SCDA: SLA-aware Cloud Datacenter Architecture for Efficient Content Storage and Retrieval

Debessay Fesehaye
Department of Computer Science
University of Illinois at Urbana-Champaign
dkassa2@illinois.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
klara@illinois.edu

ABSTRACT

With the fast growth of (online) content and the need for high quality content services, cloud data centers are increasingly becoming the preferred places to store data and retrieve it from. With a highly variable network traffic and limited resources, efficient server selection and data transfer rate allocation mechanisms become necessary. However, current approaches rely on random server selection schemes and inefficient data transmission rate control mechanisms.

In this paper we present SCDA, an efficient server selection, resource allocation and enforcement mechanism with many salient features. SCDA has prioritized rate allocation mechanism to satisfy different service level agreements (SLA)s on throughput and delays. The allocation scheme can achieve max/min fairness. SCDA has a mechanism to detect and hence mitigate SLA violation in realtime.

We have implemented SCDA in the NS2 simulator. Extensive experimental results confirm some of the design goals of SCDA to obtain a lower content transfer time and a higher throughput. The design of SCDA can achieve a content transfer time which is about 50% lower than the existing schemes and a throughput which is higher than existing approaches by upto than 60%.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]; C.2.1 [**Network Architecture and Design**]: Network communications

General Terms

Computer Systems Organization

Keywords

Cloud architecture, max/min fairness, fast transfer, SLA-aware, prioritized rate.

1. INTRODUCTION

Over the past few years there has been an exponential growth of online content and such content generation is expected to grow at 40-35% a year [19]. Users of (multimedia) content have diverse Quality of Service (QoS) requirements. Based on their QoS specifications, content users make service level agreements (SLA) with content and/or network providers. Satisfying QoS requirements with dynamic network and server loads and with limited resource capacities (link bandwidth, server storage, processing, energy) is challenging. The main problem lies in knowing *where* to *store* contents and *retrieve* them from, by *utilizing* available resources and *detecting SLA* violations.

Addressing this problem involves answering a series of questions. Some of the main questions are the following.

1. *Which* server among a group of servers at different locations is the *least loaded* and power efficient?
2. At what *rate* should content be *written* to server and *retrieved* from it in order to satisfy SLA (lower delay and higher throughput)?
3. How can max/min fairness be ensured where available resource is utilized as long as there is demand for it?
4. How can *SLA violation* be detected in realtime (milliseconds interval) and be mitigated?
5. How can such data transfer rate allocations to different users be enforced without changing routers, switches and the TCP/IP stack?

Existing attempts to solve these problems broadly fall into two categories. The first one is using large content distribution networks (CDNs) such as Akamai. Such CDNs use a large number of edge servers distributed in vast Internet locations. As explained in [18, 28] such schemes select a server for client request based on proximity and latency. Server selection is not based on best content transfer rates and lowest delays. Besides, the scalability of distribution and maintenance of edge-servers scattered in many Internet locations all over the world is costly. The work presented in [28] shows that significant consolidation of Akamai's platform into fewer large data centers is possible without degrading performance.

The second but dominant and emerging content storage and retrieval approach is using cloud data centers. There have been numerous data center architectures [12, 2] to address the above-mentioned challenges. However such architectures do not use an efficient mechanism to select the best servers in the data center. They use random switch (server) selection strategies. They also rely on the transmission control protocol (TCP) [14] to control the rates of the senders. TCP is known to have higher average file completion time (AFCT) than necessary as discussed in [6]. Besides, such approaches are restricted to specific structure of datacenter network interconnect.

In this paper we present the design of SLA-aware Cloud Datacenter Architecture (SCDA) for efficient content storage and retrieval. SCDA among other things addresses the

above five questions. The design of SCDA has two main features. The first feature enables SCDA to use multiple name node servers (NNS) using a light weight front-end server (FES) which forwards requests to the name nodes (NNS). This approach solves the weakness of current state-of-the-art cloud-computing architectures (file systems) [11, 26]. In such systems only a single NNS, which can potentially be a bottleneck resource and single point of failure, is used.

The second main feature of SCDA is its ability to avoid congestion and select the less loaded servers using a cross-layer (transport and network layers) concept unlike current well known schemes [11, 26, 12, 2] which rely on TCP and random server selection. SCDA also uses resource monitors (RM) and resource allocators (RA) to do fine grained resource allocation and load balancing. The roles of these SCDA components can be extended to constantly monitor the performance of the cloud against malicious attacks or failures. All the aggregated and monitored traffic metrics can be offloaded to an external server for off-line diagnosis, analysis and data mining of the distributed system.

The data center (cloud) resource *allocation and enforcement* mechanism of SCDA using RMs and RAs is *stateless* and does not need modifications to routers/switches or the TCP/IP stack. The scheme can *detect violation in service level agreements (SLA)* and can help cloud (data center) administrators (admins) to automatically add more resources to resolve detected SLA violations.

The SCDA resource (bandwidth) allocation mechanism is *max-min fair* in that any link bandwidth unused by some flows (bottlenecked elsewhere) can be used by flows which need it. This is a very useful quality any resource allocation mechanism needs to achieve. We also show how SCDA can do more *power aware server selection* as there is heterogeneity in power consumptions by different servers. This heterogeneity can be due to server's location in a rack, due to server age and specifications or due to other compute intensive or background tasks the servers perform. The RM and RA of SCDA are *software components* and can be consolidated into a few powerful servers close to each other to minimize communication overheads and latencies.

The rest of this paper is organized in such a way that we first present the network and content models used in the design of SCDA in section 2. In section 3 we discuss the SCDA nodes and software components. Section 4 discusses simple formulas to obtain the rate metric at which clients share resources (link bandwidth, CPU, storage). The steps used in the SCDA algorithm are presented in section 5. In the subsequent sections 6, 7 and 8 we discuss each of these steps. These steps are the rate allocation mechanism at a global and different levels of the data center tree, the server selection mechanism using the allocated rate metrics and ways to serve both outside client and internal cloud requests respectively. In section 9 we give a brief description of how SCDA can be applied to different cloud network topologies. In section 10 we present experimental results comparing the performance of SCDA against existing schemes which use random server selection and TCP (RandTCP). A related work discussion is given in section 11. We finally give summary of the paper in section 12.

2. NETWORK AND CONTENT MODEL

In this section we present description of the network and content models for which we design and analyze SCDA.

2.1 Network Model

Under SCDA, the network consists of client nodes connected to cloud data-center servers via links. The clients are connected to the cloud via dedicated tunnels as part of the service level agreement (SLA) or over the Internet. This is usually done using protocols such as the OSPFv3 as a Provider Edge to Customer Edge (PE-CE) Routing Protocol [25]. The cloud data-center servers are connected with each other via high speed links typically using a hierarchical topology similar to the one shown in figure 1. SCDA can be easily extended to work with other data-center network topologies such as [12, 23] as briefly described in section 9.

2.2 Content Model

Contents stored in cloud data centers can be classified into *active* and *passive*. A *passive content* is content which is not frequently read or written to, after its initial storage in the cloud. An *active content* on the other hand is a content which is frequently accessed due to read or write actions. The read and write frequencies to distinguish passive content from active content in our design are user defined parameters. Active contents can further be classified into *high write and high read* (HWHR), *low write and high read* (LWHR) and *high write and low read* (HWLR). Following this classification, the passive contents can be considered as *low write and low read* (LWLR). Considering an email application for instance, sent emails and attachments can be considered passive for the sender. Chatting (both text and video/audio) can be considered active content. A file which is edited by collaborative users can be considered an active content. Database tables which are constantly updated can be considered active contents. Some hot news can be considered an active content.

