BGPmon Design

Requirements (1)

- Lightweight core
  - while talking to hundreds of BGP peers
  - and answering thousands of client queries
- Extensible
  - support different input/output formats with minimal pain
  - support of different modes of operation by abstracting and chaining functionality
- Scalable
  - utilize all available cores in MP systems
  - horizontal scaling on adding nodes
  - easy deployment
- Language Agnostic Ecosystem
  - BGPmon middleware implementors should be as free as possible
  - while protecting the core from possible mistakes on their side
- Leveraging available technologies
  - rely on industry proven best practices

Requirements (2) - Core

- Talk BGP just enough to siphon messages from peers
  - everything in / keepalive out
- Client interfaces are debatable but HTTP is a popular choice
  - just the right amount of statelessness
  - directly modelling the request/response paradigm
  - REST allows us to pass around state as a normal object
  - application state should concern the user of the interface
  - resource state is the only thing the HTTP server will maintain
  - easily cacheable
- Having I/O established, what is the core???
  - request/response router

Requirements (3) - Extensibility

- XML, JSON, graphs and not_yet_invented output formats
- Only the edges of are concerned
- Different functionalities can all be requested through HTTP
- Yet follow completely different paths in the bgpmon network
- Abstracting functionality means finding the right verbs to reason with
  - *collect* BGP messages
  - *store* them
  - *serve* the user queries. those can be
    - *return* sets of stored messages
    - *reduce* sets of stored messages to results
    - *transform* messages to other formats

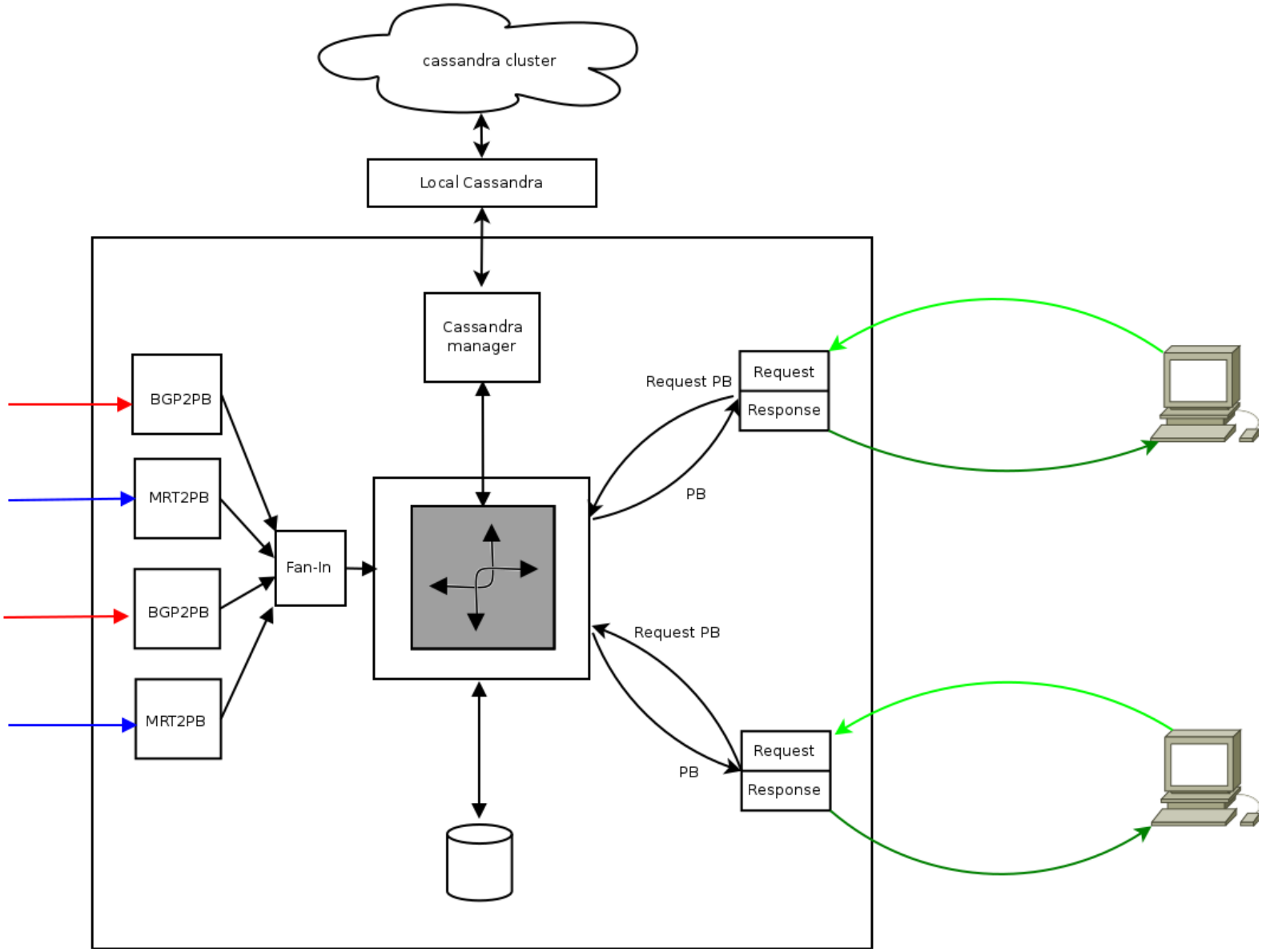
Requirements (4) - Scalability and Data Interchange

