SELDAC: Software-Defined Storage based Efficient Load Distribution and Auto Scaling in Cloud Data Centers

Renuga Kanaga and Khin Mi Mi Aung
Data Center Technology Division, A*STAR Data Storage Institute, Singapore.
Email: {Renuga_k, Mi_Mi_AUNG}@dsi.a-star.edu.sg

Abstract —Cloud computing allows the users to access the shared pool of configurable resources such as compute, network and storage on-demand basis from the cloud Data Centers. Cloud storage provides the storage resources to the cloud users over the network. The massive growth of cloud data due to the applications such as video streaming, big data processing, social networking, banking and scientific applications fueled the storage demand. An increase in storage demand resulted in the expansion of the cloud storage with additional storage nodes. The expansion of the cloud storage necessitates the migration of data across the storage nodes to keep them balanced. Efficient management and migration of tera bytes or peta bytes of such cloud data depends not only on the storage availability but also on the networking cost which is proportional to the distance and the bandwidth requirement. In this scenario, software defined environment opens up the new opportunities to orchestrate the network and storage resources control strategies to provide an effective solution. In this paper, we present the software-defined orchestration framework for efficient load distribution and auto scaling mechanism and shows that it improves the overall efficiency in terms of latency and retrieval cost. We demonstrate the effectiveness and the efficiency of the proposed mechanism using simulations.

Index Terms—Software-defined storage, software-defined networking, load distribution, orchestration, cloud data center

I. Introduction

Cloud Data Centers consisting of tens of thousands of servers with huge computing, storage, and network bandwidth resources are becoming common. Virtualization technology enables the cloud users to hire Virtual Machines (VMs) with the required computing, storage and bandwidth resources on-demand basis from the cloud data centers instead of owning them. Cloud storage is the model of data storage in which data is stored on remote servers, maintained and managed by the cloud storage service providers. It is made available to the cloud users over the network. Cloud storage is growing at the phenomenal rate due to big data, social networks and mobile devices. According to Gartner's report [1], the consumer digital storage will grow to 4.1 zettabytes in 2016, with 36 percent of this storage in the cloud.

Manuscript received June 25, 2015; revised December 8, 2015.
This work was supported by the the Singapore A*STAR TSRP project 1122804004.
Corresponding author email: Renuga_K@dsi.a-star.edu.sg
doi:10.12720/jcm.10.12.1020-1026

The rising rates of cloud adoption resulted in the expansion of cloud storage with additional nodes. As the cloud services continue to grow, the diverse I/O workloads can cause significant imbalance across the servers causing higher delays to the end users. As a result, tera or peta bytes of cloud data are shuffled per day across the server nodes in the cloud storage to balance the load. Similarly, during the expansion of the cloud storage with additional storage nodes, data can be migrated across them to keep them balanced. This results in high reconfiguration cost of data migration due to bandwidth oversubscription in the Data Center network.

In cloud computing, load balancing is required to distribute the dynamic workload evenly across all the nodes to achieve higher user satisfaction, high resource utilization ratio, short response time and high throughput. Proper load balancing helps to achieve minimal resource consumption, scalability, fail-over and oversubscription. In cloud Data Centers, efficient migration of data not only depends on the storage availability but also on the network bandwidth. Traditional storage systems use dedicated storage and network for specific applications to guarantee latency, throughput and Input/output Operations Per Second (IOPS). But, these resources are not evenly utilized. There is a need for efficient load distribution mechanism considering network knowledge along with the storage capacity.

The orchestration of both storage and network devices in cloud computing with the overall resources controlled by the common cloud controller will lead to a promising research direction. Cloud computing started with the virtualization of computing resources, followed by recent advances and rapid innovations in Software-Defined Networking (SDN) [2], [3], which aim to virtualize networking resources and separate the control plane from the data plane. Apart from that, the current trend towards Software-Defined Storage (SDS) aims to virtualize the storage resources and enforces the policies by dynamically, adjusting the storage resources across multiple cloud users.

