \{e_1, e_1, \ldots, e_l\}$ be the set of computed values for each test t_i. Also, given m as the number of messages, $MS = \{ms_1, ms_2, \ldots, ms_m\}$ is defined as the set of message sizes used by the testing tool, $X = \{x_1, x_2, \ldots, x_m\}$ as the best set of result value for each test t_t, the results set $R_i = \{r_1, r_2, \ldots, r_k\}_i$ one for each cluster node, $S = \{s_1, s_2, \ldots, s_m\}$, where $s_i = \sum_{j=i}^{k} r_{ij}$, and $B = \{b_1, b_2, \ldots, b_m\}$ as the set of best values for each test t_i. The algorithm computes the values for the test variables as follows:

```
1  foreach $t_i$ in $T$ do
2      $iter = i_i$;
3      while $iter < e_i$ do
4          $param = generate\_set(t_i, iter)$;
5          $set\_kernel\_parameters(param)$;
6          foreach $n_i$ in $N$ do
7              $start\_remote\_testing\_program(n_i)$;
8              $prepare\_local\_testing\_program(n_i)$;
9          end
10         parallel do
11             $R_i = execution\_of\_local\_testing\_programs$;
12         end
13         foreach $ms_i$ in $MS$ do
14             foreach $n_j$ in $N$ do
15                 $s_i[iter] \mathrel{+}= r_{ij}$;
16             end
17             $iter = get\_next\_iter(t_i, i_i, e_i, iter)$;
18         end
19     end
20     foreach $ms_i$ in $MS$ do
21         $iter = i_i$;
22         while $iter < e_i$ do
23             if $x_i < s_i[iter]$ then
24                 $x_i = s_i[iter]$;
25                 $b_i = iter$;
26             end
27             $iter = get\_next\_iter(t_i, i_i, e_i, iter)$;
28         end
29     end
30     $best = get\_max\_count(B)$;
31     $set\_kernel\_parameters(t_i, best)$;
32 end
```

The algorithm has two main components: the network test component corresponding to ”Network test tool” in Figure 1 and implemented by lines 3-19 from the algorithm, and the computational component corresponding to ”Parameter computation” in Figure 1 and implemented by lines 20-30. The methods used in the algorithm implements the following actions:

- generate_set: produce a new set of parameters used for network testing;
- start_remote_testing_program: launches the remote component of the testing application (NetPIPE in our case);
- prepare_local_testing_program: prepare the local component of the testing application, necessary to maximize the accuracy of the measured values;
- execution_of_local_testing_programs: runs the testing application;
- get_max_count: extract the parameter value corresponding to the maximal throughput.

Finally, the line 31 sequentially sets the kernel parameters to the best computed values on the entire cluster.

4 Implementation and experimental results

To improve the cluster communications, dynamic tests and adjustments for the following Linux kernel network parameters are performed: tcp_window_scalling, tcp_rmem and tcp_wmem. Bandwidth measurements and TCP parameters adjustments were carried out to obtain the highest bandwidth usage for each case and to determine the maximum bandwidth available for different use cases.

The environment used to test the proposed model consists in a grid cluster with the following configuration: one front-end computer with 4 x 3.66 GHz Intel Xeon processors, 4 x 146 GB hard drive and 8 GB of RAM and 12 computing nodes with 1 x 2.33GHz Intel Core2 Duo CPU, 1 x 160GB hard drive and 2GB of RAM with Gigabit Ethernet card connected with CAT6 cables via a Gigabit switch.

First implementation of the algorithm was made in Bash, but due the dificulties in working with data structures was switched to Perl. Also, to preserve measurement accuracy and performance the tests results were saved to files for further usage, like graphical presentation.

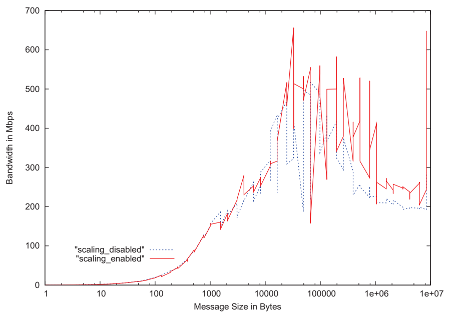
The performance results were obtained based on a test array with three elements: one for

TCP windows scaling, a second one for TCP read buffer and the third one for the TCP write buffer kernel parameters. For each of this there is a graphical presentation, where on the X and Y-axis the message size used during tests and the resulted bandwidth values are, respectively, presented.

