Performance Engineering in Cloud

Muhammed Suhail T. S.
ITA Tata Consultancy Services
Infopark Kakkanad
Kochi

ABSTRACT

For any user centric application, performance is a critical factor. This paper deals with mobile/web application where backend is configured in either cloud / classic servers and how every component interacts and affects the overall system performance. Every component designed in the architecture of a system has its own importance. However, excessive use of any resource will have impact on the complete system. Performance Engineering is the study of these critical aspects and helps in fine-tuning the application under test. This paper will give a deeper insight on the infra level components as well as real time problems which are encountered in real life scenarios.

Keywords

AUT – Application Under test, API – Application Programming Interface, DB – Database, SOA – Service Oriented Architecture, AWS – Amazon Web Services, DBMS – Data Base Management System, ODBMS – Object Database Management System, IaaS – Infrastructure as a Service, PaaS – Platform as a Service, SaaS – Software as a Service, CDN – Content Delivery Network

1. INTRODUCTION

Mobile apps & web apps dominate the global online market nowadays. The potential of the same, if effectively utilized would be unprecedented. The industry growth in the sector is huge and the total number of active mobile apps in the app store by 2020 would be close to 5 million, in play store by 2020 would be around 6 million and mobile web slated for a huge number. The total revenue generated by these mobile/web apps are in billions. As per recent studies, it was found that if the response time of an app or website is more than 3 seconds the number of users using that app decreases dramatically. As the response time increases the number of users engaging with the app or site reduces exponentially. The bounce rate of such sites are around 60%. This indicates the importance of the performance of a web/mobile app.

Majority of the apps in our day-to-day life operates by consuming data, i.e, they work leveraging Internet. APIs are the basic method by which data transfers occurs in applications. Performance of any app has a direct relationship with the APIs they consume. The purpose of application programming interfaces is that it makes it easier for developers to use certain technologies in building applications. By abstracting the underlying implementation and only exposing objects or actions the developer needs, an API simplifies programming. APIs helps to get data, write data, update data, and delete data in a database. In addition, it performs calculations and logic to provide business functionalities to the end user. In performance engineering, the goal would be to enhance the operational response of APIs, which in turn would result in enhanced system performance.

For building a service-oriented architecture in an application there are mainly two possible ways. Either the traditional server based architecture or by using cloud based services. There are many advantages of using cloud over server architecture, the most important being the cloud vendor handling infra level aspects completely. It offers very high availability and security as compared to traditional server systems. Scalability is another factor that favors cloud over traditional approach. Some of the major cloud vendors are AWS, Microsoft Azure, Google cloud platform, Adobe, IBM Cloud etc. The cloud service providers offers a whole range of services in addition to hosting, including API services, DB services, Cache services, Message Queues etc.

The server-based deployment is also still widely used as not all applications are hosted on cloud. In server-based architecture, we always have a specific limit to the traffic that could be handled by the server. Also scaling up is comparatively difficult as compared with cloud. Infra level bottlenecks are higher in this architecture. The cost would always remain constant whether the vendor consumes the required amount of resources or not. Availability is another trade off with this system. However, applications having low amount or constant amount of traffic can be configured with the server-based systems effectively as the operational costs of cloud are comparatively higher in this case. Some of the famous hosting providers of servers are inmotion hosting, liquid web, single hop, codero, bluehost etc.

2. ARCHITECTURAL COMPONENT OF SYSTEMS

All dynamic applications web and mobile leverage application programming interfaces for their functioning. Only static / standalone apps works with resources already packaged in the executable. Any app having connectivity and data transfer follows the below given architecture.

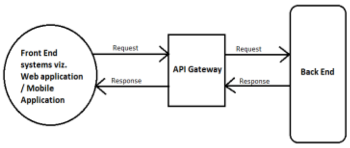

Figure 1. Basic dynamic application architecture.

In the figure, we can see that the front-end viz. Mobile/Web interfaces passes on requests to the API gateway. The API gateway on receiving a particular request validates the authenticity of the incoming request and after confirming validity passes it on to the backend. The backend system receives this request and processes it based on the business logic and returns the corresponding response back to the app. During the execution of business logic, all required infra level components are utilized in order to provide the appropriate response.

