* **Database As A Service**
* **Introduction on technology**

In the past, normally all the enterprises, universities and organizations will have their databases running on their dedicated hardware, this situation is well suited for the past because the limitation of network bandwidth, as well as for many other concerns(e.g. privacy concerns, accessibility etc.). And yet, in the past several years we saw the great improvement on network speed, virtualization technologies, and the technologies for cloud server are becoming more and more mature, big organizations are tempted to put the all their different databases into different VM, and in this way, one server will hosts many separated databases, and one DB administrator can then monitor several databases. Also, in this way, the usability of hardware improved dramatically, yet this architecture can only cater for big enterprises, and maybe wasteful for some small organizations to host such VM shared machine.

Not long ago, we witnessed the significant advancing on Cloud technologies and the enormous economic potential both for the cloud server provider and the service client themselves. The first kind of technologies immediately followed by the great advancing on virtualization technologies, of course, was the VM-cloud. In this architecture, each client's database or application are installed within a VM machine, the cloud server will provide a powerful server to host many VM from different clients, the client then no longer needed to concern the hardware, they just give their VM to the cloud service provider, and sign a contract with the cloud server provider, like "any request needed to be responded with 1 second" etc. In this way, they no longer own the physical hardware to run the database or their applications, and they don't need to hire expert database administrator to tune their database, their contract explains all, then the cloud provider provide the desire the service.

Yet the VM-cloud suffer from a lot of limitations, one clear defeat is that VM is a hard sharing of the hardware, meaning that it would be hard to divide a VM, and so one VM must be placed in one node of the cloud server, this will limit a lot of the potential concurrency ability of cloud sharing. Also, in order to use VM to host a database, each VM will need to install an OS and DBMS, therefore wasting a lot of unneeded space. Under these premises, a new cloud service is born, in this technology, the cloud service provider contains many nodes, each node hosts a DBMS, and one DBMS provide service for many clients' databases. This design has a major difference to the VM cloud method in that a client's database can be easily be partitioned, and so the pieces of a client's database can be located on several nodes on the cloud server, and so providing some potential concurrent processing ability. As before, in this design, the client also need not worrying about the physical devices and hiring a database administrator, which can be significantly more economically beneficial.

Yet another implementation of cloud sharing is for client to share a common table, but this method abandon the flexibility requirement of client databases, what if one client decide to add one column to one of the tables shared? This is a common problem, and so this implementation is never realized.

The method that different client databases sharing one DBMS in one of the many nodes in the cloud service provider, is the implementation called "Database As A Service"(DBaaS). However, in order for this service to be popular and be accept by many clients, there are many difficulties needed to be tackled.

One obvious problem is how do we provide privacy for the client? Because the client let their database run on the cloud service, some method has to be implemented in order to convince the clients that the administrator of the cloud server(the hyper supervisor) will not be able to peek into the data within their clients' databases, and also, one client's databases is entirely separated from other clients' databases. The last problem is harden because now client's database has been partitioned, and are sharing the same DBMS, without carefully design of the system, other client's data might be exposed, leading to privacy problem. As suggested in [1], one obvious solution would be using encryption on data, and the DBMS is require to do SQL query processing on encrypted data, in this way, not only other client sharing the same DBMS cannot know the data(since they did not own the keys needed to decrypt the data), but also the hyper supervisor, the database administrator on the cloud server cannot understand the client's data. Yet this implementation ensues one immediate question, that is, how do we encrypt the data in order to let the database to do query processing on these encrypted data, also, we need to do the processing on the encrypted data in an efficient way, otherwise the benefit of sharing DBMS in the cloud would be entirely ruined. Under the suggestions in [1], it has been showed that using homomorphic encryption(HOM) can enable the DBMS to perform operations on encrypted data. Also, the DBMS might employ some kind of an "onion" design on the decrypted data, that is, for each row, we use different keys to encrypt the data from several times, each layer of encryption will protect data from performing some kind of operation(for example, the outer most layer may be protecting from basic accessing of the data, then the second layer protecting update and delete, then the last layer protecting joins), only the client will have the keys for decrypting in each layer of the onion, and the client knows what kind of operations they are performing, and so, the client send only the necessary keys to the server to enable the server to do the necessary steps. This design is good in that the DBMS do not knows the keys of the encrypted data, and just doing operations on encrypted data, and different "layer" of operations are all protected and can be control by the client, providing a feasible privacy.