Our design called "software-defined Storage based Efficient Load Distribution and Auto Scaling (SELDAC)" is considered as an instance of software defined storage. It includes the load balancing module that act as the "software defining" part of the storage system and provides the latency/bandwidth, IOPS requirements of the system. It also contains the traffic

engineering module which will choose an efficient path to migrate the data to reach the destination server nodes. The main contribution of this paper is the design of the orchestrated resource control framework which has several attractive features: (1) Balanced and Efficient load distribution in a cloud storage environment. (2) Redistributing as few objects as possible in the event of storage node addition/failure (auto scale) under varying traffic conditions. (3) There is no need to maintain the central mapping table which will make the search performance more efficient and save space. It improves the overall efficiency of the system in terms of bandwidth and throughput.

The rest of this paper is organized as follows: Section II gives a review of the related work. Section III introduces the proposed orchestrated framework for load balancing in cloud. Section IV describes the performance analysis of the proposed framework. Finally concluding remarks are made in Section V.

II. Back Ground and Related Work

In this section, we first briefly describe about the software- defined networking concept and openFlow protocol and then review previous load balancing algorithms related to this work and discuss their merits and limitations.

A. Software Defined Networking

Recently, Software-Defined Networking (SDN) architectures in which the control plane is decoupled from data plane have become popular as users can intelligently control routing and resource usage mechanism. SDN concept has been introduced by academia [2], [3] and now become the industry standard [4]. By employing a central server, which uses the knowledge of switch routing information and network status, SDN manages routing algorithms separately while keeping data flows running on original network paths. Hence, SDN provides much easier modification to switch algorithms, as well as faster control over network state changes. Thus, it provides agile traffic control and management. OpenFlow, proposed by McKeown *et al*. [3], is a communication interface between SDN controllers and network devices considered as the most popular enabler for SDN as shown in Fig. 1.

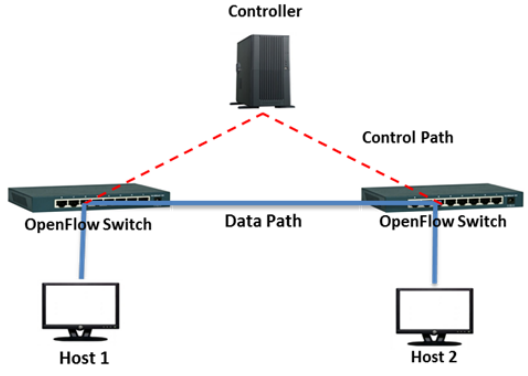

Fig. 1. SDN architecture.

OpenFlow is an open standard, which enable users to implement experimental routing protocols via software. Every OpenFlow switch maintains a flow table with entries for the flows. An entry in the table specifies a few packet fields and associated action (e.g. forward, drop, modify header) to be carried out in the event of a matching with the incoming packet. In the event when a switch is unable to find a match in the flow table, the packet is forwarded to the controller to make the routing decisions. After deciding how to route the new flow, the controller then installs a new flow entry at the required switches, so that the desired actions can be performed on the new flow.

B. Load Balancing Algorithms

Load balancing is required in the cloud computing environment to achieve short response time and high throughput. Various load balancing approaches have been proposed. Commonly used static algorithms are Round Robin (RR) and Weighted Round Robin (WRR). The static methods do not consider run time conditions. Several algorithms have been proposed with different goals like energy-saving, time-saving, network-bandwidth utilization, CPU load balancing etc.

In [4], a data placement algorithm in cloud storage system has been proposed. It selects the storage server based on the storage capacity. However; it does not consider the network utilization for selecting a storage server. In [5], a virtual cluster management has been proposed to guarantee the bandwidth requirement. In [6], network I/O load balancing strategy has been proposed to efficiently utilize the network resources.

In [7], a data placement algorithm in cloud storage system has been proposed. It selects the storage server based on the storage capacity. However; it does not consider the network utilization for selecting the storage server.

Our work differs from the above work by considering the network knowledge along with the storage capacity for efficient load distribution in the cloud Data Center.

III. System Architecture

A. SELDAC Functional Components

Fig. 2 shows the functional components of our proposed SELDAC framework. The SELDAC framework consists of three main components: Access tier, Cloud orchestrator, storage servers connected with storage pools.

- Access tier: An access tier consists of several clients or applications that are connected to a server which acts as an entry point for the frame work. The host system is serviced by cloud orchestrator, which is connected to one or more storage servers.
- Cloud orchestrator: This is the centralised controller (control plane) which plays a vital role in orchestrating network and storage control strategies. The function of this orchestrator is to distribute the

requests from the cloud users to the appropriate storage servers in a balanced manner. We briefly describe below the functions of each module in cloud orchestrator as shown in Fig. 3.

