System Demo 7.5
NZAPP

SPO-664 Deliver prototype visibility receive metrics pipeline suitable for eventual integration into SKAMPI

SP-1001 RT visibility receive signal display developed and integrated into the SDP prototype (including stretch goals)

Metric Display Architecture

Loosely coupled strawman architecture chosen for PI#7 work on SP-1001

Metric Generator

- Some of the SKA1 data rates requiring monitoring are very high
- Can't realistically push all raw data to database for metrics so one role of metric generator is to aggregate data before persisting
  - Allow metric generator to be configurable so can tune the frequency of metrics generated to control data volumes and which data of particular interest
  - Allow multiple metrics to be generated (eg output metrics for a particular baseline at higher rate or for particular channels at greater resolution while still generating broader metrics at lower rate)
- For SP-1001 in PI#7
  - Have preferred to push metrics (rather than have them pulled) so each metric generator doesn't have complication of buffering metrics/providing service
  - Have created metric_generator.py (https://gitlab.com/ska-telescope/cbf-sdp-emulator-metrics-generator) which can run stand alone or integrated with CBF-SDP Emulator (with both single and multithreaded emulation) (https://gitlab.com/ska-telescope/cbf-sdp-emulator)
  - Have added metric generator RDMAmetrics.c (https://gitlab.com/ska-telescope/rdma-data-transport) to monitor sending and/or receiving of RDMA messages, including status of all memory regions

Time Series Database

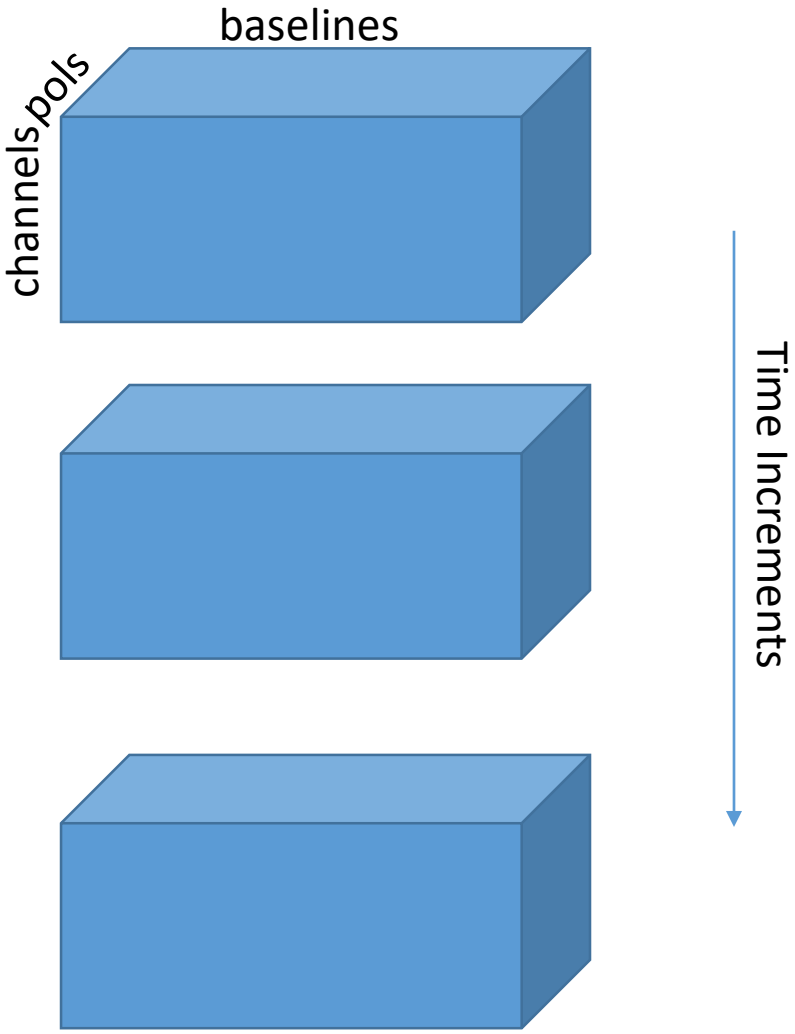
- A time series database seems the most suitable choice for persisting metrics
- Many choices available:
  - InfluxDB free for non-clustered usage, MIT licence, written in Go, supports HTTP push
  - Prometheus free, Apache licence, written in Go, uses HTTP pull
  - kdb+ commercial, written in Q
  - Graphite (formerly Whisper) free, Apache licence, written in Python
- For SP-1001 in PI#7
  - Selected InfluxDB
  - Advantages: popular, free (non-clustered version), good support for receiving pushed metrics via HTTP, compatible with assortment of visualisation tools
  - Disadvantages: bulky (in my opinion) text-based format for metrics, not free in clustered version (if over 750k writes/sec or over 100 queries/sec)