Another problem is that how do we partition the database, this question might not be first encountered in the database world, because even in the old days when the database are only located on the client's own dedicated hardware, sometime the database is so large that cannot be hosted within machine, and when that happens, the database administrator will need to consider how to partition the database and locate the pieces on different machine. But the problem occurs in DBaaS world is much more trickier, that is because the databases are handled automatically, and when in the old days the database administrator of their company's knows well how to partition the database will be optimal, now in the cloud, the hyper supervisor who are tuning all these databases without the ability to know any transaction details about it, will be especially difficult to know the best way to partition the database. But when one node in the cloud server cannot host the entire database, this database must be partitioned. And also, since we are using many nodes anyway, we want a way to partition the database to make some kind of queries can be fulfilled by different nodes, and so getting the concurrent benefit of the cloud, in some cases, these concurrent ability might let the query processing even faster than using dedicated hardware, since the dedicated hardware is unlikely to consist of many nodes and providing good concurrency. On the other hand, researches, as in [1], has shown that it is unwise to partition the database in a way that many transaction will span to many nodes, that is, the data that are frequently being request together should not be placed on different places, the reason is that the overhead in order to combine the data and locking on different nodes are a major bottleneck. Having this observation, [1] suggested a way to partition the database, they implemented a thread to periodically logging the set of records that are accessed together within the same operation or transaction, and they use a graph partition method to partition the database as follow: whenever data are being accessing from different nodes of the cloud server within one transaction, and edge is created, and a weight of the edge will be generated, basically representing the frequency of them being accessing together, and so, they will have a graph generated representing the potential wrong partition of a database. According to the graph, they run a graph partition algorithm on it, and alarms the system when some frequently fetch data is separated on different nodes. This ways on partition is innovated in that they no longer require the premise that the database administrator to know the database well, such as foreign key constraint and the nature of the transaction performed, and are well suited on cloud server because of its automatic partition ability.

Yet another problem concerns how do we relocate the pieces of databases and how do we know a "packing" of pieces of databases from different clients can be efficiently working together within a node. At the first part of this problem, assumes now we already know that in some node the packing has exceeded the capability of a single node, and we must perform the operation of migrating a piece of one client's database to another node, the main concerns here is that, we need a method to migrate the piece of database to another place without any downtime, and possibly reduce the increased latency during the migration as much as we can. Although the cloud server's nodes and usually connected with very high bandwidth network cable, given that the size of the migration piece might be large(from the discussion on the previous problem we know that we might migrate a large piece of database if all the data inside it are frequently accessed within a single transaction), the migration operation takes at least several minutes to hours, therefore we must strike for a way to do the migration "on-the-fly", but then because transferring a large piece of data will undoubtedly consume some or considerable available network bandwidth of each node, and so migration operation done in a not careful fashion will degrade the whole system's performance. And so, when we need to decide which piece of data need to be migrated, we must also take into the account of the effect it has on the whole system, and see whether it is worth to migrate at the expense of the degrade of the system's performance.

The other part of this problem is to decide whether a "packing" of pieces of clients' databases is appropriate(i.e. not exceeding the node's capacity). Let's divide this problem into two part, that is, knowing the capacity of each node, and another part is obviously, knowing the consumption of different resource of each located database piece. The former one seems obvious, the hyper supervisor will normally know a node's RAM size, CPU rate, network bandwidth and disk size etc. But we need to be more careful about this, could these physical parameter encapsulate one node's total capacity? Will there be any other concerning parameter that is not known until the run-time, and therefore blindfolding the hyper supervisor? In fact, and buffer pool can be hard to predict, also, and the total Disk I/O maybe tricky, we need to take into account whether the I/O is on a random basis or sequential basis. And so, for the cloud service architect, it is important that the system takes care of all these run-time details automatically, and let the hyper supervisor to concern only about the physical capacity, which is well suggested in [3]. This approach also having the advantage that it is no longer necessary for the hyper supervisor to have an inside into one client's database workload, because enabling the outsider to peek into the workload may give the opportunity for the outsider to guest which client's database piece it is, or what kind of operation this database is doing, and performing some malicious operations.

As have been suggested in the previous paragraph, another side of this problem is that we need to determine the total resource consumption of each of the pieces of clients' databases, which has been proven to be an non-trivial task. In the VM-cloud architecture, however, this task is considerably easier, that is because of the nature that VM machine can be hard to split and run on different nodes, and so normally the VM will be located within one single node, and also, because the VM is like the sealed box of the client's data, it is much easier for us the deduce that total resource consumption once we hack into the VM controller, the only hard part would be that since several VM are still sharing the same hard drive, and so the disk I/O might be a little bit harder to predict. In the Database As a Service architecture, the situation becomes much more tricky by the fact that now the client's databases are commonly sharing the DBMS, and the DMBS's buffered pool are also shared, the client's operations are now no longer "sealed inside a box", the sharing pattern is much more intricate. We also do not wish to achieve the monitoring of a client's database without introducing a significant change to the currently available DBMS, because it pushes too much pressure on the DBMS implementation and so becomes un-attractive to DBMS implementer, and also it result in a less flexible architecture. And so, the modern Database As a Service provider will construct their monitoring component above the available DBMS, and calculate various resource consumption profile either directly extracted from the DBMS itself, or do some estimation based on the available data.