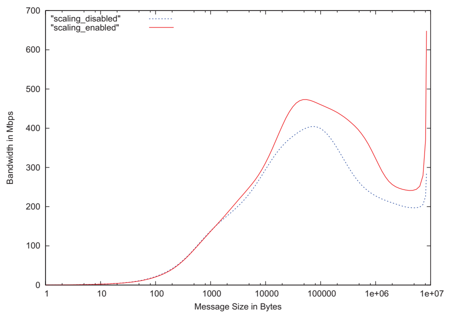
The results for the TCP windows scaling parameter are showed in Figure 2(a), where one line is for net.ipv4.tcp_window_scaling=0, and the other one is for net.ipv4.tcp_window_scaling=1. For an easier reading of the results graph, we apply a Bezier function in order to obtain the presentation from Figure 2(b).

The influence of TCP read/write buffer size over the available bandwidth are presented in Figure 3(a)/Figure4(a). In this case, there are a large number of graphic representations and the observation is very difficult, so a Bezier function was applied on the results values to make the graphical presentation more readable, as shown in Figure3(b)/Figure4(b). The red line in the graphical representations corresponds to the default value for tcp_rmem (4KB) and the blue dotted line is the best value resulted using this model, with 100Mbps more than the default value.

By applying different values for TCP buffers and running tests bandwidth variation for each of them is presented (Figure 5). TCP buffers size starts from 4KB and each algorithm step doubles the previous value.

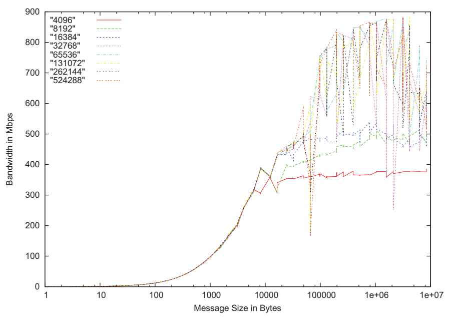


(a)

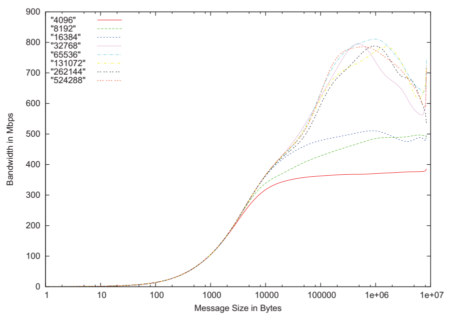


(b)

Figure 2: TCP window scaling influence over bandwidth

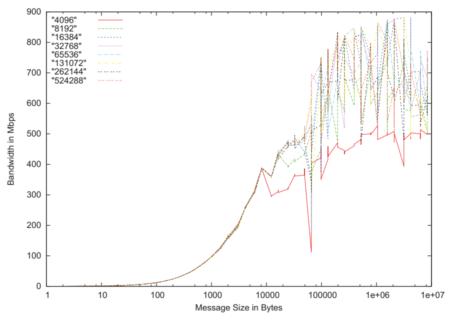


(a)

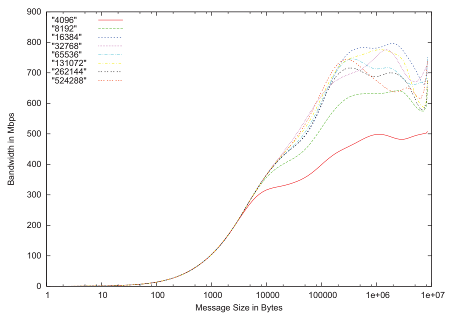


(b)

Figure 3: TCP read buffer influence over bandwidth



(a)



(b)

Figure 4: TCP write buffer influence over bandwidth

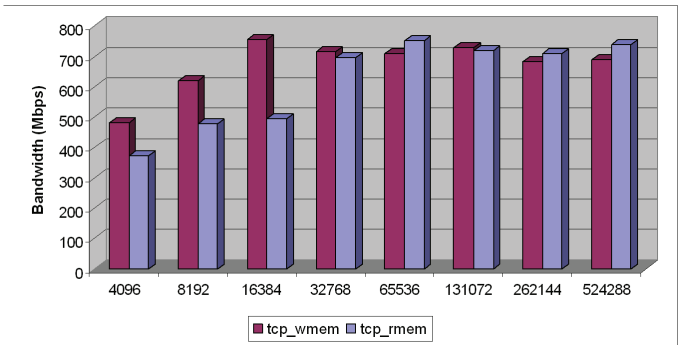
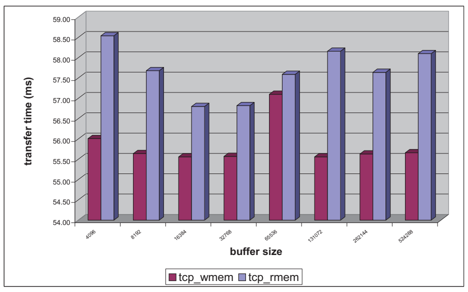
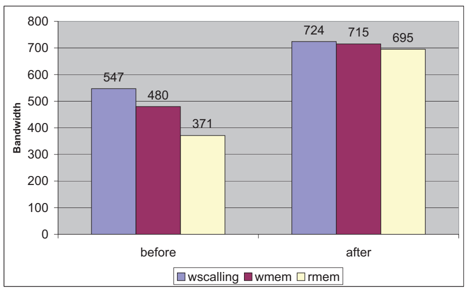


