2014 IEEE International Symposium on Parallel and Distributed Processing with Applications

BwMan: Bandwidth Manager for Elastic Services in the Cloud

Ying Liu*, Vamis Xhagjika†*, Vladimir Vlassov*, Ahmad Al-Shishtawy‡
*KTH Royal Institute of Technology, Stockholm, Sweden, {yinliu, xhagjika, vladv}@kth.se
†Universitat Politècnica de Catalunya, Barcelona, Spain, xhagjika@ac.upc.edu
‡Swedish Institute of Computer Science (SICS), Stockholm, Sweden, ahmad@sics.se

***Abstract*—The flexibility of Cloud computing allows elastic services to adapt to changes in workload patterns in order to achieve desired Service Level Objectives (SLOs) at a reduced cost. Typically, the service adapts to changes in workload by adding or removing service instances (VMs), which for stateful services will require moving data among instances. The SLOs of a distributed Cloud-based service are sensitive to the available network bandwidth, which is usually shared by multiple activities in a single service without being explicitly allocated and managed as a resource. We present the design and evaluation of BwMan, a network bandwidth manager for elastic services in the Cloud. BwMan predicts and performs the bandwidth allocation and tradeoffs between multiple service activities in order to meet service specific SLOs and policies. To make management decisions, BwMan uses statistical machine learning (SML) to build predictive models. This allows BwMan to arbitrate and allocate bandwidth dynamically among different activities to satisfy specified SLOs. We have implemented and evaluated BwMan for the OpenStack Swift store. Our evaluation shows the feasibility and effectiveness of our approach to bandwidth management in an elastic service. The experiments show that network bandwidth management by BwMan can reduce SLO violations in Swift by a factor of two or more.**

***Keywords*-Bandwidth Management, Cloud Computing, SLO**

I. Introduction

Cloud computing with its pay-as-you-go pricing model and illusion of the infinite amount of resources drives our vision on the Internet industry, in part because it allows providing elastic services where resources are dynamically provisioned and reclaimed in response to fluctuations in workload while satisfying SLO requirements at a reduced cost. When the scale and complexity of Cloud-based applications and services increase, it is essential and challenging to automate the resource provisioning in order to handle dynamic workload without violating SLOs. Issues to be considered when building systems to be automatically scalable in terms of server capabilities, CPU and memory, are fairly well understood by the research community and discussed in literature, e.g., [1], [2], [3]. There are open issues to be solved, such as efficient and effective network resource management.

In Cloud-based systems, services, and applications, network bandwidth is usually not explicitly allocated and managed as a shared resource. Sharing bandwidth by multiple physical servers, virtual machines (VMs), or service threads communicating over the same network, may lead to SLO violations. Furthermore, network bandwidth can also be presented as a first class managed resource in the context of Internet Service Provider (ISP), inter-ISP communication, Clouds as well as community networks [4], where the network bandwidth is the major resource.

In our work, we demonstrate the necessity of managing the network bandwidth shared by services running on the same platform, especially when the services are bandwidth intensive. The sharing of network bandwidth can happen among multiple individual applications or within one application of multiple services deployed in the same platform. In essence, both cases can be solved using the same bandwidth management approach. The difference is in the granularity in which bandwidth allocation is conducted, for example, on VMs, applications or threads. In our work, we have implemented the finest bandwidth control granularity, i.e., network port level, which can be easily adapted in the usage scenario of VMs, applications, or services. Specifically, our approach is able to distinguish bandwidth allocations to different ports used by different services within the same application. In fact, this fine-grained control is needed in many distributed applications, where there are multiple concurrent threads creating workloads competing for bandwidth resources. A widely used application in such scenario is distributed storage service.

A distributed storage system provides a service that integrates physically separated and distributed storages into one logical storage unit, with which the client can interoperate as if it is one entity. There are two kinds of workload in a storage service. First, the system handles dynamic workload generated by the clients, that we call *user-centric workload*. Second, the system tackles with the workload related to system maintenance including load rebalancing, data migration, failure recovery, and dynamic reconfiguration (e.g., elasticity). We call this workload *system-centric workload*.

In a distributed storage service, the user-centric workload includes access requests issued by clients; whereas the system-centric workload includes the data replication, recovery, and rebalance activities performed to achieve and to ensure system availability and consistency. Typically the

978-1-4799-4293-0/14 $31.00 © 2014 IEEE
DOI 10.1109/ISPA.2014.37

system-centric workload is triggered in the following situations. At runtime, when the system scales up, the number of servers and the storage capacity is increased, that leads to data transfer to the newly added servers. Similarly, when the system scales down, data need to be migrated before the servers are removed. In another situation, the system-centric workload is triggered in response to server failures or data corruptions. In this case, the failure recovery process replicates the under-replicated data or recover corrupted data. Rebalance and failure recovery workloads consume system resources including network bandwidth, thus may interfere with user-centric workload and affect SLOs.

