Network Topology Optimization For Data Aggregation

Soham Das and Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, USA
Email: {sdas, sahni}@cise.ufl.edu

***Abstract*—In this paper, we show that the problem of configuring the topology of a data center network to optimize data aggregation is NP-hard even when the number of aggregators is 1. Further, the approximation ratio of the algorithm proposed by Wang, Ng, and Shaikh [3] for the case of a single aggregator is (k+1)/2, where k is the degree of ToR (top-of-rack) switches and this algorithm also exhibits an anomalous behavior- increase in the switch degree may result in an increase in the aggregation time. By comparison, if topology configuration is done using the longest processing time (LPT) scheduling rule, the approximation ratio is (4/3-1/(3k)). We show that for every instance of the single aggregator topology configuration problem, the time required to aggregate using the LPT configuration is no more than that using the Wang et al. rule. By coupling the LPT rule with the rule of Wang et al., we achieve a better throughput as promised by LPT and at the same time reduce the total network traffic. Experimental results show that the LPT rule reduces aggregation time by up to 90% compared to the Wang et al. rule. The reduction in aggregation time afforded by a known improvement, COMBINE, of LPT relative to Wang et al. is up to 90.5%. More interestingly, when either of the LPT rule or COMBINE is augmented with the Wang et al. rule, total network traffic is reduced by up to 90% relative to using LPT and COMBINE with chains.**

***Keywords*-Data Center Networks; Software Defined networking; Big Data applications; Map-Reduce tasks**

I. Introduction

A large data center may be comprised of thousands of racks with each rack housing tens of computers [3]. Each rack has a top-of-rack (ToR) switch with some number (k) of optical links. Current commercial ToR switches have $k \leq 10$ [3]. Software defined networking (SDN) enables one to dynamically (re)configure the topology of the network comprised of ToR switches and their optical links. In the case of OpenFlow enabled ToR switches, this reconfiguration can be done by changing the OpenFlow rules stored in each ToR switch. In big data applications [1], [2], the time required to configure a desired network topology in this way is small compared to the time required to run the application using the configured topology. For example, in a big data MapReduce application, the time required to aggregate data from the mappers to the reducers is orders of magnitude greater than the reconfiguration time. Hence, the performance of these applications can be enhanced by reconfiguring the network topology to one that is optimized for the application rather than using a generic topology or the one used by an earlier application.

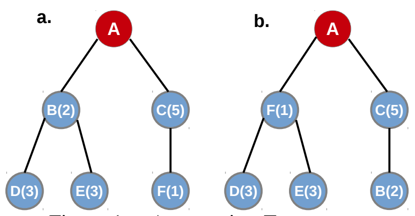


Figure 1. Aggregation Tree prototype

In this paper, we focus on network topology optimization for the data aggregation task common to many big data applications. In the data aggregation problem, data residing in several racks of a data center are to be aggregated into a single specified rack called the aggregator. In general, we may have several aggregators with each aggregating data from some subset of the racks. However, in this paper, we limit ourselves to the case of a single aggregator. Figure 1(a) shows an example aggregator tree. In this figure, the root (A) denotes the aggregator rack, the remaining nodes (B, C, D, E, and F) denote racks from which data is to be aggregated into the root, and the edges denote optical links. The shown aggregator tree requires $k \geq 3$ as node B uses three optical links to connect to nodes A, D, and E. To accomplish the required data aggregation, each node sends its data to its parent who then sends the received data to its parent and so on until all data reach the root. Suppose that node B has 2Gb of data to send to the root and that nodes D and E each have 3Gb of data to send to the root as shown in brackets in the figure. We assume that the data is transmitted in packets, a node can receive incoming data from all children in parallel, and that while a node is receiving packets from its children, it can transmit previously received packets to its parent. If each optical link has a bandwidth of 10 Gbps, the data from B, D, and E can be aggregated into A in $(2+3+3)/10 = 0.8s$. If C and F have 5Gb and 1Gb, respectively, of data that is to be aggregated into A, the time for this is $0.6s$. Since A receives data from B and C in parallel, the total aggregation time is $\max\{0.8, 0.6\} = 0.8s$. When the aggregation tree of Figure 1(b) is used, the aggregation time is $\max\{0.1+0.3+0.3, 0.2+0.5\} = 0.7s$.

We define the total network traffic to be the sum of the amounts of data moved through each optical link. For example, the total network traffic using the topology of Figure 1(a) is 8 (link AB) + 6 (link AC) + 3 (link BD)

+ 3 (link BE) + 1 (link CF) = 21Gb and that using the topology of Figure 1(b) is 7 + 7 + 3 + 3 + 2 = 22Gb. If we swap nodes D and F in Figure 1(b), the total network traffic becomes 20Gb and the aggregation time remains $0.7s$.

In the single aggregator network topology optimization (SANTO) problem, we are to determine a tree topology that minimizes the aggregation time subject to the constraint that the degree of every node (ToR) is at most k, where k is the number of optical links at each node (ToR). SANTO-NT is an extension of SANTO in which we seek a topology that minimizes total network traffic subject to the constraint that aggregation time is minimized. That is, from among the topologies that minimize aggregation time, obtain one with least total network traffic.

