Cost to Serve of Large Scale Online Systems

Andrés Paz Sampedro, Shantanu Srivastava
Microsoft Corporation
Redmond, USA

Abstract

Online systems typically provide a variety of different service offerings. For example, an internet search engine provides the service of searching web pages, videos, images, news, maps etc. Each offering can utilize different physical and/or virtual systems, networks, data centers, and so forth. Thus, a request to search videos may use some, but not all, of the resources used by a request to search images. Also, each video query will not use the same number of resources due to caching and ranking algorithms. Due to this it can become extremely difficult to ascertain the Cost to Serve (CTS) of an offering. CTS is required to understand cost of the product offerings for request per second (RPS), create rate card for partner deals, target efficiency areas and decide ROI of services. In this paper, we define the CTS methodology for Bing. In this methodology, CTS is calculated by determining operational RPS of each platform in Bing and the average number of times a type of request touches those platforms. Prior to this work, CTS was calculated by manually tagging capacity used by each offering and number of observed queries. The methodology described here can be applied to any other large scale online distributed system.

1. Introduction

Bing is a global online search engine. It is available worldwide in many languages and localized for many countries. It is currently the second most popular search engine with 21.4% market share in US for desktop searches [1]. The infrastructure that supports Bing is large and diverse, both in terms of the type of resources used and in geographic locations. It has a very complex architecture which includes serving platforms that directly serve end user queries and non-serving platforms that generates or processes data to enrich or improve query results. Additionally, there is research and development (R&D) platforms which are not involved in live traffic e.g. engineering systems used to develop, build, deploy and test.

With such a complex architecture, it is challenging to report how much it costs to generate the response for a user request. Not only that, but there are different categories of requests, which depending on a variety of factors like the source of the query (end-users, 3rd parties, automated bots), the location, or the type of information requested (web, news, images, etc) will have different costs.

When available, the Cost to Serve (CTS) of a request can be an important tool used in many ways, for example, platform owners can use it to compare efficiencies and setup goals; business development and marketing teams can use a regional CTS to determine how much to charge for partner deals; the leadership team can use it to analyze the cost of a product offering and determine its ROI.

To define a methodology to calculate CTS we looked at the airline model [2] where capacity is calculated per number of seats available be it filled or empty. We replicated this model by looking at total available capacity in the system rather than the current load of requests. In its simplest form, CTS can be calculated by:

$$CTS = \frac{Cost\ (\$)}{Volume\ (Requests)}$$

For example, the CTS of a simple application comprised of a single service that serves only one type of request can be calculated simply by collecting all the infrastructure costs of the service, and divide it by the number of requests it can handle.

More broadly, if an application is comprised of multiple services, we can calculate the CTS of each individual service and report the CTS of a request as the sum of the CTS of the services it used.

In our case, Bing is comprised of hundreds of services (we call these platforms) and hundreds of product offerings which trigger millions of requests per day. Not only that, but we want to be able to report the CTS across multiple dimensions like region, device,

partner, etc., so we had to find mechanisms to automatically detect the cost and volume of Bing's platforms; identify what platforms a given request uses and classify the incoming requests into their corresponding product offering and dimensions.

2. Cost: Bing Cloud Catalog

Our first step was to have a single and accurate inventory of Bing's resources and cost associated with them. We called this the Bing Cloud Catalog (BCC)

Each resource type is managed by a different management system which is capable to report how many resources each platform owns; we wrote tools to capture the inventory from each of these systems and collect it into a single repository. The tools also took care of normalizing the platform names across the management systems so we could have a unified reporting.

Based on different heuristics, like the name and number of resources, their utilization patterns, and other signals from the management systems, a machine learning system is able to categorize the resources into different dimensions like usage (serving, non-serving or R&D) and variable capacity % (i.e. how much capacity will scale with traffic). Teams verify the categorization given by the system, which is then fed back to improve future categorizations.