From our experimental observations, in a distributed storage system, both user-centric and system-centric workloads are network bandwidth intensive. To arbitrate the allocation of bandwidth between these two kinds of workload is challenging. On the one hand, insufficient bandwidth allocation to user-centric workload might lead to the violation of SLOs. On the other hand, the system may fail when insufficient bandwidth is allocated for data rebalance and failure recovery [1]. To tackle this problem, we arbitrate network bandwidth between user-centric workload and system-centric workload in a way to minimize SLO violations and keep the system operational.

We propose the design of BwMan, a network bandwidth manager for elastic Cloud services. BwMan arbitrates the bandwidth allocation among individual services and different service activities sharing the same Cloud infrastructure. Our control model is built using machine learning techniques [5]. A control loop is designed to continuously monitor the system status and dynamically allocate different bandwidth quotas to services depending on changing workloads. The bandwidth allocation is fine-grained to ports used by different services. Thus, each service can have a demanded and dedicated amount of bandwidth allocation without interfering among each other, when the total bandwidth in the shared platform is sufficient. Dynamic and dedicated bandwidth allocation to services supports their elasticity properties with reduced resource consumption and better performance guarantees. From our evaluation, we show that more than half of the SLO violations is prevented by using BwMan for an elastic distributed storage deployed in the Cloud. Furthermore, since BwMan controls bandwidth in port granularity, it can be easily extended to adapt to other usage scenarios where network bandwidth is a sharing resource and creates potential bottlenecks.

In this work, we build and evaluate BwMan for the case of a data center LAN topology deployment. BwMan assumes that bandwidth quotas for each application is given by data center policies. Within a limited bandwidth quota, BwMan tries to utilize it in the best way, by dividing it to workloads inside the applications. Specifically, BwMan arbitrates the available inbound and outbound bandwidth of servers , i.e., bandwidth at the network edges, to multiple hosted services; whereas the bandwidth allocation of particular network flows in switches is not under the BwMan control. In most of the deployments, control of the bandwidth allocation in the network by services might not be supported.

The contributions of this work are as follows.

- First, we propose a bandwidth manager for distributed Cloud-based services using predictive models to better guarantee SLOs.
- Second, we describe the BwMan design including the techniques and metrics of building predictive models for system performance under user-centric and system-centric workloads as a function of allocated bandwidth.
- Finally, we evaluate the effectiveness of BwMan using the OpenStack Swift Object Storage.

The rest of the paper is organized as follows. In Section II, we describe the background for this work. Section III presents the control model built for BwMan. In Section IV, we describe the design, architecture, and work-flow of BwMan. Section V shows the performance evaluation of the bandwidth manager. We conclude in Section VII.

II. OpenStack Swift

A distributed storage service provides an illusion of a storage with infinite capacity by aggregating and managing a large number of storage servers. Storage solutions [6], [7], [8], [9] include relational databases, NoSQL databases, distributed file systems, array storages, and key-value stores. In this paper, we consider an object store, namely OpenStack Swift, as a use case for our bandwidth management mechanism. Swift follows a key-value storage style, which offers a simple interface that allows to put, get, and delete data identified by keys. Such simple interface enables efficient partitioning and distribution of data among multiple servers and thus scaling well to a large number of servers. Examples of key-value storages are Amazon S3, OpenStack Swift, Cassandra [6] and Voldemort [7]. OpenStack Swift, considered in this work, is one of the storage services of OpenStack Cloud platform [8]. In Swift, there are one or many Name Nodes (representing a data entry point to the distributed storage) that are responsible for the management of the Data Nodes. Name Nodes may store the metadata describing the system or just be used as access hubs to the distributed storage. The Name Nodes may also be responsible for managing data replication, but leave actual data transfer to the Data Nodes themselves. Clients access the distributed storage through the Name Nodes using a mutually agreed upon protocol and the result of the operation is also returned by the same Name Node. Despite Name Nodes and Data Nodes, Swift consists of a number of other components, including Auditors, Updators and Replicators, together providing functionalities such as highly available storage, lookup service, and failure recovery. In our evaluation, we consider bandwidth allocation tradeoffs among these components.

