MySQL replication Topology

Parallel replication overview

One of the main grievance of replication users is that, while a well-tuned master server can handle thousands of concurrent operations, an equally tuned slave is constrained to work on a single thread. In Figure 1, we see the schematics of this paradigm. Multiple operations on the master are executed simultaneously and saved to the binary log. The slave IO thread copies the binary log events to a local log, and on such log the SQL thread executes the events on the slave database. When the master is very active, chances are that the slave lags behind, causing hatred and nightmares to the DBAs.

Single applier

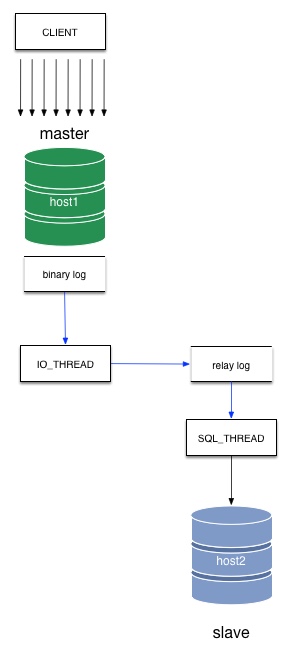


Figure 1 - Single applier

Until 2011, there was little remedy to this problem. Then we saw the first implementation of parallel apply, provided by Tungsten Replicator. The parallelism is schema-based, and provides 5x improvement on the slave performance.

Shortly after that, the same algorithm of schema-based parallel applier was implemented —with the same performance gain— in MySQL 5.6 (Figure 2). This implementation is valuable for those organizations that have data split by schemas, which can happen for those companies that have adopted sharding of data and split their logical chunks in different schemas, or for those companies that run multi-tenant services, where splitting data by schema is a necessity. This solution does not meet the needs of all users. Many systems are based on a single very active schema and perhaps a few ancillary ones with minimal traffic. For those users, parallel replication by schema is useless.

Parallel applier

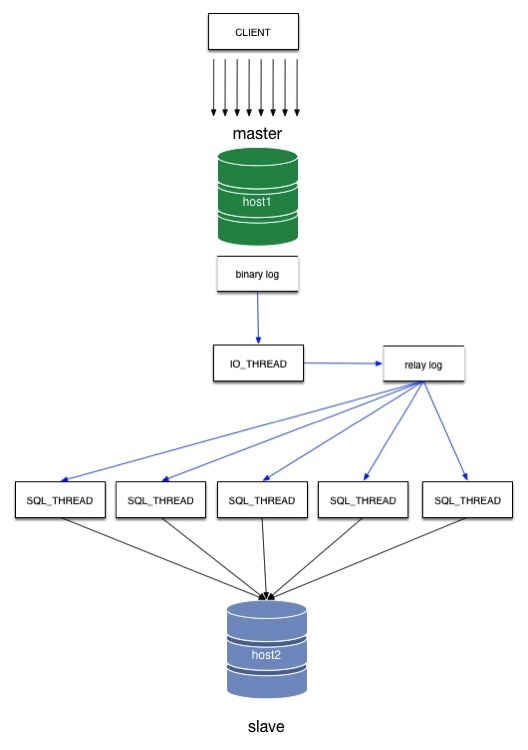


Figure 2 - Parallel applier

<http://datacharmer.blogspot.com/2015/08/mysql-replication-in-action-part-5.html>

==============================================================================

Multiple masters: attraction to the stars

In the last 10 years I have worked a lot with replication systems, and I have developed a keen interest in the topic of multiple masters in a single cluster. My interest has two distinct origins:

On one hand, I have interacted countless times with users who want to use a replication system as a drop-in replacement for a single server. In many cases, especially when users are dealing with applications that are not much flexible or modular, this means that the replication system must have several points of data entry, and such points must work independently and in symbiosis with the rest of the nodes.

On the other hand, I am a technology lover (look it up in the dictionary: it is spelled geek), and as such I get my curiosity stirred whenever I discover a new possibility of implementing multi-master systems.

The double nature of this professional curiosity makes me sometimes forget that the ultimate goal of technology is to improve the user’s life. I may fall in love with a cute design or a clever implementation of an idea, but that cleverness must eventually meet with usability, or else it loses its appeal. There are areas where the distinction between usefulness and cleverness is clear cut. And there are others where we really don’t know where we stand because there are so many variables involved.