Finally, we worked with the finance team to come up with actual cost of different resources per region by distributing cost of capacity and including warranty, hosting and lease cost.

After completing BCC, we were ready to start working on calculating the platform's volume.

2.1. Volume: Operational Requests Per Second

Determining the volume per platform or the denominator of the CTS equation is critical. We are interested in finding the number of requests per second that a platform can handle under normal operations. We call this the Operational RPS, or ORPS.

In the past, each team had to run capacity tests that would stress their serving platform (i.e. a platform that serves request from users) to the breaking point to identify the max RPS it could handle. Setting up and maintaining these tests is typically hard and expensive and it would not scale for us. Instead, we calculate the breaking point of a platform simply by defining its Service Level Agreements (SLA) and using the load, latency and utilization counters from its production servers.

As part of the service definition, each platform has a well-defined SLA on what's the maximum amount of time a response to a request must take. Typically, this is enforced via timeouts on the client.

We created a utilization reporting pipeline where we captured load, latency and utilization data and group it by service. In this context, the load is typically measured in the number of requests being serviced in a given period of time, such as RPS. Latency is the difference between the time when the request is received and when the request is fulfilled. Utilization is a measure of how busy the system is, such as CPU utilization, utilization of other system resources such as I/O, RAM, storage, and/or so forth.

We found that it is very common to have a direct correlation between load and utilization: the more requests a single server gets the more resource it will utilize. Similarly, there is typically a direct correlation between latency and utilization: the more resources a server utilizes the more time a request takes to complete. This is shown in Figure 1.

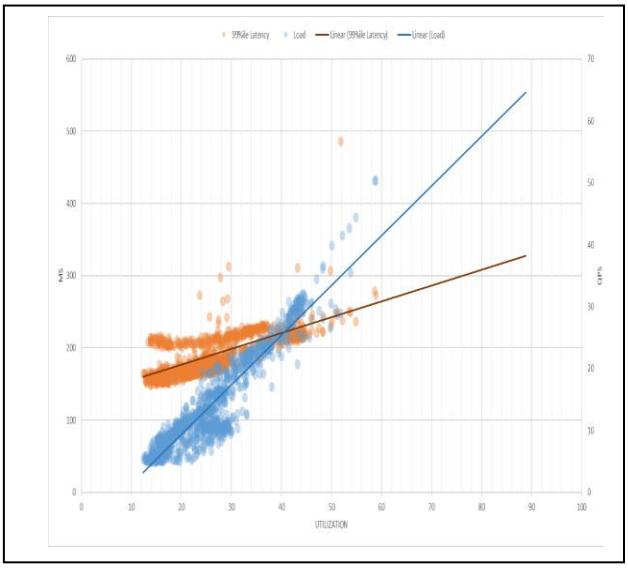

Figure 1. Load and latency vs utilization

Even more, because all servers of the same service and on the same datacenter have the same SKU (i.e. physical characteristics), and because we randomly and evenly distribute the load across all the server of the same service, the value of these correlations are the same across all servers of the same service in the same datacenter.

Using these correlations, we can calculate the load a single server can handle such that both are true:

a. latency is below or equal the platform's service level agreement (SLA)
b. the utilization is below a specific max utilization target (e.g. 90%)

Specifically, using the correlation between latency and utilization, we can predict the server's utilization level that would break the latency SLA. We either take this value or the max utilization target to now predict,

using the correlation between load and utilization, the corresponding load for this utilization level.

This value predicts the load at which a server will start breaking its latency SLA. We call this the server's MaxRPS.

For example, assuming the server in Figure 1 has a latency SLA of 300ms and a 90% max utilization target, we can predict using a linear approximation of its load vs utilization graph, that it will reach its latency SLA of 300ms at 76% utilization level; at this utilization level, and again using a linear approximation of utilization vs load, we can predict a load of 55 RPS. Therefore, the server's MaxRPS = 55.

