Towards Performance and Scalability Analysis of Distributed Memory Programs on Large-Scale Clusters

Sourav Medya[1,2], Ludmila Cherkasova[2], Guilherme Magalhaes[3], Kivanc Ozonat[2], Chaitra Padmanabha[3], Jiban Sarma[3], Imran Sheikh[3]

[1]University of California, Santa Barbara, [2]Hewlett Packard Labs, [3]Hewlett Packard Enterprise

medya@cs.ucsb.edu, lucy.cherkasova@hpe.com, guilherme.magalhaes@hpe.com, kivanc.ozonat@hpe.com, p.chaitra@hpe.com, jiban.jyoti.sarma@hpe.com, imrans@hpe.com

Abstract

Many HPC and modern Big Data processing applications belong to a class of so-called scale-out applications, where the application dataset is partitioned and processed by a cluster of machines. Understanding and assessing the scalability of the designed application is one of the primary goals during the application implementation. Typically, in the design and implementation phase, the programmer is bound to a limited size cluster for debugging and performing profiling experiments. The challenge is to assess the scalability of the designed program for its execution on a larger cluster. While in an increased size cluster, each node needs to process a smaller fraction of the original dataset, the communication volume and communication time might be significantly increased, which could become detrimental and provide diminishing performance benefits. The distributed memory applications exhibit complex behavior: they tend to interleave computations and communications, use bursty transfers, and utilize global synchronization primitives. Therefore, one of the main challenges is the analysis of bandwidth demands due to increased communication volume as a function of a cluster size. In this paper[1], we introduce a novel approach to assess the scalability and performance of a distributed memory program for execution on a large-scale cluster. Our solution involves 1) a limited set of traditional experiments performed in a medium size cluster and 2) an additional set of similar experiments performed with an "interconnect bandwidth throttling" tool, which enables the assessment of the communication demands with respect to available bandwidth. This approach enables a prediction of a cluster size, where a communication cost becomes a dominant component, at which point the performance benefits of the increased cluster lead to a diminishing return. We demonstrate the proposed approach using a popular Graph500 benchmark.

1. INTRODUCTION

In the last few years, graph algorithms have received much attention and become increasingly important for solving many problems in social networks, web connectivity, scientific computing, data mining, and other domains. The numbers of vertices in the analyzed graph networks have grown from billions to tens of billions and the edges have grown from tens of billions to hundreds of billions. A traditional way for improving performance application is to store and process its working set in memory. As the problem size increases and it cannot fit into memory of a single server, the distributed computing and memory resources are required for holding the entire dataset in memory and processing it. This leads to a scale-out execution, where each machine handles a portion of the complete dataset, and needs to communicate with each other to synchronize the executions.

Message passing interface (MPI) is a standard programming paradigm for scale-out, distributed memory applications. Complex MPI-based programs interleave computations and communications in inter-tangled patterns which makes it difficult to perform an accurate analysis of communication layer impact on application performance and predict scaling properties of the program. Due to asynchronous, concurrent execution of different nodes, many communication delays between the nodes could be "hidden" (i.e., do not contribute or impact the overall application completion time). It happens when some nodes are still in their "computation-based" or "processing" portions of the code, while the other nodes already perform communication exchanges. Equally difficult is to analyze the utilized (required) interconnect bandwidth during the execution of MPI program due to a variety of existing MPI collectives and calls that could involve different sets of nodes and communication styles. At the same time, performance of such distributed memory applications inherently depends on a performance of a communication layer of the cluster.

Designing and implementing an efficient and scalable distributed memory program is a challenging task. Typically, during the initial implementation and debugging phases, a programmer is limited to experiments on a small/medium size cluster for application testing and profiling. The challenge is to assess (predict) the scalability of the designed program during its execution on a larger size cluster. This scalability problem had existed for decades and some elaborate and sophisticated ensembles of tools and simulators [11, 10, 6, 12, 5, 9, 2, 4, 1, 13, 14] were proposed by HPC community to attack this challenging problem.