Figure 5: Bandwidth achieved for different TCP buffer size

A ram drive on all computers was built in order to test the results without any delays introduced by the hard disk drive. Because of the 2GB memory limits on the computing nodes, the file transferred and the ram drive size was 512 MB. In the Figure 6(a) transfer time for the file is shown. In this test the TCP buffers size was changed from 4KB to 512 KB and data transfer starts in both directions for each value, from the frontend to cluster nodes and back.



(a)



(b)

Figure 6: (a)Transfer time for a 512MB file between cluster; (b)The benefits of parameters adjustments

The best transfer time was obtained when both tcp_rmem and tcp_wmem values were 16KB or 32KB. During the tests, this solution provides all values for the considered TCP parameters, which can be useful for other scenarios. In Figure 6(b), the bandwidth improvement is presented.

Using only the default values, the cluster internal network available bandwidth is not optimally used with a strong impact on the overall computing performance. Using this optimization model, the bandwidth available in the cluster is efficiently used.

5 Conclusions and future work

Using the proposed model, the communication between cluster nodes has been improved. All results are considered for the specific needs of mentioned cluster, where a significant amount of data needs to be transferred between cluster nodes. For other applications, like a web server farm, the final results may be slightly different, but can be optimized by adjusting the test tool for those specific needs. The network kernel parameters computed can be used later if the use case is changed, i.e. a web server farm. The algorithm can be used for IPv6 too, however the authors doesn't implemented nor tested.

The further development of presented application will follow two directions: one is to extend its capabilities to support UDP traffic performance adjustment; and the second one is to support tuning parameters other than the ones related to the Linux kernel.

Bibliography

[1] T. Dunigan, M. Mathis, B. Tierney , A TCP tuning daemon, *Conference on High Performance Networking and Computing, Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, 2002

[2] B.L. Tierney, TCP tuning guide for distributed application on wide area networks, *Usenix*;login. `http://www-didc.lbl.gov/tcp-wan-perf.pdf.`, 2001

[3] V. Jacobson, R. Braden, D. Borman, RFC1323 TCP Extensions for High Performance, May 1992

[4] B.L Tierney, D. Gunter, J. Lee, M. Stoufer, J.B. Evans, Enabling network-aware applications, Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, Page: 281-288, 2001, ISBN: 0-7695-1296-8

[5] NetPIPE, webpage: `http://www.scl.ameslab.gov/netpipe/`

[6] A. Tirumala, L. Cottrell, "Iperf Quick Mode", `http://www-iepm.slac.stanford.edu/bw/iperfres.html`

[7] Netperf homepage, `http://www.netperf.org/netperf/NetperfPage.html`

[8] D. Turner, A. Oline, X. Chen, and T. Benjegerdes,Integrating New Capabilities into NetPIPE, *Lecture Notes in Computer Science*, Springer-Verlag, September 2003, pp. 37-44.

[9] D. Turner, X. Chen, Protocol-Dependent Message-Passing Performance on Linux Clusters, *Proceedings of the IEEE International Conference on Cluster Computing*, September 2002, pp. 187-194.

[10] Q.O. Snell, A. Mikler, J.L. Gustafson, NetPIPE: A Network Protocol Independent Performance Evaluator, *ASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[11] H. Sivakumar, S. Bailey, R. L. Grossman,PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks, *Proceedings of IEEE Supercomputing 2000*, Nov., 2000, `http://www.ncdm.uic.edu/html/psockets.html`

[12] J. Postel, *RFC793 Transmission Control Protocol*, September 1981

[13] R. Braden , *RFC1122 Requirements for Internet Hosts – Communication Layers*, October 1989

[14] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, *RFC 2330 Framework for IP Performance Metrics*, May 1998

[15] E. Ciliendo, T. Kunimasa, B. Braswell, *Linux Performance and Tuning Guidelines*, IBM, July 2007.