- Our design will do a large amount of message passing
- The problem can be modelled in a concurrent fashion
- To avoid locking as much as possible we ulitize CSP style syncronization
  - non-blocking I/O on channels
  - green threads that are spawned often and live short lives
  - multiple cores are naturally utilized
- Storage?
  - all these BGP messages can't be held indefinitely in memory
  - and shared efficiently among all the BGPmon nodes
  - replication is required, because of the important nature of the data and the catastrophic failure modes that can occur
- Once data leaves the process or even more so, the host, it must be in a commonly agreed format
  - easy Marshalling/Unmarshalling
  - easy parsing of the important parts of a message
  - optional verbose form of the whole packet to assist network level debugging

Technologies

- Language: Golang
  - mature (7 years old)
  - compiled and static typed
  - concurrency primitives (typed channels/goroutines)
  - rich standard library (net/, bufio/, encoding/{json, xml})
  - multiplatform (Linux,{Free, Open, Net}BSD, OSX, Windows for x86/arm/amd64)
- Storage: Cassandra
  - elastic scalability
  - transaction support
    - atomicity
    - tunable consistency
  - automatic cluster syncronization
- Internal Format: protocol buffers
  - compilers for most languages
  - heavily optimized for efficient transfer over the wire
  - automatic json decoration for debugging

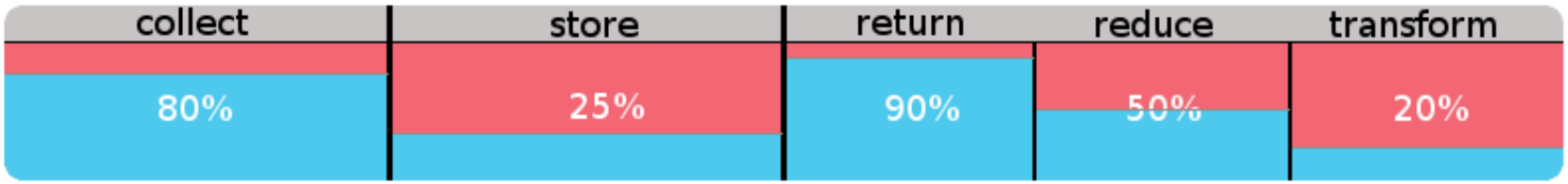
Node Architecture



BGPmon networks

- Data sharing between nodes in a monitoring network
  - Just configuring their cassandras to be in the same cluster
    - automatic replication
  - CSU will maintain such an infrastucture with public data
    - users can choose between running their own bgpmons and having their cassandras join an existing cluster
    - or just peer with a bgpmon and access their data over the RESTful interface
    - pushing data to the dbs has to be done by an approved user of our public cluster
  - if an organisation so desires, they can run their own private cluster/instance
    - they will be still be able to pull all the public data we provide

So where are we now?