Visualisation Tool

- Three visualisation options considered
  - Use a popular time-series visualisation solution such as Grafana or Kibana
  - Use a client-side JavaScript library for drawing graphs, such as ChartJS, Metric-Graphics, Recharts, C3js, React-vis, Metabase, Plotly (note some of these can be used inside a visualisation solution, such as Poltly inside Grafana)
  - Use or adapt custom code, likely custom JavaScript within a browser, such as done by MeerKAT signal displays (https://github.com/ska-sa/katsdpdisp/blob/master/katsdpdisp/html/figure.js#L475) to produce graphs closest to legacy observatory monitoring implementations
- For SP-1001 in PI#7
  - Selected simplest solution using Grafana
  - If found to not be suitable then will have to move to other options (more customisable displays but also much more effort)

Visibility Metric Generator

- cbf-sdp-emulator-metrics-generator pushes multiple metrics to InfluxDB-Grafana for:
  - Stand alone using GLEAM datasets
  - csp-sdp-emulator in single threaded config as a consumer
  - csp-sdp-emulator in multithreaded config as a consumer (special thanks to Seth and Adam for integrating this already!)
- Each metric can independently:
  - Filter (select) specified polarisations and baselines
  - Aggregate across any number of specified channel intervals
  - Aggregate across specified number of time increments
  - Operate on visibility real, imaginary, amplitude, phase
  - Perform operations mean, max, min, variance
- Has exposed some limitations of python for generating metrics
  - Task was not suitable for NumPy
  - Easy to lose SPEAD heaps



RDMA Metric Generator

- RDMA receiver also pushes data to InfluxDB-Grafana
  - Gives a way to stress-test the InfluxDB-Grafana solution
  - Provides NIC queue utilisation, CPU utilisation, network bandwidth, messages transferred and missing, and snapshot of current status of every memory region
  - Hundreds of fields output each push
  - Have tested pushing dozens of times/sec so far, with Grafana pulling up to 10 times/sec
- Uses C libcurl package for posting metrics to InfluxDB
- Found that the thread coordinating the RDMA message transfers needs to focus on its task to achieve 100G rates otherwise packet loss increases
  - Regardless of RDMA this give alarm bells about using a solution for receiving visibilities that may introduce jitter (eg garbage collection, OS interrupts)
  - 300k messages at 100G can loop through all memory buffers in ≈1ms, so jitter should be kept under few 100µs (rough guideline only)
  - HTTP now performed in separate threads
  - RDMA with separate HTTP threads working fine in this C implementation on commodity Ubuntu

Demonstrations

- Visibility Metric Generator standalone with GLEAM large dataset (four channels) pushing two metrics
  - Mean amplitude and mean phase
- Visibility Metric Generator with csp-sdp-emulator in multithreaded configuration (10 receivers) pushing single metric (to avoid overburdening emulator)
  - Mean amplitude with 50000 channels aggregated into intervals 5000 wide
  - Note the multithreaded emulator provides heaps with constant 0.5+0i visibilities
- RDMA Metric Generator
  - Pushing data live from actual CIPA FPGA at sustained 98G, with metrics pushed across internet to a dedicated InfluxDB-Grafana thick-provisioned storage server