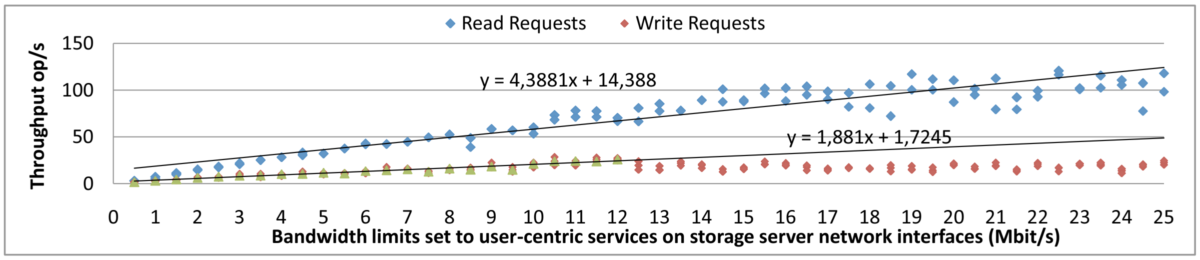


Fig. 1. Regression Model for System Throughput vs. Available Bandwidth

III. Predictive Models of the Target System

BwMan bandwidth manager uses easily-computable predictive models to foresee system performance under a given workload in correlation to bandwidth allocation. As there are two types of workloads in the system, namely user-centric and system-centric, we show how to build two predictive models. The first model defines correlation between the user-oriented performance metrics under user-centric workload and the available bandwidth. The second model defines correlation between system-oriented performance metrics under system-centric workload and the available bandwidth.

We define user-oriented performance metrics as the system throughput measured in read/write operations per second (op/s). As a use case, we consider the system-centric workload associated with failure recovery, that is triggered in response to server failures or data corruptions. The failure recovery process is responsible to replicate the under-replicated data or recover corrupted data. Thus, we define the system-oriented performance metrics as the recovery speed of the corrupted data in megabyte per second (MB/s). Due to the fine-grained control of network traffic on different service ports, the bandwidth arbitration by BwMan will not interfere with other background services in the application, such as services for failure detection and garbage collection.

The mathematical models we have used are regression models. The simplest case of such an approach is a one variable approximation, but for more complex scenarios, the number of features of the model can be extended to provide also higher order approximations. In the following subsections, we show the two derived models.

A. User-oriented Performance versus Available Bandwidth

First, we analyze the read/write (user-centric) performance of the system under a given network bandwidth allocation. In order to conduct decisions on bandwidth allocation against read/write performance, BwMan uses a regression model [2], [3], [10] of performance as a function of available bandwidth. The model can be built either off-line by conducting experiments on a rather wide (if not complete) operational region; or on-line by measuring performance at runtime. In this work, we present the model trained off-line for the OpenStack Swift store by varying the bandwidth allocation and measuring system throughput as shown in Fig. 1. The model is set up in each individual storage node. Based on the incoming workload monitoring, each storage node is assigned with demanded bandwidth accordingly by BwMan in one control loop. The simplest computable model that fits the gathered data is a linear regression of the following form:

$$Throughput[op/s] = \alpha_1 * Bandwidth + \alpha_2 \quad (1)$$

For example, in our experiments, we have identified the weights of the model for read throughput to be $\alpha_1 = 4.388$ and $\alpha_2 = 14.38$. As shown in Fig. 1, this model approximates with a relatively good precision the predictive control function. Note that the second half of the plot for write operations is not taken into consideration, since the write throughput in this region does not depend on the available bandwidth since there are other factors, which might become the bottlenecks, such as disk write access.

B. Data Recovery Speed versus Available Bandwidth

Next, we analyse the correlation between system-centric performance and available bandwidth, namely, data recovery speed under a given network bandwidth allocation. By analogy to the first model, the second model was trained off-line by varying the bandwidth allocation and measuring the recovery speed under a fixed failure rate. The difference is that the model predictive process is centrally conducted based on the monitored system data integrity and bandwidth are allocated homogeneously to all storage servers. For the moment, we do not consider the fine-grained monitor of data integrity on each storage node. We treat data integrity at the system level.

The model that fits the collected data and correlates the recovery speed with the available bandwidth is a regression model where the main feature is of logarithmic nature as shown in Fig. 2. The concise mathematical model is

$$RecoverySpeed[MB/s] = \alpha_1 * ln(Bandwidth) + \alpha_2 \quad (2)$$

Fig. 2 shows the collected data and the model that fits the data. Specifically, in our case, the weights in the logarithmic regression model are $\alpha_1 = 441.6$ and $\alpha_2 = -2609$.

