Congestion Control and Energy Conservation of Data Centers

Sina Astani

April 5, 2018

1 Abstract

"Swarm-based Incast Congestion Control in Datacenter Serving Web Applications" and "On Energy Conservation in Data Centers describe algorithms which attempt to solve two major issues related to data centers: response time and energy use. Data centers can be roughly defined as facilities that contain networks of computers, which are used to "organize, process, store, and disseminate data"[1]. Given the large number of servers which are intertwined, these issues can be quite complex.

2 Introduction

Response time is particularly important for web applications such as search engines, social networks, and e-commerce websites which are expected to provide results in a timely matter to users, despite the large number of requests. Latency is crucial to companies like Amazon, where an increase in latency of 100 milliseconds can effect sales by as much as one percent. Traffic congestion in the topologies of these data centers is better known as incast congestion. In their paper, Wang, Shen, and Liu suggest a system for dealing with incast congestion.

Energy conservation is another issue related to data centers. Its estimated that data centers use as much as 1.5% of all electricity that is used around the world, while servers are only used 20-40% of any 24 hour period. It is likely that companies with less web traffic have server usage towards the lower end of this spectrum. A way to deal with this problem is to switch servers from an active to inactive state, whenever they are not being used and from an

inactive to active state whenever they are needed. Energy conservation of servers is essentially an optimization problem of minimum cost flow.

3 Description of Problem

A traditional model of a data center consists of a network of one front-end server and many back-end data servers. The front-end server is a single server which receives requests from users and forwards the requests to the many back-end servers, which are typically databases. Thus, one server is connected to many different servers at the same time. Incast congestion is an issue that occurs when there are a high number of concurrent requests by the front end to the backend, typically resulting in packet loss and an overall increase in client response time [2]. Several methods for dealing with incast congestion have been previously presented. The first of these is the sliding window protocol. Essentially, the sliding window protocol works by establishing a limit on number of packets that can be sent and received [3] between the two types of servers at any time. Wang, Shen, and Liu note that this is not an effective approach as it does not meet low latency requirements. Other methods include the data reallocation and data replication methods, but neither of these methods take into account the issues that can occur with web applications that serve hundreds of thousands of users, particularly the additional delay or overhead that they introduce. The authors formulate their own methodology called *SICC*, which is composed of three different methods.

The energy use of data centers, which often consist of servers with states of varying power consumption rates, is another problem. Servers consume a measurable amount of energy with each power up and save a measurable amount with each power down (to either an idle or sleep state), which are the two types of state transitions. In order to save energy, data centers need to match the consistently changing demand of computing resources with the appropriate number of active servers. Prior research to this problem did not account for changing demands or servers of differing power consumption rates. The authors form an optimization problem called DPM, or dynamic power management, which is similar to a minimum cost flow problem as a means to finding a solution.

4 Model and Assumptions

We will first consider *SICC*, which modifies the typical configuration of one to many communication between a front end server and back end server by introducing a data server called a hub. These sort of act as a middle man for a swarm, forwarding data from the back end servers to the front end server. *SICC* works by combining all target data servers that are in the same rack into a swarm. It in turn decreases the number of data servers connected to the front end at the same time.

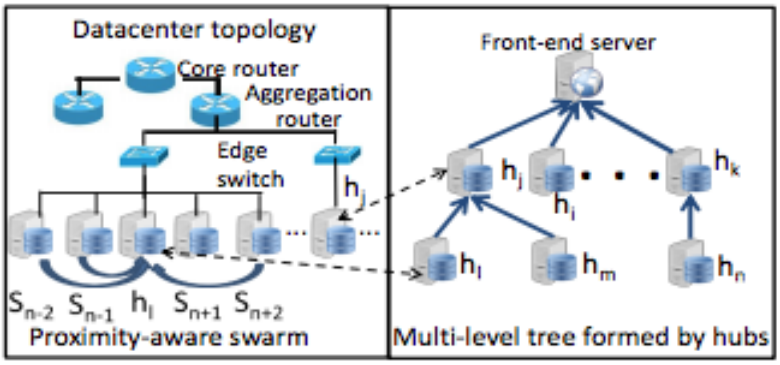


Figure 2: The multilevel tree with proximity-aware swarms.

The diagram above depicts how hubs are formed. The left shows how the servers, as well as network devices, are typically arranged in a data center, while the right shows the tree structure of hubs that is formed.

Next we consider the model for conserving energy in data centers. The authors define $S = S_1, ..., S_m$ as the set of heterogenous servers with $s_{i,0}, ..., s_{i,\sigma_i}$ as the states of some server, S_i, in the set. Each problem instance of DPM, I, consists of a set of servers, S and changing computing demands, D over some time interval: $I = (S, D)$. D is officially the *demand profile* over a time interval. DPM considers optimal schedules, which specify the state of each server in the set (S_i) at some time t, for each problem instance. A state transition (a change from power up to power down, or the opposite) only occurs if there is a change in necessary computing resources, which is one of the assumptions. The other assumption is that a server never makes a transition while its in a sleep or inactive state.

