Volume 4, Issue 4, April 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Strategic Model to Enhance the Vertical Scalability in Cloud Environment

K. P. Yadav
Mangalmay Institutions
Greater Noida, India

Monika Sainger
Inderprastha Engineering College
Ghaziabad, India

Abstract- ***In spite of the claimed advantages, the cloud still needs some work with regards to a variety of different issues. In this paper we try to address the issue of vertical scalability of an application in a cloud. Improving the vertical scalability of applications is important in achieving the benefits of a reduction in costs associated with cloud computing and virtualization. Applications that fail to vertically scale well may end up costing more when deployed in the cloud because of the additional demand on compute resources required as demand increases. In this paper we introduce a strategic model DOCS2 to enhance the vertical scalability in cloud environment. This model consists of five strategies to improve the vertical scalability in different phases of an application.***

Index Terms— vertical scalability, strategic model, fine-grained parallelism, optimization, application delivery controller, API manager, network-side scripting.

I. INTRODUCTION

Computer systems or individual applications have capacity limits. Scalability is the ability of a particular system to fit a problem as the scope of that problem increases.[6] Scalability, simply, is about doing something in a bigger way. Application Scalability is a measure of the ability of an application to expand to meet our business needs. Generally, we can scale a given application by adding more (or bigger) resources when needed. Almost all elements of IT can be scaled. A few of them are CPU% allocated for a task, amount of on-chip memory (RAM), space on the hard disk, number of application instances, number of end users, number of communication ports, communication bandwidth and number of software licenses [1]. Scaling a web application is all about allowing more people to use an application. In other words, it is the ability to increase the application throughput in proportion to the hardware that is being used to host the application. If an application is able to handle 100 users on a single CPU hardware, then the application should be able to handle 200 users when the numbers of process are doubled. Scalable applications are able to operate normally as they grow and can have more resources added at any time to service more customer demand. Applications that are **not** scalable encounter performance and service availability problems as demand increases. These kinds of applications may not be able to take advantage of more resources.

A. Horizontal Scalability Vs Vertical Scalability

There are two key primary ways of scaling web applications which are in practice today.

Horizontal Scalability refers to adding multiple logical units of resources and making them work as a single unit. Most clustering solutions, distributed file systems; load-balancers help you with horizontal scalability. Horizontal scalability is the ability of an application to be scaled up to meet demand through replication and the distribution of requests across a pool or farm of servers. It's the traditional load balanced model, and it's an integral component of cloud computing environments.

Vertical Scalability refers to adding resource within the same logical unit to increase capacity. An example of this would be to add processors to an existing server, or expanding storage by adding hard drive on an existing RAID/SAN storage. Vertical scalability is the ability of an application to scale under load; to maintain performance levels as the number of concurrent requests increases. While load balancing solutions can certainly assist in optimizing the environment in which an application needs to scale by reducing overhead that can negatively impact performance (such as TCP session management, SSL operations, and compression/caching functionality) it can't solve core problems that prevent vertical scalability.

Horizontal scaling is particularly effective for building high-throughput web-centric applications. The web-tier, e.g. a social networking website is able to take advantage of this approach, because such web applications generally use small non-shared memory space having many independent threads [2]. Also they have a loosely-coupled external structure and run possibly on many OS's. Google, Yahoo, eBay and Amazon all rely such cluster architecture for their computational needs [6].

Vertical scaling is especially efficient for relational database tiers, due to the large shared memory space, many dependent threads and the tightly-coupled internal structure [6]. The major server vendors continue to invest heavily in building bigger and more powerful shared-memory servers [2]. The problem is that a single database table or SQL query that is poorly constructed can destroy vertical scalability and actually increase the cost of deploying in the cloud. Because user generally pay on a resource basis, if the application isn't scaling up well it will require more resources to maintain performance levels and thus cost a lot more.

