International Journal of Computer Applications (0975 – 8887)
Volume 52– No.2, August 2012

A Challenge in Improving the Consistency of Transactions in Cloud Databases - Scalability

R. ANANDHI
Registered Scholar in Computer Science and Engineering
SCSVMV University
Kanchipuram, Tamil Nadu, India

K. CHITRA
Phd, Assistant Professor in Computer Science
Government Arts College, Melur
Madurai, Tamil Nadu, India

ABSTRACT

Cloud computing is a automated on-demand self-service paradigm, allowing a pay-per-use model on shared resources. Scalability is one of the challenges faced by cloud computing to be achieved at full strength. The “cloud scalability” is slowly gaining weight in cloud industry since the load of traffic is usually unpredictable. Load balancers are appointed to monitor the traffic and scale accordingly. This paper will give the types of scalability, choosing the correct scalability and other issues.

General Terms

Scalability, Cloud Computing

Keywords

Horizontal scalability, Vertical Scalability, Scalability factor.

1. INTRODUCTION

There are many factors used to improve the performance of a Database. One of them is **SCALABILITY**. Scalability means doing what in a bigger way. Scaling an application indicates allowing more people to use the application. There are mainly two types of scaling an application. They are:

- **Vertical Scalability**: It denotes the addition of extra resources within the same logical unit to increase the capacity. E.g.: Addition of CPU to an existing server, expanding storage by adding hard drive. Therefore it increases the capacity of existing hardware and software. It is the ability of the application to be scaled under load. Here, the database uses many cores and CPUs that share RAM and disks.
- **Horizontal Scalability**: It denotes the concept of addition of multiple units of resources and treats them as a single unit. E.g.: Distributed systems. It denotes the strength of the application to be scaled in order to face the demands by making the copies of application (replication) to satisfy the increasing user demands.. It is a traditional load balancing model and it`s an integral component of cloud computing environment. Therefore it refers to the ability to distribute both the data and the load of simple operations over many servers with no RAM or disk shared among those servers.

2. SCALABILITY FACTOR

Every component whether it is processors, servers, storage or load-balancers that is to be scaled need some kind of management overhead. During scaling, it is important to note what percentage of resource is actually scalable. It is called as **SCALABILITY FACTOR**. E.g.: If we lose 5% of the processor power every time we add a CPU to the system, then the scalability factor is 0.95.

Scalability can be further classified into four types based on the scalability factor. They are:

- **Linear scalability:** Here scalability remains constant in spite of scaling.
- **Sub-linear scalability:** Here scalability factor falls below 1.0.
- **Supra-linear scalability:** It is possible to get good throughput just by adding one resource (which is very rare) is called as supra-linear scalability.
- **Negative scalability:** If the performance of an application degrades when the application is scaled is called as negative scalability.

3. CHOOSING OF SCALABILITY

If an application needs urgent scaling, going to the choice of vertical scalability is wise. But the cost of vertical scaling is directly proportional to the size of application. Horizontal scalability is comparatively works at low budget. It implies that the horizontal scalability can be done using the existing commodity storage and server solutions. Going deep into the technology, horizontal scalability is also not a cheap one since the application has to be built from the scratch to view as a single unit. The horizontal scalable system faces two problems as “Split brain” and “Hardware failure”. Achieving infinite vertical and horizontal scaling is totally impossible. If we are writing an application for a pre-determined number of users, choosing vertical scalability is wise. But if the number of users grows to millions, going vertical will be a difficult one as it becomes very expensive.

Selection of correct scalability depends on the factor how much we want to scale that application. There is no “one size fits all” solution for scalability. Many people aim to improve the processing power when considering the concept of scalability. But for a successful scalable application, all layers like storage layer, the database layer, application layer, the web layer, load balancer, firewall etc are all have to be scaled equally. Usually focus is on horizontal scalability since the number of cores is limited in sharing the memory and also it is less expensive and use commodity servers.