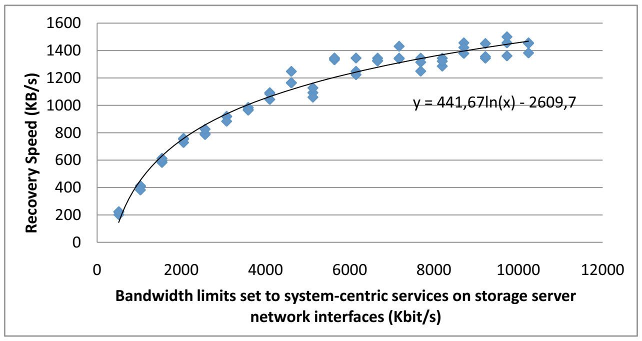


Fig. 2. Regression Model for Recovery Speed vs. Available Bandwidth

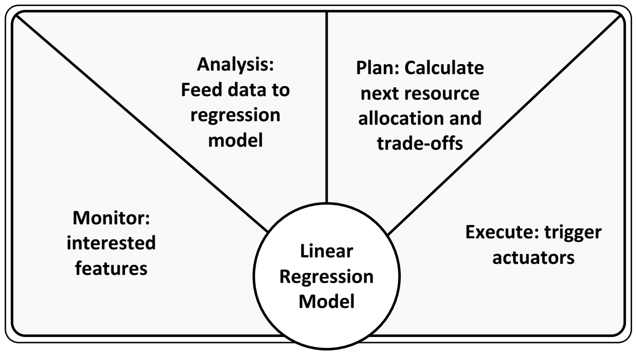

Analysis: Feed data to regression model
Plan: Calculate next resource allocation and trade-offs
Monitor: interested features
Linear Regression Model
Execute: trigger actuators

Fig. 3. MAPE Control Loop of Bandwidth Manager

IV. BwMan: Bandwidth Manager

In this section, we describe the architecture of BwMan, a bandwidth manager which arbitrates bandwidth between user-centric workload and system-centric workload of the target distributed system. BwMan operates according to the MAPE-K loop [11] (Fig. 3) passing the following phases:

- Monitor: monitor user-defined SLOs, incoming workloads to each storage server and system data integrity;
- Analyze: feed monitored data to the regression models;
- Plan: use the predictive regression model of the target system to plan the bandwidth allocation including tradeoffs. In the case when the total network bandwidth has been exhausted and cannot satisfy all the workloads, the allocation decisions are made based on specified tradeoff policies (explained in Section IV-B);
- Execute: allocate bandwidth to sub-services (storage server performance and system failure recovery) according to the plan.

Control decisions are made by finding correlations through data using two regression models (Section III). Each model defines correlations between a specific workload (user-centric or system-centric) and bandwidth.

A. BwMan Control Work-flow

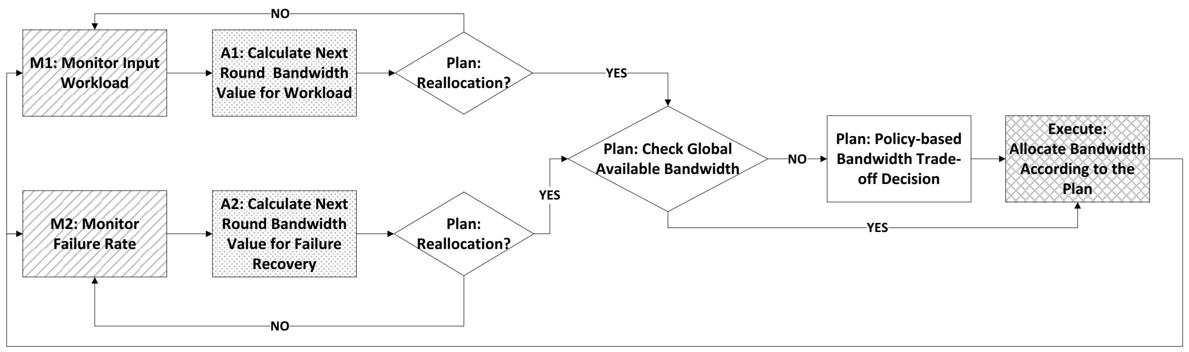
The flowchart of BwMan is shown in Fig. 4. BwMan monitors three signals, namely, user-centric throughput (defined in SLO), the workload to each storage server and data integrity in the system. At given time intervals, the gathered data are averaged and fed to analysis modules. Then the results of the analysis based on our regression model are passed to the planning phase to decide on actions based on SLOs and potentially make tradeoff decision. The results from the planning phase are executed by the actuators in the execution phase. Fig. 4 depicts the MAPE phases as designed for BwMan. For the Monitor phase, we have two separate monitor ports, one for user-centric throughput (M1) and the other one for data failure rates (M2). The outputs of these stages are passed to the Analysis phase represented by two calculation units, namely A1 and A2, that aggregate and calculate new bandwidth availability, allocation and metrics to be used during the Planning phase according to the trained models (Section III). The best course of action to take during the Execution phase is chosen based on the calculated bandwidth necessary for user-centric workload (SLO) and the current data failure rate, estimated from system data integrity in the Planning phase. The execution plan may include also the tradeoff decision in the case of bandwidth saturation. Finally, during the Execution phase, the actuators are employed to modify the current state of the system, which is the new bandwidth allocations for the user-centric workload and for the system-centric (failure recovery) workload to each storage server.