One of such cases is a star topology, where you have many master nodes, which are connected to each other through a hub. You can consider it a bi-directional master/slave. If you take a master/slave topology, and make every node able to replicate back to the master, then you have almost a star. To make it complete, you also need to add the ability of the master of broadcasting the changes received from the outside nodes, so that every node gets the changes from every other node. Compared to other popular topologies, say point-to-point all-masters, and circular replication, the star topology has the distinct advantage of requiring less connections, and of making it very easy to add a new node

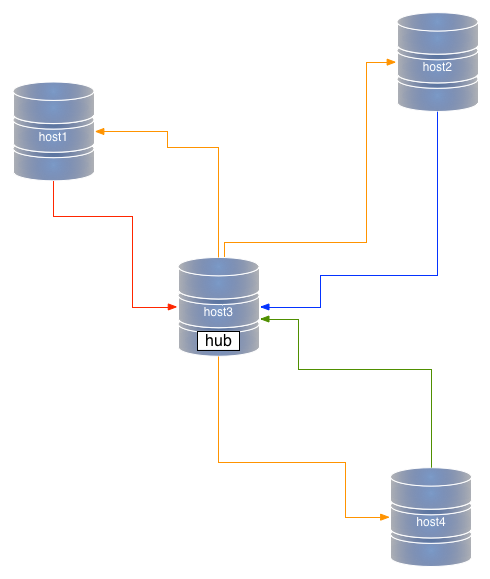
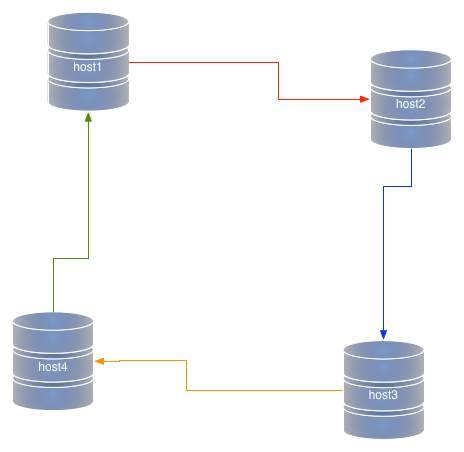


Figure #1: Star topology

However, anyone can see immediately one disadvantage of the star topology: the hub is the cornerstone of the cluster. It’s a single point of failure (SPOF). If the hub fails, there is no replication anywhere. Period. Therefore, when you are considering a multi-master topology, you have to weigh in the advantages and disadvantages of the star, and usually you consider the SPOF as the most important element to consider.

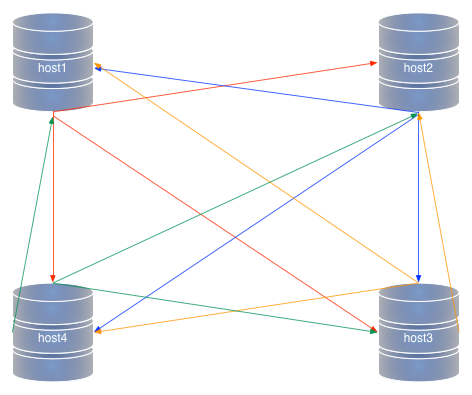
Depending on which technology you choose, though, there is also another important element to consider, i.e. that data must be replicated twice when you use a star topology. It’s mostly the same thing that happens in a circular replication. If you have nodes A, B, C, and D, and you write data in A, the data is replicated three times before it reaches D (A->B, B->C, and C->D). A star topology is similar. In a system where A, B, and D, are terminal nodes, and C is the hub, data needs to travel twice before it reaches D (A->C, C->D)



*Figure #2: Circular replication*

This double transfer is bad for two reasons: it affects performance, and it opens to the risk of unexpected transformations of data. Let’s explore this concept a bit. When we replicate data from a master to a slave, there is little risk of mischief. The data goes from the source to a reproducer. If we use row-based-replication, there is little risk of getting the wrong data in the slave. If we make the slave replicate to a further slave, we need to apply the data, generate a further binary log in the slave host, and replicate data from that second binary log. We can deal with that, but at the price of taking into account more details, like where the data came from, when to stop replicating in a loop, whether the data was created with a given configuration set, and so on. In short, if your slave server has been configured differently from the master, chances are that the data down the line may be different. In a star topology, this translates into the possibility of data in each spoke to be replicated correctly in the hub, but to be possibly different in the other spokes.