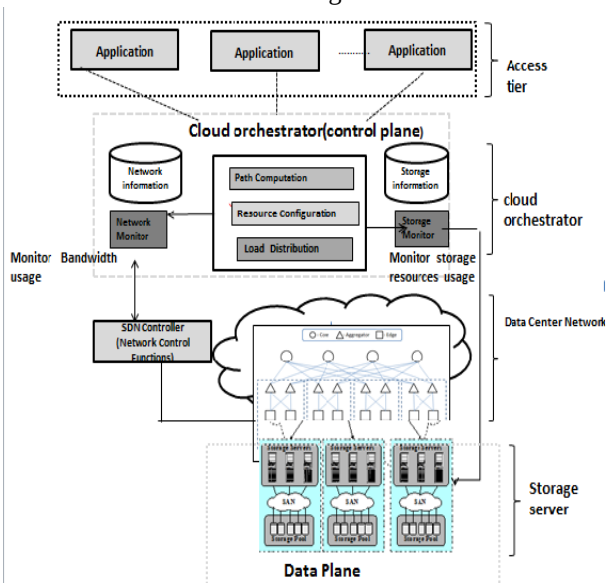

Fig. 2. SELDAC framework.

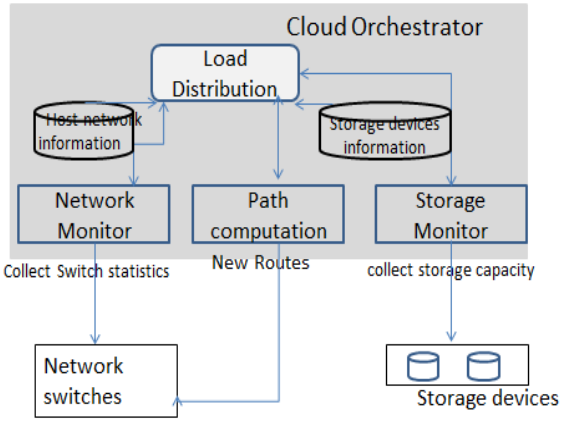

Fig. 3. Cloud orchestrator functional modules.

- *Resource Configuration Module* is responsible for creating the volumes based on the replication policies on the storage systems and allocating these volumes to the virtual machines (VMs).
- The *storage Monitor component* which will monitor the number of Read and Write requests, the latencies involved and the usage of the storage volume. Based on the Storage Monitor values, the *load distribution* module selects the storage servers for request distribution.
- The *Storage information* module keeps track of the information about the different types of storage devices, volume information and the available storage capacity.
- The *Network Monitor component* which leverages on Software-Defined Networking (SDN) will monitor the bandwidth availability across the network links. The responsibility of this module is to collects, stores, and queries statistics from all OpenFlow switches that will be used by the *Path computation* module to compute and compare the load on various links. These statistics are polled at fixed intervals and stored in cloud orchestrator as snapshot objects. Every snapshot will be assigned a sequence number, which increments by one after each interval. The two most recent snapshots will be maintained in memory at any instant of time.
- *Network Information module* keeps track of all the servers detected on the network. It stores the MAC address, IP address of the interface of the servers connected to the network, the OpenFlow switch id and port number each of the servers is connected to.
- *Path computation* is also responsible for finding the optimal path with enough available bandwidth for routing the migrated data when a new storage node is added in the cloud storage. As the current cloud Data Centers adopting the multi-rooted tree architecture [8], [9] with multiple paths between the servers, it is required to find the optimal routing path with enough available bandwidth among the multiple paths available between two end hosts. It computes the least-loaded shortest candidate paths between any pair of end hosts based on the statistics collected from the switches by the Network Monitor module.
- *Load Distribution module* makes the decision on the selection of the storage servers and the optimized network paths between the storage servers based on the monitored values. Our algorithm named *weighted bit-window based load distribution* is used in this module to distribute the load across the storage servers in a balanced manner, which is described in next section.