As shown in [16] for HDFS logs in one of Yahoo!âĂŹs enterprise Hadoop clusters, about 60% of content was not accessed at all in a 20 day window. Hence SCDA takes content diversity into account when selecting storage or replica servers for each request to store or retrieve content. SCDA uses different server selection strategies for the active and passive contents.

3. SCDA COMPONENTS

The architecture of SCDA [10] is presented in figure 1. As shown in the figure, the SCDA architecture assumes a tree structure of the data center networks for cloud computing as is the case with most data center networks today. Our SCDA scheme also works with other cloud and data center network topologies. The solid lines in figure 1 show the physical cable connections. The arrows show logical control flow communications between SCDA components. Like existing popular large scale distributed file systems [11, 26], SCDA consists of a network of block servers (BS). Unlike GFS and HDFS, SCDA uses a light weight front end server (FES) and more than one name node server (NNS). This enables SCDA to solve the potential problems of GFS and SCDA in being bottlenecked at the single NNS. SCDA also achieves its efficient resource allocation and load balancing schemes and energy efficiency using rate monitors and rate allocators. We next discuss the nodes and the resource monitors and allocators of SCDA.

3.1 Nodes

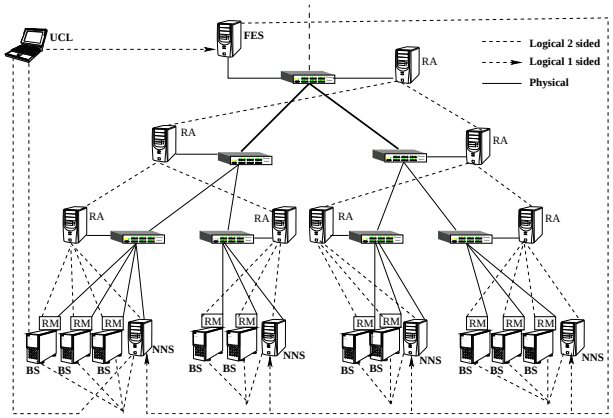

Figure 1: SCDA Architecture

The nodes in SCDA consist of the front end server (FES) which receives external requests to and from the local cloud and forwards them to the respective name node server (NNS). Each NNS keeps metadata information, for example, which block of data is stored in which block server (BS). Each BS stores data blocks assigned to it by the NNS. To help balance load among all NNS, the FES may also be assisted by the NNS to forward requests to other NNS. The UCL node is a user client which requests cloud services. The functionality of FES can be moved to the UCLs or to the NNS. FES agents associated with the UCL clients can forward the client requests to the corresponding NNS. When an FES agent is associated with the NNS, a UCL can connect to any of the NNSs. If the hashing function maps the UCL request to the receiving NNS, the NNS serves the request. Otherwise the NNS hashes the request and forwards it to the corresponding NNS. Multiple FES with different IP address for different regions can also be employed. The DNS then chooses the nearest matching FES.

3.2 Resource Monitors and Allocators

The resource monitor (RM) of SCDA is a software component responsible for monitoring and sending resource load information from the BS to the resource allocators (RA). The RAs on the other hand gather resource load information from each BS via the RMs and other information from the switches and calculate SCDA rate allocation metrics at each level of the tree.

4. SCDA RATE METRIC

To define the rate metric, we first present the following notations.

For each SCDA parameter $X \in \{R, C, Q, \hat{N}, N, n^j, R^j\}$, we use the notation

$$X_{d,u} = \begin{cases} X_d & \text{if } X \text{ is a downlink parameter,} \\ X_u & \text{if } X \text{ is a uplink parameter.} \end{cases} \quad (1)$$

We next give short descriptions of the SCDA parameters.

Given the above SCDA parameters, each RA and RM calculate the rates $R_d(t), R_u(t)$ of the down (d) and up (u) links associated with their local switches as follows:

Variables	Description
$C_{d,u}$	Link capacity in bits/sec
τ	Control interval in sec
$Q_{d,u}(t)$	Link queue size from the current interval (round) in bits
$R_{d,u}(t)$	Link rate allocation of the current interval in bits/sec
$N_{d,u}(t)$	Number of flows in the link during the current round
$\hat{N}_{d,u}(t)$	Effective number of flows in the link for the current round
$R^j_{d,u}(t)$	Rate of flow j for the current round in bits/sec
$S_{d,u}(t)$	Sum of flow bottleneck rates in current round in bits/sec
$\Lambda_{d,u}(t)$	Total current arrival rate to the link in bits/sec
$L_{d,u}(t)$	Total number of bits at a link in the current interval
$\wp^j_{d,u}$	Priority weight of flow (stream or chunk) j
$M^j_{d,u}$	Minimum rate requirement of content flow j
α, β	Stability parameters

Table 1: SCDA Parameters

The down-link (d) and up-link (u) rates

$$R_{d,u}(t) = \frac{\alpha C_{d,u} - \frac{\beta Q_{d,u}(t-\tau)}{d}}{\hat{N}_{d,u}(t-\tau)} \quad (2)$$

where

$$\hat{N}_{d,u}(t-\tau) = \frac{S_{d,u}(t)}{R_{d,u}(t-\tau)}, \quad (3)$$

$$S_{d,u}(t) = \sum_{j}^{N_{d,u}(t-\tau)} R^j_{d,u}(t) \quad (4)$$

and

$R^j_{d,u}(t) = min\left(R^j_{send,other}(t), R_{e2e}, R^j_{recv,other}(t)\right)$. Here, the R_{e2e} is the end-to-end link rate of flow j obtained using max/min algorithm discussed in section 6.1 below. The $R^j_{send,other}(t)$ and $R^j_{recv,other}(t)$ are the flow rates at the sender and receiver sides of the tree due to other bottleneck resources (CPU computation, disk storage). The CPU of the server which sends or receives flow j may be too busy with internal computations to serve external write or read requests at the e2e link rate, R_{e2e}. Or the server may not have enough disk space. The application generating flow j may also not have enough data to send or cannot send at the e2e link rate.

As shown in Figure 1, each RA and RM get the values of $Q_d(t-\tau)$ and $Q_u(t-\tau)$ from the local switch (router) with which they are connected (associated). This doesn't need any change to the switches as all switches maintain the queue length in each of their interfaces. Each RM computes the effective number of up-link and down-link flows using equation 3. Each RM reports the values of $S_d(t)$ and $S_u(t)$ to its parent RA. Each RA adds these values from each of its children to find its $S_d(t)$ and $S_u(t)$ values. Each RA also sends its accumulated sum $S_d(t)$ and $S_u(t)$ for both the down-link and up-links to its parent RA. This continues until the highest level RA. After the first time RM sends its $S_d(t)$ and $S_u(t)$ values, it can send the difference Δ_d and Δ_u values to its parents for all other rounds (if there is a change in the rate values). This is to minimize the overhead by sending the difference which is a smaller number than the the sum of the rates. Each RA can also do the same by sending the difference instead of the actual effective number of flows to its parent RA.

Each RM and RA perform the computation of equation 2 periodically every control interval τ. This control interval for the RM can be estimated as the average of the round

trip times (RTT) of the flows of its BS or it can be a user defined parameter. For instance the maximum RTT can be used. Each RA at level h computes its $R_{d,u}(t)$ after it gathers the $S_d(t)$ and $S_u(t)$ information from all its children or after a certain timeout value T_o expires.