In this work, we discuss a new approach for assessing the scalability and performance of distributed memory programs. We analyze a recently introduced by HPC community Graph500 benchmark [3] for measuring and comparing computer's performance in memory retrieval. It implements a Breadth First Search algorithm on graphs and uses as an input a synthetically generated scale-free graph, which could be easily scaled to extremely large sizes. Our approach is based on performing a limited set of traditional experiments in a small/medium size cluster for assessing a baseline

[1]This work was originated and largely completed during S. Medya's internship at Hewlett Packard Labs in summer 2015.

ICPE 2016, March 12-18, 2016, Delft, Netherlands.
 ISBN 978-1-4503-4080-9/16/03 ...$15.00
DOI: http://dx.doi.org/10.1145/2851553.2858669.

program scalability. For deriving the interconnenct bandwidth demands by the program in a larger size cluster and assessing these demands' scaling trend, we perform an additional set of similar experiments augmented with the "interconnect bandwidth throttling" tool [15], which helps to expose the communication demands of the program with respect to required (utilized) interconnect bandwidth. By combining the insights from these two complementary sets of experiments, we could project the application performance for larger cluster sizes, and in particular, the size, where a communication cost becomes a dominant component. At this point, the performance benefits of the increased cluster size provide a diminishing return. The remainder of the paper presents our approach and results in more detail.

2. WHAT MATTERS FOR APPLICATION PERFORMANCE AND SCALABILITY?

In this work, we focus on the performance and scalability analysis of distributed shared memory programs, and graph algorithms in particular. We demonstrate the problem and our approach by considering the Graph500 benchmark [3] that implements Breadth First Search (BFS) algorithm. BFS is an important algorithm as it serves a building block for many other algorithms such as computation of betweenness centrality of vertices, shortest path between two vertices, etc. Breadth First Search is a typical graph traversal algorithm performed on an undirected, unweighted graph. A goal is to compute a distance from a given source vertex s to each vertex in the graph, i.e., finding all the vertices which are "one hop" away, "two hops" away, etc. When reasoning about program performance and its execution efficiency on a large-scale distributed cluster, the following factors are critically important (see Figure 1):

1. **Selected Underlying Algorithm**: Graph problems could be implemented in many different ways in terms of graph data partitioning for parallel processing as shown in the top layer of Figure 1. Data partitioning and algorithm details play an important role in scalability analysis as it impacts algorithm's communication style and the communication volume (in a distributed scenario), and thus the application completion time. There are a few well-known data partitioning approaches (e.g., 1-D and 2-D) proposed for parallel processing of BFS [8]. The 2-D partitioning algorithm was theoretically proven to be a scalable algorithm [8].

2. **Implementation Code**: Program performance and its scalability further depends on the code that implements a selected/designed algorithm. This is the middle layer shown in Figure 1. In spite of excellent theoretical properties of the 2-D partitioning algorithm, its inefficient implementation may result in a poorly performing and badly scaling program. Therefore, the implementation details are critical part of application performance and scalability.

3. **Underlying System Hardware and Software**: Finally, the underlying system hardware, that is available and targeted for program execution, is extremely important for program performance and its scalability as shown by the bottom layer in Figure 1. Specially designed systems, such as Blue Gene and K Computer, have proprietary, custom-built interconnects which provide enhanced support for MPI collectives, and therefore, demonstrate superior performance compared to commodity clusters.

Therefore, the scalability analysis and performance prediction depends on the underlying graph problem/algorithm, its parallel implementation, and the underlying system software and hardware. Figure 1 illustrates these critical factors with some examples from each category.

In the paper, we demonstrate our approach by using the 2-D partitioning IBM implementation of BFS algorithm [3] executed on the commodity cluster (shown by the green boxes in Figure 1).

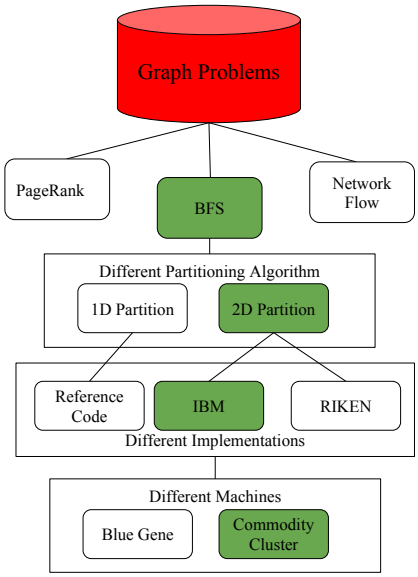


Figure 1: Performance and scalability: example of critical factors.