B. Tradeoff Scenario

BwMan is designed to manage a finite resource (bandwidth), so the resource may not always be available. We describe a tradeoff scenario where the bandwidth is shared among user-centric and system-centric workloads.

In order to meet specified SLOs, BwMan needs to tune the allocation of system resources in the distributed storage. In our case, we observe that the network bandwidth available for user-centric workload directly impact user-centric performance (request throughput). Thus, enough bandwidth allocation to the user-centric workload is essential to meet SLOs. On the other hand, system-centric workload, such as failure recovery and data rebalance, are executed in order to provide better reliability for data in a distributed storage. The rebalance and replication process moves copies of the data to other nodes in order to have more copies for availability and self-healing purposes. This activity indirectly limits user-centric performance by impacting the internal bandwidth of the storage system. While moving the data, the available bandwidth for user-centric workload is lowered as system-centric workload competes for the network bandwidth with user-centric workload.

By arbitrating the bandwidth allocated to user-centric and system-centric workloads, we can enforce more user-centric performance while penalizing system-centric functionalities or vice versa. This tradeoff decision is based on policies specified in the controller design.



NO
M1: Monitor Input Workload
A1: Calculate Next Round Bandwidth Value for Workload
Plan: Reallocation?
YES
Plan: Check Global Available Bandwidth
NO
Plan: Policy-based Bandwidth Trade-off Decision
Execute: Allocate Bandwidth According to the Plan
YES
M2: Monitor Failure Rate
A2: Calculate Next Round Bandwidth Value for Failure Recovery
Plan: Reallocation?
YES
NO

Fig. 4. Control Workflow

The system can limit the bandwidth usage of an application by selecting the requests to process and those to ignore. This method is usually referred as admission control, which we do not consider here. Instead we employ actuators to arbitrate the bandwidth between user-centric workload and system-centric workload.

V. Evaluation

In this section, we present the evaluation of BwMan in OpenStack Swift. The storage service was deployed in an OpenStack Cloud in order to ensure complete isolation and sufficiently enough computational, memory, and storage resources.

A. OpenStack Swift Storage

As a case study, we evaluate our control system in OpenStack Swift, which is a widely used open source distributed object storage started from Rackspace [12]. We identify that, in Swift, user-centric workload (system throughput) and system-centric workload (data rebalance and recovery) are not explicitly managed. We observe that data rebalance and failure recovery mechanisms in Swift are essentially the same. These two services adopt a set of replicator processes using the "rsync" Linux utility. In particular, we decide to focus on one of these two services: failure recovery.

B. Experiment Scenarios

The evaluation of BwMan in OpenStack Swift has been conducted under two scenarios. First, we evaluate the effectiveness of BwMan in Swift with specified throughput SLO for the user-centric workload, and failure rates that correspond to system-centric workload (failure recovery), under the condition that there is enough bandwidth to handle both workloads. These experiments demonstrate the ability of BwMan to manage bandwidth in a way that ensures user-centric and system-centric workloads with maximum fidelity.

Second, a policy-based decision making is performed by BwMan to tradeoff in the case of insufficient network bandwidth to handle both user-centric and system-centric workloads. In our experiments, we give higher priority to the user-centric workload compared to system-centric workload. We show that BwMan adapts Swift effectively by satisfying the user-defined SLO (desired throughput) with relatively stable performance.

C. Experiment Setup

1) Swift Setup: We have deployed a Swift cluster with a ratio of 1 proxy server to 8 storage servers as recommended in the OpenStack Swift documentation [13]. Under the assumption of uniform workload, the storage servers are equally loaded. This implies that the Swift cluster can scale linearly by adding more proxy servers and storage servers following the ratio of 1 to 8.

2) Workload Setup: We modified the Yahoo! Cloud Service Benchmark (YCSB) [14] to be able to generate workloads for a Swift cluster. Specifically, our modification allows YCSB to issue read, write, and delete operations to a Swift cluster with best effort or a specified steady throughput. The steady throughput is generated in a queue-based fashion, where the request incoming rate can be specified and generated on demand. If the rate cannot be met by the system, requests are queued for later execution. The Swift cluster is populated using randomly generated files with predefined sizes. The file sizes in our experiments are chosen based on one of the largest production Swift cluster configured by Wikipedia [15] to store static images, texts, and links. YCSB generates requests with file sizes of 100KB as like an average size in the Wikipedia scenario. YCSB is given 16 concurrent client threads and generates uniformly random read and write operations to the Swift cluster.

3) Failure Generator and Monitor: The injected file loss in the system is used to trigger the Swift's failure recovery process. We have developed a failure generator script that uniformly at random chooses a data node, in which it deletes a specific number of files within a defined period of time. This procedure is repeated until the requested failure rate is reached.

To conduct failure recovery experiments, we customized the swift-dispersion tool in order to populate and monitor the integrity of the whole data space. This customized tool functions also as our failure recovery monitor in BwMan by providing real-time metrics on data integrity.