Horizontal scalability achieved through the implementation of a load balancing solution is easy. It is the vertical scalability that has always been and remains difficult to achieve, and it is even more important in a cloud computing or virtualized environment because now it may cause the user pay more for the same thing. If an application doesn't vertically scale well, it's going to increase the costs to run in the cloud.

II. VERTICAL SCALABILITY: The Domain of the Application Developer

In cloud computing environment the issue of scalability needs to be discussed because there are different strategies to solving problems related to achieving application scalability and not every piece of application or computing resource scales with the same success. For example in cloud computing, the advertised performance for "infinite scalability" looks very different if applied to a traditional, relational database (found in most web applications) vs. a distributed, large scale non-relational, multi-dimensional database such as Google's BigTable. Also cloud computing is not going to magically optimize code or database queries or design database tables with performance in mind, that's still squarely in the hands of the developers regardless of whether or not cloud computing is used as the deployment model.

Cloud computing and virtualization can certainly address vertical scalability limitations by using horizontal scaling techniques to ensure capacity meets demand and performance level agreements are met. But doing so may cost you dearly and eliminate many of the financial incentives that led you to adopt cloud computing or virtualization in the first place. Also, cloud computing providers can't, and probably wouldn't if they could (it makes them money, after all), address vertical scalability issues because they are peculiar to the application. No external solution can optimize code such that the application will magically scale up vertically. External solutions can improve overall performance, certainly, by optimizing protocols, reducing protocol and application overhead, and reducing bandwidth requirements, but it can't dig into the application code and rearrange the order in which joins are performed inside an SQL query, rewrite a particularly poorly written loop, or refactor code to use a more efficient data structure. Thus, vertical scalability, whether the application is deployed inside the local data center or out there in the cloud, is still the domain of the application developer. Here we discuss a strategic model DOCS2 to enhance the vertical scalability in cloud environment.

III. DOCS2 MODEL - A Strategic Model to enhance vertical scalability

Improving the vertical scalability of applications is important in achieving the benefits of a reduction in costs associated with cloud computing and virtualization. Applications that fail to vertically scale well may end up costing more when deployed in the cloud because of the additional demand on compute resources required as demand increases. In this paper we have introduced a strategic model DOCS2 (Fig. 1). DOCS2 discusses five stages where the application developer can work to improve vertical stability.

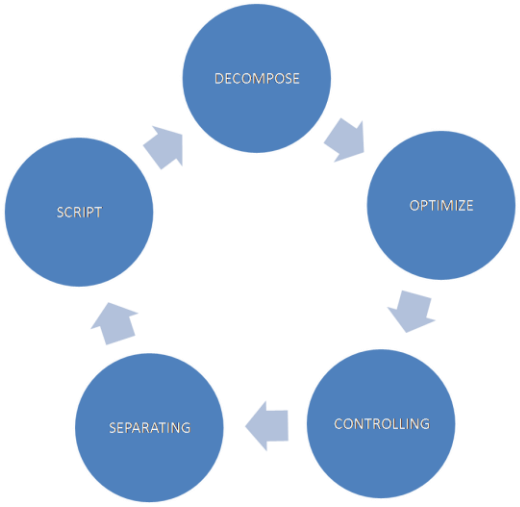

Fig. 1 DOCS2 Model

A. Decompose

Decomposing applications into too finely grained services is one of the best approaches to extracting the excellent performance from the underlying hardware. By using this approach the business applications speeds up. To make effective use of fine-grained parallelism, it becomes important to think of processing not so much in terms of long functional threads but smaller-sized tasks that can be handed to a thread pool. Fine-grained parallelism approaches threading, not at the thread level, but at the task level. Most threading APIs have some concept of a thread pool although the design and quality of implementation varies. When the thread pool is started up by the program, it creates a certain number of so-called "worker" threads. These threads are just regular threads, except that when they complete their

assigned task, the thread pool does not let them die off. Rather, it keeps them alive by putting them into a dormant state until there is new work to be done. These threads, in other words, stay alive as long as the thread pool is alive, which is generally until the program shuts down. This approach avoids the cost of creating and terminating threads, which is invariably an expensive process. By assigning tasks to individual threads, you enable those tasks to run in parallel. This gives you better overall performance and, because it relieves the main thread from doing all the work, it permits faster and snappier visual presentation.