Bigger hardware is not always fast but it can handle more loads. It is similar to a highway with more lanes. The reason is that SQL response times will not automatically improve on bigger hardware. Horizontal Scalability can be achieved through load balancing solution. It's the vertical scalability which is difficult to achieve in a cloud computing or virtualized environment.

The problem is that a single database or SQL query that is poorly constructed can destroy vertical scalability and actually increase the cost of deploying in the cloud because we are paying for the resources as per the usage. Cloud computing does not going to do any magic in optimization of codes or queries or database tables; but it lies in the developers hand to choose whether or not the cloud computing is used as the deployment model.

4. CLOUD SCALABILITY

When considering cloud computing, the concept of vertical scalability is very important since the charges are based on the usage of resources like the old mainframe model. If the application is not properly scaled vertically, it is going to increase the cost of the application to run in the cloud. Cloud computing providers also would not address the vertical scalability unless they are peculiar to the application. No external solution will help to optimize the code; they can raise the overall performance by normalizing protocols, network bandwidth etc; but it can't go deep into the code like rearrange the joins, rewrite the poor loop, refactoring a data structure etc.

Vertical scalability is in the domain of the application developer whether the application is going to be deployed inside the data center or outside the cloud. The developers can make use of the technologies like network side scripting, application delivery solutions etc to help the efforts for vertical scalability; but there is also limitation to address the root cause of the application's failure to vertically scale.

Improving the vertical scalability is important in achieving the low investment on cloud computing and virtualization. Applications not responding to vertical scalability will finally become expensive when implements in cloud services as requirement on resource increases with demand of application. Cloud computing and virtualization have impacts on vertical scalability by using horizontal scalability techniques to give the surety that capacity meets demand and performance level agreements are met. Improving vertical scalability can be achieved by optimizing SQL queries, understanding the bottlenecks associated with the programming languages used, using API, taking advantage of offload capabilities of application delivery solutions available and aware about the decomposition of the applications into finely grained services.

5. SCALING AT MESSAGING SYSTEM

We can improve the scalability of messaging system by adding multiple brokers to the system, thus escaping the inherent resource limits of a broker deployed on a single machine. Brokers can be combined into a network by adding network connectors between the brokers, which enables to define broker networks with an arbitrary topology. When brokers are linked together as a network, routes from producers and consumers are created dynamically, as clients connect to and disconnect from the network. Therefore the consumer can connect to any broker in the network and the network automatically routes messages from producers attached at any other point in the same network.

There are two approaches for routing messages through a broker network namely Static Propagation and Dynamic Propagation.

- ❖ **Static propagation**: It wants to explicitly specify the routes, by telling the broker where to forward messages for specific queues and topics using pattern matching. So, we can configure the brokers to disable advisory messages altogether, which eliminates the scalability problems associated with advisory messages.
- ❖ **Dynamic propagation**: It necessitates sending advisory messages throughout the broker network, which the brokers then use to figure out the optimal route in a dynamic way. It is more flexible than static propagation. There is a risk that the advisory messages could swamp the traffic in the broker network as scaling is done on the network.

6. ALIAS FOR HORIZONTAL SCALABILITY

An alternative horizontal scaling strategy is to deploy multiple brokers, but to leave them isolated from one another, so that there is not broker network. So the decision can be taken by the client for this case to decide which broker to send messages to or receive messages from. This approach requires the partitioning of messages into various categories based on the receiver's name; so the messages belonging to a particular receiver can be handled by a separate broker.

The merits of this approach are:

- ❖ Usage of tuning techniques for vertical scaling.
- ❖ Achievement of better horizontal scalability than a network of brokers since there is meager crosstalk among brokers.

The demerit of this technique is the need of complex clients

- ❖ as they have to know the way of partitioning of messages and
- ❖ selection of appropriate broker for routing messages to a particular destination.

Query response time can be considerably decreased by proper indexing of data. Proper indexing aims to exploit the logarithmic scalability of the B-Tree index to its full extent. Response time problems are caused by sloppy indexing. The response time difference is stunning. It is hardly possible to improve it by scaling horizontally however it would be easy to cut the response time by adding more servers. The horizontal performance gains of the so-called NoSQL systems are mostly on the write side—often reached with the eventual consistency model. They allow temporary inconsistencies that will finally become consistent. But SQL insists on the very rigid consistency. This increases response times for write operations but not give worse throughput.

