MySQL Scale-Out by application partitioning

Oli Sennhauser

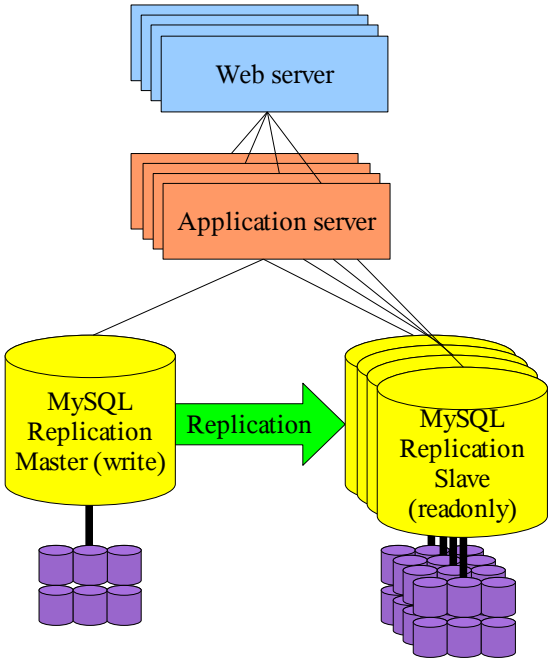
Rebenweg 6
CH – 8610 Uster
Switzerland
oli.sennhauser@bluewin.ch

Introduction

Eventually every database system hit its limits. Especially on the Internet, where you have millions of users which theoretically access your database simultaneously, eventually your IO system will be a bottleneck.

Conventional solutions

In general, as a first step, MySQL Replication is used to scale-out in such a situation. MySQL Replication scales very well when you have a high read/write (r/w) ratio. The higher the better.



But also such a MySQL Replication system (let us call it "MySQL Replication cluster" [4] rather than "MySQL Cluster" in this paper) hits its limits when you have a huge amount of (write) access.
Because database systems have random disk access, it's not the throughput of your IO system that's relevant but the IO per second (random seek). You can scale this in a very limited way by adding more disks to your IO system, but here too you eventually hit a limit (price).

Scale-out possibilities

So we have to think about other possibilities to scale-out. One possibility would be to use MySQL Cluster. This solution can be very fast because it is not IO bound. But it has some other limits like: amount of available RAM, and joins not performing to well. If these limitations were not applicable, MySQL Cluster would be a good and performant solution.
An other promising but more complex solution with nearly no scale-out limits is application partitioning. If and when you get into the top-1000 rank on alexa [1], you have to think about such solutions.

Application partitioning

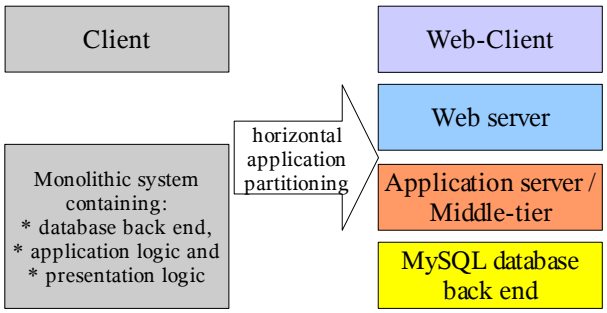
What does "application partitioning" mean? Application partitioning means the following:

"Application partitioning distributes application processing across all system resources..."

There are 2 different kinds of application partitioning: horizontal and vertical application partitioning.

Horizontal application partitioning

Horizontal application partitioning is also known as Multi-Tier-Computing [2] which means splitting the database back end, the application server (middle tier), the web server, and the client doing the display. This nowadays is common sense and good practice.
But with horizontal application partitioning you still have not avoided the IO bottleneck on the database back end.



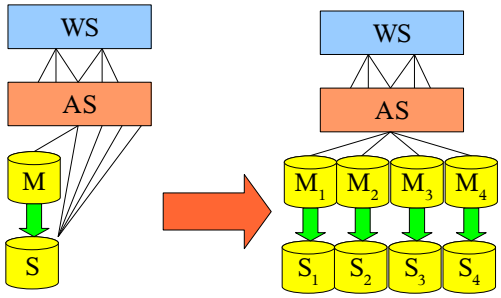
Vertical application partitioning

With vertical application partitioning you can scale-out your system to a nearly unlimited degree. The more loosely coupled your vertical application partitions are, the better the whole system scales [3].

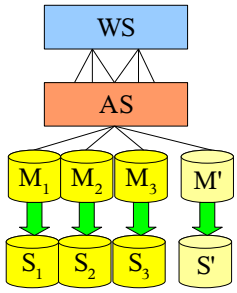
But what does vertical application partitioning now mean? For example suppose you have an on-line contact website with 1 million users. Some of them, let's say 20%, are actively searching for contacts with other people. Each of these active searching users does 10 contact requests per day. This gives approximately 2 million changes into the back end (23 changes per second). In general one contact request results in more than one change in the database and also people are doing this contact search during peak hours (1/3 of the day). This can result easily in several hundred changes per second on the database during peak time. But your I/O system is roughly limited by this formula:

250 I/O's /s per disk * #disks = #I/O /s

When you are using MySQL Replication, some caching mechanism (MySQL query cache, block buffer cache, file system cache, battery buffered disk cache, etc.) can help and when you follow the concept of “relaxation of constraints” you can increase this amount of I/O by some factors. You can handle these 1 million users on an optimized MySQL Replication Cluster system (when you have tuned it properly).
But what happens when you want to scale by factor 10 or even 100? With 10 million users your system definitely hits its limits. How do we scale here?
In this case we can only scale if we split up one MySQL Replication Cluster into several pieces. This splitting can be done for example by user (user_id).