For servers with only two states, DPM is relatively simple.The authors state that an optimal solution, or schedule, for a problem instance, I can be

found in polynomial time. For each problem instance, there is a network $N(I)$ that consists of components C_i, where one exists for each server (see below). The components each have an upper path and a lower path, which are the active and inactive states, respectively. State transitions are represented as directed edges between upper and lower paths.

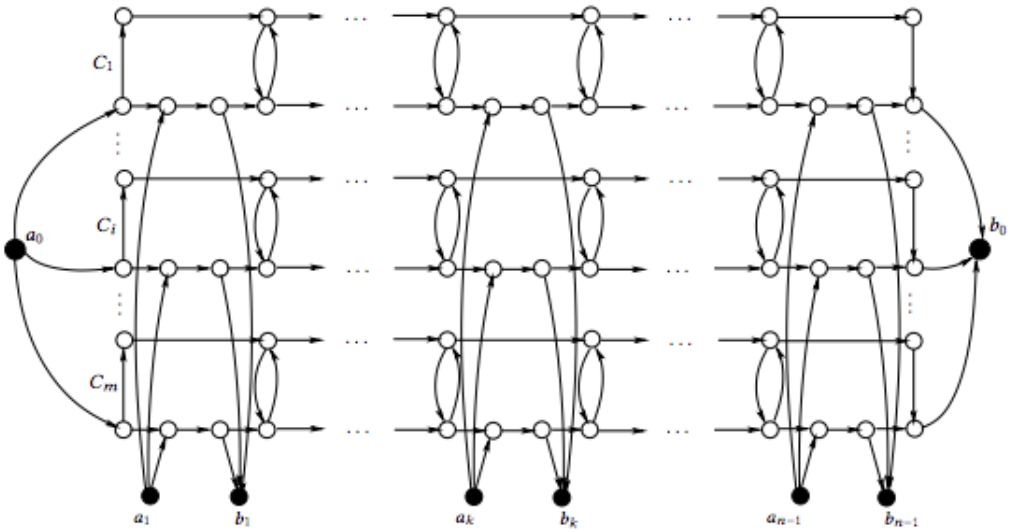
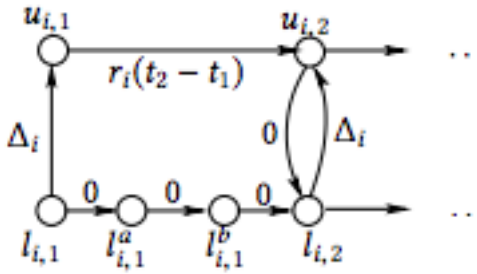


Figure 2: The network $N(I)$

The diagram below depicts a component for some server during its initial time interval:



As can be seen above, a transition from an upper path, u_i, k, to a lower path, l_i, k or a power down, has a cost of 0, while the opposite, a power up, has a cost Δ_i. In this diagram, $k = 1$, but k can equal between 1 and m.

DPM for servers with multiple states in relatively more complex. Th key differences are S_i represents a server type or class, not a single server and each component, and C_i, corresponds to a different server class. In each class, there is a total of m_i independent and asynchronous servers. The

authors propose that the schedule for problem instance can be calculated "in polynomial time based on a min-cost two-commodity flow computation".

5 Results

The first part of *SICC*, the most important part, is *proximity aware based data transmission*. This is the formation of the multilevel tree structure of hubs mentioned previously. It is important to note that this structure can consistently change and is not set in stone. When a swarm is created, a data server is chosen as a hub based on its ability to handle I/O as well as another server in the rack as a backup, in case the original hub fails. The algorithm for creating this structure is as follows:

Algorithm 1: Building a multi-level tree from hubs.

```
1  Cluster target data servers in each rack into a swarm;
2  /*Selecting a hub from each swarm*/
3  for each swarm do
4      Select the data server with the largest number of requested
       data objects as the hub; Enqueue the hub in to queue Q_h;
5  Sort the hubs in Q_h in an ascending order of the number of
   stored requested data objects;
6  /*Creating multi-level tree from the hubs*/
7  while |Q_h| > N do
8      Dequeue a hub h_i from Q_h;
9      Select a
         hub h_j with the smallest number of data objects and under
         the same aggregation router as h_i; Link h_i as a child to h_j;
10     while h_j has less than N children and h_i has children do
11         Transmit the last child from h_i to be a child of h_j;
12     Update h_j's number of requested data objects by add h_i's;
13     Update h_j's position at Q_h accordingly;
```

The key calculation in this algorithm is to determine the number of hubs, N, for a request. It is found by $N = M/m$ where M is the largest number of possible hubs connected to the front-end server divided by m or the average number of requests sent from the front-end server. If more than N hubs are connected there is a tendency for incast congestion.