In this paper, we study the SANTO and SANTO-NT problems and obtain the following results:

- SANTO is NP-hard for $k \geq 2$. From this it follows that SANTO-NT is also NP-hard for $k \geq 2$.
- The approximation ratio for the algorithm proposed by Wang et. al. [3] for SANTO is $(k+1)/2$ and this bound is tight.
- The SANTO algorithm of Wang et al. [3] may result in an increased aggregation time when the number of optical links per rack is increased.
- SANTO may be solved by using any of the approximation algorithms proposed for minimum finish time scheduling of identical machines. When the popular LPT (Longest Processing Time) scheduling rule [4], [5] is used, the approximation ratio for SANTO is $(4/3 - 1/(3k))$. Further, for every SANTO instance, the aggregation time of the constructed LPT topology is no more than that of the topology constructed by the algorithm of Wang et al. [3]. The COMBINE scheduling rule can also be used to solve every SANTO instance and it has an approximation ratio of $10/9$, which is proved for two machines in [6]. Once again the constructed topologies always have at least as short an aggregation time as those obtained by the algorithm of Wang et al. [3].
- For SANTO-NT, we propose using either LPT or COMBINE to partition the nodes (ToRs) into k sets and then using the algorithm of Wang et al. [3] to arrange the nodes in each partition into a tree topology. We show that this algorithm minimizes total network traffic subject to the partitioning obtained by LPT or COMBINE.
- We show via experimentation that LPT and COMBINE can reduce aggregation time as well as total network traffic by up to 90% relative to the algorithm of Wang et al. [3].

The remainder of this paper is organized as follows. In Section II, we briefly review related work. SANTO is shown to be NP-hard in Section III. The approximation ratio for the SANTO algorithm of Wang et al. [3] is derived in Section IV and its anomalous behavior (i.e, possible increase in aggregation time with increase in the number of optical links per rack) is demonstrated in Section V. In Section VI, we show how the scheduling algorithms LPT and COMBINE may be used to obtain a tree topology for SANTO. The approximation ratio for this use of the algorithms stays the same as for their native use in scheduling. In Section VII, we show that using the algorithm of Wang et al. [3] to obtain a tree topology for a given partitioning of nodes minimizes the total network traffic for that partitioning. Our experimental results are presented in Section VIII and we summarize our conclusions in Section IX.

II. Related Work

Customizing data center network topology to better suit the needs of an application and thereby enhancing performance has been the focus of intense recent research. For example, Greenberg et al. [8] have proposed the use of programmable commodity switches to reduce network cost and enhance performance through the use of Valiant Load Balancing, Al-Fares et al. [9] propose using commodity Ethernet switches to support the full aggregate bandwidth of large clusters, Guo et al. [10] use servers as nodes in the interconnect, Das et al. [11] explore the use of OpenFlow to control routing according to application need and Webb et al. [12] propose to isolate applications and use different routing mechanisms for them in fat-tree based data-centers. The flow of big data traffic in data center networks has also been studied. For example, Kavulya et al. [13] have analyzed 10-months of Map-Reduce logs from the M45 supercomputing cluster at Yahoo! and Benson et al. [14] have conducted an empirical study of the network traffic in 10 data centers that include university, enterprise, and cloud data centers.

Our work is most closely related to the work of Wang, Ng and Shaikh [3] who describe an ”integrated network control for big data applications” that comprises ”OpenFlow-enabled top-of-rack (ToR) switches”. Wang et al. [3] propose an algorithm for SANTO as well as for the more general case of multiple aggregators. Their algorithm for SANTO has two steps. In the first step, the racks are sorted into decreasing order of the amount of data they need to send to the aggregator and in the second step, the racks are placed into the tree topology in this order. The objective is to place racks with ”higher demand closer to the aggregator”. Consequently, the tree topology used is necessarily a tree in which the root has degree k and every other non-leaf node (with the exception of possibly one such non-leaf) has degree $k - 1$.

III. SANTO is NP-hard

An instance of SANTO is characterized by k, which is the number of optical links in a ToR switch; n, which is the number of racks from which data is to be aggregated into the aggregator; and d_i, which is the amount of data in rack i, $1 \leq i \leq n$, that is to be aggregated. We shall prove

SANTO NP-hard by using the known NP-complete problem Partition in which we are given a multi-set of non-negative integers $s_i, 1 \leq i \leq m$ and seek to determine whether or not the multi-set can be partitioned so that the sum of the numbers in each partition is $\sum s_i/2$.

Theorem 1. *SANTO is NP-hard for $k \geq 2$.*

Proof: Consider any instance $s_i, 1 \leq i \leq m$ of the Partition problem. Without loss of generality, we may assume that $\sum s_i$ is even as otherwise there can be no partition. From this instance, we construct, for any k, an instance of SANTO with $n = m + k - 2$ racks, $d_i = s_i$, $1 \leq i \leq m$, and $d_i = \sum s_i/2$, $m < i \leq n$. Every aggregation tree for this instance has k subtrees (some may be empty). Let D_i be the sum of the d_js for the racks in subtree i. The aggregation time is $\max\{D_i\}/B$, where B is the link bandwidth [1]. From this, it is easy to see that the constructed SANTO instance can be aggregated in $\sum s_i/2/B$ time iff the Partition instance has a partition. Hence, SANTO is NP-hard. ■

Since SANTO is NP-hard, its extension SANTO-NT is also NP-hard.

IV. Approximation Ratio For SANTO of [3]

Let I be a SANTO instance with k optical links per rack. Let $T^*(I)$ be the minimum aggregation time for this instance and let $T_A(I)$ be the aggregation time for the topology computed for this instance by some algorithm A. The *approximation ratio* of A is defined to be $\max\{T_A(I)/T^*(I)\}$, where the max is taken over all instances I that have k optical links per rack.

As mentioned in Section II, the SANTO algorithm of Wang et al. [3] places racks as close to the aggregator as possible in decreasing order of d_i. To be more precise, we may assume the aggregator tree is constructed beginning with the root node (i.e., the aggregator) and adding in the $n \geq k$ remaining rack nodes one at a time top to bottom and within a level from left to right. The degree of the root is k and that of the remaining non-leaf nodes (except possibly the last one) is $k - 1$. The degree of the last non-leaf node is at most $k - 1$.