In our experiments, we use a **32-node cluster** connected via FDR InfiniBand (56 Gbits/s). Each node is based on HP DL360 Gen9 servers with two sockets, each with 14 cores, 2 GHz, Xeon E5-2683 v3, 35 MB last level cache size, and 256 GB DRAM per node.

3. BASE LINEAR REGRESSION MODEL

There are different ways to formulate "scaling" of a particular program. One of the classical methods is ***strong scaling***. In strong scaling, the problem data size is kept fixed and the number of processors (nodes) to execute the program is increased. In a general case, the completion time of a distributed memory program can be modeled as follows:

$$CompletionTime = ProcessingTime + CommunicationTime$$

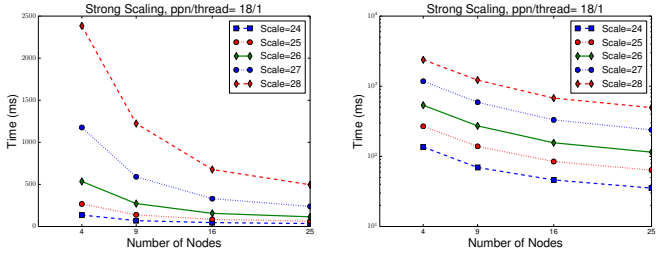
As the number of processors in the cluster is increased to p, one would expect that the *ProcessingTime* in this equation will improve by p times. With the assumption that the data is evenly distributed over the nodes, the processing time can be approximated as $O(\frac{1}{p})$.

To estimate the *CommunicationTime* of a distributed memory program as a function of number of processors p is a more challenging task. To our rescue comes a theoretical analysis of 2-D partitioning implementation [8]: its communication pattern is well known−the number of messages per processor is $O(\sqrt{p})$. We exploit this asymptotic analysis and include this factor in our model. So, the communication time can be accounted as $O(\frac{1}{\sqrt{p}})$.

Base Linear Regression Model: we can derive the formulation for *Completion Time* as linearly dependent on $\frac{1}{p}$ and $\frac{1}{\sqrt{p}}$:

$$Completion\ Time(p) = C_1 * \frac{1}{p} + C_2 * \frac{1}{\sqrt{p}} \qquad (1)$$

Figure 2 (a) shows measured completion times of Graph500 benchmark executed in our cluster for different graph sizes and number of nodes in the cluster. We configured each node in the cluster to execute 18 MPI processes, each with 1 thread. The legend "scale" denotes the size of the graph [3]. The scale s defines the graph with 2^s vertices and $16 \cdot 2^s$ edges, e.g., graph of scale 27 has 134 Million vertices and 2.1 Billion edges, graph of scale 28 has 268 Million vertices and 4.2 Billion edges, etc.



(a) Regular linear-scale format. (b) Logscale (Y-axes) format.

Figure 2: The Graph500 completion time in a strong scaling scenario.

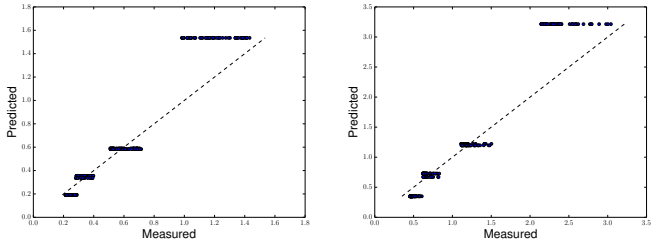
Note, that Figure 2 (b) shows the same measurements with Y-axes in logscale format. If $CompletionTime(p)$ would scale as $O(\frac{1}{p})$ (i.e., no communication overhead) then in Figure 2 (b) one can expect a straight line with a negative slope near 1 [7]. However, it is not the case, and the communication time represents an essential component of the overall completion time of Graph500 execution.