- Storage server: Each storage server (Data plane) can be connected to one or more heterogeneous storage devices. Each storage server has the necessary storage driver to communicate with the storage devices. The storage servers send "keep alive" messages to the orchestrator at frequent intervals so that the orchestrator will know how many servers are available for the client request. They also exchange the change in the storage capacity information of the storage pools that are attached to. The storage servers are grouped according to the storage capacity. The purpose of this is to assign the large request to the server with large capacity and so on.

B. The Proposed Load Distribution Algorithm

As we stated earlier, load distribution is an important problem that need to be addressed in the growing cloud storage environment. We developed a new algorithm named *weighted bit-window based load distribution with routing* works in two phases. For example, a storage server with more capacity tends to have large number of objects than a storage server with low capacity. This helps to reduce access time and congestion.

In Phase I, when an orchestrator receives a request from the access tier, first it attempts to find a suitable group of storage servers with varying storage capacity to host a request based on the information obtained from storage monitor. We consider the number of active storage servers and the storage resource usage within the servers are the main factors in this phase. Our *weighted bit-window* algorithm then distributes the load across the servers in a balanced manner.

In Phase II, the impact of the load distribution on network traffic with the assumption of single shortest path routing is considered. We note that the network congestion is largely influenced by the data migration when a new storage server is added during the cloud storage expansion. Based on the path load statistics obtained from *network monitor* component, the least loaded path which is the path with the minimum load among all the candidate paths is selected for data migration in this phase. The cost of the path is calculated as the sum of the utilization of the links along the path. If there is more than one path with the same minimum cost, the path with maximum bandwidth is preferred. We reduce the network congestion by taking the load into the consideration.

1) Storage server grouping and load distribution

In phase 1, the incoming requests are mapped to the appropriate server group using our *weighted bit-window load distribution* method. Once the storage server is chosen, the request will be placed in that server. In this section, we briefly explain the working of *weighted bit-window load distribution* algorithm and explain how it supports operations like server node addition during the cloud storage expansion and storage node deletion during node failure.

Let n_1 be the number of existing storage servers with index from 0 to $n_1 - 1$. Let X be the total number of requests. We group the servers according to the available resource capacity 'c'. These servers are referred as active servers. The server indices and the corresponding server IP addresses are maintained in a table at the orchestrator. The cloud client requests are translated into an h-bit identifier using a base hash function SHA-1. We call the identifier as request_id. The hashed *h*-bit request_id is a fixed size string of 'h' bits. The *h*-bit identifier is divided into number of bit windows of size $k = \lceil \log_2 (n_1) \rceil$. While the server_ids 0 through $n_1 - 1$ are said to be active servers, the remaining server_ids n_1 through $2^k - 1$ are called inactive server_ids that correspond to inactive storage servers. The size of the bit window depends on the number of the storage servers and it varies according to the addition or deletion of storage servers. The bit windows are labeled starting from 0 from the right end, denoted as BW_0, $BW_1 \ldots BW_{v-1}$, where *v* is the maximum number of windows used. The bit-windows of a request_id are examined one by one starting from BW_0. If the value of the BW_0 is valid then the request_id is placed in the storage server whose index is equal to that value. If the value of BW_0 is not valid, then we continue to examine BW_1, BW_2…… BW_{v-1} until an active storage server is found. If no valid storage server index is found among the *v* windows, the request_id is placed at the storage server whose index is obtained by inverting the most significant bit (MSB) of BW_0.

To count for varying storage capacity, we use logical server ids. A server with capacity 'c' will have 1 id whereas a server with capacity '2c' will have two ids and so on. This method does not require any table to maintain the relation between the request_id and the server index and thus save the memory usage for storing the table. As the demand grows, the system will be scaled with a new server automatically. When a new server node is added, our algorithm migrates less number of objects across other server nodes to keep them balanced.

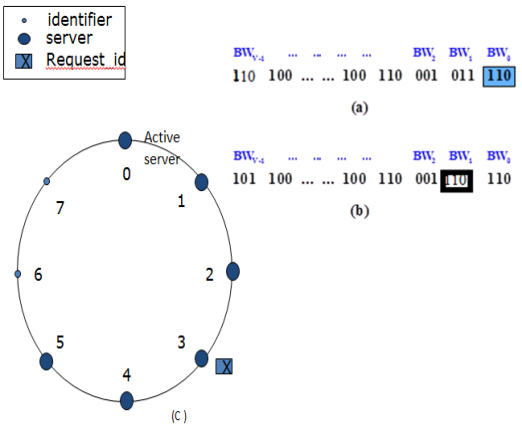

Fig. 4. Illustration of Bit –Window algorithm (N=6). (a) BW0 is searched - invalid. (b) BW1 is searched- valid. (c) Assigned the request_id to server node 3.