Theorem 2. *The approximation ratio of the SANTO algorithm of [3] is $(k + 1)/2$ and this bound is tight.*

Proof: Consider the SANTO instance $d_1 \geq d_2 \geq \cdots \geq d_n$ with k optical links per rack. Let B be the link bandwidth and let $a_i = d_i/B$, $1 \leq i \leq n$. Let T^* be the optimal aggregation time and let T_W be the aggregation time using the aggregation topology constructed by the algorithm of [3]. We first observe that $T^* \geq \max\{a_1, \sum a_i/k\}$.

[1]Note that the aggregation time is independent of the structure of the individual subtrees and depends only on the D_is. In particular, the aggregation time is the same for subtrees that are chains and for those that are well balanced.

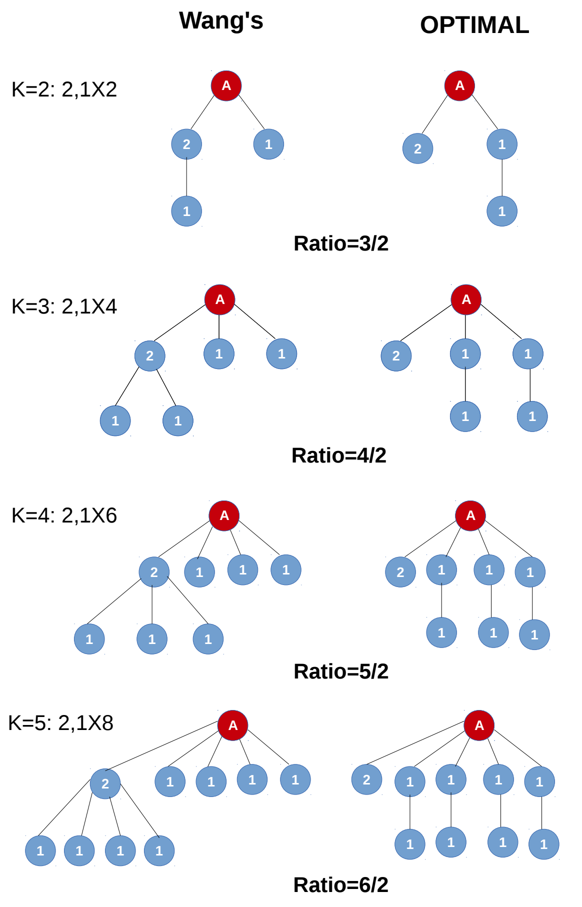


Figure 2. Tightness of the Wang's Bound

First, consider the case $k = 2$. When n is odd,

$$\begin{aligned} 2T_W &= 2a_1 + 2a_3 + 2a_5 + \ldots + 2a_n \\ &\leq a_1 + a_1 + a_2 + a_3 + a_4 + a_5 + \ldots a_n \\ &= a_1 + \sum_1^n a_i \\ &\leq T^* + 2T^* = 3T^* \end{aligned}$$

When n is even,

$$\begin{aligned} 2T_W &= 2a_1 + 2a_3 + 2a_5 + \ldots + 2a_{n-1} \\ &\leq a_1 + a_1 + a_2 + a_3 + a_4 + a_5 + \ldots a_{n-1} \\ &= a_1 + \sum_1^{n-1} a_i \\ &\leq T^* + 2T^* = 3T^* \end{aligned}$$

So, when $k = 2$, $T_W/T^* \leq 3/2$.

Generalizing this proof to arbitrary k, we get

$$\begin{aligned}2T_W &= 2a_1 + 2(a_{k+1} + a_{k+2} + \cdots a_{2k-1}) + \cdots \\ &\leq a_1 + a_1 + a_2 + \cdots a_{2k-1} + \cdots \\ &\leq a_1 + \sum a_i \\ &\leq T^* + kT^* = (k+1)T^*\end{aligned}$$

So $T_W/T^* \leq (k+1)/2$.

The tightness of this bound is established by demonstrating an instance for each k for which this bound is achieved. Figure 2 gives these instances for $2 \leq k \leq 5$. The shown instances are easily generalized to higher values of k. ■

V. Anomalous Behavior of SANTO Algorithm Of [3]

A surprising observation regarding the behavior of the algorithm in [3] is that the aggregation time may increase with increase in the number of optical links, k, per rack. To see this, consider the SANTO instance with $n = 7$ racks and $d_{1:7} = \{10, 9, 8, 7, 6, 5, 4\}$. Figure 3 shows the constructed aggregation tree for $k = 3$ and 4. The aggregation time (assuming $B = 1$) is 23 when $k = 3$ and 25 when $k = 4$. When all d_is are the same and $n = 7$, the aggregation time for the case $k = 4$ is 25% more than when $k = 3$ (4 nodes are placed in the leftmost subtree when $k = 4$ compared to 3 when $k = 3$).

