Performance and Fault Tolerance Documentation

1. an explanation of your choices for QoS levels for each publication and subscription.

For our publisher our Quality of service level is 0 (at most 1), however since we are logging our requests that we send there is room to extend this to 1 (at least once) if we create a mechanism to react to the visualiser/broker going down and save the “unreceived” requests and sending them whenever the visualizer is online again.

Our pipe and filter uses the quality of service level 0 as well for both publishing and subscription, to keep it as simple as possible.

Our visualiser currently has a quality of service level of 0 for subscriptions. If we create a fault tolerance to ignore duplicate requests (same ID) and we extend our QoS level for the publisher to 1 to make sure requests are always going out at least once it is possible for us to extend the QoS level on the visualiser to 2.

2. an explanation of the fault-tolerance mechanism(s) you have implemented in the system.

For our emitter fault-tolerance is ensured through a disconnect/reconnect method that tries to automatically reconnect with an interval of 5 seconds for 10 tries if the component disconnects from the broker. This component also logs everything to prevent data loss in case the visualizer stops receiving the requests from either disconnecting or subscription is stopped.

In the visualiser, in case of disconnection, a red banner notifies the user that the connection is lost, and the user then has reconnect manually. Also if a topic stops publishing you can always change the topic, this is all done manually through the user interface. The code is written in a way where a sudden stop of data-flow or a stop to the subscription won’t affect the application in any other way than to just stop plotting on the map.

In the pipe and filter component there is a mechanism in place that tries to automatically reconnect to the broker(of the same address as specified when first connecting) automatically every 5 seconds for 10 tries, so in the case of a broker disconnecting it has 50 seconds to go back online for the pipe and filter to reconnect. If it doesn’t reconnect within 50 seconds it has to be restarted.

3. a discussion about the potential impact of your architecture as well as the choices of 1. and 2. above on scalability, throughput, latency, availability and consistency.

Our latency at the moment is good because we don’t really care about lost requests, we currently don’t have any mechanisms in place to recognize lost data. If we were to increase our QoS levels to a 1 or a 2 and save logs to later resend this would require some computation as well as communication between components which could increase latency in our requests, however it would result in an increase of consistency.

We have put a lot of focus on availability in our system to make sure it is reliable in use. Every component has a mechanism in place to recognize disconnects and trying to reconnect to the system again. Our publisher also logs every requests that it sends so that no data goes lost.

Since our components currently don’t have much local computation we can process our data in a very efficient manner. The visualiser is able to handle 1 data point per 0.1 seconds without problems.

Since our components stand on their own and do not communicate directly to each other (communication goes through the broker) our systems scalability is really good, especially when you factor in our emphasis on availability.