Performance is a factor that depends deeply on the backend part of the system under consideration. Although optimizations could be made in front-end the impact that it can create is minimal as compared to backend enhancements. Hence taking a deeper look onto the components that are acting in the backend is required before proceeding further. The below figure represents the basic back end components that an API interacts with.

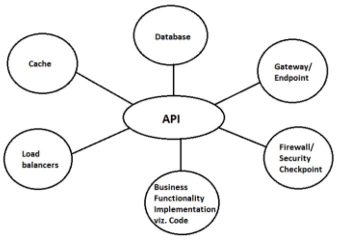
Database
Cache
Gateway/ Endpoint
API
Load balancers
Firewall/ Security Checkpoint
Business Functionality Implementation viz. Code

Figure 2. API Infra components.

Database

Database is an integral part of an application. All data related aspects are stored and retrieved via databases. A DBMS software is required for administration of databases. In order to define, create, update, and delete entries in a DB, DBMS systems are used. An example of DBMS software is the famous SQL. There are different database classifications based on the way in which they operate, rather than their data classification structure. They are:

1. Navigational Databases
2. Relational Databases
3. Object Databases
4. No SQL Databases

In the Navigational Model, it relies on the "manual" navigation of a linked data set, which is formed into a large network. Data retrieval occurs by use of a primary key, navigating relationships from one record to another or by scanning all the records in a sequential order.

Relational Databases are one of the widely used database model in the market now. It solved the complexities of the navigational model and introduced the ease of use with the table, row and column structures. It had a much-defined model. However, the performance of the DB degrades when huge amount of data is in store for processing. This led to the creation of the next model, i.e. The NoSQL databases.

NoSQL databases are often very fast, do not require fixed table schemas, avoid join operations by storing denormalized data, and are designed to scale horizontally. The most popular NoSQL systems include MongoDB, Couchbase, Riak, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra, and HBase.

Object Databases are a form of database in which information is represented in the form of objects. It is different from the table model of relational databases. The ODBMS comes coupled with the programming language and is more programmer friendly. It avoids the clear distinction between the data and the application layer.

Now, from the performance aspect, database is a very crucial component in the API performance. Every operation that includes the accessing of DB to write data during the API invocation would require higher amount of execution time as DB write operations are slow. Although, doing a single operation might not produce considerable impact, having simultaneous multiple operations would indeed result in lowering performance.

Next is the queries with joins, when large number of joins are used when retrieving data, then the time required for processing the query increases. There are certain steps to be used while writing joins. Some of them are:

1. Ensure to use EXPLAIN.
2. Every “leg” of join should use an index. Ideally, it should be joining on the primary key.
3. Find the appropriate storage engine to use, which maximizes the performance.

Simple read operations would never take higher response time as data would be retrieved faster. Also, ensure that the hosted systems have enough CPU and memory to handle the incoming requests.

Gateway/Endpoint

The gateway is the initial access point for an API request. All Authorization & Authentication occurs at the gateway level. It validates whether a request needs to be transferred for further processing or whether it is a malicious request. Security policies also are validated via the gateway. A gateway can provide functionalities like collecting analytics data and providing caching. A gateway often includes a transformation engine to orchestrate and modify the requests and responses on the fly. The responses are transferred back to the requestor via the gateway.

From the performance standpoint of an API, the gateway has an important role. Since the initial authentication, authorization and security/throttling validations are done at this level, it could become a possible performance bottleneck. This also would be visible when the system under test is loaded to its capacity full capacity or beyond that. As this is also a component in the infra level architecture of the API system a latency would be induced which could cause a performance glitch when system is on load.

Firewall/Security Checkpoint

Firewalls are the basic security mechanism to prevent the system from being compromised. There are ways to configure firewalls to check for different types of threats viz. Denial of Service attacks, Distributed Denial of service attacks, ransomware etc. In configuring these threat detection and avoidance calculations at the firewall level can result in a performance glitch. Suppose in a direct access situation the request takes 100 mille seconds to execute and on adding the security validations the response got increased by 10 mille seconds. Here the impact for a single request is negligible. However once the number of requests received by the system increases, i.e. if the number of simultaneous requests increases at the firewall validation this results in a bottleneck which could result in increase of response time of the system by a maximum of 100% to 200%. So while configuring the firewall rules ensure to avoid high complex threat analysis at the infra level.