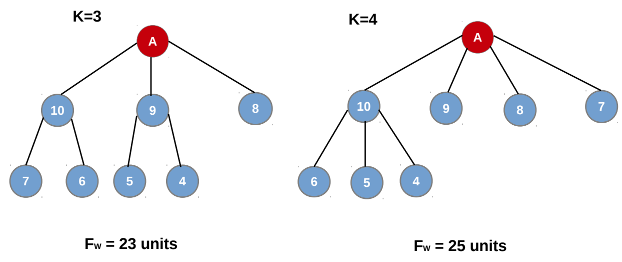


Figure 3. Anomalous behavior of SANTO algorithm of [3]

VI. SANTO Using LPT and COMBINE

As noted in Section III, the aggregation time is $\max\{D_i\}/B$, where D_i is the amount of data to be transmitted to the aggregator by the racks in subtree i, $1 \leq i \leq k$, of the root of the aggregator tree. So, to minimize the aggregation time we can use any of the many heuristics developed to partition a multiset of numbers into k subsets so that the size of the largest subset is minimum. This k-partitioning problem is identical to the well studied problem of scheduling k identical and independent machines so as to minimize the finish time. In this paper, we focus on the LPT (longest processing time) rule and the COMBINE algorithm whose approximation ratios have been derived in [4]–[6].

A. *Longest Processing Time First (LPT)*

The LPT rule [4], [5] assigns jobs to machines one job at a time. Jobs are assigned in decreasing order of their length (i.e., processing time). When a job is considered for assignment, it is assigned to that machine on which it will finish earliest (i.e., the machine for which the sum of the processing times of the jobs so far assigned is least). In the context of SANTO, the LPT rule becomes: (a) assign racks to the subtrees of the aggregator one at a time, (b) assign racks to subtrees in decreasing order of d_i, (c) when a rack is considered for assignment, assign it to the subtree for which the sum of the d_is of the racks already assigned to it is least. Once the partitioning into subtrees has been done, the racks assigned to individual trees may be organized using any topology for that subtree. The approximation ratio for the LPT rule is $(4/3 - 1/(3k))$ and this bound is tight [4], [5]. Note that this bound applies not only to the machine scheduling problem for which it was originally developed but also to SANTO. Comparing the performance of the LPT rule and that of the algorithm of Wang et al. [3], we see that when $k = 2$, the LPT rule may generate aggregator trees whose aggregation time is at most 16.7% more than optimal while the aggregation time using the algorithm of Wang et al. [3] may be as much as 50% more than optimal. When, $k = 10$, these percentages become 30% and 450%! Interestingly, there is no SANTO instance for which the algorithm of Wang et al. [3] results in a smaller aggregation time than obtained using the LPT rule. This is established below.

Lemma 1. *Let D_i be the sum of the d_is for racks assigned to subtree i, $1 \leq i \leq k$ of an aggregator tree T. Suppose that kj, $j \geq 1$, new racks are added to these subtrees using the LPT rule. Let the amount of data to be transmitted from these new racks to the aggregator, respectively, be $b_1 \geq b_2 \geq \cdots \geq b_{kj}$ and let D'_i be the sum of the data associated with the racks in subtree i following the assignment of these new kj racks. Then,*

$$max\{D'_1, D'_2 \cdots D'_k\} \leq max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{j} b_i$$

Proof: We use induction on j.
Induction Base. $j = 1$ and $b_1 \geq b_2 \geq \cdots \geq b_k$ are added to T using the LPT rule. We have the following cases:

1) *At least one subtree of T gets no new rack*: Let X be the value of D_i for one of the subtrees of T to which none of the b_is are assigned by LPT. Since, LPT assigned no new rack to this subtree, it follows that
$$\begin{aligned}\max\{D'_1, D'_2 \cdots D'_k\} &\leq X + b_k \\ &\leq \max\{D_1, D_2 \cdots D_k\} + b_k \\ &\leq \max\{D_1, D_2 \cdots D_k\} + b_1\end{aligned}$$
2) *Each subtree of T gets a new rack*: Since there are k subtrees and k new racks, each subtree gets assigned

exactly one new rack by LPT. So,

$$\max\{D'_1, D'_2 \cdots D'_k\} \leq \max\{D_1, D_2 \cdots D_k\} + b_1$$

Induction Hypothesis. Assume that the lemma is true for $j = q \geq 1$.

Induction Step. We show that the lemma is true when $j = q + 1$. Let D'_i be the sum of data in the racks assigned to subtree i following the addition of $b_1 \geq \cdots \geq b_{kq}$ to tree T. From the induction hypothesis, it follows that

$$\max\{D'_1, D'_2 \cdots D'_k\} \leq \max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{q} b_i$$

Let T' be the aggregation tree following the addition of these kq racks using LPT. Now, add the remaining k racks to T' using LPT. The associated bs are $b_{kq+1} \geq b_{kq+2} \cdots \geq b_{kq+k}$. Let D''_i be the sum of the data associated with the racks in subtree i following the assignment of these k racks. We consider the same two cases as in the induction base.

1) *At least one subtree of T' gets no new rack*: Let X be the value of D'_i for one of the subtrees of T' to which none of the k new b_is is assigned by LPT. Since, LPT assigned no new rack to this subtree, it follows that

$$\begin{aligned}
& \max\{D''_1, D''_2 \cdots D''_k\} \\
\leq\ & X + b_{kq+k} \\
\leq\ & \max\{D'_1, D'_2 \cdots D'_k\} + b_{kq+1} \\
\leq\ & \max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{q} b_i + b_{q+1} \\
\leq\ & \max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{q+1} b_i
\end{aligned}$$

2) *Each subtree of T' gets a new rack*: Since T' has k subtrees and k new racks are added, each subtree gets assigned exactly one new rack by LPT. So,

$$\begin{aligned}
& \max\{D''_1, D''_2 \cdots D''_k\} \\
\leq\ & \max\{D'_1, D'_2 \cdots D'_k\} + b_{kq+1} \\
\leq\ & \max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{q} b_i + b_{q+1} \\
\leq\ & \max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{q+1} b_i
\end{aligned}$$

■

Lemma 2. *Let k, $d_1 \geq \cdots \geq d_n$, where $n = k((k-1)^j - 1)/(k-2)$ for some integer $j \geq 1$, be an instance of SANTO. Let $D_W(j)$ and $D_L(j)$ denote the maximum data in any subtree of the aggregation tree generated by the algorithm of Wang et al. [3] and the LPT rule, respectively. $D_L(j) \leq D_W(j), j \geq 1$.*

Proof: Note that when $n = k((k-1)^j - 1)/(k-2)$, the aggregation tree constructed by the algorithm of Wang et al. [3] has exactly $j + 1$ levels and that each level is full (i.e., the root has k child racks, each child of the root has $k-1$ children racks and $(k-1)^2$ grandchildren racks, and so on). We prove $D_L(j) \leq D_W(j), j \geq 1$, by induction on j.

