View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Technical Disclosure Common

Technical Disclosure Commons

Defensive Publications Series

August 2020

DYNAMIC TELEMETRY WITH LATENCY AWARE NETWORK OPTIMIZATION

Thomas Vegas

Anirban Karmakar

Giacomo Trifilo

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

Vegas, Thomas; Karmakar, Anirban; and Trifilo, Giacomo, "DYNAMIC TELEMETRY WITH LATENCY AWARE NETWORK OPTIMIZATION", Technical Disclosure Commons, (August 24, 2020)
https://www.tdcommons.org/dpubs_series/3542

This work is licensed under a Creative Commons Attribution 4.0 License.

This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

DYNAMIC TELEMETRY WITH LATENCY AWARE NETWORK OPTIMIZATION

AUTHORS:

Thomas Vegas
Anirban Karmakar
Giacomo Trifilo

ABSTRACT

A Sender uses a continuous feedback loop to estimate remote processing latency. With that estimated value, the Sender continuously adapts its sending behavior to optimally send updates and reduce network consumption. Implementing this dynamic telemetry, the overall system latency is untouched and only the necessary updates are sent.

DETAILED DESCRIPTION

Data pipeline characterization is a well-studied topic. Error rate, data throughput and latency are metrics used to evaluate the health of a data pipeline.

In one technique, a data set may be evaluated. This data set is sent for each update of its member. The data set could be a replicated database record, or it could be a data blob sent over the network towards a processing system. The case of a data set describes a state, and that state is evaluated. Various states can be collapsed, meaning the state is a scheme of eventual consistency. Missing some updates can be tolerated as the long as the latency remains acceptable for eventual consistency.

Some latencies of a data pipeline are induced at the receiver side. The latencies may be due to some periodic batched processing, or more or less regular processing interval, the result of another data pipeline, having its own constraints.

The data sender might not be able to act on those latencies. However, with the knowledge of the latency, the data sender can act to opportunistically send the data over the pipeline, so as to drastically reduce the amount of data sent, while minimizing, or not impacting, the actual functional impact on the processing on the receiver side.

Problem Modeling

This paper considers the eventual consistency of a dataset between a Sender and a Receiver.

A bidirectional system is defined that is composed of two components exchanging the latest version of a dataset over a pipeline (ipc/network/linked list...). The actual data set is sent from the Sender towards the Receiver only.

The Sender produces updated versions of the dataset and sends it to the Receiver. The Receiver reads the latest version and queues it for further processing. The Receiver is performing other tasks, potentially unrelated, but that may have an impact on its overall processing latency/period.

A few timestamps are defined that are carried along with the data being transferred:

- T1: time when the data content change was first done, on the Sender side.
- T2: time when the data set, including the change, was first sent towards the Receiver side, on the Sender side.
- T3: time when the data set's corresponding version was received and queued on the Receiver side.
- T4: time when the Receiver has fully completed the processing of the corresponding data set received.

Without configuration, the Sender estimates, with eventual feedback from the Receiver:

- Network latency: T3-T2.
- Past processing times: collection of T4-T3.
- Observer dataset latency: after the fact, establish Sender / Receiver dataset latencies: T4-T1, could be instantiated in case of multiple dataset.

From this estimation, the Sender predicts next Remote processing time minus network latency:

- Only sends latest event at the last moment.

Without configuration, by estimating the time when the Remote entity is going to process the events, the Sender can locally collapse the updates, and only send at the last moment the latest event.

Estimation Through Data Graph Clustering Identification

On the Receiver side, once the data set has been processed, the data timestamps are represented as data clusters. More specifically, the data is represented as graph with dots representing event with position as x:T1 and y:T4.

Known clustering detection techniques are used to identify latency pattern, representing moments when the receiving system is actually processing the data cluster (T4). With processing latencies and Receiver queues, horizontal blobs may appear. With that, the biggest T1 and its corresponding T2 may be used, knowing that all the events of the blobs that were sent before, but in the same processing, could have been skipped.

For a system that periodically processes the events/data set versions, the graph may have a stairs-like shape. For a system that processes events as soon as they arrive, purely linearly aligned dots on the graph (T4=T1+network_delta) should be observed. The solution presented herein addresses the case where there are latencies at the Receiver side.

The timings are transferred back to allow some estimation on the Sender side, which can adapt the periodic time of sending T2 to be as late as possible to fall before the next likely/probable T4 processing time. By doing this, more data set updates are allowed to be collapsed locally, only sending the latest version. This minimizes the pipeline throughput and eases the processing power required on the Receiver side.

This also positively impacts the processing time by generally reducing load. This will save cost on transmitting data.

Practical Use Cases

1. Local database replication on the same node. Replication can be done through the use of IPC. It would be useful to be able to relieve the Sender side when the replicating process at the Receiver side is introducing fixed latency. The database eventual consistency collapse is moved from the Receiver side to the Sender side.

2. From network devices to cloud telemetry. Often, it is necessary to repeatedly send a given updated database record. It would be beneficial to take into account the fact that the Cloud entity has processing delays. This will enable the Sender to throttle the sending rate (i.e., skipping the intermediate record state sending) while still maintaining the same end-to-end observed replication latency.

This saves operational expenses associated with maintaining the cloud service as every byte is subject to a charge. Simulation shows that 3x network bandwidth reduction can be achieved with maintained latency.