Equation 3 enables SCDA to be a *max-min fair* protocol where resources (link bandwidth) unused by flows bottlenecked at other resources (links) can be utilized by flows which need it. For instance, if $R_u^j(t)$ is a bottleneck rate of flow j which is not bottlenecked at a link which allocated $R_u(t-\tau)$, then this link counts flow j as $\frac{R_u^j(t)}{R_u(t-\tau)}$ which is less than 1 flow.

The simplified SCDA rate metric can also be given by

$$R_{d,u}(t) = \frac{(\alpha C_{d,u} - \beta \frac{Q_{d,u}}{d}) R_{d,u}(t-\tau)}{\Lambda_{d,u}(t)} \quad (5)$$

where $\Lambda_{d,u}(t) = L_{d,u}/\tau$ is total packet arrival rate to the router during the control interval τ. In this simplified version of SCDA each RA and RM can also get the values of $L_{d,u}(t)$ from the corresponding switch or router. Hence, for this simplified version of SCDA, the RMs and RAs do not need to report the rate sum values $S_d(t)$ and $S_u(t)$ of flows to their parent nodes (RA).

4.1 Prioritized Rate Allocation for a Desired QoS level

SCDA can also achieve a desired quality of service (QoS) value by allocating different rates to different flows. This is done by using the priority $\wp_{d,u}^j$ weight of flow j in equation 4 as shown by equation 6.

$$S_{d,u}(t) = \sum_{j}^{N_{d,u}(t-\tau)} \wp_{d,u}^j R_{d,u}^j(t) \quad (6)$$

The source of each flow specifies the priority weight values using the RM to achieve a desired rate value. If the source j gets the bottleneck rate $R_{d,u}^j(t)$ discussed above (section 4) and if it wants to set its rate in the next round $t+\tau$ to $R_{d,u}^j(t+\tau)$, it sets it priority as

$$\wp_{d,u}^j = \frac{R_{d,u}^j(t+\tau)}{R_{d,u}^j(t)}.$$

This way a source can achieve the desired rate of its flow j by increasing or decreasing the corresponding $\wp_{d,u}^j$. This approach can adaptively and implicitly implement many scheduling policies in a distributed manner. For instance something like a shortest file (job) first (SJF) and early deadline first (EDF) scheduling algorithms can be implemented by assigning higher target rate $R_{d,u}^j(t+\tau)$ for short or early deadline flows resulting in higher priority weight $\wp_{d,u}^j$ for such flows.

Equation 6 is also very important for *detecting and ensuring service level agreement (SLA)*. A SLA violation is *detected* if the sum $S_{d,u}(t)$ of a link exceeds the link capacity $capacity = \alpha C_{d,u} - \beta \frac{Q_{d,u}}{d}$ in equations 2 and 5. The RM detects SLA violation if its $S_{d,u}(t)$ exceeds the *capacity* of the link it is associated with.

We denote RM level of the tree with level 0. The RAs which are direct parents of RMs are at level 1. The highest level RA is at level h_{max}. The value of h_{max} of figure 5 is 3. The RAs at level 1 detect SLA violation if the sum of $S_{d,u}(t)$ from their children RMs exceeds the *capacity* of the link they are associated with. The RAs at levels higher than 1 detect the SLA violation if the sum of $S_{d,u}(t)$ from their children RAs exceeds the *capacity* of the link they are associated with. Once the SLA violation is detected, the corresponding RM or RA automatically requests for more bandwidth allocation in its link or other alternative links. The SLA violation report by an RM or RA can also be handled by the NNS assigning a different BS for the requesting node. Such selected BS must have enough available bandwidth to support the new request. The data center can also maintain reserve, backup or recovery links to resolve SLA violations automatically. The weights of prioritized flows can then be adaptively adjusted by each distributed source at every RTT to achieve the desired rate of a specific flow.

4.2 SCDA with OpenFlow

SCDA can also implement the QoS Prioritization by using the functionalities of current OpenFlow switches [24]. Each OpenFlow switch maintains packet count Cnt_j for each flow j. So to implement SJF scheduling, the switch can approximate a small size flow to be the flow which has sent fewer packets. The OpenFlow switch then always serves the packets of the flow with smaller packet count. As the packets of the flows which already sent more packets are delayed, such flows reduce their sending rates due to delayed ACK packets. Each RM can also send the priorities of its flows to its RA. The RA can then inform the OpenFlow switch to schedule the packets of the flows according to the priorities.

4.3 QoS By Explicit Reservation

In the SCDA scheme, some sources can also reserve minimum rate $M_{d,u}^j$ required. In this case, for each requesting flow j, the total available link capacity to be shared by other flows is reduced by $M_{d,u}^j$. So if we have $N_{d,u}^{Res}$ such flows reserving capacity, the $C_{d,u}$ in equations 2 and 5 is replaced with $C_{d,u} - \sum_k^{N_{d,u}^{Res}} M_{d,u}^j$. Each RM first sums the $M_{d,u}^j$ values of its node. It then sends the sum to its RA. Each RA also sends these values to its parent RA. When the top level RA receives the sum of the reservations, all flows will have their desired bandwidth reserved in the data center (cloud). The remaining capacity can then be allocated among all flows sharing links.

5. SCDA ALGORITHM

The SCDA algorithm adaptively performs

- per flow resource (link, storage, processing) rate allocation at a global and h-level of the cloud data center tree,
- decision of how and where in the cloud to store data using the allocation information,
- decision of which (replica) server in the cloud to retrieve stored data from and how.

In the following sections we discuss each of these SCDA algorithms in detail.

6. GLOBAL AND H-LEVEL RATE ALLOCATION

Each NNS needs to decide (a) which BS at level h to choose to store each block of data and (b) at what rate to send data from one BS to another BS or to/from an external agent. To do this, the NNS asks the RA at level h, $0 \leq h \leq h_{max}$ of the tree as shown in figure 1. Hence each RA needs to maintain the best down-link and up-link rate values and the address of the BS or BSes with these rate values. For global allocation, the highest level values are needed. Here h_{max} is the maximum level value in the tree like cloud topology starting from the BS nodes. For such three tier topology, $h_{max} = 3$. In such topology, the block servers (BS) are at level 0.

Each NNS among other things also needs to decide at what rate to replicate data from one BS in one level of the cloud tree to another BS in another part of the cloud by asking each RM. Hence each RM also needs to keep the up-link and down-link bottleneck rate values upto each level of the tree. To achieve this, each RA forwards its rate values obtained using equation 2 to its children. Besides, each RA needs to forward to its children the minimum of its rate and the rates forwarded to it from its higher level parents. Finally, these rates of each level of the cloud tree are received by each RM.

The above best h-level rate values stored at each RA and RM are obtained using a max-min scheme as follows.

6.1 Obtaining the Rate values using Max/Min Algorithm

Here is how the rate metric at different levels of the network tree are obtained as also described in figure 2.

To get the metrics kept by the RAs:

- Each RM j at level $h = 0$ sets its downlink (d) and uplink (u) $\hat{R}^{hj}_{d,u}$ rate values to its the minimum of $R^{hj}_{d,u} = R_{d,u}(t)$ which is obtained using equation 2 or equation 5 and its R^{hj}_{other}. The rate value R^{hj}_{other} is a function of the CPU and disk loads. If either the available CPU speed or disk speed are too low, R^{hj}_{other} decreases accordingly. For instance R^{hj}_{other} can be measured from the previous control interval. It can as well be the weighted average of previous intervals. The CPU and disk usage can be profiled to get what CPU and/or usage can serve what link rate. This approach allows SCDA to be a *multi-resource allocation* mechanism. If link bandwidth is the only bottleneck resource, we set $\hat{R}^{hj}_{d,u} = R^{hj}_{d,u}$.