Fig. 4 illustrates the weighted *bit-window load balancing* algorithm with the available number of servers N= 6. The server_ids 0 thru 5 are now active servers as shown in Fig. 3. The request_id generated by SHA1 is 110100….100110001011110 as shown in Fig. 4 (a). The size of the Bit-Window is given by k=3. The value of bit window BW_0 is 6 which is not a valid node index. So the search continues with BW_1 as shown in Fig. 4 (b). The value of the BW_1 is 3 which is a valid index and hence the request_id is assigned to node 3 as shown in Fig. 4(c). Let us now analyze how the objects are evenly distributed in the case new server node addion/deletion.

Server node addition: As the demand grows, cloud storage is expanded by adding more sever nodes. When new node joins (assigned index n_1) and this new node addition does not change the bit window size, then $c * \frac{X}{n_1 + 1}$ requests will be migrated to the new node labelled n_1. If the new server node addition resulted in the expansion of the bit-window, $c * \frac{X}{2n_1}$ requests will be migrated.

Server node deletion: When a node with an index $n_1 - 1$ is deleted due to node failure, and this deletion does not change the size of the bit-window, then c* $\frac{X}{n_1}$ requests will be migrated to other server nodes.

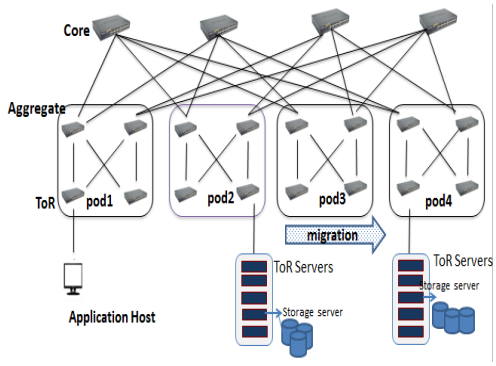

Fig. 5. Cloud data center network topology

2) Path selection and routing

We consider a multi-rooted Data Center network as shown in Fig. 5. The network is comprised of switches interconnected in three layers- Top of Rack (ToR), aggregate, and core- as shown in the Fig. 5. When a new storage server is added to the Cloud Data Center due to the increase in demand, the migration operation is initiated to migrate the requests from the existing storage server to the newly added storage server as shown in the Fig. 5. We develop an adaptive routing method which selects appropriate routes for the incoming request based on the current network state. Based on the load statistics at the switches and based on the loads on the paths , the minimum cost path is selected. The path computation module as described earlier chooses the least-loaded path which is the path with minimum load among all candidate paths. The shortest-hop paths are considered as candidate paths. We note that there exist many shortest-hop paths in a Data Center network. The cost of a path is calculated as the sum of the utilization of the links along the path. If there is more than one path with the same minimum cost, the path with maximum bandwidth is preferred. Each $link_k$ has a link load of L_k bps computed from the statistics, and link capacity C_k bps. Mathematically, the cost of the link is given as,

$$route\_cost_k = \sum_{\text{for all } k} (L_k/C_k)$$

After selecting the minimal cost path, the *Load distribution* module migrates the requests to the newly joined server node.

IV. Performance Study

In this section, we present the simulation results of the *weighted bit-window based load distribution* algorithm. The *ns3 simulator* [10], [11] is used for simulation.

A. Simulation Setup

We considered storage servers of varying capacity and objects with varying sizes. To show that the algorithm can meet its design goals, we considered two scenarios. In the first scenario, we want to show that objects distribution and average retrieval latency is fairly even although servers may be of varying capacity. In this scenario, we create 24 server nodes, a number somewhere in between two bit-windows, and 1000 objects. The server nodes have capacity varying from 50 to 470 GB, while the objects have size between 4k and 10GB.

In the second scenario, we first created 31 server nodes and 50,000 objects. The capacity of the servers ranged between 6000 to 14,000 GB, while the objects size remained between 1 and 10GB. We then added another node to determine the migration of objects when there is no change to the bit-window size.