Maintaining a strict consistency in a distributed system—E.g. like scaling horizontally with multiple servers requires the members to coordinate all changes in a synchronous manner. But the drawbacks of this approach are:

1. Synchronous communication adds latencies and increases response times.
2. Reduces the overall availability since every change needs many clients to participate.

A distributed SQL database achieves consistency by two-phase commit (2PC) protocol. 2PC allows each system to start a global transaction that modifies data in both systems. The global transaction maintains the consistency. It wants the all or nothing property. It will not succeed if one system is unavailable—the overall system availability is reduced. Even if all are available, 2PC increases the response time due to the additional effort. The troublesome on strict consistency is directly proportional to the number of nodes in the network. In practical applications, strict consistency is hard to apply across more nodes. Dropping of strict consistency will lead to solve the availability problem as well as reduce the overall response time. The idea is to have global consistency after executing write operation. So consistency is eventually reached by handling conflicts and not by preventing conflicts.

Having more hardware will not improve the response time; rather it will result only in the latencies decreasing the response time. Therefore the more complex the infrastructure, the more latencies arise, the slower the response time. Disk Seek Time also has a impact on the response time since moving the disk head requires considerable amount of time.

7. HOW TO IMPROVE CLOUD SCALABILITY?

- ❖ **Auto Scaling**: It is one of the unique features of cloud computing and implemented in Amazon’s EC2. The components needs are a load balancer and a couple of web servers. Setup the auto-scaling algorithm and initialize the threshold value based on the network traffic. When the setup threshold value is PASSED, Amazon’s EC2 will spins a new web server and automatically roll it into the load balancer pool. Similarly when the traffic falls below the threshold value, Amazon will take a server from the allotted pool.
- ❖ **Scale the Database tier Horizontally**: NoSQL offers a number of solutions for this approach. Configure NoSQL in a master-master active passive cluster, also known as circular replication. All the completed transactions will be sent to the other server in the cluster by NoSQL. The passive server will handle the read transactions and thereby increasing availability and scalability. If the network traffic increases, add a additional read-only server.
- ❖ **Striped Elastic Block Storage (EBS):** EBS is a good approach to bring the flexibility of a storage area network in traffics. To improve EBS performance, use Linux RAID technology. Since EBS already has redundancy, we can use striping or RAID 0 across a number of EBS volumes (usually 4).

8. CONCLUSION

Therefore throughput and response time are the two views of measuring the performance of a system while scaling. More hardware will not improve query response time while proper indexing is the best way to improve. There are also other types of scalability like container-level scalability and database scalability to be dealt. Network scalability is a rare topic to be touched by the experts. Cloud applications also depend upon the network resources during consistency and scalability performance issues.

9. REFERENCES

[1] H. C. Lim, S. Babu, and J. S. Chase, “Automated control for elastic storage,” in ICAC10. New York, NY, USA: ACM, 2010, pp. 19–24.

[2] P. Marshall, K. Keahey, and T. Freeman, “Elastic site:Using clouds to elastically extend site resources,” Cluster Computing and the Grid, IEEE International Symposium on, vol. 0, pp. 43–52, 2010.

[3] E. Berger and J. C. Browne, “Scalable load distribution and load balancing for dynamic parallel programs,” in IWCBC99: In Proceedings of the International Workshop on Cluster-Based Computing 99, Rhodes/Greece, 1999.

[4] The University of Melbourne, “Dynamically Scaling Applications in the Cloud”.

[5] R. Cattell, “Scalable SQL and NoSQL Data Stores”.

[6] University of Technology, Sydney, Australia,”Availability and Load Balancing in Cloud Computing”.

[7] Stonebraker and R. Cattell, “Ten Rules for Scalable Performance in Simple Operation Datastores”.