- The RMs also calculate $S^{hj}_{d,u} = S_{d,u}(t)$ using equation 6.

- Each RM sends its $\hat{R}^{hj}_{d,u}$ and $S^{hj}_{d,u}$ values to its parent RA which is associated with the switch the RM and RA are directly associated with (connected to).

- Each RA j at level h calculates its $S^{hj}_{d,u}$ by summing up the $S^{(h-1)j}_{d,u}$ of its children in the RA-RM tree as shown in figure 2. The RA then calculates its $R^{hj}_{d,u} = R_{d,u}(t)$ using equation 2.

- Each RA j at level h sets its $\hat{R}^{hj}_{d,u}$ to the minimum of its $R^{hj}_{d,u}$ and the highest $\hat{R}^{(h-1)j}_{d,u}$ obtained from its children.

- Each RA j at level h then stores its $\hat{R}^{hj}_{d,u}$ values and sends them to its parent RA along with the ID of the corresponding BS. The parent also does the same.

- By the time this process reaches the RA at level h_{max} which is the highest level RA associated with with a switch/router at the entry point to the cloud, each RA j at level h has the best h-level $\hat{R}^{hj}_{d,u}$ and the ID of the corresponding best BS. These values are useful for the NNS to decide *where to store (write) data*.

To get the metrics kept by the RMs:

- The highest level RA (at level $h = h_{max}$) sends its $R^{hj}_{d,u}$ values along with its level number down to its children RAs. Each (child) RA at level $h - 1$ also forwards the minimum of its rate and each of its higher level rates along with the level numbers to its children. Finally each lowest level RA forwards these values to its children RM.

- At this point each RM knows the best h-level up-link and down-link rate values $\check{R}^{hj}_{d,u}$ along with the level numbers. These values are helpful for the NNS in deciding where *to read replicated data from* and to update the *rates of on-going flows* to and from the main cloud (data center) using the information in the RM.

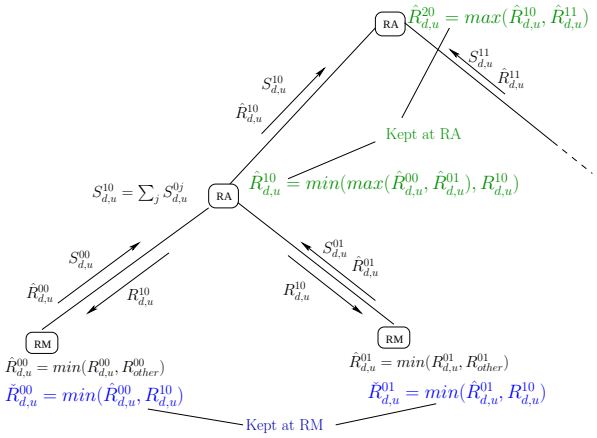

Figure 2: The SCDA Max/Min Scheme

7. CLOUD SERVER SELECTION

After the rate values are obtained using a max-min algorithm discussed in the above section and stored at each RA and RM, SCDA selects a cloud server to store the data of a requesting client. The server is selected in such a way that transfer, retrieval and processing of the data is fast and efficient. To achieve this goal, SCDA treats active and passive contents discussed in section 2.2 differently. Passive data is with low write and low read (LWLR) frequency. *Interactive content* is where write and read operations are interleaved in less than a few seconds interval with high frequency (HWHR). We consider a maximum interactivity interval of 5 seconds to decided whether or not a content is interactive. A *semi-interactive content* is either with HWLR or LRHW. The client applications can specify the type of content or the RMs of the servers can learn the type of content from the

server access frequencies (of writes and reads) by the content. We next show the server selection strategies for each type of content.

7.1 Interactive Content

For interactive applications, the RA at level $h+1$ also keeps the highest of the $min(\hat{R}_d^{hj}, \hat{R}_u^{hj})$ of all its children RM or RA where min is a minimum function. Here, $\hat{R}_d^{hj}$ is the downlink rate and $\hat{R}_u^{hj}$ is the uplink rate of a link at level h (child node of the RA) as shown in figure 2. This is because for interactive applications, the rate at which the interaction is done is limited by either the uplink rate to or downlink rate from the selected server, whichever is smaller. As this process goes up the RA tree hierarchy, all RAs, including the highest level RA (at level $h = h_{max}$), keep $\hat{R}_{min}^{hj} = min(\hat{R}_d^{hj}, \hat{R}_u^{hj})$. SCDA then chooses a BS with highest $\hat{R}_{min}^{hj}$ to serve requests for interactive contents.

7.2 Semi-interactive Content

For semi-interactive applications where either the write or the read operations is very frequent, the server selection is done in two stages. In the first stage, the RA chooses the server at level h with the best downlink rate $\hat{R}_d^{hj}$. This server is the server to which content (data) writing by clients is the fastest. In the second stage, the server to which data is being written chooses another replication server with the best uplink rate $\hat{R}_u^{hj}$. This ensures that the content retrieval is fast. So for these kind of applications, writing is done to the server where data transfer and writing is the fastest. Content reading (retrieval) is done from the (replica) server which offers the fastest reading (upload) rate.

7.3 Passive Content

A content with low write and read frequency (passive content) is replicated at *dormant* servers. *Dormant* servers are in low-power, high-energy-saving inactive power modes as there are more idle periods of server utilization. By sending passive content to dormant servers, SCDA saves energy by reducing latencies associated with the power state transitions. So SCDA can save energy by scaling down some servers with passive content.

Server selection for passive content requests is also done in two stages. In the first content write stage, the server with the highest download rate $\hat{R}_d^{hj}$ is selected. This ensures that data is written fast. The server to which this data is written then selects a replication server which has an upload rate $\hat{R}_u^{hj}$ greater than the scale down threshold rate R_{scale}. The value of R_{scale} is user specified depending on how aggressive the scale down needs to be. For highly aggressive scale down R_{scale} is small. This scale down value can also be set adaptively.

As long as there are passive contents, interactive and semi-interactive contents do not use servers whose upload rates $\hat{R}_u^{hj}$ are greater than R_{scale}. For these applications the RMs of servers to which data is written, select other servers with $\hat{R}_u^{hj} < R_{scale}$ for content replication. This leaves the least loaded servers (servers with very high $\hat{R}_u^{hj}$) for the passive data. This essentially keeps the dormant servers dormant resulting in effective scale down of servers.

Passive content which is initially written to the active servers can be totally moved to the dormant servers after the active servers learn the low frequency of the content. The RM of the active servers to which data is initially written can obtain the frequency (popularity) of contents by counting the number of accesses.

7.4 More Power Efficient Server Selection

In this section we will discuss how to handle heterogeneity in servers energy consumption. This heterogeneity can be due to location of a server in a rack or room, specifications and age of the server hardware and other (processing) tasks the server is doing [17]. So SCDA takes such diverse energy consumption by each server into account while selecting server for each requesting client application. To do this, SCDA relies on measurements of each server's energy consumption. This measurement can be done by (heat or temperature) sensors in the servers. Denoting the heat measurement at time t with $H(t)$, the power consumption during a control interval τ is given by $P(t) = \frac{H(t)}{\tau}$.