As mentioned in the previous paragraph, monitoring each client database's resource consumption profile has proven to be non-trivial. This is mainly result from the fact that current DBMS do not provide enough useful details of resource consumption, or even if the DBMS provides the data, it is not on a shared per-client basis. [3] has shown that based on the OS profile and some plugin or enhancement of some DBMS, some of the required information can be directly extracted from, these resource data might includes CPU usage, network bandwidth usage, the number of lock acquired on a per table basis and RAM, buffered pool consumption. And yet for some other kind of information, it is hard to be extracted directly from neither the OS nor the DBMS, and require a more sophisticated method. In [3], it has been suggested that estimating disk write is an non-trivial task, we obverse that there are basically two kind of disk write operations performed on a normal database, one is the log write, which provide the undo and redo ability, another kind is the disk write of dirty pages. Since the fact that log write are mostly sequential and are linearly related to the number of transactions executed, it would be easy to deduce the number of log write once we hack into the log documented by the DBMS. Yet the disk write for the dirty pages is much more tricky, because it tend to be random writes, and the trace of the write operations are scatter in different places. The authors in [3] make a important observation that at the time the system is in a stable state, the speed of all the pages are getting dirty are the same with the speed of the pages that are flushed into the disk. Since tracking the rate of pages being dirtied is trivial, we have a approximation of the disk write for data. Although this method neglect some facts such as when dealing with databases which number of disk write are highly fluctuated, it makes a good approximation on general databases.

Another solution of estimating the resource usage profile has been given in [2], in the resource metric system they devise, they have also discovered that some sub-set of the required information can be directly extract from the OS or the DBMS, the information used in this scheme includes the Write Percent( the percentage of write operation perform against all operations), the Average Operation Complexity( the average number of pages touched in one transaction), the Percent Cache Hit (as the name suggested, is the percentage of cache hit, the paper, however, does not specific whether it is L1 cache or other kind of cache), the Buffer Pool Size, the OS Page Cache Size, the Database Size, the Throughput (the average number of transaction completed by a single client in one second). The main metric that they used to calculate the resource consumption pattern for each client is expected disk IOPS, throughput, and operation complexity. They use all the information extracted above to calculate each of these metric. Oddly enough, they have also discovered that the disk I/O rate cannot be non-trivially extracted from neither OS nor DBMS directly, and instead of making the assumption on the page dirtied rate and page flushing rate being the same at the steady state as in [3], the author in [2] devise an estimation equation for approximating the disk I/O per second. Let P(A) denote the buffer pool miss probability, that is, on a page request, the page cannot find in the buffered pool, let P(B) denote the cache miss probability. The greatest assumption on this equation is that we assume event A and event B are independent to each other, and so the expected disk I/O per second is given as P(A ∩ B) = P(A)P(B). This equation will be further expanded, eventually it will be able to calculated provided that we have extracted the aforementioned information from the OS or the DBMS.

Once we have generate the detail resource consumption profile for each of the tenant in one node, many further operation becomes viable, the most noticeably viable operation now is to decide in run-time, whether the "packing" of different tenant in one node has been exceeded the maximum capacity of that node, and when this situation happens, we take the migration operation. The run time property can only be done based on the resource profile extracted above is that, cloud server normally contains a large number of tenant, and the hyper supervisor has little inside into the tenant query's nature. Requiring the manually adjustment of the placement of pieces of databases in these many nodes would be nearly impossible. And so an automated solution is required. In [2], the author has provided us with an automated solution. As mentioned in the previous paragraph, the three main metric used in [2] is expected disk IOPS, throughput, and operation complexity, and for each of these three metric, we classify the recourse consumption of one tenant into high, medium and low group, and so one possible profile of a tenant might be "high disk IOPS; high throughput; medium operation complexity". Also, for each node, it can be in one of the three states:{under utilized, good utilized, over utilized}. We use a vector representing the number of tenant in each of the resource usage profile group, for example, the vector [2,0,3] may represent there are 2 tenant with "high disk IOPS, high throughput and medium operation complexity" and 3 tenant with low in all these three metric. Once the node's resource consumption has exceeded some threshold on either disk IOPS, throughput or operation complexity, the node will be label "over utilized" are will the subject to migrating tenant from. The thresholds of these three metric required the administrator of the cloud service provider to define, yet given that now the hyper supervisor only concerns the overall aspect of one node, and the administrator is suppose to know these three metric given that they are comparatively "visual" to the administrator, and so this requirement could be easily met. Once an over utilized node has been detected, a search algorithm is perform, to expedite the finding of the recipient, [2] suggested that we should use the hill-climbing algorithm, and consider all the immediate neighbors, the algorithm will result with a destination node with the greatest improvement on the overall performance of the system. If no destination could be found, it simply suggested that the administrator should make plan for acquiring more nodes for the system.