Induction Base. $j = 1$. Both algorithms assign exactly one rack to each subtree of the aggregator tree. So, $D_W(1) = D_L(1) = d_1$.

Induction Hypothesis. Assume that the lemma is true for $j = q$, where q is some integer ≥ 1.

Induction Step. We show that the lemma is true for $j = q+1$. Let $d'_1 \geq d'_2 \geq \cdots \geq d'_{ks}$, $s = (k-1)^q$ be the last racks (i.e., racks with the smallest data) in the SANTO instance. The algorithm of Wang et al. [3] places these racks at the lowest level of the aggregation tree. It is easy to see that

$$D_W(q+1) = D_W(q) + \sum_{i=1}^{s} d'_i$$

Let D_i be the amount of data in the racks assigned by LPT to subtree i of the aggregation tree following the assignment of the first $n - ks$ racks. From the induction hypothesis, it follows that

$$D_L(q) = \max\{D_1, D_2 \cdots D_k\} \leq D_W(q)$$

Using Lemma 1, we get

$$\begin{aligned}
D_L(q+1) &\leq max\{D_1, D_2 \cdots D_k\} + \sum_{i=1}^{s} d'_i \\
&\leq D_W(q) + \sum_{i=1}^{s} d'_i \\
&= D_W(q+1)
\end{aligned}$$

■

Theorem 3. *For every SANTO instance the aggregation time using the LPT tree is less than or equal to that using the aggregation tree generated by the algorithm of Wang et al. [3].*

Proof: For SANTO instances with number of racks equal to $n = k((k-1)^j - 1)/(k-2)$ for some integer $j \geq 1$, the theorem follows from Lemma 2. When other values of n, we add fictitious racks with 0 data so that the total number of racks (actual plus fictitious) equals $k((k-1)^j - 1)/(k-2)$ for some integer $j \geq 1$. The addition of these fictitious racks does not alter the behavior of either LPT or the algorithm of Wang et al. [3] and the aggregation tree obtained by these algorithms for the original instance may be obtained from that for the augmented instance by simply discarding the fictitious racks from the aggregation tree for the augmented instance. This discarding of the fictitious racks does not alter the aggregation time. From Lemma 2, we know that the LPT aggregation time is no more than the Wang et al. aggregation time for the augmented instance. Hence, the same is true for the original instance. ■

B. COMBINE

The k-machine scheduling algorithm of Lee at al. [6] uses the better of the schedules obtained by LPT and the Multifit scheduling algorithm of Coffman et al. [7]. Multifit

Algorithm 1 COMBINE Algorithm [6]

```
Input: $n$ racks with $d_1, d_2, \cdots d_n$ data and $k$.
Output: A topology of the aggregation tree that minimizes
    aggregation time.
1: Apply LPT and let $T$ be the aggregation time.
2: if $M \geq 1.5A$ then
3:     Stop.
4: end if
5: Apply Multifit in interval $high = T$ and $low = \max\{A, max_i\{a_i\}, T/(4/3 - 1/(3m))\}$.
```

schedules the k machines by using the first-fit-decreasing (FFD) heuristic for bin packing. This is done by attempting to pack the jobs into k bins of size S using FFD. Note that S represents the target finish time of the schedule. To find the smallest S for which such a packing is possible using FFD, a binary search for the smallest S is done in the interval $\max\{A, max_i\{a_i\}\}$ and $\max\{2A, max_i\{a_i\}\}$, where a_i is the duration of job i and $A = \sum a_i/k$. Coffman et al. [7] have shown that the approximation ratio of Multifit is $8/7$ and that, in practice, 7 iterations of the binary search are sufficient. We describe COMBINE in Algorithm 1. By its very nature, COMBINE guarantees schedules at least as good as those produced by LPT. From Theorem 3, it follows that the aggregation topologies obtained by COMBINE when used to solve SANTO are at least as good as those obtained by the algorithm of Wang et al. [3]. The approximation ratio of COMBINE is proved to be $10/9$ for two machines in [6]. Hence, using COMBINE in place of LPT gives us a better approximation ratio for SANTO at the expense of increased run time. Note that when Multifit is limited to 7 iterations, its asymptotic complexity is the same as that of LPT but the actual run time is larger by a constant factor.

VII. SANTO-NT

In SANTO-NT, the primary objective is to minimize aggregation time and the secondary objective is to minimize total network traffic. We propose using Algorithm 2 for SANTO-NT. This algorithm first obtains a k-partition of the racks using either LPT or COMBINE. The use of this good partition guarantees an aggregation time with an approximation ratio corresponding to that of either LPT or COMBINE. Next, for each partition, the aggregation subtree is obtained by placing racks with more data closer to the root as is done by the algorithm of Wang et al. [3].