While the software-based, fine-grained multithreaded environments have demonstrated superior performance over heavyweight, OS-level threads for irregular applications, software-based lightweight multithreading has its limits where the overhead of thread management becomes significant. To solve this problem, researchers have proposed an architectural approach called lightweight chip multi-threading (LCMT), to maximize the on-chip fine-grained parallelism by providing direct hardware support for fine-grained threads [3]. The proposed LCMT architecture [3] targets throughput computing. Therefore, it is designed as an in-order multithreaded chip multiprocessor. LCMT provides direct architectural support for fine grained multithreading to maximize the fine grained parallelism on-chip and achieve improved performance. LCMT is based on a mainstream multithreaded processor, specifically Niagara [5], with very small additional hardware overhead. This enables it to evolve with the mainstream designs with minimal additional cost. LCMT can leverage legacy software with few changes. Therefore, current applications can be easily modified to run on the LCMT architecture and achieve better performance. On comparing the LCMT architecture with Niagara-like baseline architecture, results show up to 1.8X better scalability, 1.91X better performance, and more importantly, 1.74X better performance per watt.

B. Optimize

This stage of model is about optimizing the database queries. SQL statements are generally used to retrieve data from the database. We can get same results by writing different SQL queries. But use of the best query is important when performance or scalability is considered. So we need to SQL query tuning based on the requirement. SQL optimization techniques can be used to optimize the queries for better performance. Following are the query optimization techniques on different aspects of the query (Fig.2).

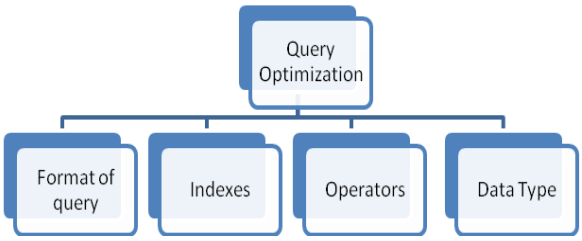

Fig. 2 Query Optimization Techniques

1) Format of query**:** The SQL query becomes faster if we use the actual columns names in SELECT statement instead of '*'.
   - Try to minimize the number of subquery blocks in the query.
   - Use non-column expression on one side of the query because it will be processed earlier.
   - To write queries which provide efficient performance follow the following general SQL standard rules:
     - Use single case for all SQL verbs.
     - Begin all SQL verbs on a new line.
     - Separate all words with a single space.
     - Right or left aligning verbs within the initial SQL verb.

   - We can optimize the query with wildcard by doing a postfix wildcard instead of pre or full wildcard. For example,

     *SELECT * FROM TABLE WHERE COLUMN LIKE 'hello%';*

     That column must be indexed for such optimize to be applied. Doing a full wildcard in a few million records table is equivalent to killing the database.
   - Another common way of optimizing a query is to minimize the number of row return.
   - In MySQL the columns have default values. So insert values explicitly only when the value to be inserted differs from the default. This improves the insert speed.
2) Operators
   - HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.
   - Use operator EXISTS, IN and table joins appropriately in the query as usually IN has the slowest performance. Also IN is efficient when most of the filter criteria is in the sub-query whereas EXISTS is efficient when most of the filter criteria is in the main query.

- Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.
- Try to use UNION ALL in place of UNION in the queries.
- Symbol operator such as >,<,=,!=, etc. are very helpful in the query. We can optimize some of our queries with symbol operator provided the column is indexed. For example,

*SELECT * FROM TABLE WHERE COLUMN > 16*

Now, the above query is not optimized due to the fact that the DBMS will have to look for the value 16 THEN scan forward to value 16 and below. On the other hand, a optimized value will be

*SELECT * FROM TABLE WHERE COLUMN >= 15*

This way the DBMS might jump straight away to value 15 instead. It's pretty much the same way how we find a value 15 (we scan through and target ONLY 15) compare to a value smaller than 16 (we have to determine whether the value is smaller than 16; additional operation).