It should be considered that the splitting is done by the entity with the smallest possible interaction. Otherwise a lot of synchronization work has to be done between the concerned database nodes. It should also be considered that some data can or must be kept redundant (general static information like for example geographic information). This can also be done by a separate replication tree:

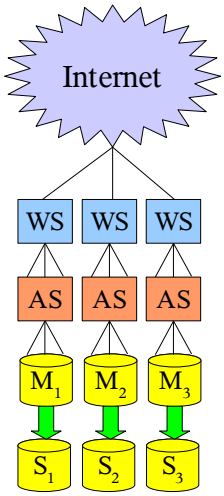


The disadvantage of this solution is, that you have to (keep) open at least 1 connection from each application server (AS) to each Master (M_n) and Slave (S_n) of each MySQL Replication cluster. So the limitation of this system is roughly:

#Conn./Server : #AS Conn. = #Replication clusters

1000 : 50 = 20

When this limit too has been hit, a much more sophisticated solution with distribution of the users in the AS and WS tier has to be considered:



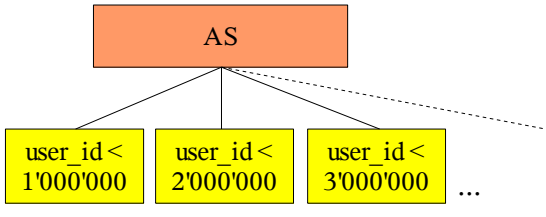
But in this concept something like an “asynchronous inter MySQL Replication Cluster” protocol has to be established.

How to partition an entity

An entity can be split up in several different ways:

Partition by RANGE

Users are distributed to their MySQL Replication cluster, for example by their user_id. For every 1 million users you have to provide a new MySQL Replication cluster:



Advantages:

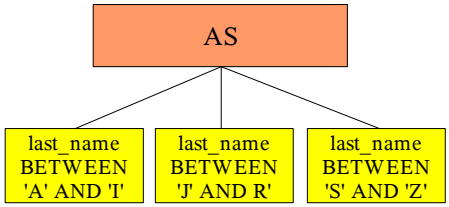
- No redistribution of users during growth needed. You only have to add a new MySQL Replication cluster.
- Improves slightly locality of reference [5].
- Easy to understand.
- Easy to locate data.
- Likelihood of hot-spots is low.
- Simple distribution logic can be implemented.

Disadvantages:

- On the "old" MySQL Replication clusters it is likely that you get less and less activity. So you either have to waste hardware resources -- which is not too serious because these machines are depreciated after some years and "only" consume some power and space in your IT center -- or you have to migrate users from the oldest MySQL Replication Clusters once in a while -- which causes a lot of traffic and probably some downtime on these 2 machines.
- Resource balancing causes a lot of migration work.
- When resource balancing is done, simple distribution logic does not apply anymore. Then a lookup mechanism is needed.

Partition by a certain CHARACTERISTCS

Users are distributed by certain characteristics for example last name, birth date or country.



Advantages:

- Easy to understand.
- Easy to locate data.

Disadvantages:

- You can get "hot-spots" for example on the server with the last name starting with "S" or some countries like US, JP, D etc., and get unused resources on servers with for example birth date February 29th, last names with "X" and "Y" or countries like the Principality of Liechtenstein, Monaco or Andorra. This can cause a necessity for redistribution of data.
- This can be avoided by merging some of the values into one MySQL Replication Cluster but then some look-up table must exist.
- Resource balancing is difficult.

Partitioning by HASH/MODULO

An entity can also be split up by some other functions like MODULO. The MySQL Replication Cluster is determined by either:

Cluster = user_id MOD #Clusters

or

Cluster = HASH(last_name) MOD #Clusters

Splitting up by DIV is already discussed in "Partitioning by RANGE".

Advantages of HASH:

- Random distribution, thus no hot-spots

Disadvantages of HASH:

- For rebalancing the whole system must be migrated!
- Hot-spots can happen if done wrong for example HASH(country) MOD # Clusters

Advantages of MOD:

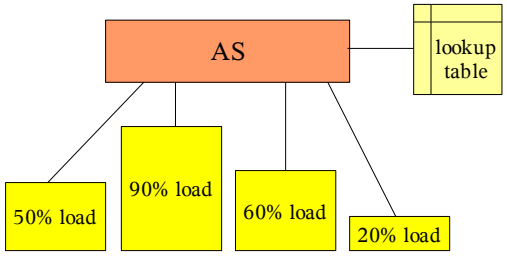
- Deterministic distribution, target cluster is easily visible "by hand".

Disadvantages of MOD:

- For rebalancing the whole system must be migrated!

Partition by LOAD (with lookup table)

A dynamic way to partition users is measuring the load of each MySQL Replication cluster (somehow) and distributing new users accordingly (similar to a load balancer). For this, for every user a more or less static lookup table is needed to determine on which MySQL Replication cluster a user is located.



Advantages:

- New MySQL Replication cluster is automatically loaded more until it reaches saturation.
- No data redistribution is need.

Disadvantages:

- When old users are not removed after some posting peaks can happen on the old systems.

Literature

[1] Alexa top 500 ranking: http://www.alexa.com
[2] Multi-Tier-Computing: http://en.wikipedia.org/wiki/Three-tier_%28computing%29
[3] Loose coupling: http://en.wikipedia.org/wiki/Loose_coupling
[4] Cluster: http://en.wikipedia.org/wiki/Computer_cluster
[5] Locality of reference: http://en.wikipedia.org/wiki/Locality_of_reference