Each RA j of SCDA at level h can then select a server with the highest rate to power ratio by replacing $\hat{R}_{d,u}^{hj}$ in section 6.1 with $\frac{\hat{R}_{d,u}^{hj}}{P(t)}$. Other functions of power and rate can also be used to perform server selection. The value of $P(t)$ can be obtained as a running average or with more weight to the latest power consumption measurement.

We next show the steps involved in serving write and read requests by an external requesting client and by the internal cloud (data center) servers.

8. SERVING REQUESTS

In this section we discuss how requests for cloud data center resources are served. The requests can be external to write to and read data from the cloud servers. The request can also be internal to replicate or move data from one cloud server to another server in the same cloud. We next discuss how SCDA serves such requests.

8.1 Serving External Write Request

To serve an external request of a user client (UCL) to use cloud resources, SCDA performs the following steps which are also presented in figure 3.

1. The UCL sends its ID (IP Address) along with the request to write into a cloud (data center) server.
2. The FES hashes the UCL ID and forwards it to the corresponding NNS. The matching NNS can for instance be the server with the ID equal to *hash(UCL_ID) mod* N_{NNS} where N_{NNS} is the number of NNS in the cloud data center and *mod* is the modulo operation.
3. The NNS asks the RA at a level where it wants to select a cloud server from. If the NNS wants to select a server at a specific rack, it asks the RA at level 1 of the corresponding rack for the best server in that rack. This best server is the server to which sending data is the fastest among those in the rack. If the NNS wants to select the best server in the data center, it asks the RA at level $h_{max} = 3$ for the best server (3 tier).
4. The RA selects a block server (BS) with the best rate.
5. The NNS then forwards the UCL ID to the selected BS.
6. The selected BS asks its RM for the download rate all the way from the highest level router (RA) in the data center (cloud).

7. The RM responds with rate which it obtained from the highest level RA via intermediate RAs.

8. The BS sets its receive window size (*rcvw*) to the product of the downlink rate it obtained from its RM and the RTT of the flow. The initial value of the RTT can be updated with more packet arrivals. The receiving cloud server can obtain the RTT from the time stamp values in the headers of the packets it receives from the sender.

9. The selected BS then contacts the requesting peer (UCL) to start the connection which the UCL uses to write data to the BS. While doing so, the the BS sends its receive window size (rcvw) in the packet header.

10. The UCL asks its RM for the upload rate.

11. The RM responds with the upload rate which is the minimum rate upto the highest level RA router (switch).

12. The UCL then sets its congestion window (cwnd) to the product of the upload rate and its RTT. As the UCL also receives the rcvw from the destination BS, it sets its sending window size to the minimum of the *cwnd* and *rcvw*. If the UCL has no RM (no dedicated tunnel), then setting the *rcvw* can ensure that the UCL does not send more than what the datacenter can handle.

13. The UCL then starts writing its data to the selected server.

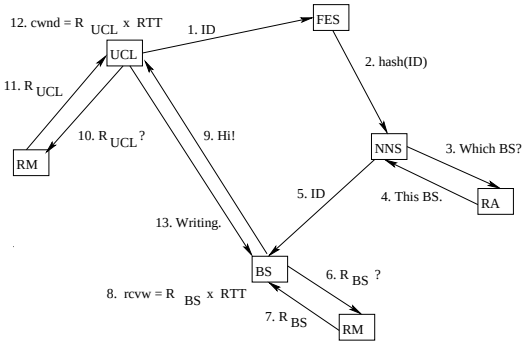

Figure 3: Serving External Write Request

8.2 Serving Internal Write Request

Once a UCL writes (uploads) data content to the cloud BS which offers the best upload rate, the BS decides to replicate or move the content to another BS which can offer the best upload rate with minimum energy consumption. To do this SCDA follows the following steps which are also described in figure 4.

1. First the BS (e.g. BS11) which wants to replicate the content contacts the NNS of the content by sending hash of the content ID.

2. The NNS selects a block server (BS23) based on the content selection algorithm discussed in section 7 (server which offers high upload rate) to ensure that future client read requests are fast.

3. The rest of the steps are similar with the steps in serving external write request discussed in section 8.1 with BS11 instead of the UCL and BS23 as the BS in figure 3.

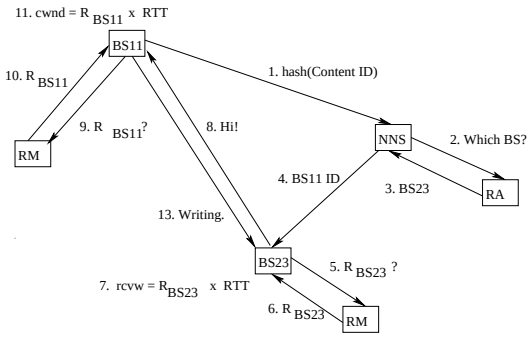

Figure 4: Serving Internal Write Request

8.3 Serving External Read Request

Once the UCL writes data to the selected cloud BS and after this BS replicates the content, the UCL request to read data is served using the following steps which are also presented in figure 5. The server selection is based on the server selection mechanism discussed in section 7.

1. The UCL requests for a certain content to read from the cloud by sending its ID.

2. The FES hashes this UCL ID and forwards it the responsible (corresponding) NNS which has the metadata of the requested content.

3. The NNS can either maintain the best BS for each of the contents whose metadata it keeps. It can also request (poll) the RMs of the BSs which have the content for their upload rates. It then chooses the best BS based on the server selection mechanism discussed in section 7 (server with the content which has high upload rate). The NNS forwards the UCL ID to the selected BS.

4. The selected BS asks its RM for the upload rate.

5. The RM provides its BS with the upload rate.

6. The BS sets its cwnd to the product of this rate and its RTT. If the UCL has no RM (not using a dedicated tunnel), the BS just sets its maximum congestion window size to the product of the rate and its RTT.

7. The BS starts writing to the requesting UCL.

8. The UCL asks its RM for its download rate.

9. The UCL gets the download rate from its RM.

10. The UCL sets its rcvw to the product of this download rate and its flow's RTT and continues to read (download) the content.

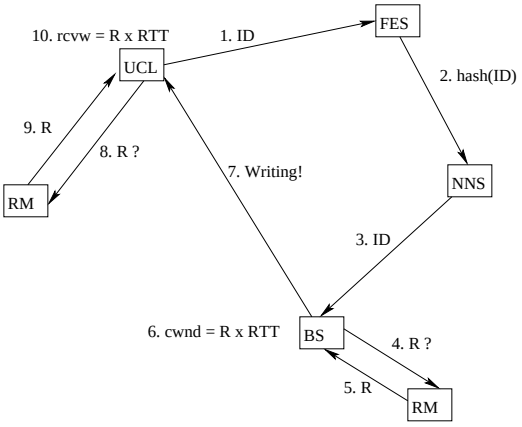

Figure 5: Serving External Read Request

8.4 Updating Rate of On-going Flows

To update the rates at which on-going flows in the cloud should send data, both the sender and the receiver have to update their windows. Suppose the lowest level parent (switch/router) both the sender and receiver share is at level h. The sender sets its *cwnd* to the product of the level h upload rate it obtains from its RM and the current RTT of the flow. Besides, the receiver sets its receive window to the product of the h level download rate it obtains from its RM and the current RTT. These two window updates in each BS are done by the RM of each BS every control interval τ.

9. GENERAL NETWORK TOPOLOGIES

In the above sections of this paper, we have shown how SCDA works with tree type network topologies described by figure 1. The design of SCDA also applies to a general data center network topology such as figure 8 of [2]. For such topologies, the RM associated with each BS obtains the values of $S_{d,u}(t)$ given by equation 6 for each group of flows sharing the same path upto the highest level switches. To form these groups, the RMs can use their routing tables.