- Try to avoid NOT operator in SQL. It is much faster to search for an exact match (positive operator) such as using the LIKE, IN, EXIST or = symbol operator instead of a negative operator such as NOT LIKE, NOT IN, NOT EXIST or != symbol. Using a negative operator will cause the search to find every single row to identify that they are ALL not belong or exist within the table. On the other hand, using a positive operator, search will just stop immediately once the result has been found.
- Max and Min operators look for the maximum or minimum value in a column. We can use Max or Min on columns that already established such Indexes. If a particular column is frequently in use, having an index should help speed up such searching and at the same time speed max and min operators. This makes searching for maximum or minimum value faster. Deliberate having an index just to speed up Max and Min is always not advisable.

3) Indexes

- Indexing the column is a common way to optimize the search result. However one must fully understand how indexing does work in each database in order to fully utilize indexes as useless and simply indexing without understanding how it work might just do the opposite.
- Some database such as MySQL search better with columns that are unique and indexed. Hence, it is best to remember to index those columns that are unique.
- Don't use a subquery within the IN operator as Doing this is very expensive because SQL query will evaluate the outer query first before proceed with the inner query. Instead use a dummy table to store intermediate result or an EXIST operator.
- Indexes lose their speed advantage when using them in OR-situations in MySQL at least. On the other hand, using Union will utilize Indexes i.e. runs faster.

4) Data Types

- Use the most efficient (smallest) data types possible. For example, use the smaller integer types if possible to get smaller tables. MEDIUMINT is often a better choice than INT because a MEDIUMINT column uses 25% less space. On the other hand, VARCHAR will be better than long text to store an email or small details.)

To most effectively optimize queries, one should start by identifying the queries that have the longest duration. This can be done by using SQL Profiler. Next, analyze the queries to determine where they are spending their time and whether they can be improved. One can use the SQL Query Analyzer to help analyze query behavior.

C. Controlling

The idea here is to choose an appropriate application delivery controllers and server load balancing products to ensure efficient and effective website infrastructure to meet today's needs, while ensuring the right upgrade path for tomorrow's business requirements. An application delivery controller is a data center network device that helps manage client connections to complex Web and enterprise applications. Enterprises typically deploy an application delivery controller behind a firewall and in front of one or more application servers. The application delivery controller has the core functionality of a server load balancer, which directs clients to individual servers based on total existing connections, CPU utilization and other factors. It also performs computationally intensive application delivery and session management functions that negatively affect server performance, such as application acceleration, SSL VPN offloading and application layer security.

Application delivery solutions were built to address the challenges associated with website infrastructure complexity, performance, scalability and security. Application delivery solutions are quite diverse. They may be known as application delivery controllers (ADC). ADCs provide the ability to direct Internet users to the best performing, most accessible servers. If a server becomes inaccessible due to any type of failure, the ADC will take that server or application off-line, while automatically re-routing users to other functioning servers. This process is essentially seamless to the user, and critical to servicing the customer. In addition, by using various load balancing algorithms, an ADC can distribute users to servers that offer the best possible performance. The ADC can dynamically interrogate key server elements such as the number of concurrent connections and CPU/memory utilization.

To further enhance, and secure the user experience, more-advanced ADCs provide SSL offload/acceleration. SSL acceleration in the ADC enables you to offload the SSL handshake and encryption/decryption processes from the servers. This offloading dramatically increases the servers' performance, while decreasing the time and costs associated with the

server's SSL certificate management. Application delivery controllers use various techniques to distribute the traffic load between two or more servers, routers, firewalls, and other networked resources, to optimize resource utilization and improve website performance and response time.