4) The Actuator: Network Bandwidth Control: We apply NetEm's tc tools [16] in the token buffer mode to control the inbound and outbound network bandwidth associated with the network interfaces and service ports. In this way, we are able to manage the bandwidth quotas for different activities in the controlled system. In our deployment, all the services run on different ports, and thus, we can apply different network management policies to each of the services.

D. User-centric Workload Experiment

Fig. 5 presents the effectiveness of using BwMan in Swift with dynamic user-centric SLOs. The x-axis of the plot shows the experiment timeline, whereas the left y-axis corresponds to throughput in op/s, and the right y-axis corresponds to allocated bandwidth in MB/s.

In these experiments, the user-centric workload is a mix of 80% read requests and 20% write requests, that, in our view, represents a typical workload in a read-dominant application.

Fig. 5 shows the desired throughput specified as SLO, the bandwidth allocation calculated using the linear regression model of the user-centric workload (Section III), and achieved throughput. Results demonstrate that BwMan is able to reconfigure the bandwidth allocated to dynamic user-centric workloads in order to achieve the requested SLOs.

E. System-centric Workload Experiment

Fig. 6 presents the results of the data recovery process, the system-centric workloads, conducted by Swift background process when there are data corruption and data loss in the system. The dotted curve sums up the monitoring results, which constitute the 1% random sample of the whole data space. The sample represents data integrity in the system with max value at 100%. The control cycle activation is illustrated as triangles. The solid curve stands for the bandwidth allocation by BwMan after each control cycle. The calculation of bandwidth allocation is based on a logarithmic regression model obtained from Fig. 2 in Section III.

F. Policy-based Tradeoff Scenario

In this section, we demonstrate that BwMan allows meeting the SLO according to specified policies in tradeoff decisions when the total available bandwidth is saturated by user-centric and system-centric workloads. In our experiments, we have chosen to give preference to user-centric workload, namely system throughput, instead of system-centric workload, namely data recovery. Thus, bandwidth allocation to data recovery may be sacrificed to ensure conformance to system throughput in case of tradeoffs.

In order to simulate the tradeoff scenario, the workload generator is configured to generate 80 op/s, 90 op/s, and 100 op/s. The generator applies a queue-based model, where requests that are not served are queued for later execution. The bandwidth is dynamically allocated to meet the throughput SLO for user-centric workload.

Table I
PERCENTAGE OF SLO VIOLATIONS IN SWIFT WITH/OUT BWMAN

SLO confidence interval	Percentage of SLO violation	
	With BwMan	Without BwMan
5%	19.5%	43.2%
10%	13.6%	40.6%
15%	8.5%	37.1%

Fig. 7 and Fig. 8 depict the results of our experiments conducted simultaneously in the same time frame; the x-axis shares the same timeline. The failure scenario introduced by our failure simulator is the same as in the first series of experiments (see data integrity experiment in Fig. 6).

Fig. 7 presents the achieved throughput executing user-centric workload without bandwidth management, i.e., without BwMan. In these experiments, the desired throughput starts at 80 op/s, then increases to 90 op/s at about 70 min, and then to 100 op/s at about 140 min. Results indicate high presence of SLO violations (about 37.1%) with relatively high fluctuations of achieved throughput.

Fig. 8 shows the achieved throughput in Swift with BwMan. In contrast to Swift without bandwidth management, the use of BwMan in Swift allows the service to achieve required throughput (meet SLO) most of the time (about 8.5% of violation) with relatively low fluctuations of achieved throughput.

Table I summarizes the percentage of SLO violations within three given confidence intervals (5%, 10%, and 15%) in Swift with/out bandwidth management, i.e., with/out BwMan. The results demonstrate the benefits of BwMan in reducing the SLO violations with at least a factor of 2 given a 5% interval and a factor of 4 given a 15% interval.

VI. RELATED WORK

The benefits of network bandwidth allocation and management is well understood as it allows improving performance of distributed services, effectively and efficiently meeting SLOs and, as consequence, improving end-users' experience with the services. There are different approaches to allocate and control network bandwidths, including controlling bandwidth at the network edges (e.g., of server interfaces); controlling bandwidth allocations in the network (e.g., of particular network flows in switches) using the software defined networking (SDN) approach; and a combination of those. A bandwidth manager in the SDN layer can be used to control the bandwidth allocation on a per-flow basis directly on the topology achieving the same goal as the BwMan controlling bandwidth at the network edges. Extensive work and research has been done by the community in the SDN field, such as SDN using the OpenFlow interface [17].