B. Load Distribution

We use load distribution as the performance metric to evaluate the effectiveness of our algorithm. Fig. 6 shows the load distribution in the first scenario with 24 server nodes of varying capacity. It can be observed that there is an even distribution of objects according to varying server capacity. The servers with lower capacity tend to have lesser number of objects and the servers with higher capacity tend to have more number of objects.

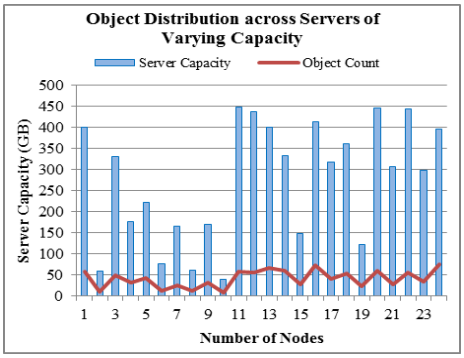

Fig. 6. Load distribution across servers of varying capacity.

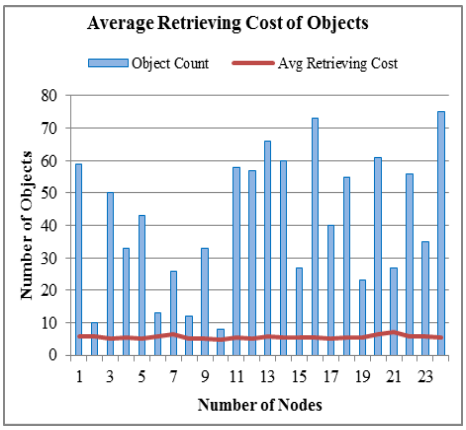

Fig. 7. Average retrieval cost

C. Average Retrieval Cost

We evaluate the performance of our algorithm in terms of average retrieval cost. In Fig. 7, we plot the average retrieving cost for the objects across the server nodes. We define average retrieval cost as the sum of the storage access time and latency. We can see that the average

retrieving cost curve remains flat across the server nodes despite the variation in the number of objects on each node. This shows that the weighted data placement algorithm is capable of distributing objects across servers of varying capacity while maintaining similar retrieval coast across all servers.

D. Load Migration

Fig. 8 to Fig. 9 show the distribution and migration of objects when server nodes are added to the environment.

In Fig. 8, we plot the number of objects that are distributed across 31 server nodes to show the initial distribution. The objects distribution across the nodes varies with the server nodes capacity. As described previously, the servers with lower capacity tend to have lesser number of objects and the servers with higher capacity tend to have more number of objects.

We consider the scenario of adding the new server to the network. When a new server node is added and the number of server nodes increases to 32, some of the objects will be migrated from existing nodes to server node 32. Fig. 9 shows the number of objects that are migrated out of each of the previous 31 nodes and moved to the newly added node. We can see that the number of objects that needs to be migrated from the existing nodes is quite small. When there is no change in bit-windows size, the number of objects that needs to be migrated whenever a new node is added is fairly insignificant.

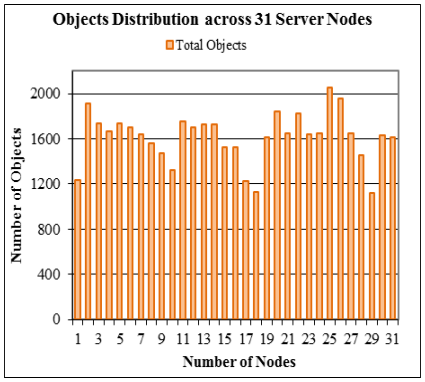

Fig. 8. Load distribution across 31 servers

E. Impact of Window Size

We evaluate the impact of the bit-window size on load distribution. From Fig. 9, we understand that when there is no change in bit-windows size during new server addition, the number of objects that needs to be migrated is small. To study the impact of the new server node addition that causes the change in bit window size, another node is added to the system subsequently bringing the total number of server nodes to 33. Fig. 10 shows the distribution and migration of objects when the 33rd server node is added which causes a change in the bit-window size. The change in the bit-window size means that objects that were previously hashed into a node may now be hashed into a different node. To reduce the number of migration, if an object is currently on a node that is adjacent to the new node that it is supposed to be hashed into; the object will not be migrated and can remain on the current node.