Eq. 1 provides the formulation for *CompletionTime* as linearly dependent on $\frac{1}{p}$ and $\frac{1}{\sqrt{p}}$, where C_1 and C_2 are constants which need to be derived from the asymptotic analysis. We aim to find these constants via linear regression. Using a medium size cluster N with n processors in total we obtain measured completion times for BFS code on all possible sub-cluster configurations with p processors, where $p \leq n$. So, we have data points as a pair, $(time, number\ of\ processors)$. We use these experimental data in the set of equations (as shown below) and solve this set of equations for finding the coefficients C_1 and C_2 via linear regression:

$$CompletionTime_1(p_1) = C_0 + C_1 * \frac{1}{p_1} + C_2 * \frac{1}{\sqrt{p_1}}$$
$$CompletionTime_2(p_2) = C_0 + C_1 * \frac{1}{p_2} + C_2 * \frac{1}{\sqrt{p_2}}$$
$$\ldots \qquad \ldots \qquad \ldots \qquad \ldots$$

where $CompletionTime_i$ is the corresponding completion time when p_i processors are used. C_0 is added in regression methods to characterize noise. A popular method for solving such set of equations is Least Squares Regression, which we use here. In statistics, this is an approach for modeling the relationship between a scalar dependent variable (e.g., *CompletionTime* here) and one or more independent variables (e.g., $\frac{1}{p}$ and $\frac{1}{\sqrt{p}}$). The set of coefficients C_0, C_1 and C_2 is the model that describes the relationship.

For a fixed problem data size in Graph500, we perform experiments with different number of nodes in our 32-node cluster (using the same configuration per node), collect measurement data, and then solve Eq. 1 with linear regression for finding constants C_0, C_1, and C_2. Figures 3 (a)-(b) show the regression results for problem scales 27 and 28 respectively. The solution is based on collected measurements of approximately 1000 data points.



(a) Scale = 27 (b) Scale = 28

Figure 3: Computing the Constants via Regression.

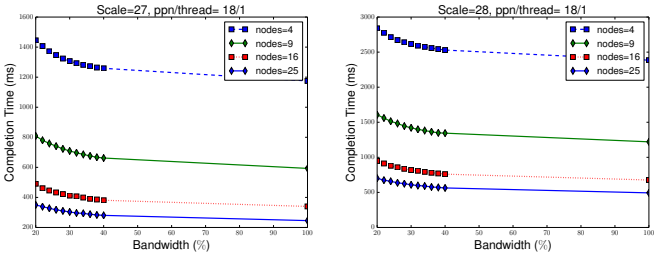
Two well studied measures to show quality of regression are R^2 and "mean square" error. R^2 is better being close to 1. We get R^2 as 0.95 and 0.97 for scales 27 and 28 respectively. The corresponding "mean square" errors (close to 0 is better) are 0.04 and 0.19. Both types of errors reflect a high quality of regression results. The computed coefficient C_1 (57.67 and 102.1) is greater than C_2 (3.8 an 10.3) in both cases (for scales 27 and 28 respectively). This result shows that the processing time dominates the benchmark execution time (i.e., *CompletionTime*) in a small/medium cluster.

4. ESTIMATING THE COMMUNICATION BANDWIDTH DEMANDS

With an increased number of cluster nodes the communication volume becomes a dominant component in Eq 1. Past literature [7] shows that in a strong scaling scenario, the program performance gets additionally impacted when system interconnect bandwidth starts affecting the communication time. We need to assess the increased bandwidth demands of a communication volume as a function of the increased cluster size. Unfortunately, there are no existing tools or common approaches to analyze the utilized (required) interconnect bandwidth during the execution of general MPI program. It is a very challenging task due to a variety of existing MPI collectives and MPI calls that could involve different sets of nodes and communication styles.

To overcome this challenge, we apply *InterSense* [15] - a special interconnect emulator, which can control (throttle) the interconnect bandwidth to determine how much bandwidth the program needs before its completion time becomes impacted. It enables us to accurately estimate the required (needed) bandwidth by the program.