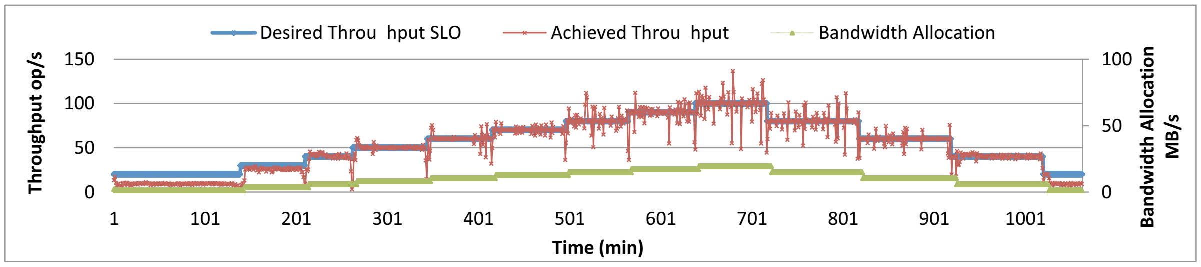


Fig. 5. Throughput under Dynamic Bandwidth Allocation using BwMan

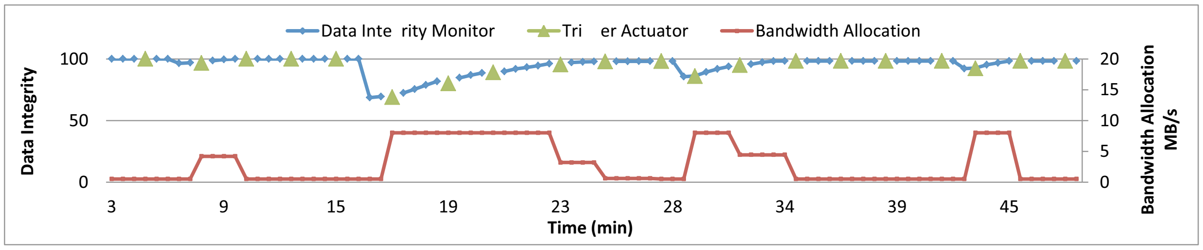


Fig. 6. Data Recovery under Dynamic Bandwidth Allocation using BwMan

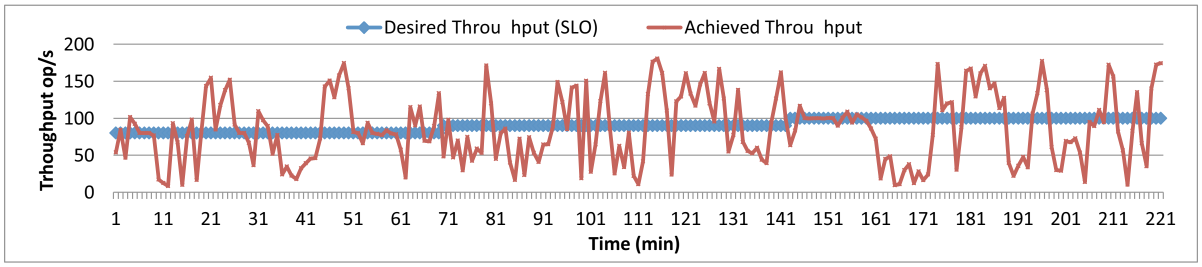


Fig. 7. Throughput of Swift without BwMan

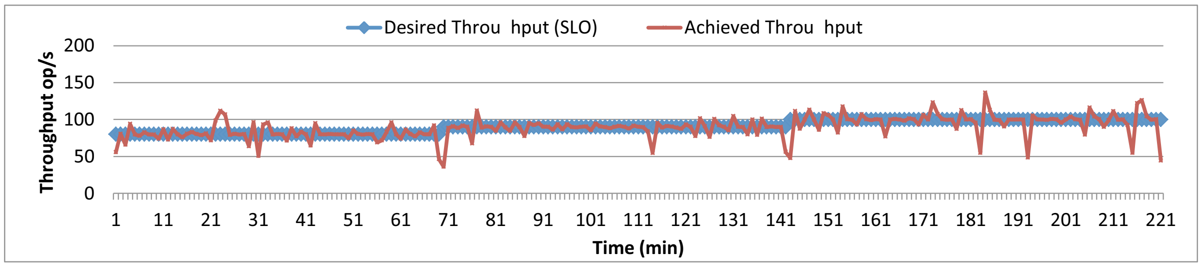


Fig. 8. Throughput of Swift with BwMan

A typical work of controlling bandwidth allocation in the network is presented in Seawall [18]. Seawall uses reconfigurable administrator-specified policies to share network bandwidth among services and enforces the bandwidth allocation by tunnelling traffic through congestion controlled, point to multipoint, edge to edge tunnels. In contrast, we propose a simpler yet effective solution. We let the controller itself dynamically decide the bandwidth quotas allocated to each services through a machine learning model. Administrator-specified policies are only used for tradeoffs when the bandwidth quota is not enough to support all the services on the same host. Using machine learning techniques for bandwidth allocation to different services allows BwMan to support the hosting of elastic services in the cloud, whose demand on the network bandwidth varies depending on the incoming workload.