The routing tables can be calculated by each RM and RA (distributed). They can also be obtained by a central agent (controller) and shared among all RMs and RAs. To form a topology for route computation, each RM and RA share the weights of the links they represent. This can be done using message passing (inter-process communication) if the RMs and RAs are located in the same server system. Each RM and RA can also send the weights of the links they represent to a central server (controller) which forms the topology with link weights from which the shortest paths are computed.

The weight of each link is the value of $R_{d,u}(t)$ of that link given by equations 2 or 5. In this case, a max/min algorithm has to be used to find the best path and the rate in that path. This is done by first finding the minimum rate of each path and then taking the path with the maximum such rate as shown in [7].

10. EXPERIMENTAL RESULTS

We implemented SCDA in the NS2 simulation package. We use the network topology described by figure 6. We run experiments using content size and flow arrival rate traces and well known distributions. In the experiments we show how SCDA compares with RandTCP, a random server selection and TCP rate control approach used by well known architectures such as VL2 [12] and Hedera [2]. We next discuss our initial experimental results.

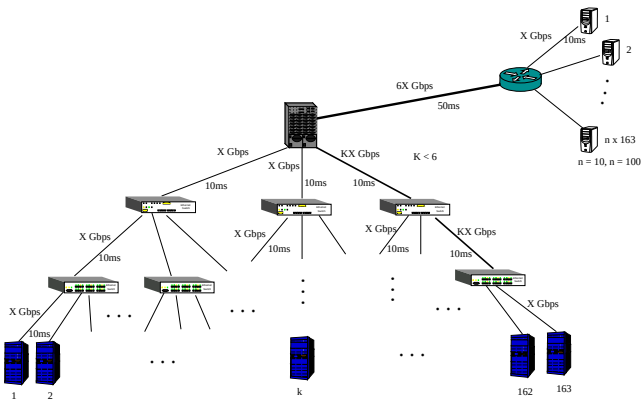

Figure 6: SCDA experimental network topology

10.1 Using File Size and Flow Arrival Traces

In this section we will first discuss experimental results based on CDN YouTube video traces and then using general datacenter traffic traces.

10.1.1 Video Traces

For the first group of experiments we use CDN traces for the file sizes [27] and flow arrival rates [22]. The file size traces belong to control flows which are less than 5KB and YouTube video flows which are greater than or equal to 5KB. A bandwidth factor of $K = 3$ is used for these experiments. By varying this bandwidth multiplier of some links in the right side of the topology given in figure 6, we show that SCDA is not restricted to equal bandwidth datacenter architectures. We calculate arrival rates to 20 of the 2138 YouTube servers considered in [27] proportionally to scale our simulation. For these set of experiments we use the base bandwidth of $X = 500\,Mbps = 0.5Gbps$.

The first set of video trace experiments includes both the control and video flows. The control flows are HTTP messages exchanged between the Flash Plugin and a content server before a video flow starts. Figure 7 shows that SCDA achieves higher average instantaneous throughput than RandTCP based schemes. Figure 8 shows that most of SCDA flows finished in a much shorter time when compared with RandTCP based schemes. A combination of random server selection and TCP behavior causes the performance decline of RandTCP based approaches. Figure 9 also shows that SCDA can achieve a smaller AFCT (average file completion time). AFCT of flows of some size is obtained by taking the average completion times of all flows with that size which finish within simulation time.

We have also conducted trace based experiments excluding the video control flows (only YouTube video flows) as shown in figures 10,11 and 12.

From figures 7 and 10 it can be seen that upto 50% higher average throughput than RandTCP based schemes can be obtained. Figures 8 and 11 as well as figures 9 and 12 show that SCDA can result in FCT (file completion time) which is more than 50% lower than that of RandTCP based schemes.

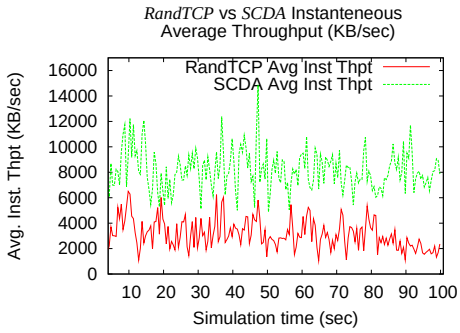

Figure 7: Instantaneous throughput comparison of SCDA and RandTCP based: Using Video Traces with control flows

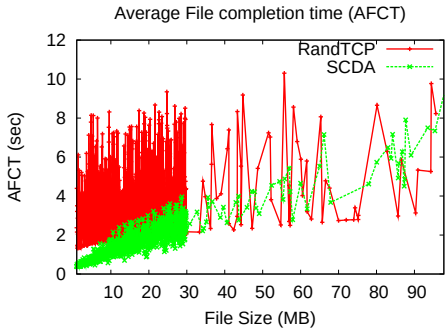

Figure 9: AFCT comparison of SCDA and RandTCP based: Using Video Traces with control flows

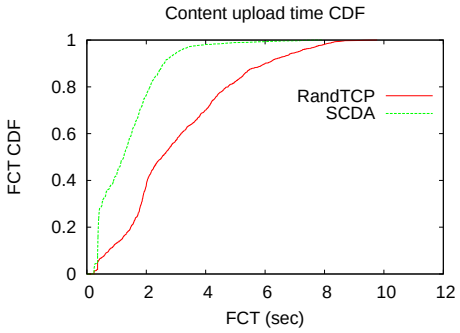

Figure 8: FCT CDF comparison of SCDA and RandTCP based: Using Video Traces with control flows

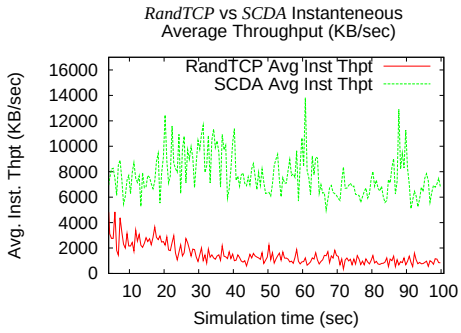

Figure 10: Average Instantaneous Throughput comparison of SCDA and RandTCP based: Using Video Traces without control flows

As shown in [27, 5], there is a maximum size limit of about 30MB for most YouTube video files. The AFCT in figures 9 and 12 show that the transfer (upload or download) time of these files is more than 60% smaller than RandTCP based schemes for the topology given in figure 6. The transfer times of the very few files which are larger than 30MB is also not larger than that of the RandTCP based schemes. The wild fluctuations of the AFCT of the RandTCP based schemes is because of the random server selection and the behavior of TCP in not knowing the appropriate sending rate. On the other hand SCDA gets explicit bottleneck rate share information of each flow from the interactions of the RM (resource monitors) and RA (resource allocators).

10.1.2 General Datacenter Traces

We have also evaluated the performance of SCDA using datacenter file size and flow inter-arrival traces obtained from [12] and [3] respectively. For the first set of experiments shown in figures 13 and 14 we use a bandwidth factor of $K = 1$. Similar to the plots in section 10.1.1, these plots also show that SCDA achieves a FCT which is upto 50% lower than RandTCP based schemes.