The next two parts of *SICC* are *two level data transmission speed control* and *packet compression and object query redirection*. The first of these is

a way to avoid incast congestion which can still occur due to bandwidth overflow by setting the speed of data transmission of each hub. The latter of these involves putting multiple data objects into one packet, since data objects tend to be small while the payload of a packet is much higher.

Next, we will look at DPM as applied to a network of problem instances of servers with two states. It considers feasible flows in the network of each problem instance and finds a flow which matches up to a feasible schedule and is also optimal. While all flows are integral (i.e. the flow along edges are always an integer values), not all flows are consistent. Consistency occurs when the sum of the flows of a parallel upper path and lower path (two paths during the same time interval) equals zero and integral flows are consistent only when they are consistent for all time intervals.

The authors look at algorithms which turn inconsistent flows into consistent ones. The first algorithm, A_1 modifies flows in two ways. The first modifies flow such that the node right before the last node in the lower path of a component is rerouted to a sink, b_k as seen in the diagram below.

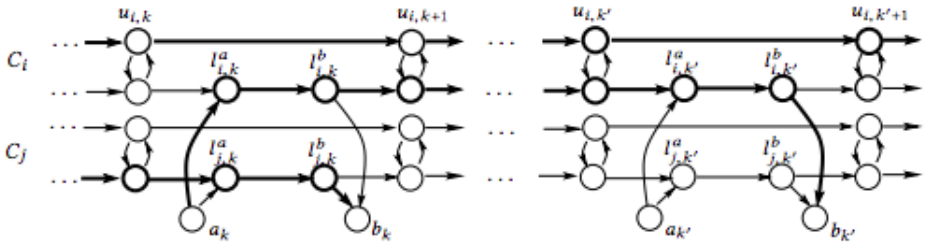


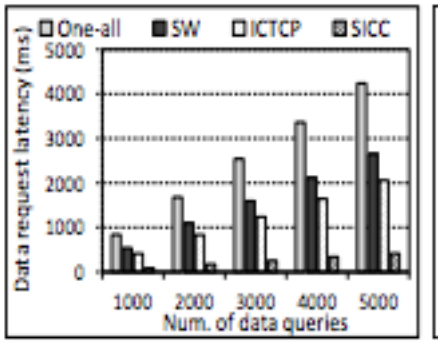
Figure 3: A flow that is not consistent in $[t_{k+1}, t_{k+2})$.

The second redirects the flow that goes into the last node of the lower path, l_j, k. It now goes to the sink, b_k.

The second algorithm of DPM deals with servers of multiple states. Feasible flows can now be fractional, instead of integral. There are three steps to modifying flow: flow scaling, rounding flow of upper paths, and packing flow on lower paths. Scaling multiplies the minimum-cost flow by $/tau$ or the number of components. Rounding occurs when flows on the upper path of a component are turned into integral flows, while packing turns flows on lower paths of a component into integral flows.

6 Conclusion

In their paper, Wang, Shen, and Liu compare *SICC* with previous methods for dealing with incast congestion. Compared to one-all, sliding window protocol, and ICTCP, *SICC* incurs much less latency than these approaches. This difference is exaggerated when the number of data queries increases. The performance they observed in their simulation of a datacenter of 3000 dataservers is very similar to the performance on an actual testbed, which can be seen below.



(a) Data request latency

Thus it can be concluded that Swarm-based Incast Congestion Control, or *SICC* is the best solution to this problem.

Albers' solution for DPM, or dynamic power management, cannot be compared in the same way, as previous work did not consider changing computing demands. Her paper has more of a theoretical approach and defines two methods of finding schedules for switching servers from active to inactive states in order to conserve energy. The idea with both of these methods is to find a minimum cost flow by constructing networks for each set of servers or server types and their corresponding demands.

7 References

[1] Roger Godinho and Stephen Bigelow. *Data Center*. `https://searchdatacenter.techtarget.com/definition/data-center`.

[2] Centre for Advanced Internet Architectures. *Incast congestion control*. `http://caia.swin.edu.au/urp/incast/`.

[3] Susanne Albers. *On Energy Conservation in Data Centers*. SPAA '17 Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures Pages 35-44. July 24 - 26, 2017.

[4] Haoyu Wang, Haiying Shen, and Guoxin Liu. *Swarm-based Incast Congestion Control in Datacenter Serving Web Applications*. In textitProceedings of SPAA 17, Washington DC, USA, July 24-26, 2017.