Aster*x: Load-Balancing as a Network Primitive

Nikhil Handigol
Stanford University
nikhilh@stanford.edu

1. INTRODUCTION

With the emergence of large web applications in the late 1990s, there was much interest in load-balancing to spread incoming requests across a set of identical web servers. Usually, they exploit some trick in the network (e.g. DNS, anycast, etc) to make it work without altering the network logic [5]. Many commercial load-balancing products have been built that sit on the path of incoming requests and spread them over a set of servers [2]. Load-balancing is used increasingly for other tasks beyond balancing web requests too. For example, it is used in CDNs for serving content from multiple servers. It has become a commonly used element of all scale-out network services.

Current load-balancing methods make a number of assumptions about the services:

- Requests enter through a single point in the network; the load-balancing device is placed at a choke point through which all traffic must pass. For networks where this condition does not hold true, operators end up using many of these expensive devices. Yet, they create congestion in the network. Figure 1 shows a typical datacenter architecture [1] with load balancers placed at choke points.
- The network structure is regular. In practice, it may be (e.g. a datacenter); but enterprise networks are not.
- The congestion is at the servers, not within the network. Again, this may be true in datacenters hosting only one service. In a cloud datacenter with many services, the network may be congested differently in different places. Often datacenters don't have full bisection bandwidth and can be congested. In an enterprise network, there can be many choke points, like egress connections to the WAN, campus backbones, etc.
- The servers are static. Operators of virtualized datacenters move VMs around to make efficient use of their servers. As VMs move, the load-balancers need to track their location so as to direct new requests to the right place.
- The network load is static. Most load-balances spread traffic obliviously through the network, using static schemes like ECMP. This is suboptimal when some parts of the network are congested.
- All services need the same load-balancing algorithm. This means the service provider has to use the same scheme for load-balancing, say, HTTP requests and video requests; or he has to install two load-balancers. In virtualized data centers this will be much harder as more services will be deployed by different users, and they will be moving around.

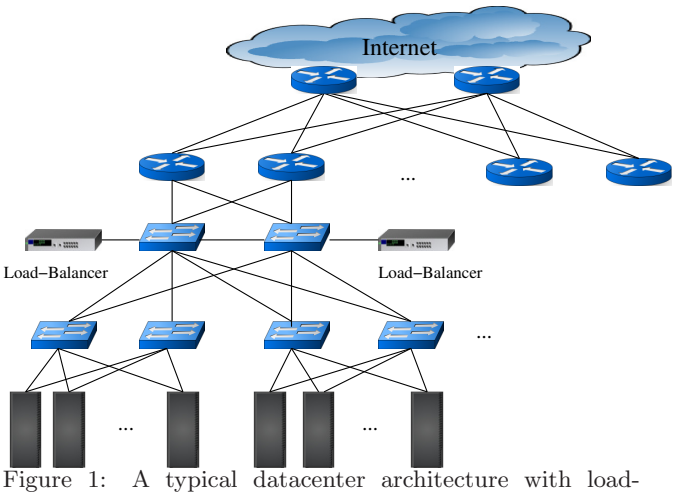

Figure 1: A typical datacenter architecture with load-balancers placed at choke points.

Our work is premised on the following observation: load-balancing is essentially congestion-aware routing (based on network and server congestion). It, therefore, leads us to the belief that load-balancing should be an integral property of the network. If we think of the network datapath as implementing a basic small set of "plumbing primitives" (e.g. forward a packet to one or more ports) then load-balancing fits nicely into this model. It just means that the datapath has to intelligently (and dynamically, and quickly) decide which outgoing port to send a request to, and make sure all the packets associated with the request follow the same path to the same server. We therefore seek a solution with the following characteristics:

- **Distributed**: If every switch in the network can load-balance incoming requests, we don't need to re-design the network to accommodate the load-balancers – it simply becomes a property of the network. We can do away with choke points and constrained routing. It is then naturally scalable too - just add more switches.
- **Dynamic**: It must react to network and server congestion, and pick routes and servers accordingly.
- **Auto-configured**: It must automatically adapt and scale when servers and network capacity are added or deleted. It must continue to operate as servers move around (e.g. VM migration).
- **Flexible**: It must balance load in a way that is optimized for each service or application; the service creator must be able to decide how load is balanced. This means new algorithms must easily be created, tested and deployed.

We describe *Aster*x*, a prototype distributed load-balancer. Aster*x is premised on the belief that every switch and router should easily be able to do load-balancing, and that it is cheap to do so. Our approach builds on the growing trend towards "software-defined networking", such as OpenFlow [4]/NOX [3]. In these approaches, the network switches are treated as a dumb, minimal flow-based datapath, under the control of a remote, software control plane. OpenFlow is the common, narrow, vendor-agnostic interface to the flow switches; NOX is the control plane upon which services like Aster*x are built.