Load Balancers

Load balancers help to improve the distribution of workloads across resources such as servers, clusters etc. This in turn results in enhanced performance of the system and ensures 100% availability. It also reduces the chances of system failures as distributing load across reduces over pressuring of a single server. There are different possible ways to configure on how the distribution should occur. Simple algorithms include random choice, round robin, or least connections. Advanced algorithms operate taking into consideration factors like servers load, up time and down time, current response of the server, active connections, geo location etc. Nevertheless, the load balancers are another component that is added to the overall network architecture. Hence performance factor has dependency on this component as well.

A possible problem that might occur with load-balanced system is routing of a single user's requests to different servers. This will affect the system performance. One possible way to solve is either by using shared database resources or by using in memory database. Another option is to route switch servers based on session identity. I.e. For a single session, all requests should be routed to one server and for another session, it should be routed to the next server. However, both these approaches require some validation check in the load balancer, which in turn results in inducing a latency to the incoming request and its response.

Cache

Cache is a hardware or software component, which stores data to be used for future computations. The data stored in cache is the result of an earlier calculation or the duplicate of a data stored elsewhere. Two common terms associated with cache is cache hit & cache miss. If a particular data is found in the cache then it is called a cache hit. Cache miss occurs when the requested data is not found in the cache and have to be fetched from the lower memory hierarchy. Cache hits are extremely faster than other memory compute operations. This results in improved performance when cache hits are more.

Cache is a component in the architecture, which if used effectively can considerably enhance the performance of systems. The only limitation is that caches are not as big other memory units. There is tradeoff between speed and size. An L2 cache is generally close to 100 times faster than the RAM. As a huge gap exists between the processing speed development and the memory development, the cache approach has a significant traction. Similarly, in API architecture as well instead of always relying on the DB to get the required data, the frequently accessed data could be stored in a cache and accessed accordingly. Although this increases the complexity in the system, it helps to save valuable time and enhances user experience.

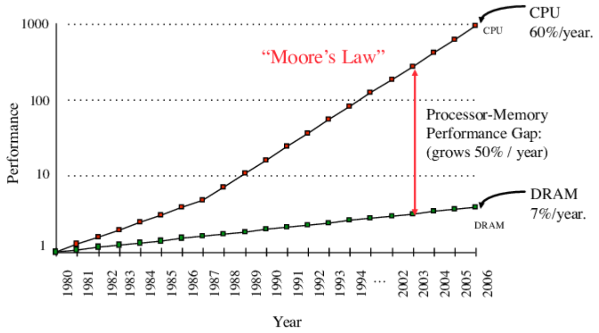

Fig 3. Memory processor gap.

However, it is never advised to put all your data into the cache and adopt a cache only design as the cache is very much expensive to use compared to the ordinary storage devices. By effectively leveraging the cache the overall system performance can be enhanced.

Business Functionality viz. Code

Business functionality is the most important part of the application. It is responsible for the end to end functioning of the application and where complex operations involving data exists. Performance of the code is of most importance. Before jumping into any code level optimization first part of performance engineering is to tune the infra level aspects. Once infra is optimized to the maximum, and even then, if the performance is lagging then a closer look into the code is required.

The business code optimization can happen at different levels. They are.

1. Design Level
2. Algorithms & Data structures
3. Source code level
4. Build Level
5. Compile Level
6. Assembly level
7. Run time

The Design level optimization occurs at the highest level. This is considered during the design phase of the system. Based on the high-level architecture design the system is developed and deployed. So any changes in this level needs to be considered initially as a rework on the design requirements and will have considerable impact on the cost and timeline of the project if implemented at a later stage. The optimization at the higher hierarchies, like design, would yield the highest performance enhancements. As the levels decreases the percentage increase in performance decreases considerably. There exists an inverse relationship between higher-level optimization and performance enhancement percentage. Once high performance enhancements are made with the higher hierarchies, the next level increase in performance becomes increasingly difficult.