For the second set of experiments shown in figures 15 and 16 we use $K = 3$. As can be seen from figures 13 and 15, the AFCT of RandTCP shows some wild fluctuations as a result of random server selection and TCP behavior. The random server selection may result in assigning flows (requests) to servers which are congested with long-lived (elephant) flows. As a result AFCT of SCDA is upto 50% lower than that of the RandTCP based schemes. The CDF figures 14 and 16 also show that more than 60% of SCDA flows achieve upto 50% smaller transfer times than RandTCP based approaches.

10.2 Using Pareto File Size and Poisson Flow Arrival Distributions

We have also used Pareto distribution to generate the file (content) sizes and Poisson distribution to generate the flow inter-arrival times. We set the base bandwidth value $X = 200\,Mbps$ and the bandwidth factor $K = 3$ for this experiments. File sizes are Pareto distributed with mean $500KB$ and shape parameter of 1.6. Flow arrival rates are Poisson distributed with mean 200 flows/sec. Consistent with the trace based plots in section 10.1, the distribution based figures 17 and 18 show that SCDA outperforms RandTCP based schemes.

11. RELATED WORK

In this section we discuss existing server selection and congestion control mechanisms in data center networks. As also discussed in [4], in the Fat-Tree architectures [1, 23], each switch in the lower level of the topology regularly (every second) measures the utilization of its output ports. This measurement is done at regular interval (every 10 second). If the utilization of the output ports are mismatched, the switch

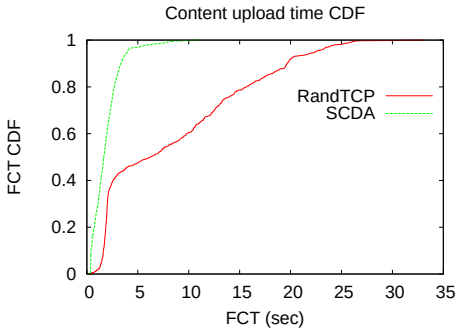

Figure 11: FCT CDF comparison of SCDA and RandTCP based: Using Video Traces without control flows

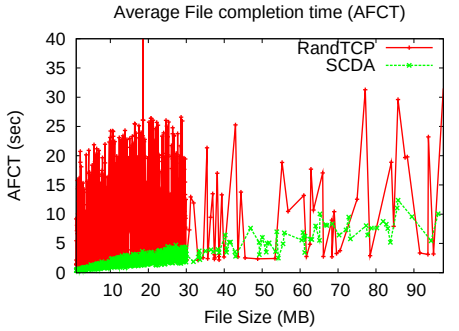

Figure 12: AFCT comparison of SCDA and RandTCP based: Using Video Traces without control flows

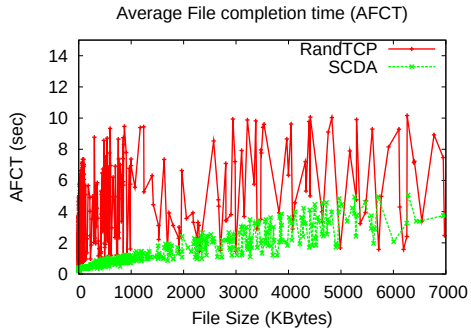

Figure 13: AFCT comparison of SCDA and RandTCP based: Using Datacenter Traces with $K = 1$

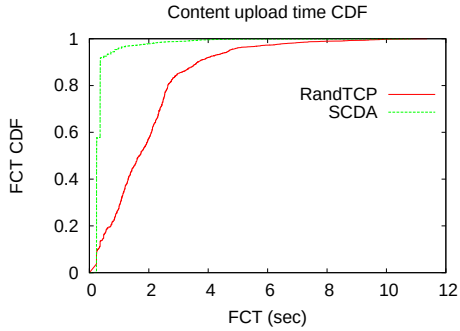

Figure 14: FCT CDF comparison of SCDA and RandTCP based: Using Datacenter Traces with $K = 1$

reassigns a minimal number of flows to the ports. Fat-Tree uses this local heuristic to balance load across multiple shortest paths. However, as discussed in [4], this heuristic results in a 23% performance gap from the optimal value resulting in possible packet losses and congestion. This demands for globally optimal decisions as also pointed out in [4]. Our SCDA scheme has an adaptive global and local view of the cloud data center to achieve optimal resource allocation.

The VL2 architecture [12] randomly chooses intermediate switches to forward flows to servers using equal-cost multipath routing (ECMP) [13] and valiant load balancing (VLB) [20]. As also pointed out by the authors of VL2 and in [2], both ECMP and VLB schemes of random placement of flows to servers can lead to persistent congestion on some links while other links are under-utilized. This is specially the case with "elephant flows" in the network or in a network where there is multimedia video streaming. One of the reasons for this is the inability of ECMP to track and adapt to instantaneous traffic volume. It should be noted that per-flow VLB which is the case with VL2 becomes equivalent to ECMP, with both utilizing random switch and hence server selection mechanism.

The Hedera [2] flow scheduling utilizes ECMP for short-lived flows and a centralized approach to route large flows (with over 100MB of data). Leaving the complexity of classifying flows and detecting the amount of data, flows can send before they send it, aside, the work in [4] showed that the Hedera scheme performed comparable to (not better than) the ECMP as most of the contending flows had less than 100MB of data to send. Unlike VL2 and Hedera, the SCDA server selection approach adaptively takes many resource constraints into account.

A traffic engineering scheme, called MicroTE, is also proposed in [4] to address the problems of the above data center architectures. Just like our initial work [8, 9], the MicroTE approach uses a controller which aggregates network traffic information from the top-of-the-rack (ToR) switches in the data center. The controller then tries to solve a usual capacity constrained linear programming problem using a heuristic approach. The approach first sorts predictable ToR pairs which exchange traffic according to their traffic volume. The prediction is done over 2 seconds interval. Leaving the complexity involved in doing this prediction aside, for N nodes and E edges in the network, MicroTE has a computational complexity of $\bigodot(PNlog(N)+P+PlogP)$ for P predictable ToR (source-destination) pairs. Besides the link weight used for the optimization problem is the inverse of available capacity. If there are two links with the same available capacity, and if one of them has more flows sending data to it, then it is not a good idea to consider the two links the same way. This implies that inverse of available capacity is not a good link metric as it does not take into account the number of flows sharing the link. To deal with such link weight issues, SCDA uses a flow rate as a link weight. Besides, SCDA uses a distributed and adaptive scheme to implicitly solve the optimization problem.

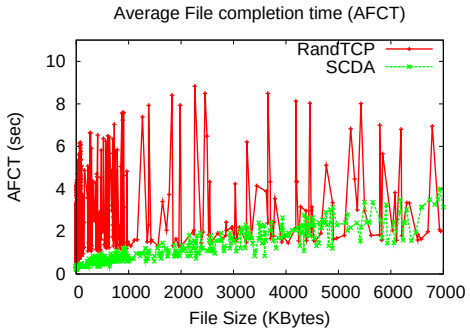

Figure 15: AFCT comparison of SCDA and RandTCP based: Using Datacenter Traces with $K = 3$

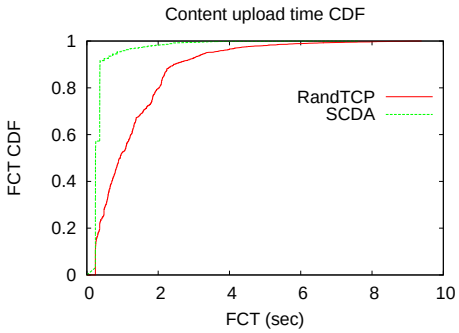

Figure 16: FCT CDF comparison of SCDA and RandTCP based: Using Datacenter Traces with $K = 3$