Theorem 4. *For every k-partition, the total network traffic is minimized by placing racks with more data as close to the root as possible as is done by the algorithm of Wang et al. [3].*

Proof: The total network traffic is $\sum d_i h_i$, where d_i is the data to be sent by rack i to the aggregator and h_i is the number of hops from rack i to the root of the aggregation tree. Within each partition, the algorithm of Wang et al. [3] places the maximum possible number of racks at level 1, then the maximum possible number at level 2, and so on. Further, the placement is done in decreasing order of d_i. A proof by induction on the number of racks in a partition establishes the minimality of the network traffic generated by the tree constructed for that partition. ■

Algorithm 2 Algorithm for SANTO-NT

```
Input: $n$ racks with $d_1, d_2, \cdots d_n$ data and $k$.
Output: A topology of the aggregation tree that minimizes
    aggregation time and total network traffic.
1: Use either LPT or COMBINE to $k$-partition the racks.
2: Use the SANTO algorithm of Wang et al. [3] to arrange
    the racks in each partition into a tree of degree $k - 1$.
3: Make the constructed $k$ trees the subtrees of the aggregator.
```

We note that the proof of Theorem 4 also establishes the optimality of the algorithm of Wang et al. [3] for minimizing total network traffic. Consequently, the total network traffic of the aggregation tree generated by Algorithm 2 cannot be less than that of the aggregation tree generated by the algorithm of [3]. This is not a concern as our primary objective is to minimize aggregation time and not total network traffic.

As an example, consider the SANTO-NT instance $n = 10$, $k = 3$ and $d_{1:10} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Figure 4(a) shows the aggregation tree generated using the algorithm of Wang et al. [3]. The node labeled A (i.e., the root) is the aggregator. Its aggregation time is $24/B$ and the total network traffic is 84 units. A possible 3-partition obtained by LPT is (we say possible because different partitions are possible depending on how we implement the tie breaker) $\{10, 5, 4\}$, $\{9, 6, 3\}$, and $\{8, 7, 2, 1\}$. No matter how the 3 subtrees of the aggregation tree are configured the aggregation time is $19/B$ so long as the racks are assigned to the subtrees using this partition. Configuring the subtrees as chains with racks assigned in decreasing order of d_i top-to-bottom gives us the aggregation tree of Figure 4(b) for which the total network traffic is 94. When the subtrees are configured according to the algorithm of Wang et al. [3], the aggregation tree is as in Figure 4(c). The total network traffic for this aggregation tree is 84.

VIII. Experiments

We conducted experiments to assess the reduction in aggregation time that could be expected in practice using either LPT or COMBINE instead of the algorithm of Wang et al. [3]. We also assessed the reduction in the total network traffic when we use the Wang's algorithm in combination with LPT and COMBINE.

Depending on the application, the data distribution among the racks could range from fairly uniform to widely variable. For example, when the racks are reporting the frequency of

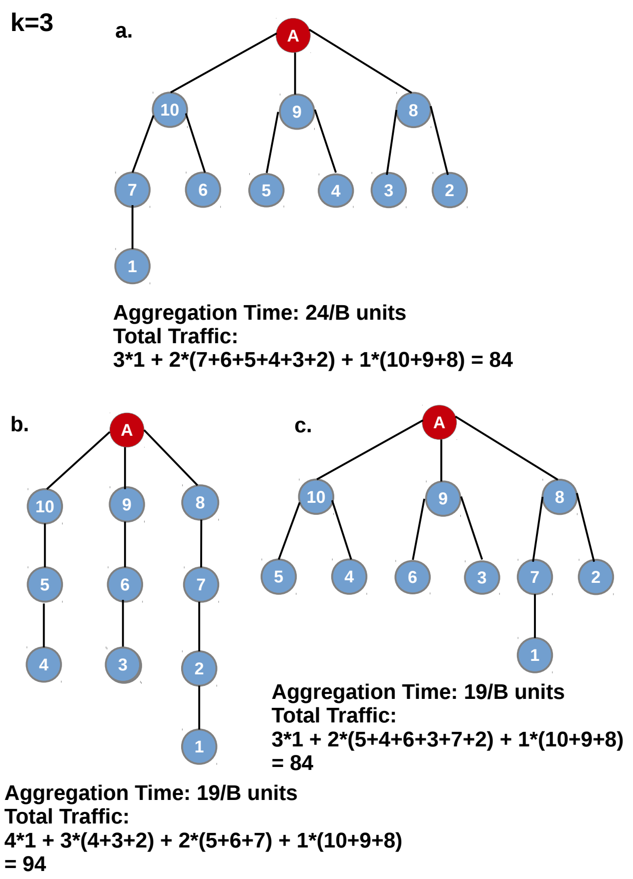
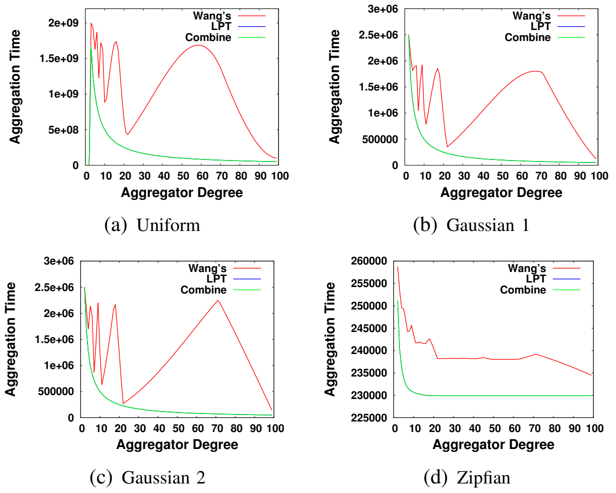


Figure 4. Total Traffic in Network

occurrence of each entry in a list of keywords in a target text corpus that has been distributed across the racks, each rack sends to the aggregator a fixed-size list of counts. On the other hand when the racks are returning to the aggregator the records in their local databases that satisfy a query, the size of data being transmitted could vary widely from one rack to the next. In an attempt to assess relative performance across the range of data that aggregation instances may exhibit, we use the following data sets:

- *Uniform.* The d_is are drawn from a uniform distribution with values in [1,1000000].
- *Gaussian 1.* The d_is are drawn from a truncated Gaussian distribution with mean 500 and standard deviation 1000 truncated in $[200, 800]$.
- *Gaussian 2.* The d_is are drawn from a truncated Gaussian distribution with mean 500 and standard deviation 1000 truncated in $[400, 600]$.
- *Zipfian.* The d_is are drawn from a Zipfian distribution with parameter 2.