3. CLOUD VS SERVER ARCHITECTURE

Application backend was earlier hosted on secured standalone servers. However, due to the cost of setting up these systems and maintaining the same was considerably higher, distributed hosting services aka cloud services gained traction. Both have their advantages and tradeoffs.

Fig 4. Cloud vs Dedicated.

Considering the appropriate system based on the requirement is an enterprise call, which should be made. The features of cloud and standalone server architecture is explained below.

Cloud

Cloud servers are created using virtualization software that divides physical servers to multiple virtual servers. Some of its features are.

1. High performance.
2. Easy to scale.
3. Built in redundancy.
4. Hardware configurable.
5. Future proofed.

There are certain tradeoffs for the cloud server.

1. Physically not isolated.
2. For high usage costs are high.

Dedicated Servers

Often known as a bare metal server, a dedicated server is a single physical server that is not shared or used by anyone else. Its features are.

1. High performance.
2. Hardware configurable.
3. Physically isolated.
4. Setup costs are constant.

Some disadvantages of dedicated systems are.

1. Low scalability or scaling requires high effort cost and down time.
2. No built in redundancy.
3. It is not future proofed.

4. PERFORMANCE ENGINEERING WITH AWS

Amazon Web Services (AWS) is a subsidiary of Amazon that provides on demand cloud computing resources to the consumers. It includes IaaS, PaaS and SaaS services. The consumer is billed based on the amount of resources consumed. It is a pay as you use platform. As of 2017, AWS accounts for 34% of the whole IaaS and PaaS services offered over cloud, which is higher than that of the next 3 competitors combined viz. Microsoft Azure, Google Cloud and IBM. Hence, the focus of performance engineering on AWS does have a significant traction.

A closer view of how an API works with AWS will provide a way to understand the architecture of the system and methods to improve the performance aspects of the same. The following figure represents how a basic API request performs with the AWS cloud backend.

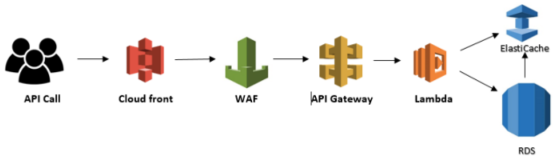

Fig 5. AWS API basic architecture with security components.

In the above figure, the API call originates from the user end and first hits the cloud front in the AWS architecture. Cloud Front is a global Content Delivery Network (CDN) provided by AWS. It accelerates delivery of websites, APIs, video content, or other web assets. It can be integrated with other AWS products to provide businesses an easy way to accelerate content to end users with no minimum usage commitments. It uses a global network of edge locations to deliver website, including dynamic, static, streaming and interactive content. Requests for the content are automatically routed to the nearest edge location. Amazon CloudFront is optimized to work with other services in AWS and it also works seamlessly with 3rd party servers not associated with Amazon.

In the performance space, the Cloud Front service helps to improve the overall system performance by routing the incoming request to the nearby edge location whereby the overall content delivery is made faster.

WAF is the Web Application Firewall provided by AWS. It is a web application firewall service that helps protect the websites and web apps that you deliver with Amazon CloudFront and ELB Application Load Balancers. It also helps to control access to the content. Custom rules can be added to mitigate all threats happening to the system. Basic security offers protection against cross site scripting, DDoS attack, SQL injection etc. It can be used to block IP addresses, filter based on the headers, identify and block requests originating from a particular geo location and also based on string and regex occurring in the headers of the request. However, WAF is a component that requires certain amount of calculation to be executed. This in turn adds latency to the incoming request as the complexity of the custom rule goes beyond a specific level, the flooding of incoming request can have adverse effects on the response time. Configuring WAF optimally is an important step in AWS performance engineering.

The Amazon API Gateway acts as a front door for all incoming API calls. It can handle hundreds of thousands of concurrent API calls simultaneously providing authorization and authentication, traffic management, monitoring and API version control. The gateway is responsible for routing the incoming requests to the appropriate functions, which contain the business logic to be executed. Since API gateway is a single point of handling immense number of incoming requests, it can become a choke point as the application scales. So while doing performance analysis on components in AWS, API gateway configuration has a critical role.