Compare this with a point-to-point all-masters. In this topology, there are no SPOFs. You pay for this privilege by having to set a higher number of connections between nodes (every node must connect to every other node), but there is no second-hand replication. Before being applied to the slave service, the data is applied only once in the originating master.



*Figure #2: Point-to-point all-masters topology*

Where do I want to go from all the above points? I have reached the conclusion that, much as user like star topologies, because of their simplicity, I find myself often recommending the more complex but more solid point-t-point all-masters setup. Admittedly, the risk of data corruption is minimal. The real spoiler in most scenarios is performance. When users realize that the same load will flow effortlessly in a point-to-point scenario, but cause slave lags in a star topology, then the choice is easy to make. If you use row-based replication, and in a complex topology it is often a necessary requirement, the lag grows to a point where it becomes unbearable.

As I said in the beginning, all depends on the use case: if the data load is not too big, a star topology will run just as fine as point-to-point, and if the data flow is well designed, the risk of bad data transformation becomes negligible. Yet, the full extent of star topologies weaknesses must be taken into account when designing a new system. Sometimes, investing some effort into deploying a point-to-point all-masters topology pays off in the medium to long term. Of course, you can prove that only if you deploy a star and try it out with the same load. If you deploy it on a staging environment, no harm is done. If you deploy in production, then you may regret. In the end, it all boils down to my mantra: don’t trust the theory, but test, test, test.

<http://datacharmer.blogspot.com/2014/01/multiple-masters-attraction-to-stars.html>

==============================================================================

MySQL replication in action - Part 2 - Fan-in topology

In the latest releases of MySQL and MariaDB we have seen several replication improvements. One of the most exciting additions is the ability to enhance basic replication with multiple sources. Those who have used replication for a while should remember that one of the tenets of the “old” replication was that a slave couldn’t have more than one master. This was The Law and there was no escape ... until now. The only way to work around that prohibition was to use circular replication, also known as ring replication, where each node is slave of the previous node and master of the next one.

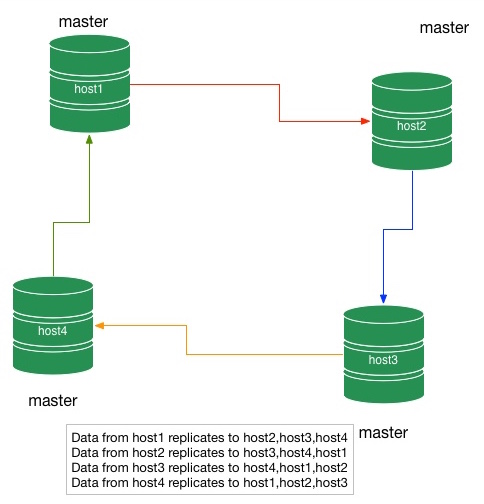


Figure 1: Circular replication

This topology can work, but it is quite fragile: if one node breaks, the replication flow is also broken. Every change down the chain will continue up to the broken node, but it does not reach the nodes up the chain.

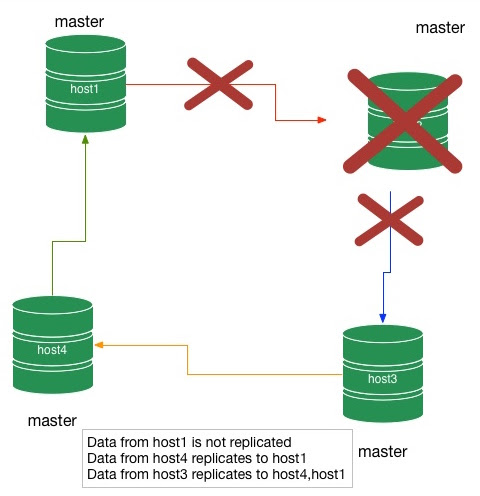


Figure 2: Circular replication with a broken node

Of course, you can fix a broken circular replication deployment, but it is not easy, and has several tricky points that make this task one of the least liked by DBAs.

Despite this limitation, circular replication has been used in production sometimes, mostly because there was no alternative (well, there is Tungsten Replicator, now part of VMWare Continuent, but not everybody was ready to embrace an external replicator), and because users were trying to solve the HQ-branch problem, also known as the fan-in topology.