Figures 4 (a)-(b) show the outcome of bandwidth throttling experiments. We execute Graph500 benchmark for two different dataset scales, 27 and 28, and four different cluster sizes. Each line shows the benchmark completion times at different (controlled by *InterSense*) interconnect bandwidth percentage. These plots show that there is a non-linear relation between completion times and available interconnect bandwidth. To get accurate estimates on the required interconnect bandwidth, we experiment with 2% interval in the range from 20% to 40%. As expected, the completion times are higher with larger scales and smaller process configurations.



(a) Scale = 27 (b) Scale = 28

Figure 4: Bandwidth Impact on the Completion Time.

For a particular configuration, we define *Completion Time Increment* (**CTI**) at an available bandwidth (*bw*) as the increment in percentage w.r.t the completion time when 100% bandwidth is available.

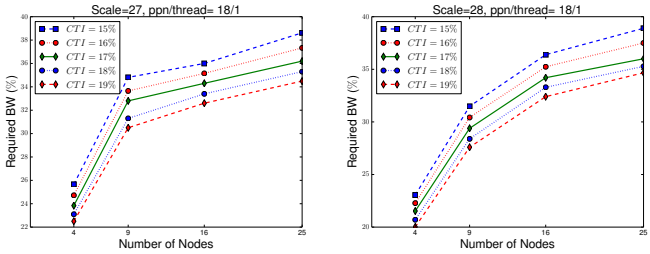
$$CTI_{bw} = \frac{Completion\ Time\ at\ bw - Completion\ Time\ at\ 100}{Completion\ Time\ at\ 100} \quad (2)$$

Bandwidth Demand ($BW_{CTI,p}$) is defined as the percentage of bandwidth required to achieve the predefined CTI for a particular processor configuration p.

Figure 4 shows that different cluster configurations have different completion times when the available interconnect bandwidth is

varied. **The question is** how *CTI* is related to bandwidth demands in a cluster with different number of nodes? So, **the goal is** to predict the required interconnect bandwidth by Graph500 benchmark in the cluster with increased number of nodes. Then we can incorporate the impact of increased bandwidth demands into the increased communication time. As a result, we can estimate the cluster size, where a communication cost becomes a highly dominant component, at which point the performance (scalability) benefits in the further increased cluster would lead to a diminishing return.

We aim to build a model of required interconnect bandwidth for predicting these bandwidth demands in a larger size cluster. For a particular problem scale (i.e., graph size), different CTIs have similar trends as shown in Figure 5.



(a) Scale = 27 (b) Scale = 28

Figure 5: Interconnect Bandwidth Demands.

We compute a single *DemandConstant (DC)* for all different CTIs to predict the interconnect bandwidth demand for a larger cluster configuration. First, we determine how two processors' configurations and their required bandwidth demands are related. We follow the relation to determine a constant DC_{CTI,p_1,p_2} for a particular *CTI*, and two processors' configurations :

$$\frac{BW_{CTI,p_1}}{BW_{CTI,p_2}} = DC_{CTI,p_1,p_2}\frac{\sqrt{p_1}}{\sqrt{p_2}}$$

where $p_1 > p_2$, and p_1 and p_2 are the number of processes in the executed configurations. The intuition of the above relation comes from 2-D partitioning algorithm, where a number of messages per processor is $O(\sqrt{p})$. *DC* is taken as an average over all such DC_{p_i,p_j}. Once *DC* is computed, one can use the following equation to find the bandwidth demand ($BW_{CTI,p'}$) for a larger cluster size and a given *CTI*:

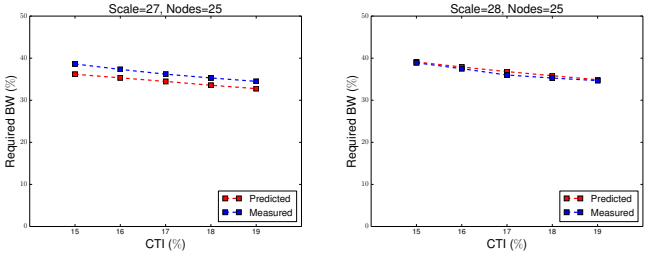
$$BW_{CTI,p'}/BW_{CTI,p*} = DC\frac{\sqrt{p'}}{\sqrt{p*}} \quad (3)$$

where $p*$ is a number of processes in the smaller cluster size configuration available.