Next is Lambda. Lambda is where the entire code execution takes place. All business logic is written inside the lambda service using java, python, node, C #, or Go. Lambda is a compute service of AWS. Once the function is created in lambda, it takes care of provisioning and managing the servers to run the code upon invocation. Ease of use is the most important feature of the lambda service. All code level optimization needs to be handled at this component. This would be the last place to search for performance improvement once all other infra level components have been fine-tuned. Lambda functions are easy to monitor using the cloud watch metrics. It can provide the number of invocations, error count, invocation duration, concurrent execution etc.

Amazon RDS is the relational data base service provided by AWS. It is easy to setup, operate and scale as the DB is in cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks. There are six familiar DB engines to choose in Amazon RDS viz. Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle and Microsoft SQL Server. Some of the advantages of the Amazon RDS are.

1. Fast and easy to administer.
2. Highly Scalable.
3. Available and Durable.
4. Secure.
5. Inexpensive.

Database operations are always time-consuming tasks. Hence, there is a huge impact for the same in the defining the performance of the application. Some APIs perform DB write operations and some DB read. DB write would always have higher response times as compared to DB reads. However, if complex business operations were involved in a service call of DB read, this would again result in increased response time. So simplifying the business logic to be performed with ease would be part of the re-engineering process involved in the project/API development. The basic requirement is to bring the API responses around the usual business standards. A recent study indicated that the likeliness of users using an application that have a response time > 3 seconds is less than 10%. Hence bringing the API response under 500 mille seconds is a critical aspect to the success of a web/mobile application. Operations like sign up would always be a time consuming operation if performed with hundreds of submits simultaneously as it would always be a DB intensive task. By reducing the direct DB calls, it is possible to an extent to enhance the performance of the APIs. Also by using appropriate cache designs, faster response time could be achieved.

Amazon ElastiCache is a service, which helps to easily deploy in memory cache in cloud. Unlike the normal disk based DB, the in memory cache is extremely faster. This in turn helps in improving web application performance as information retrieval occurs from fast, managed, in-memory cache. There are mainly two types cache supported by ElastiCache. They are.

1. Redis
2. Memcached

Redis is a fast, open source, in-memory data store and cache. ElastiCache is fully managed, scalable and secure. Hence, it is best suited to power high performance mobile, web, game app, ad-tech & IoT.

ElastiCache is a protocol compliant with Memcached so that the existing systems using memcache can work with elasticache memcache seamlessly.

Appropriate cache design is a brilliant method to enhance the system/API performance. However, the cost of cache is very higher as compared to the ordinary disk storage. Hence, putting all data in cache to increase performance is never a sustainable approach. Striking the appropriate balance between DB disk operation and cache execution can result in efficient performance results.

5. BASIC PERFORMANCE BOTTLENECKS CHECKS

Until now, a view of how basic cloud backend & server backends work with respect to APIs is detailed. However, there are many basic steps to be followed while doing performance bottleneck analysis.

After working with few projects for their performance bottleneck identification and enhancements, there are some checklists/guidelines, which could be applied across any performance enhancement projects. They include.

- Initial analysis should be done with low transaction limits on the system to understand the basic performance.
- Test post and get requests separately with load to understand the response time for DB intensive and business logic calculations.
- Analyze and monitor DB usage when performing testing and capture the CPU, Memory and I/O graphs.
- Monitor the hosted backend system performance metrics viz. CPU and Memory consumption during test.
- Monitor the gateway to understand the number concurrent requests being received at the backend and any errors.
- Monitor the cache metrics, especially cache hits and cache misses to understand how many requests are routed to the DB for calculation and what the current cache efficiency is.
- Monitor the network speed when performing tests as slowness in network speed can result in increased response times.
- Tools like New Relic could be used to monitor the code/query calls being made, that consumes the most time for execution. Analyze whether any optimization could be done on long intensive queries/operations.
- Another basic thing which could be missed is the log writing processes done in the code side. For example, during development there would code snippets in the source code for writing the logs which could result in huge performance lags when on load. Always ensure to reduce the amount of logging before pushing code to performance.