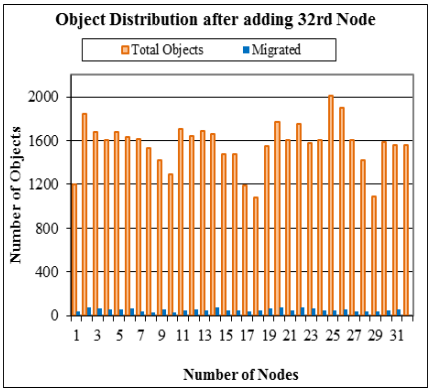

Fig. 9. Distribution and migration of objects when a server node is added without bit-window size change

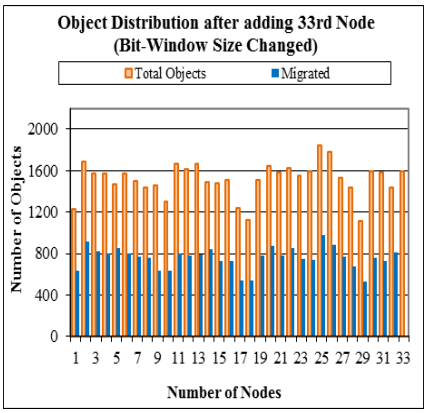

Fig. 10. Distribution and migration of objects when a server node is added causing bit-window size change.

V. Conclusions

In the paper, we presented the design, implementation and the evaluation of *SELDAC*, a software-defined orchestration framework that can adapt to the changes in the cloud storage expansion. We developed an efficient weighted bit-window based load distribution algorithm in cloud Data center environment that ensures even distribution of objects in proportion to storage server resource capacity and network bandwidth availability. It supports storage server node additions and deletions with low number of object migrations. We studied the performance of the proposed mechanism through simulation results.

Acknowledgment

This work was supported mainly by the Singapore A*STAR TSRP project 1122804004.

References

[1] Gartner Says That Consumers Will Store More Than a Third of Their Digital Content in the Cloud by 2016. (2012). [Online]. Available: http://www.gartner.com/newsroom/id/2060215

[2] N. McKeown, T. Anderson, B. Hari, G. Parulkar, *et al.*, "OpenFlow: Enabling innovation in campus networks," in *Proc. SIGCOMM*, 2008.

[3] A. Tavakoli, M. Casado, and T. Koponen, "Applying NOX to the datacenter," in *Proc. 8th ACM Workshop on Hot Topics in Networks*, 2009.
[4] Open Networking Foundation. (2013). Open Networking Foundation. [Online]. Available: https://www:opennetworking:org/
[5] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, *et al*., "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th International Conference on Emerging Networking Experiments and Technologies*, USA, 2010.
[6] W. Song, F. Wang, X. H. Sri, H, Lin, and Z. W. Wang, "Network I/O load based virtual machine placement algorithm in HPC Cloud," *China Science: Information Science*, vol. 42, pp. 290-302, 2012.
[7] J. Myint and T. Niang, "A data placement algorithm with binary weighted tree on PC cluster-based cloud storage system," in *Proc. IEEE International Conference on Cloud and Service Computing*, 2011.
[8] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, *et al*., "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM Conference on Data Communication*, New York, NY, USA, 2009, pp. 39–50.
[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, *et al*., "VL2: A scalable and flexible datacenter network," in *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, New York, NY, USA, 2009, pp. 51–62.
[10] T. R. Henderson, M. Lacage, and G. F. Riley, "Network simulations with the ns-3 simulator," in *Proc. ACM SIGCOMM*, 2008.
[11] URL: The ns-3 Network Simulator. [Online]. Available: http://www.nsnam.org/

Renuga Kanagevelu received a PhD degree of Computer Engineering from Nanyang Technological University, in 2015. She is currently a Senior Research Engineer with A*STAR, Data Storage Institute, Singapore. Her research interests include Software- defined networks, Data Center and Network Storage Technologies.

Khin M. M. Aung received a PhD degree of Computer Engineering from Korea Aerospace University, in 2006. She is currently a Scientist with A*STAR, Data Storage Institute, Singapore. Her research interests include Data Center and Network Storage Technologies.