3. Internet of Things (IoT) devices reporting data, reporting periods evaluated might be much longer than in points 1 and 2 above. 3x network bandwidth reduction means almost 3x more power time.

<u>Simulation</u>

In a simulation script written in Python, the observed replication latency was measured for a state collapsed/eventual consistency database. Several sending decisions logic were tried which were designed to take advantage of existing latency to minimize network throughput.

This is intended as example, other estimation strategies could also be designed.

The model includes two replicated databases, some network latency around 100ms (gaussian sigma=10ms), an event generation 10 Hz and a final remote processing 3sec period (gaussian sigma=1sec).

Sending methods:

1. JustSendIT():
    1. Sending all updates
2. JustSkip(10):
    1. Simply sending one update out of ten
3. JustSkip(20):
    1. Simpling sending one update out of twenty
4. ConsumerProcessingEstimator():
    1. With a feedback loop, continuously estimating network latency and what will be the next remote processing time.

       Intent is to wait for the last moment to send the latest event, right when it will be processed remotely. Sending any other events would not improve latency: they would be waiting in remote pipeline.

Conclusion from Simulation

Network throughput is divided by 3 (9.99% versus 3.06%) and maximal latency is lower. Average latency is not worsened.

JustSendIt(): 100.0% network utilization, latency 1751.0/7005.4 (avg/max)
JustSkip(10): 9.99% network utilization, latency 2264.4/6905.6 (avg/max)
JustSkip(20): 5.0% network utilization, latency 2648.8/7206.2 (avg/max)
ConsumerProcessingEstimator(): 3.06% network utilization, latency 2243.0/5512.4 (avg/max)

Figure 1 below shows the cumulative histogram for the various latencies:

- Solid blue: Sending every update on the network (network 100%)
- Yellow and Green: Sending one update every 10 / 20 events (network 10%/5%)
- RED: Consumer processing estimator, latency is maintained, max latency is lower (network < 3%)

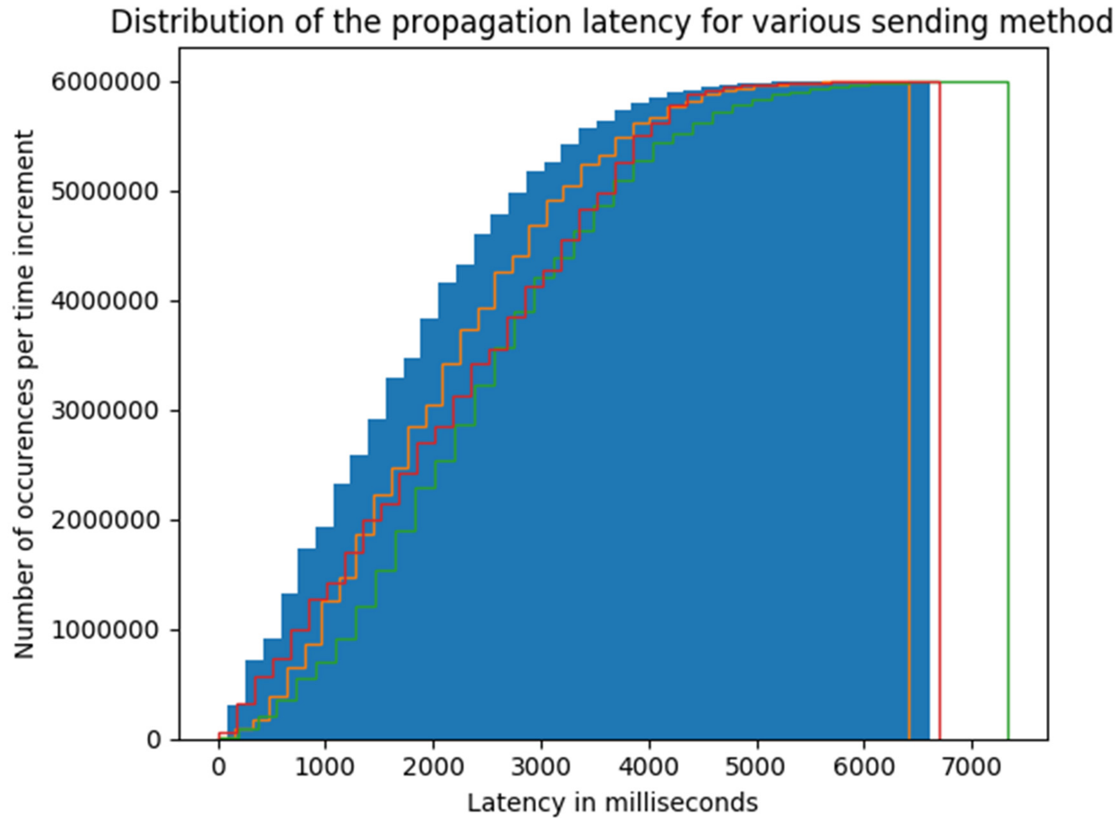

Figure 1

In summary, a Sender uses a continuous feedback loop to estimate remote processing latency. With that estimated value, the Sender continuously adapts its sending behavior to optimally send updates and reduce network consumption. Implementing this dynamic telemetry, the overall system latency is untouched and only the necessary updates are sent.