In AWS cloud, ensure to test with and without WAF rules enabled. Example, if too many WAF rules are added, it can result in latency at the WAF on flooding of requests.

Once all these checks are done, an analysis of the system could be made which would contain details on the identified infra level and basic code level issues. After rectifying the same re-test to understand what is the performance enhancement achieved. If the desired result is achieved, then we can increase the loads to test the system and see the response graphs. If all basic checks have been done and still there is performance lag in the system then serious code level analysis needs to be done to improve the performance.

6. CONCLUSION

Performance engineering is a niche field with lot of opportunities. Especially as currently in the mobile/web applications space, people always want speed rather than too much functionality. There are many examples where despite being a brilliant product, due to the low ease of use and response the application got phased out of the market. The market success of WhatsApp is the story of a brilliant simple and super-fast messaging app, which outgrew all its competitors with in much time and now is the industry leader in the messaging space. Even before the introduction of WhatsApp, there were plenty of messaging application.

However, this new one had an edge over the rest. Thanks to the performance of the app at scale that the users loved it.

Further analysis in code level could be done which indeed is a matter for another paper. Considering the importance of performance engineering, especially with cloud back ends, there is a long way to go.

7. ACKNOWLEDGEMENTS

Thanks to all the people who had given me opportunities to work with different projects and helped to understand AWS cloud as well as performance engineering concepts in deep. Thanks to the team who had developed the format of this paper.

8. REFERENCES

[1] Murray Woodside, Greg Franks, Dorina C. Petriu *The Future of Software Performance Engineering*, Carleton University, Ottawa, Canada.{cmw | greg | petriu}@sce.carleton.ca.

[2] Robert S. Hanmer John P. Letourneau, *A best practice for performance engineering.*

[3] Erwin Laure, Heinz Stockinger and Kurt Stockinger, *Performance Engineering in Data Grids,* CERN, European Organization for Nuclear Research, Geneva, Switzerland.

[4] *Performance Engineering and Testing The Challenges on Mobile Platforms* Pavlo Bazilinskyy University of St Andrews School of Computer Science pb52@st-andrews.ac.uk, Markus Brunner University of St Andrews School of Computer Science mb246@st-andrews.ac.uk

[5] *Cloud Based Performance Testing,* FungayiDonewell Mukoko, , Abhaya2, Kaushal Kumar 3, Ankush Jain4 .

[6] *A Brief Survey on Web Application Performance Testing Tools Literature Review* Isha Arora M.Tech Scholar, Department of Computer Science and Engineering , PIET, PANIPAT, INDIA Vikram Bali Department of Computer Science and Engineering, PIET, PANIPAT, INDIA.

[7] *Performance Test Workload Modeling* by agileload.com

[8] *Performance Testing of Software Systems*, Filippos I. Vokolos AT&T Labs - Applied Tech. filip@att.com Elaine J. Weyuker AT&T Labs - Research weyuker@research.att.com

[9] *Performance Testing: A Comparative Study and Analysis of Web Service Testing Tools*, Shikha Dhiman Pratibha Sharma Research Scholar Asstt Professor Department of Computer Science Department of Computer Science Career Point University Hamirpur, India Career Point University Hamirpur, India.

[10] *A Survey on Performance Testing Approaches of Web Application and Importance of WAN Simulation in Performance Testing*, Dr. Ramakanth Kumar P. Head of Department, Department of ISE, RVCE Bangalore, INDIA Email-id: ramakanthkp@rvce.edu.in Kalpan Bhargav M.Tech(SE),4 th semester , Department of ISE, RVCE Bangalore, INDIA E-mail-id: kalpan.bhargav@gmail.com

[11] Wikipedia, the free encyclopedia, https://en.wikipedia.org/

[12] https://www.clook.net/

[13] https://d1.awsstatic.com/whitepapers/aws-overview.pdf

[14] https://aws.amazon.com/lambda/?p=tile https://en.wikipedia.org/wiki/Software_as_a_service

[15] https://en.wikipedia.org/wiki/Amazon_Web_Services