A recent work of controlling the bandwidth on the edge of the network is presented in EyeQ [19]. EyeQ is implemented using virtual NICs to provide interfaces for clients to specify dedicated network bandwidth quotas to each service in a shared Cloud environment. Our work differs from EyeQ in a way that clients do not need to specify a dedicated bandwidth quota, instead, BwMan will manage the bandwidth allocation according to the desired SLO at a minimum bandwidth consumption.

The theoretical study of the tradeoffs in the network bandwidth allocation is presented in [20]. It has revealed the challenges in providing bandwidth guarantees in a Cloud environment and identified a set of properties, including min-guarantee, proportionality and high utilization to guide the design of bandwidth allocation policies.

VII. Conclusion and Future Work

We have presented the design and evaluation of BwMan, a network bandwidth manager providing model-predictive policy-based bandwidth allocation for elastic services in the Cloud. For dynamic bandwidth allocation, BwMan uses predictive models, built from statistical machine learning, to decide bandwidth quotas for each service with respect to specified SLOs and policies. Tradeoffs need to be handled among services sharing the same network resource. Specific tradeoff policies can be easily integrated in BwMan.

We have implemented and evaluated BwMan for the OpenStack Swift store. Our evaluation has shown that by controlling the bandwidth in Swift, we can assure that the network bandwidth is effectively arbitrated and allocated for user-centric and system-centric workloads according to specified SLOs and policies. Our experiments show that network bandwidth management by BwMan can reduce SLO violations in Swift by a factor of two or more.

In our future work, we will focus on possible alternative control models and methodology of controller designs for multiple Cloud-based services sharing the network infrastructure in Clouds and Cloud federations. In addition, we will investigate impact of network topology and link capacities on the network bottlenecks within or between data centers, and how to integrate controlling bandwidth on edges of the network with bandwidth allocation and with allocation in the network topology using SDN approach.

Acknowledgement

This work was supported by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the Education, Audiovisual and Culture Executive Agency of the European Commission under the FPA 2012-0030, by the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research under the contract RIT10-0043, and by the FP7 project CLOMMUNITY funded by the European Commission under EU FP7 GA number 317879.

References

[1] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proc. ICAC*, 2010.

[2] B. Trushkowsky, P. Bodík, and et al., "The scads director: scaling a distributed storage system under stringent performance requirements," in *Proc. FAST*, 2011.

[3] A. Al-Shishtawy and V. Vlassov, "ElastMan: Elasticity manager for elastic key-value stores in the cloud," in *Proc. CAC*, 2013.

[4] B. Braem, C. Blondia, and et al., "A case for research with and on community networks," *ACM SIGCOMM Computer Communication Review*, 2013.

[5] I. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.

[6] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, 2010.

[7] R. Sumbaly, J. Kreps, and et al., "Serving large-scale batch computed data with project voldemort," in *Proc. FAST*, 2012.

[8] (2013, Jun.) Openstack cloud software. [Online]. Available: http://www.openstack.org/

[9] Y. Wang, A. Nandi, and G. Agrawal, "SAGA: Array Storage as a DB with Support for Structural Aggregations," in *Proceedings of SSDBM*, Jun. 2014.

[10] P. Bodık, R. Griffith, and et al., "Statistical machine learning makes automatic control practical for internet datacenters," in *Proc. HotCloud*, 2009.

[11] IBM Corp., *An architectural blueprint for autonomic computing*. IBM Corp., 2004.

[12] K. Pepple, *Deploying OpenStack*. O'Reilly Media, 2011.

[13] (2013, Jun.) Openstack swift's documentation. [Online]. Available: http://docs.openstack.org/developer/swift/

[14] B. F. Cooper, A. Silberstein, and et al., "Benchmarking cloud serving systems with ycsb," in *Proc. SOCC*, 2010.

[15] (2013, Jun.) Scaling media storage at wikimedia with swift. [Online]. Available: http://blog.wikimedia.org/2012/02/09/scaling-media-storage-at-wikimedia-with-swift/

[16] S. Hemminger *et al.*, "Network emulation with netem," in *Linux Conf Au*. Citeseer, 2005.

[17] N. McKeown, T. Anderson, and et al., "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[18] A. Shieh, S. Kandula, and et al., "Sharing the data center network," in *Proc. NSDI*, 2011.

[19] V. Jeyakumar, M. Alizadeh, and et al., "Eyeq: Practical network performance isolation at the edge," in *Proc. NSDI*, 2013.

[20] L. Popa, G. Kumar, and et al., "Faircloud: Sharing the network in cloud computing," in *Proc. SIGCOMM*, 2012.