**What’s a fan-in topology**

In a regular master-slave topology, we have one master and one or more slaves. This setup is useful for many scenarios: reducing load of a database backing a web server (load balancing), providing the basis for a rapid replacement of a failed master (disaster recovery), spreading the data to more users than one server could bear, and more

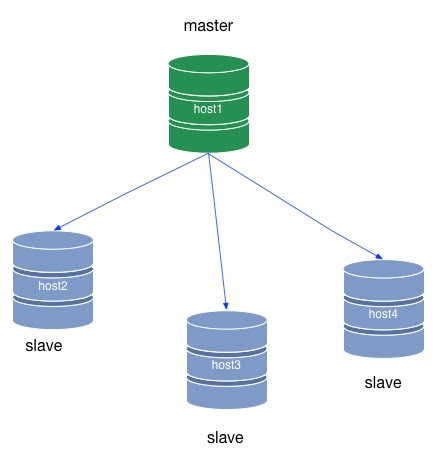


Figure 3: master-slave topology.

One thing that regular replication cannot do is getting data from many input points. The best example is the headquarters of a company, where users need to have data in real time from various branches.

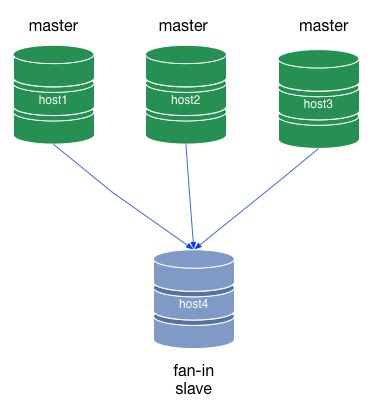


Figure 4: fan-in topology.

You have a visual resemblance between regular replication and fan-in replication. One is the mirror image of the other. While in regular replication the data is produced in one node (master) and conveyed to many consumers (slaves), in a fan-in topology we have many producers (masters) and one consumer (slave). Things are not always this simple. We can have various degrees of fan-in, where many masters replicate to one or more slaves. We could actually have more slaves than masters, if we want. But the main criteria that defines this topology is to have more than one master for each slave

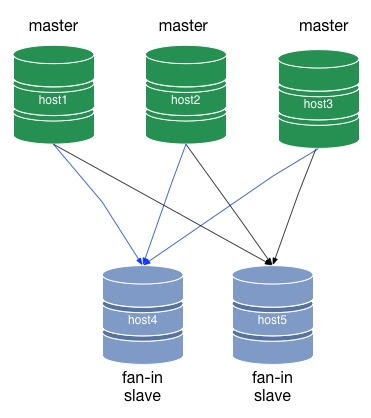


Figure 5: enhanced fan-in topology.

We will revisit the enhanced fan-in topology soon, as this is the basis for a more complex deployment.

<http://datacharmer.blogspot.com/2015/08/mysql-replication-in-action-part-2-fan.html>

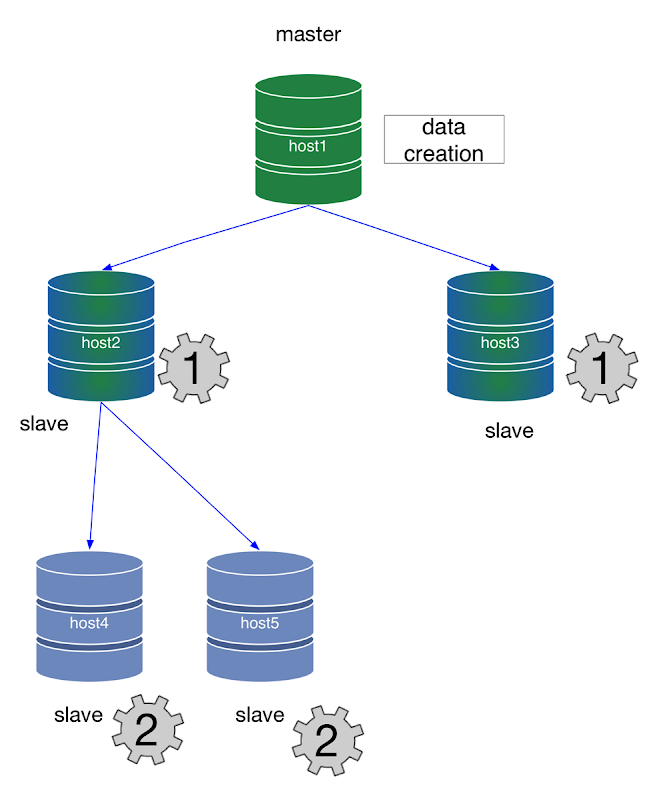
==============================================================================