Aster*x treats each individual request - or a bundle of aggregated requests - as a flow, and decides how to route the flow. The flow could be, for example, a single HTTP request, or it could be all the requests for a particular service. Aster*x can decide whether to route each individual request, or use ECMP-like oblivious load-balancing in any combination. For example, it could choose to send all HTTP requests to one pool of servers, and all video requests to another pool; and then do ECMP-like load-balancing over each pool. Or it could choose to do oblivious load-balancing over different regions of a data center, then do careful, per-request load-balancing within one region. The key here is that Aster*x can be used flexibly to define how the load-balancing is done, under the control of the service or application. It is not predefined by a fixed-function load-balancer. Aster*x has the following characteristics:

- **Distributed** throughout the network: If the switches are low-cost OpenFlow-enabled switches, then they can do load-balancing. It is therefore very scalable.
- **Logically centralized**: Load-balancing is defined in one place for the entire network – in the control plane app. It is easy to scale by replicating the control plane.
- **Flexible**: Each service can have its own load-balancing algorithm. Each type of request can be load-balanced by a combination of oblivious and intelligent (flow-by-flow) schemes depending on the needs of the service, and the capabilities of the datapath and controller. The service creator is free to decide.
- **Build now, change later**: Because the datapath supports general load-balancing, we don't need to decide how load-balancing will work when the datacenter is built; we can evolve it over time to suit the services deployed.

2. DESIGN AND IMPLEMENTATION

Aster*x uses the OpenFlow architecture to *measure* the state of the network, and to directly *control* the paths taken by requests. The load-balancing logic resides in the Aster*x controller, built as a NOX application.

One of the design goals of Aster*x is to let the service providers choose the best way to load-balance their requests based on their application type and performance metric. Some of the options that applications have in Aster*x are:

- **Pro-active vs. reactive**: The load-balancing decision can be made pro-actively, or reactively upon arrival of each request.
- **Individual vs. aggregated**: The load balancing decision can be made on individual requests, or on aggregated bundles of requests, or any combination of the two.
- **Static vs. dynamic**: The load-balancing strategy can be completely static, oblivious to the network or server load (e.g., ECMP-like uniform spreading), or load-aware and dynamic in any combination.

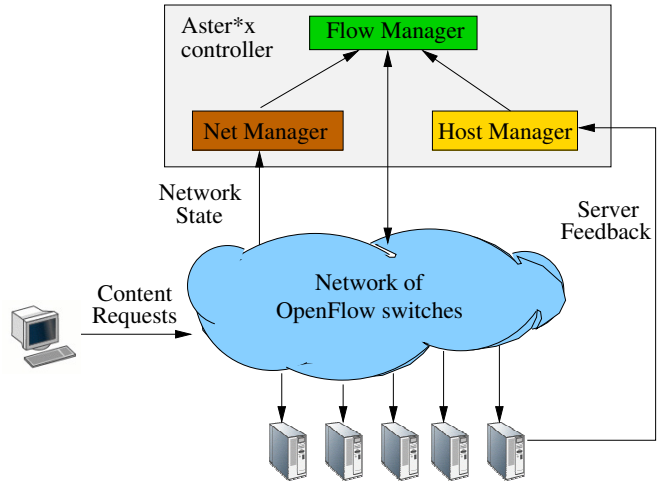

Figure 2: Design of the Aster*x controller which implements the main control logic of the whole system.

To perform load-balancing, the Aster*x controller relies on three functional units, as shown in Figure 2:

- **Flow Manager**: This module manages and routes flows based on the specific load-balancing algorithm chosen.
- **Net Manager**: This module is responsible for keeping track of the network topology and its utilization levels.
- **Host Manager**: This component tracks the individual servers in the system and monitors their state and load.

The following is a short video (8 minutes) of a live demonstration of Aster*x that we presented at the 9th GENI Engineering Conference, held at Washington DC on November 3, 2010: **http://goo.gl/1U3hg**

3. FUTURE WORK

As future work, we plan to understand the performance trade-offs of various load-balancing schemes provided by Aster*x for different types of services - HTTP, streaming media, etc. - and their combinations. We also plan to explore novel load-balancing algorithms for different networks and demand patterns.

4. REFERENCES

[1] Cisco Systems - Data center: Load balancing data center services. `https://learningnetwork.cisco.com/docs/DOC-3438`, 2004.

[2] F5 BIG-IP Local Traffic Manager. `http://www.f5.com/products/big-ip/product-modules/local-traffic-manager.html`.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, and N. McKeown. Nox: Towards an operating system for networks. In *ACM SIGCOMM CCR*, July 2008.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[5] K. Salchow Jr. Load Balancing 101: The Evolution to Application Delivery Controllers. F5 White Paper.