We use a link bandwidth of 1 so that the data transmission time equals the amount of data transmitted through a link. As we have mentioned earlier, this does not affect our results as we analyze percentage change in the values of aggregation time and total network traffic. All our test data



(a) Uniform (b) Gaussian 1 (c) Gaussian 2 (d) Zipfian

Figure 5. Average Aggregation Time with Aggregator Degree

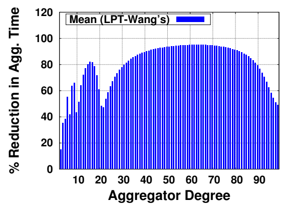
have $n = 10000$ racks and the number, k, of optical links per rack is varied between 2 and 100. For each choice of k and the data distribution, we experimented with 20 randomly generated instances and report the average aggregation time. Our experiments were conducted on a 64-bit PC with a 2.80 GHz AMD Athlon(tm) II X2 B22 processor and 8GB RAM.

A. *Aggregation Time*

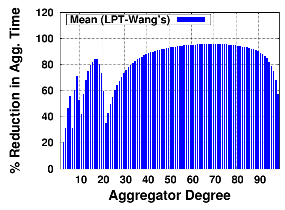
Figure 5 compares the average aggregation times of the aggregator trees constructed by the 3 algorithms with the degree of the aggregator (k). As expected, the aggregation time using the algorithm of [3] is always greater than that using the LPT or COMBINE algorithms. Further, the anomalous behavior of the algorithm of [3] noted in Section V is exhibited on our test data. However, neither LPT nor COMBINE exhibits anomalous behavior. Note that in the absence of anomalous behavior, an increase in k will not increase the aggregation time. Figure 6 shows the percentage reduction in aggregation time resulting from the use of LPT compared to the algorithm of [3]. LPT reduces aggregation time by as much as 90% for the random and the two Gaussian data-sets. However the Zipfian dataset shows a lower reduction of upto 50%. Figure 7 shows the percentage reduction in aggregation time obtained using the aggregation trees generated by COMBINE compared to those generated by the LPT. For our data sets, this reduction was at most 0.4 to 0.5% for the 2 Gaussian data-sets the random and the Zipfian data-sets however show no improvement and are not shown.

B. *Total Network Traffic*

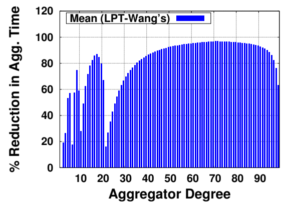
Figure 8 shows the percentage reduction in the total network traffic using the optimal aggregation tree topology for each of the k-partitions generated by LPT and COMBINE as in Algorithm 2, compared to that generated when we configure each partition as a chain from top to bottom in decreasing order of the d_is, in that partition. The reduction varies from a low of about 60% to a high of about 90%.



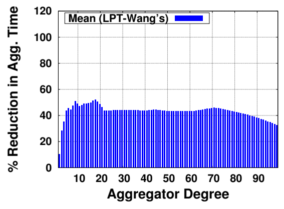
(a) Uniform



(b) Gaussian 1

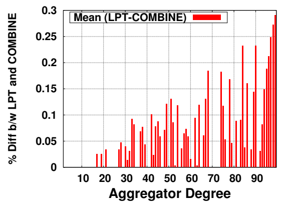


(c) Gaussian 2

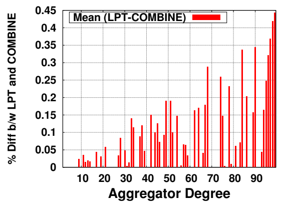


(d) Zipfian

Figure 6. Percentage Reduction in Average Aggregation Time of LPT relative to [3]



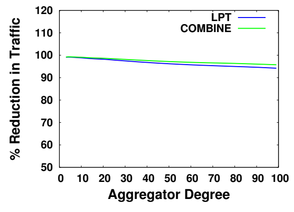
(a) Gaussian 1



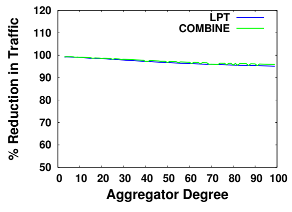
(b) Gaussian 2

Figure 7. Percentage Reduction in Average Aggregation Time of COMBINE relative to LPT

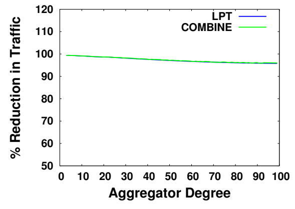
More interesting is the cost, in terms of increased network traffic, of providing the reduced aggregation times using the aggregation trees of LPT and COMBINE over using an aggregation tree that minimizes total network traffic (i.e., using the aggregation tree generated by the algorithm of [3]). Figure 9 shows this increase. COMBINE has a maximum increase of around 90% for the Zipfian dataset, 9% for the Gaussian1 data-set and even lower for the other data-sets.