MySQL replication in action - Part 3: all-masters P2P topology

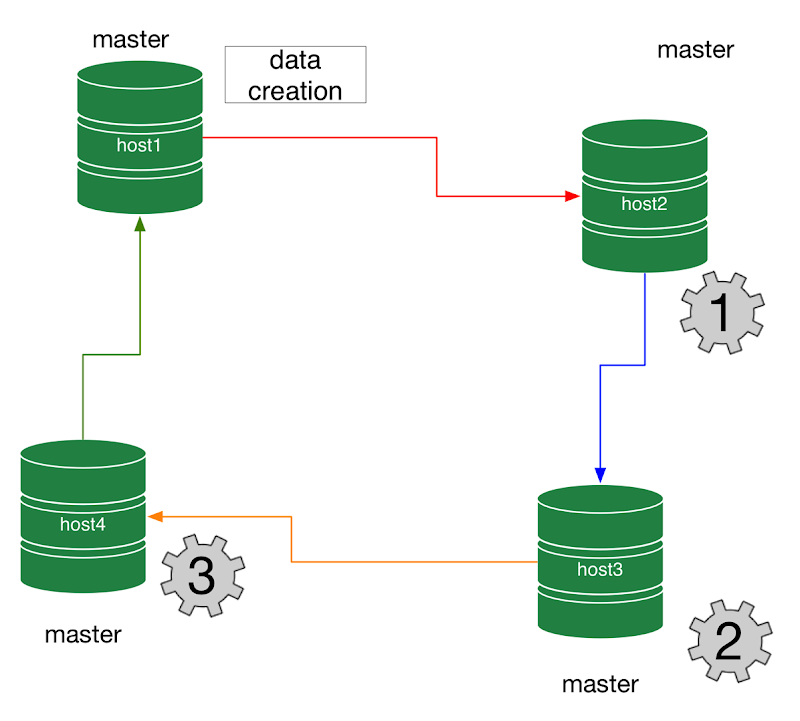
In the previous article, we saw the basics of establishing replication from multiple origins to the same destination. By extending that concept, we can deploy more complex topologies, such as the point-to-point (P2P) all-masters topology, a robust and fast way of moving data.

**Introduction to P2P all-masters topology**

A P2P (Point-to-point) topology is a kind of deployment where replication happens in a single step from the producer to the consumers. For example, in a master/slave topology, replication from the master (producer) reaches every slave (consumer) in one step. This is simple P2P replication. If we use a hierarchical deployment, where every slave that is connected to the master is also replicating to one or more slaves, we will have a 2-step replication (Figure 1). Similarly, in circular replication, we have as many steps as the number of nodes minus one (Figure 2.)



*Figure 1 - Hierarchical replication depth of processing*

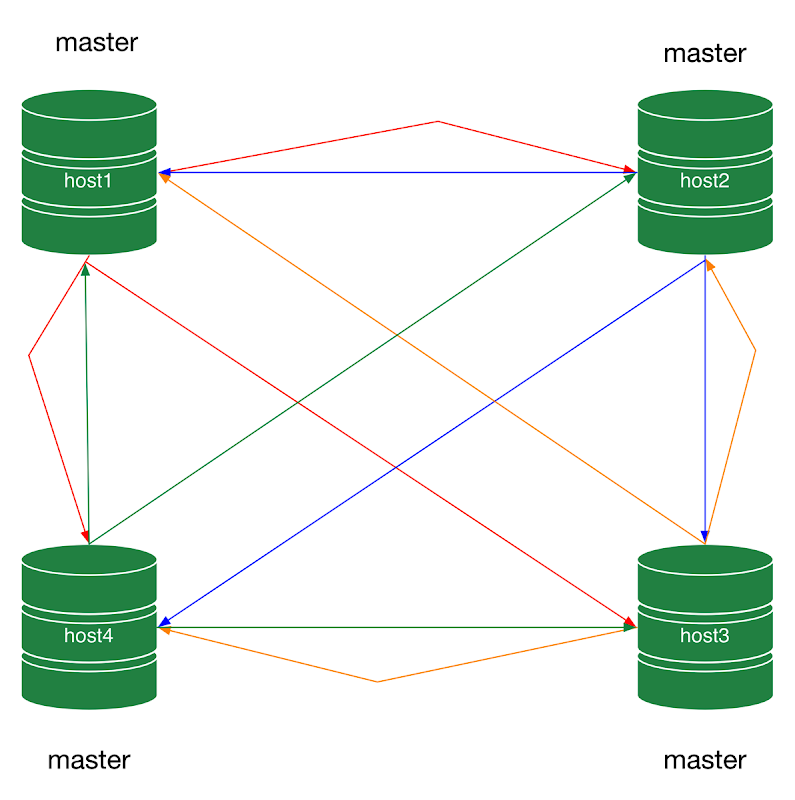


*Figure 2 - Circular replication depth of processing*

Why is this important? The number of steps affects performance, resilience, and, potentially, accuracy.