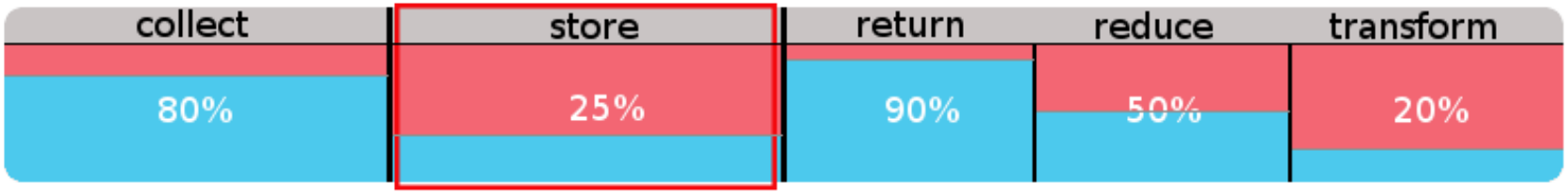
gobgpd/gomrt

- Fully fledged concurrent bgpd implementation
- Mainly developed by OSRG, NTT, japan
- JSON configuration and control over HTTP

Our additions to it include:

- gomrt
  - pure golang library to read and write MRT data
  - builds on the bufio interfaces
- RPC control plane

cassandra / RV mirror

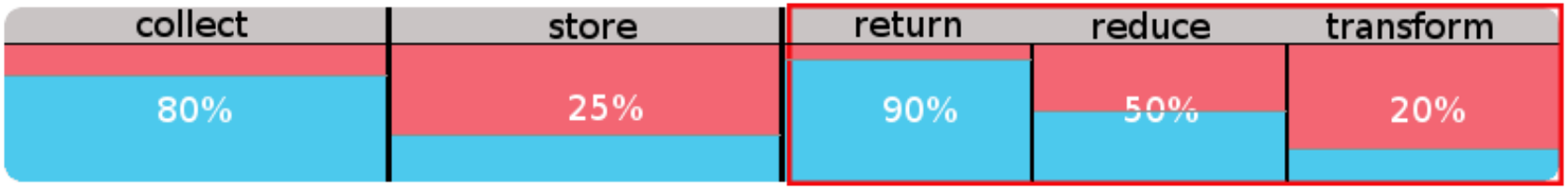


- Currently experimenting with the optimal schemas to store our BGP data
- Cassandra allows building a schema with the indexing variables living next to the whole protobuf blob

Thanks to the routeviews project we have over 10years of historical BGP data from multiple collectors (11TB as of 2015)

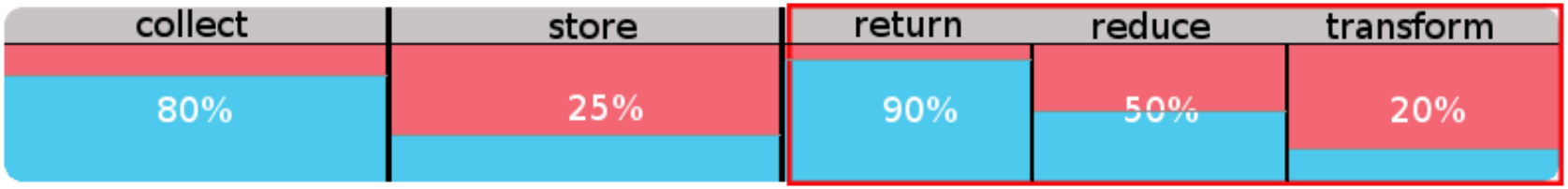
- Provides MRT files exported from modified Quagga instances on regular intervals
- We became an official mirror
- This data will be imported in our cluster and made available to the new BGPmons
- Dates for the provided data live on the filename.

archive



- Created an HTTP interface to expose the data in a unified format that abstracts away file structure
- It is intended to be the prototype for the RESTful BGPmon interface
- you specify date ranges and the format that you expect your data
- uses gomrt to parse deeper in MRT and figure out the correct dates
- it returns it over an HTTP 1.1 chunked encoding channel
- allows caching of the most used files using memcached
- handles thousands of concurrent requests on pretty much commodity hardware

stats



- The stats server is again an HTTP interface that is at the same time a client to the archive
- Uses the exact same daterange request format it calculates statistics
- Utilizes gomrt and gobgp to parse into the BGP messages and figure out types like
  - withdraws
  - NLRI
  - mpreach/mpunreach
- Produces statistics in JSON about messages in the requested daterange
- Those stats are consumed by client-side javascript and render graphs on the browser

testing/deployment

- Everything we do is dockerized
- We can easily spin up hundreds of containers for testing
- We provide docker images that bundle cassandra, golang, protobuf compilers and our gobgp code
- Apart from the usual source and binaries we will distribute BGPmon as a docker image too