Applications of weakly consistent distributed data structures over shared logs

Pranav Maddi

Abstract

1 Background

I chose these SDN systems because they are open source and well documented

1.1 Shared logs

1.2 Consistency

2 Software Defined Networking

In general, SDN allows central control of a network.

2.1 OpenFlow

Dec 2

2.2 ONOS

ONOS is an open source network "operating system" that aims to have distributed control of the network. Like the other systems covered here, ONOS uses the OpenFlow control-data plane interface. The overall architecture of ONOS is composed of a network view that contains a network graph abstraction, OpenFlow managers, as well as a Zookeeper registry for coordination.

To interact with the OpenFlow interface, ONOS uses modules from Floodlight, which is another SDN implementation. This includes the switch manager, the IO loop, and link discovery. These modules as well as shared Floodlight module manager means that ONOS is slightly more compatible with other SDN controllers.

The network view data model was stored in an ONOS graph database built on top of a distributed and persistent key-value store called *RamCloud*. RamCloud exposes the Blueprint graph API which applications use to access the network state. The system guarantees eventual consistency in the graph view.

The transition semantics of the graph database are maintained to correspond with structural nature of the graph, such as requiring that an edge connects two nodes. To justify that eventual consistency is good enough for the graph view, some of the ONOS experiments show that when using redundant and identical flow path computation on all instances, the computation modules on each instance converge to the same path. Also, legacy networks have embedded control and are not logically centralized, so it's eventual consistency doesn't impede it's normal operations. However, the authors of the paper admit that there could be some benefit to having some sequencing guarantees that provide deterministic state transition in the network.

(We could potentially allow for causal consistency in the graph view with our logs. Whether this is actually useful is questionable though.)

Out of the box, ONOS has support for a distributed controller architecture. It can run on multiple servers, where each is the master OpenFlow controller for some switches, and any switch has exactly one master controller. Therefore, an ONOS instance must propagate state changes from the switches in its domain to the network of ONOS controllers. The system also aims to be scalable so that as data plane capacity grows or control plane demand increases, more ONOS instances can be created.

ONOS uses Zookeeper as a coordination mechanism for general fault tolerance and master election for switches. Since ONOS is distributed, it has secondary controllers for each switch in case the primary controller goes down. That way, the load of a primary failure are distributed to the remaining controllers. During runtime failure of any nodes detected by the lack of a keep-alive message, a replacement leader or follower is elected to replace it.

Since ONOS assigns a switch to multiple instances, and exactly one master, it also uses a master election process for failure handling. The switch's master connection communicates all the relevant information about the switch to the network view, and the secondary controllers for the switch are only required to elect a new master. The authors emphasize that using Zookeeper to coordinate this election ensures that switches always have a unique master.

The data model behind the graph abstraction represents each network object in a different table. This was implemented as a means to reduce the number of updates, since a previous version stored all the network objects in a single table.

(We could store elements of the network in separate logs, since they don't have dependencies among each other)

(An issue with using a fuzzy log in this system to represent network state is that polling is inefficient. The authors found that they needed to build an event notification system so that the central database wouldn't need to be polled for changes. In a fuzzy log situation, reads require going to the central database, so it may be less efficient.)

¡onos diagram¿

The Network View API

2.3 Ryu

Dec 2

2.4 OpenDaylight

The OpenDaylight defines a model driven approach for managing software defined networks. It aims to create a more modular system for SDN, where subscribers and providers for services are plugins to the system. This Model Driven Software Engineering approach creates a framework for relationships between models, standard mappings and patterns that enable model generation. An advantage of the system, therefore, is that it can create code and API's directly from models. The

OpenDaylight controller uses YANG as a data modeling language to specify these cross platform portable data types, which can describe network operation. While YANG was initially developed for this purpose, it effectively describes other network elements, including policy, protocol, and subscriptions to services. Rather than being an object-oriented structure, YANG is tree-structured and includes complex data types like lists and unions.

¡yang example¿

To modify the data types defined by YANG, OpenDaylight uses the NETCONF and RESTCONF protocols. NETCONF is a network management protocol that defines the configuration and operations on a data store. It also has the ability to run remote procedure calls and send notifications about the underlying data. (more about netfconf from papers?) NETCONF encodes its messages and data in XML. RESTCONF is similar to NETCONF, except that it exports a REST interface over HTTP for use by external applications.

¡opendaylight schematic¿

Service abstraction layer Network devices and applications go through plugins. Applications interact with the data store through RESTCONF Plugins can be binding aware or binding independednt Datastore can be pugged into any implementation Can get topology information from a bgp feed

Plugins? Applications: Bgp and topology Model to model Generic model adoption

2.5 Comparison of SDN systems

¡chart?¿

¡The OpenDaylight controller is built using the Java OSGi framework. OSGi adds a modular framework that allows starting, stopping, loading and unloading of Java modules without bringing down the entire running JVM platform. Ryu controller does not offer this comodity. In Ryu we need to stop the controller and run it again with the needed modules for the REST methods we want to execute or we need to build a REST API with all REST modules included in it, which runs once and provides all REST methods without stopping the controller. ¿

3 Common Data Structures

ugh

4 SDN extensions

4.1 ONOS

make graph structure causally consistent (topology representations with a log backing...)

build a log based event notification system?

4.2 Ryu

4.3 OpenDaylight

OpenDaylight's modularity and data store agnostic setup makes it well suited for weakly consistent structures.

References