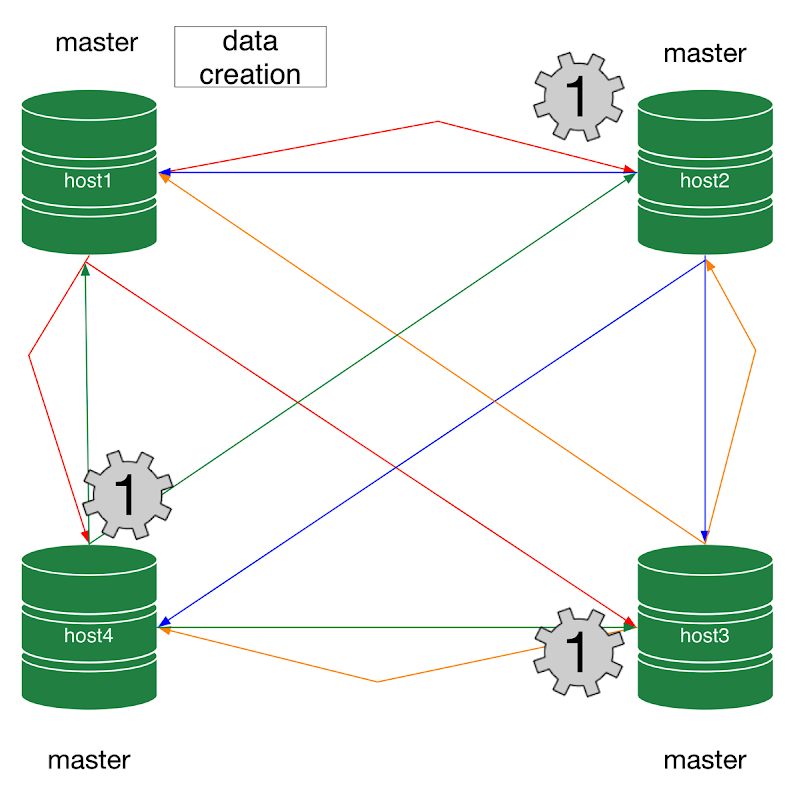
* Performance depends on the number of steps. Before the final leaf of the topology graph gets the data, it will replicate N times, one for each step. In figure 1, host4 will be updated twice as slower as host2. In Figure 2, host4 will be three times slower than host2, as it has to wait for two steps before data reaches its tables.
* Resilience, or the capacity to withstand failures, also depends on the number of intermediate steps. Intermediate masters are single points of failure (SPOF) that can break a branch of the topology graph, or the whole deployment. In this context, a master/slave deployment has one SPOF; the topology in figure 1 has 2, and the circular replication has 4 of them.
* Accuracy can be different if the data goes from master to slave directly, compared to the data going through one or more intermediaries. If data is applied and then extracted again, its chances of reaching the final destination unchanged depend on the intermediate masters to have exactly the same configuration as its predecessors in the chain.

With multi-source replication, we can overcome the limitations of circular topologies, and create a functionally corresponding deployment that has no SPOF, and it is, by virtue of its direct connections, faster and potentially more accurate than its predecessors.



*Figure 3 - All-masters P2P replication*

An all-masters P2P topology is a lot like fan-in topology, but with the number of nodes, masters, and slaves being the same. If all the nodes are fan-in slaves, and are also masters at the same time, every node can get data from the others and can send data at the same time.



*Figure 4 - All-masters P2P replication depth of processing*

In an all-masters P2P topology, each node replicates to every other node. Compared to circular replication, this deployment requires more connections per node (it's a small price to pay) but the data flows faster and more cleanly, as the origin of each transaction is easier to track.

<http://datacharmer.blogspot.com/2015/08/mysql-replication-in-action-part-3-all.html>