A small-to-medium sized organization should look for some functionality as the criteria when choosing a network and application delivery solution such as High Availability (Hot Standby), Layer 4 Load Balancing, Layer 7 Content Switching, Layer 4 & 7 Persistence, SSL Acceleration, Using Persistence with SSL, Windows Terminal Services Load Balancing, Persistence and Session Directory Management, Number of LAN Ports, Maximum Real Servers, Maximum Real Servers per VIP, Throughput, Maximum SSL TPS , Direct Server Return (DSR), Transparency, HTTP Compression, HTTP Caching, Intrusion Prevention, Resource-based Load Balancing. The below diagram maps forecasted Application Delivery Controller End-User Spending, Worldwide, 2005-2010.

D. Separating

Application scalability, performance, and reliability today are just as much reliant on architecture of its infrastructure as it is its design and implementation. Architecture of web 2.0 sites that hope to grow into monsters needs to start earlier, before it ends up standing on the verge of collapse because of high demand. And while load-balancing is certainly a part of that architecture, it's necessary to move higher up the stack, into the application layer, in order for network-focused products to be a part of the solution from day one. Such solutions need intelligence, and application fluency that can be leveraged in the original architecture such that scalability *is* as easy as adding new servers again, especially in the light of the growing popularity of "built to fail" infrastructures.

In this stage of model we consider separating APIs from the presentation layer of a site and running the processes that generate RSS feeds on separate servers from APIs and the general site. Load-balancing such an environment is going to need some intelligent applications to intelligently direct requests to the right server internally that is further going to require some intelligence to understand individual requests and the network conditions that exist in real-time, such that optimization and acceleration features like compression and caching can be employed when it makes sense and when it will be a benefit.

WSO2 API Manager is a complete solution for publishing APIs, creating and managing a developer community and for routing API traffic in a scalable manner. The crews at WSO2 have a firm grip on this reality[8]. Therefore when designing and developing the WSO2 API Manager, they made scalability of the end product a top priority. In general the API Manager can scale under following circumstances.

- Growing number of API subscribers (growth of the user base)
- Growing number of APIs (growth of metadata and configurations)
- Growing number of API calls (growth of traffic)

Following schematic provides the architecture of WSO2 API Manager and how it can scale against the factors listed above.

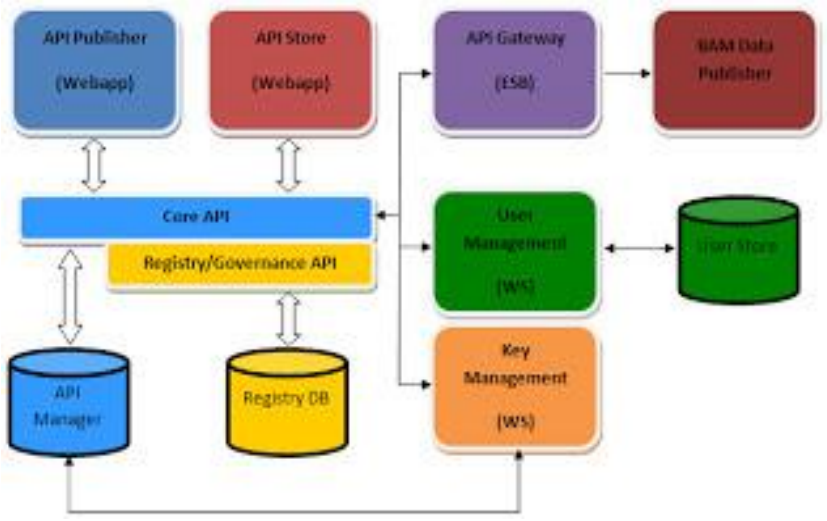

Fig. 3 Architecture of WSO2 API Manager

WSO2 API Manager (Fig. 3) uses 2 main databases - the registry database and the API management database. The registry database is used by the underlying registry components and governance components to store system and API related metadata. API management database is primarily used to store API subscriptions. In a scalable setup, it will be necessary to create these databases elsewhere, ideally in a clustered and high available database engine. One may use a MySQL cluster, SQL Server cluster or an Oracle cluster for this purpose. In a scalable deployment we might cluster some of the internal components of the WSO2 API Manager. Therefore there will be more than one JVM involved. All these JVMs can share the same databases created in the same clustered database engine.