Next, we validate our model by executing a set of experiments with varying interconnect bandwidth, size of the graph, and number of nodes in the cluster.

Prediction Accuracy of Bandwidth Demands: We aim to predict the interconnect bandwidth demands using Eq. 3 for a 25-node cluster, and evaluate the accuracy of the designed model. Using the collected measurements for clusters with 4, 9, and 16 nodes (where each node is configured with 18 MPI processes, i.e., with 72, 162, 288 MPI processes respectively), we can obtain 3 different combinations for each considered CTI (15 in total). Actual *DC* is averaged over these 15 DC_{p_i,p_j}s. Bandwidth demands for 25 nodes (with 450 processes respectively) are computed using *DC*, Eq. 3, and $p*$ as 288 (16 nodes). The *DC* for scales 27 and 28 are 0.8 and 0.86 respectively.

Figures 6 (a)-(b) show the accuracy of the prediction. The error (difference between predicted and measured) is lower than 2% and 0.5% for scales 27 (Fig. 6 (a)) and 28 (Fig. 6 (b)) respectively.



(a) Scale = 27 (b) Scale = 28

Figure 6: Prediction of Required Bandwidth Demands.

5. CONCLUSION AND FUTURE WORK

Designing and implementing an efficient and scalable distributed memory program is a challenging task. In this work, we discuss a new approach for assessing the scalability and performance of distributed memory programs by using Graph500 benchmark as a motivating example. We show a set of critical factors that needs to be taken into account for scalability analysis of a distributed memory program. Since a scalability of many distributed programs is limited by their communication volume and the available interconnect bandwidth, we show how one can derive the estimates on the required interconnect bandwidth in a larger cluster from the experiments performed in a small/medium cluster with an "interconnect bandwidth throttling" tool. By combining the outcome of these two components, we can estimate the cluster size, where a communication cost becomes a dominant component, at which point the performance benefits of the increased cluster lead to a diminishing return. In our future work, we plan to incorporate the dataset size (i.e., graph size) as a scalability problem parameter.

6. REFERENCES

[1] Dimemas: predict parallel performance using a single cpu machine. `http://www.bsc.es/computer-sciences/dimemas`.

[2] Extrae instrumentation package. `http://www.bsc.es/computer-sciences/extrae`.

[3] Graph500. `http://www.graph500.org/`.

[4] Paraver:Performance Analysis Tools: Details and Intelligence. `http://www.bsc.es/computer-sciences/paraver`.

[5] L. Adhianto, S. Banerjee, M. W. Fagan, M. Krentel, G. Marin, J. M. Mellor-Crummey, and N. R. Tallent. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6), 2010.

[6] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho. Using Performance Modeling to Design Large-Scale Systems. *Computer*, 42, Nov., 2009.

[7] A. Buluc and J. R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal on Scientific Computing*, 34(4), 2012.

[8] A. Buluç and K. Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, 2011.

[9] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In *Proc. of Intl. Conf. for High Perf. Computing, Networking, Storage and Analysis (SC'13)*, 2013.

[10] M. Casas, R. M. Badia, and J. Labarta. Automatic Analysis of Speedup of MPI Applications. In *Proc. of the 22nd Intl. Conf. on Supercomputing*, 2008.

[11] C. Coarfa, J. M. Mellor-Crummey, N. Froyd, and Y. Dotsenko. Scalability analysis of SPMD codes using expectations. In *Procs. of the 21th Annual International Conference on Supercomputing, (ICS 2007)*, 2007.

[12] M. Geimer et al. The scalasca performance toolset architecture. *Journal on Concurr. Comput.: Pract. Exper.*, 22(6), Apr., 2010.

[13] C. Rosas, J. Gimenez, and J. Labarta. Scalability Prediction for Fundamental Performance Factors. *Journal on Supercomputing Frontiers and Innovations*, 1(2), 2014.

[14] C. Rosas, J. Gimenez, and J. Labarta. Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale. *Journal on Future Generation Computer Systems*, 30, 2014.

[15] Q. Wang, L. Cherkasova, J. Li, and H. Volos. InterSense: Interconnect Performance Emulator for Future Scale-out Distributed Memory Applications. In *Proc. of the 23th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2015.