Recently Fred published a post ( http://lefred.be/content/mysql-group-replication-is-sweet-but-can-be-sour-if-you-misunderstand-it) in which he was stating, I had publish my blog ( http://www.tusacentral.net/joomla/index.php/mysql-blogs/191-group-replication-sweet-a-sour.html) which contains few “misunderstanding”.

All the people that knows me, also know I have no problem to admit if I do a mistake, at the end doing mistake is a way of learning and be afraid of the ones who don’t do them.

But at the same time, I need to have proof of it. As I provide proof of what I wrote with numbers and tests (all available on github).

Let us put down the basis of the discussion with facts, not personal speculation or assertions.

Facts:

1) From MySQL official documentations (https://dev.mysql.com/doc/refman/5.7/en/group-replication-use-cases.html)

*Group Replication enables you to create fault-tolerant systems with redundancy by replicating the system state throughout a set of servers. Consequently, even if some of the servers fail, as long it is not all or a majority, the system is still available, and all it could have degraded performance or scalability, it is still available. Server failures are isolated and independent. They are tracked by a group membership service which relies on a distributed failure detector that is*

*able to signal when any servers leave the group, either voluntarily or due to an unexpected halt. There is a distributed recovery procedure to ensure that when servers join the group they are brought up to date automatically. There is no need for server fail-over, and the multi-master update everywhere nature ensures that not even updates are blocked in the event of a single server failure.* ***Therefore MySQL Group Replication guarantees that the database service is continuously available*** .

2) Still from MySQL ( https://dev.mysql.com/worklog/task/?id=9838 ) *Group Replication introduced flow control to avoid excessive buffering and to maintain group members reasonably close to one another. For several reasons, it's more efficient to keep buffering low but, as mentioned before, it is not a requirement that members are kept in sync for replication to work: once a slave becomes delayed it will just increase the number of pending transactions in the relay log.*

*The flow control mechanism enters the scene to bound the back-log members can accumulate, in terms of transactions to certify and transactions to apply. Once those bounds are exceeded, and while they remain exceeded, the flow-control mechanism limits the throughput of writer members to adjust to the capacity of the slowest members of the group. By imposing conservative bounds it tries to make sure all members are able to keep-up and can return safely to within the bounds defined by the user.*

So given the above, I, as “standard” user, read that as “when using MySQL and GR, and setting (eventually) the Flow Control correctly, I will achieve to have a data platform that ***is continuously available***
given the use of GR”.

Cool, right? So what I was doing in my tests? Well two main things, one is to see if I can use GR for scaling reads as I was doing (and hundreds of customers as well) with PXC, the other is to see if in case of crash of my Master, I can safely fail-over another slave.

This doesn't mean I am comparing the way the two product works. I cannot care less at this stage, as I am sure most of the customer will not care. What they care is what they can do SAFELY, and with what. All the mambo-jambo about the technical details (how much sync or async I can be) is not their concern.

So the point was and is… Given I am used to product X can I move to product Y and if so how and what I should be aware of?

Once more I was trying to answer to the question "*If GR and InnoDB cluster has to work as alternative to other (virtually) synchronous replication mechanism, what change or shift our customers must consider if they want move from one to the other*
."

The outcome of the tests is pointing to answer that, period.

Let us clarify two things more:

I am perfectly aware (Fred talking to you) that GR use a different mechanism for FC, and that the numbers set in the group_replication_flow_control_certifier_threshold/ group_replication_flow_control_applier_threshold are then use to calculate the Quota. Still they are user threshold, and *Specifies the number of waiting transactions in the applier/certifier queue that trigger flow control* which are connected to the final statements reported above: *By imposing conservative bounds it tries to make sure all members are able to keep-up and can return safely* ***to within the bounds defined by the user.***

Bound that as for the manual can go from 0 to 2147483648.
As such setting it to 25 (I did try also 1000 or more with even worse results) is perfectly legit and I have some issues in considering it a mistake.

I was measuring the lag with the only tools MySQL/Oracle had give us, in the article I said I had used:
"*select @last_rec:=SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX( Received_transaction_set,':',-2),':',1),'-',-1) last_received FROM performance_schema.replication_connection_status WHERE Channel_name = 'group_replication_applier'; select (@last_rec - @last_exec) as real_lag* "

Which use the only information available at the moment regarding the incoming and the current executed transactions.
Fred says that is not valid because the certification and *This means that a transaction listed on last_received_transaction_set is certified and queued to apply on this member. On the other members it may already be certified/applied or soon be.*
I agree that it may not be perfect .. and I should have said: *is for sure the last apply or soon to be given the certification on the master a node can know about*
.
But here we are not talking of 1 or 10 entries, but in most cases lag of hundreds or thousands entries. So also if this is not perfect and I can miss in a way or another a couple or entries because still processing the certification, I still think it is a valid way to evaluate the lag given the dimensions we are talking about.

BTW if this is not … well please help us and provide the right way to calculate the REAL lag between each node in a GR cluster and the writing master, with the most accurate precision.

About the comment of the node dimension, well thanks and yes you are right here I forgot to put the details.

The four nodes are VMS on separated hosts, so the gr1/2 where hosted on Host1 while gr3/4

hosted on the another host. Nothing else running on the hosts while test was running. Applications and monitor where running on a 3 host. The exact dimension is not really relevant given all of them were the same.

I want to say that I can setup the perfect environment for GR or PXC or NDB or whatever you are testing and showing how cool that is.

But I think we should consider that real life is far to be like that, and that we must see/test is how a product is behaving in adverse conditions, or if not adverse challenging.

Given all the above, in my opinion the tests clearly shown that also if the Flow Control mechanism in GR is the coolest thing ever conceptualize, at the end it does not what it is suppose to, no matter what.

Was I setting the thresholds wrong? Well not sure about that, given the results I got.

If I have 4 nodes and 2 of them (so … no majority) are lagging behind of hundreds or even thousands of entries, while my threshold is in the order of hundreds or less, this means that the cool mechanism is not doing his job, period.

One think is write down whatever about this or that but another is doing tests and provide numbers and evidences, not just words.

As said in a comment (not published in the Fred blog) I am more than happy to perform more tests and do them in conjunctions with anyone else.

Then if I am wrong I am wrong… but until I have different evidence, just saying a car can fly because it has very nice and colorful wings, doesn't make it fly.