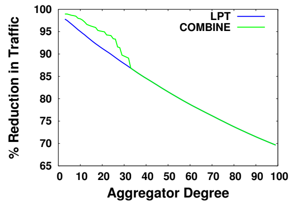
(a) Uniform



(b) Gaussian 1

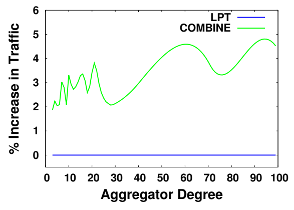


(c) Gaussian 2

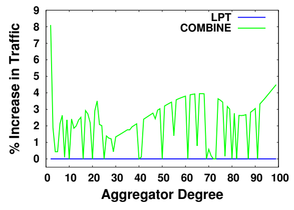


(d) Zipfian

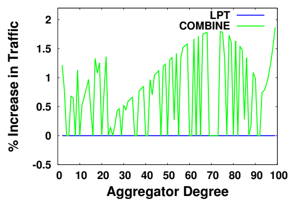
Figure 8. Percentage Reduction in Network Traffic for Algorithm 2 relative to Chains



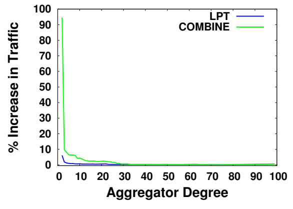
(a) Uniform



(b) Gaussian 1



(c) Gaussian 2



(d) Zipfian

Figure 9. Percentage Increase in Network Traffic Algorithm 2 relative to [3]

LPT however shows little increase in total network traffic- a maximum of 6% for the Zipfian dataset and almost no increase for the other datasets.

IX. Conclusion

We have shown that SANTO and SANTO-NT are NP-hard even when the number of aggregators is 1. Since SANTO is identical to the independent machine scheduling problem, we can use the algorithms developed for the latter problem to solve SANTO. In this study, we have focused on using the scheduling algorithms LPT and COMBINE and compared their performance to that of the algorithm proposed by Wang et al. [3]. We have derived a tight approximation ratio, $(k+1)/2$ for the algorithm of [3]. By comparison, the approximation ratio for LPT is $(4/3-1/(3k))$. Further, LPT never generates an aggregation tree that is inferior to that generated by the algorithm of [3]. For SANTO-NT, we propose using a good algorithm for machine scheduling to obtain a k-partition of the racks and then using the algorithm of [3] to place the racks in each partition into a subtree of the aggregation tree. The placement of racks in this way minimizes total network traffic subject to the constraints of the obtained k-partition. Experiments conducted by us using the LPT and COMBINE scheduling algorithms indicate a reduction in aggregation time of up to 90%. This reduction in aggregation time is accompanied by an increase in total network traffic of up to 9% relative to using the algorithm of [3] for COMBINE. On the other hand LPT generally does not show an increase in the traffic except for few extreme data-sets as illustrated in section VIII.

Acknowledgment

This research was supported, in part, by the National Science Foundation under grants CNS0829916 and CNS0905308.

References

[1] Apache Hadoop, *http://hadoop.apache.org*.

[2] Apache HBase, *http://hbase.apache.org*.

[3] Wang, Guohui and Ng, T.S. Eugene and Shaikh, Anees, *Programming your network at run-time for big data applications*, Proceedings of the first workshop on Hot topics in software defined networks HotSDN '12, 2012.

[4] R. L. Graham, *Bounds on Multiprocessing Timing Anomalies*, SIAM JOURNAL ON APPLIED MATHEMATICS, 1969, volume 17, number 2, pages 416–429.

[5] Coffman,Jr., E. G. and Sethi, Ravi, *A generalized bound on LPT sequencing*, Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation, SIGMETRICS '76, 1976.

[6] Chung-Lee and David Massey, *Multiprocessor scheduling: combining LPT and MULTIFIT*, Discrete Appl. Math., July, 1988.

[7] Jr., Edward G. Coffman and Garey, M. R. and Johnson, David S., *An Application of Bin-packing to Multiprocessor Scheduling*, SIAM J. Comput.

[8] Greenberg, Albert and Lahiri, Parantap and Maltz, David A. and Patel, Parveen and Sengupta, Sudipta, *Towards a next generation data center architecture: scalability and commoditization*, Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, PRESTO '08, 2008.

[9] Al-Fares, Mohammad and Loukissas, Alexander and Vahdat, Amin, *A scalable, commodity data center network architecture*, Proceedings of the ACM SIGCOMM 2008 conference on Data communication, SIGCOMM '08, 2008.

[10] Guo, Chuanxiong and Wu, Haitao and Tan, Kun and Shi, Lei and Zhang, Yongguang and Lu, Songwu, *Dcell: a scalable and fault-tolerant network structure for data centers*, Proceedings of the ACM SIGCOMM 2008 conference on Data communication, SIGCOMM '08, 2008.

[11] Das, S. and Yiakoumis, Y. and Parulkar, G. and McKeown, N. and Singh, P. and Getachew, D. and Desai, P.D., *Application-aware aggregation and traffic engineering in a converged packet-circuit network*, Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference, 2011.

[12] Webb, Kevin C. and Snoeren, Alex C. and Yocum, Kenneth, *Topology switching for data center networks*, Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Hot-ICE'11, 2011.

[13] Kavulya, S. and Tan, J. and Gandhi, R. and Narasimhan, P., *An Analysis of Traces from a Production MapReduce Cluster* Cluster Cloud and Grid Computing (CCGrid), 2010.

[14] Benson, Theophilus and Akella, Aditya and Maltz, David A., *Network traffic characteristics of data centers in the wild*, Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC '10, 2010.