E.Scripting

This phase of DOSC2 model discusses the use of network-side scripting. Network-side scripting is implemented by advanced Traffic Management devices. These are typically software proxies for application traffic, although some vendors only provide the technology through proprietary hardware appliances. The network is one of the many internal interfaces in a typical web-based application. We can control the entire application, both server-side and client-side, if we can inspect and manipulate the traffic on the network. For certain types of problems, the network is an ideal interface to inject a little code to control and modify how the application behaves. In all such cases network-side scripting is a powerful mechanism through which we can architect more scalable, secure, and peformant infrastructures capable of handling high-volumes of requests while protecting server-side infrastructures from the sometimes adverse affects of sudden spikes in user concurrency. Programmable proxies, a la F5's BIG-IP Local Traffic Manager, that provide a scripting language such as iRules, are simultaneously client-side *and* server-side, with the best definition to describe their placement in architectures being *network-side* scripting.

Network-side scripting executes *in the network* rather than on the client or the server. It has a view of both the client and the server and the data being exchanged between them because of its unique placement in the communication channel. It can inspect and modify both client and server side data for myriad purposes including optimization, security, and to assist with implementing application-specific logic. For example, network-side scripting can react to server-focused data like HTTP responses, cookies, and session information while simultaneously taking into consideration client-side information - HTTP requests, cookies, submitted data, and even the network conditions currently being experienced by that specific client. Thus, network-side scripting allows you to implement application functionality *in the network* that can provide benefits like improved application scalability and performance; it can enable agility in both your network and application infrastructure; it can provide centralization of functionality in a service-oriented manner through reusable network-side scripts that can be applied as necessary to one, two, or all applications with minimal effort.

IV. CONCLUSIONS

The issue of vertical scalability is very important when considering the use of cloud computing because user is often charged based on compute resources used, much like the old mainframe model. If an application doesn't vertically scale well, it's going to increase the costs to run in the cloud because of the additional demand on compute resources required as demand increases. Thus, improving the vertical scalability of applications is important in achieving the benefits of a reduction in costs associated with cloud computing and virtualization. External solutions can improve overall performance, certainly, by optimizing protocols, reducing protocol and application overhead, and reducing bandwidth requirements, but it can't rewrite the application code. It is still the domain of the application developer whether the application is deployed inside the local data center or out there in the cloud. Application developer can use technologies like network-side scripting, query optimization, advanced application delivery controller and intelligent use of APIs to address the issue of vertical scalability up to quite an extent.

REFERENCES

[1] Joyent-Performance-and-Scale-in-Cloud-Computing- A White-Paper.
[2] Maged M.; Moreira, J.; Shiloach, D.; Wisniewski R.; Scale-up x Scale-out: A Case Study using Nutch/Lucene; IBM Thomas J. Watson Research Center; IEEE 1- 4244- 0910-1/07; 2007
[3] Lightweight Chip Multi-Threading (LCMT):Maximizing Fine-Grained Parallelism On-Chip Sheng Li, Member, Shannon Kuntz, Jay B. Brockman and Peter M. Kogge, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 22, NO. 7, JULY 2011.
[4] Web 2.0: Integration, APIs, and Scalability by Lori MacVittie.
[5] OpenSPARC T1 Microarchitecture Specification, technical report, Sun Microsystems, Inc., July 2005.
[6] Davis M.; Scaling up vs. Scaling Out; http://weblogs.java.net/blog/malcolmdavis/archive/2006/07/ sca le_up_vs_sca.html ; July 2006
[7] "Handbook of Cloud Computing" ; Borko Furht , Armando Escalante ; ISBN 978-1-4419-6523- Springer New York Dordrecht Heidelberg London.
[8] WSO2APIManager-v1.0.0-GettingStarted pdf.
[9] "What is an Application Delivery Controller (ADC) ?" JetNEXUX.