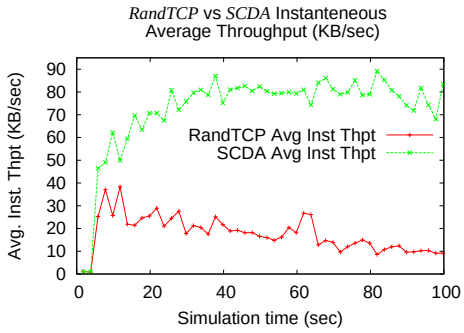

Figure 17: Throughput comparison of SCDA and VL2: Using Pareto and Poisson Distributions

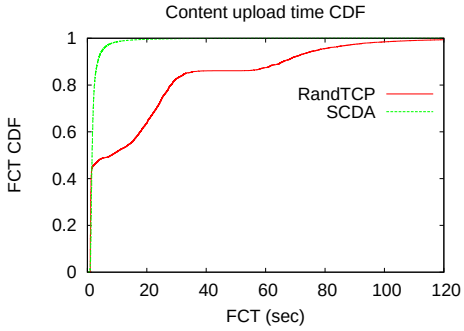

Figure 18: File Completion Time (FCT) comparison of SCDA and VL2: Using Pareto and Poisson Distributions

Besides, all of the above architectures depend on TCP to determine sending rates of the flows. Among other weaknesses, TCP results in very high AFCT [6]. Clean slate congestion control protocols such as XCP [15] and RCP [6] require modifications to routers/switches and the TCP/IP stack. The SCDA architecture mitigates the drawbacks of TCP without requiring changes at the routers/switches and TCP/IP stack. All of the above schemes do not achieve max/min fairness and do not have a mechanism to detect SLA violation in realtime. The RA and RM software components along with some mathematical formulation allows SCDA to achieve max/min fairness and to detect SLA violations without the need of hardware or TCP/IP changes.

12. SUMMARY

In this paper we have presented the design of SLA-aware cloud data center architecture (SCDA) for efficient content storage and retrieval. Current large scale distributed file systems such as the Google File System (GFS) and its derivative, the Hadoop File System (HDFS), rely on a single name node server (NNS) to manage metadata information of all chunks stored in all block (chunk) servers (BS) in the cloud. This design can make GFS and HDFS bottlenecked at the single NNS. The design of SCDA solves this problem by introducing a light weight front end server (FES) which forwards requests to multiple NNS.

Existing schemes such as GFS, HDFS, VL2, Hedera rely on TCP to avoid congestion and determine the sending rates of flows. The SCDA architecture uses efficient congestion control and server selection mechanism to decide *where in the cloud (distributed system)* to store data and *at what rate to transmit* data. This design enables SCDA to efficiently balance load among all data and name node servers automatically. The resource monitor (RM) and resource allocator (RA) components of SCDA also allow SCDA to be implemented without the need to change network switches, routers and the TCP/IP packet header format.

The design of SCDA has other important features. It can adaptively achieve max/min fairness which is described to be NP hard in the current literature. It can automatically detect SLA violation in realtime. It is scalable as all its components can run independently by exchanging messages. The design of SCDA is extended to be more energy efficient. SCDA can serve as a multi resource allocation scheme where the bottleneck resource can be other than the link bandwidth. The prioritized allocation mechanism of SCDA allows it to easily make QoS rate assignments.

We have implemented SCDA in the NS2 simulator [21] and compared it against well known existing schemes using random server selection and TCP (RandTCP). Simulation results show how SCDA outperforms RandTCP based approaches such as VL2 and Hedera in terms of content transfer time and throughput.

13. REFERENCES

[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 63–74.

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX NSDI'10* (2010), pp. 19–19.

[3] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 267–280.

[4] Benson, T., Anand, A., Akella, A., and Zhang, M. Microte: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, ACM, pp. 8:1–8:12.

[5] Cheng, X., Dale, C., and Liu, J. Statistics and social network of youtube videos. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on* (june 2008), pp. 229 –238.

[6] Dukkipati, N., and McKeown, N. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev. 36*, 1 (2006), 59–62.

[7] Fesehaye, D., Gupta, I., and Nahrstedt, K. A Cross-layer Routing and Congestion Control for Distributed Systems. Technical report, University of Illinois at Urbana-Champaign (UIUC), Nov. 2008.

[8] Fesehaye, D., Malik, R., and Nahrstedt, K. EDFS: a semi-centralized efficient distributed file system. In *Middleware (Companion)* (2009), p. 28.

[9] Fesehaye, D., Malik, R., and Nahrstedt, K. A Scalable Distributed File System for Cloud Computing. Technical report, University of Illinois at Urbana-Champaign (UIUC), 03 2010.

[10] Fesehaye, D., Malik, R., and Nahrstedt, K. Efficient Distributed File System (EDFS). Technical report slides, University of Illinois at Urbana-Champaign (UIUC), 07 2010.

[11] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev. 37* (Oct. 2003), 29–43.

[12] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM '09* (2009), pp. 51–62.

[13] Hopps, C. Analysis of an equal-cost multi-path algorithm, 2000.

[14] Jacobson, V. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols* (New York, NY, USA, 1988), SIGCOMM '88, ACM, pp. 314–329.

[15] Katabi, D., Handley, M., and Rohrs, C. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev. 32*, 4 (2002), 89–102.

[16] Kaushik, R. T., and Bhandarkar, M. Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In *Proceedings of the 2010 international conference on Power aware computing and systems* (Berkeley, CA, USA, 2010), HotPower'10, USENIX Association, pp. 1–9.

[17] Kaushik, R. T., and Nahrstedt, K. T: a data-centric cooling energy costs reduction approach for big data analytics cloud. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Los Alamitos, CA, USA, 2012), SC '12, IEEE Computer Society Press, pp. 52:1–52:11.

[18] Krishnan, R., Madhyastha, H. V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., and Gao, J. Moving beyond end-to-end path information to optimize cdn performance. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2009), IMC '09, ACM, pp. 190–201.

[19] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. Internet inter-domain traffic. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2010).

[20] López-Ortiz, A. Valiant load balancing, benes networks and resilient backbone design. In *Proceedings of the 4th conference on Combinatorial and algorithmic aspects of networking* (Berlin, Heidelberg, 2007), CAAN'07, Springer-Verlag, pp. 2–2.

[21] McCanne, S., and Floyd, S. ns-2. `http://www.isi.edu/nsnam/ns/`.

[22] Mori, T., Kawahara, R., Hasegawa, H., and Shimogawa, S. Characterizing traffic flows originating from large-scale video sharing services. TMA'10, Springer-Verlag, pp. 17–31.

[23] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM '09* (2009), pp. 39–50.

[24] Pfaff, B., and et. al. Openflow switch specification v1.3.0. https://www.opennetworking.org/, April 2012.

[25] Pillay-Esnault., P., Moyer, P., Doyle, J., Ertekin, E., and Lundberg, M. Ospfv3 as a provider edge to customer edge (pe-ce) routing protocol, 2012.

[26] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010), pp. 1–10.

[27] Torres, R., Finamore, A., Kim, J. R., Mellia, M., Munafo, M. M., and Rao, S. Dissecting video server selection strategies in the youtube cdn. ICDCS '11, IEEE Computer Society, pp. 248–257.

[28] Triukose, S., Wen, Z., and Rabinovich, M. Measuring a commercial content delivery network. In *Proceedings of the 20th international conference on World wide web* (2011), WWW '11, pp. 467–476.