Once the max load that a server of a given service can handle (MaxRPS) is known, and because most of our services can scale linearly simply by adding more machines, we define the Ceiling RPS or CRPS of a platform as:

Equation 2: Platform's CRPS

$$\text{CRPS} = MaxRPS * \#machines$$

CRPS represents the max RPS a platform can handle on a datacenter without breaking its SLA.

In Bing, all our platforms need to leave enough buffer to be able to handle traffic in case of other datacenter outages. This buffer is represented as the Business Continuity Plan RPS (BCPRPS). The BCPRPS is different per platform and per datacenter, and it's based on the peak traffic observed by the platform across the different datacenters it is deployed in the last 90 days.

For CTS we want to use the volume of requests that is *available* for end users, therefore we used the ORPS which is calculated by:

Equation 3: Platform's ORPS

$$\text{ORPS} = CRPS - BCPRPS$$

2.2. Volume of Non-Serving Platforms

In Bing, we use a lot of resources on non-serving platforms, for example building an index of all content on the internet or processing logs to improve our algorithms, therefore we also wanted to calculate the CTS for non-serving platforms such that we could include them in the calculation of a Product Offering CTS. For non-serving platforms, though, we can't define an ORPS as by definition they serve no traffic.

For non-serving platforms we decided to use observed Peak RPS in the last 90 days as their CTS' volume, as this is the volume of queries any non-serving platform eventually need to handle.

2.3. Cost Per Request

Once we have all platforms' CTS, we calculate the cost of a single request as the sum of the cost of the serving platforms by the number of times the platform was used:

Equation 4: CTS of individual request

$$R = \sum_{i=1}^{N} M_i * CTS_i$$

Where:

- R is the CTS for a request;
- M_i is the number of times the i^{th} platform was used in filling the request;
- CTS_i is the CTS for the i^{th} platform
- N is the number of platforms.

For non-serving platforms, we used 1 as the number of times the platform was used for all requests. For serving platforms, we leveraged Bing's server logs to calculate how many times a serving platform was hit. More broadly, using Bing server logs for each request we can identify:

- Resources used and/or accessed by the request (i.e., which servers);
- The product offering associated with the request (i.e., the page name for the offering);
- Information about the entity that sent the request such as any combination of: an identifier associated with the requesting entity; the region where the request originated; the language of the request (i.e., English, Chinese, etc.); other entity information; and
- Other information that can be used to break down the backend resource cost calculation such as which data center the request was routed to, etc.

Using BCC, it was simple to map individual servers to a platform, this combined with a platform's CTS allowed us to calculate the cost of an individual request.

2.4. Product Offerings CTS

Bing is the brand for our facing consumer product, but internally it has many product offerings which can be used or syndicated individually.

As explained, Bing's server logs already categorized queries based on the product offering and other dimensions. We define a product-offering CTS as the average of the CTS of all the queries for that product offering:

Equation 5: CTS of a Product Offering

$$PO_{cts} = \text{Avg(R)}$$

Similarly, if we want to compute the CTS across other dimensions like region, devices, partners, we calculate its CTS as the average of the queries' CTS that are part of such dimension.

2.5. Statistical Sampling

Bing processes billions of requests per day. It is neither practical nor cost effective for us to calculate the cost of each one of them. Instead, throughout the day we are constantly collecting a random sample of requests and use statistical methods to calculate the actual cost of the entire population.

3. Conclusion

CTS can be a powerful metric used to drive efficiencies and business priorities. Calculating it at scale on a large online application can be challenging considering the amount of services and data. We believe the biggest breakthroughs of our methodology are to calculate CTS not based on peak traffic, but on ORPS, which makes it more reliable and consistent, and to calculate ORPS based on server logs and counters from normal production traffic without the need of running capacity tests for each platform.

4. Acknowledgment

The overall success of this project wouldn't have been possible without the help and feedback of many at Microsoft, including Etta Mends, Atin Kothari, Andre Briggs, Bob Wyler, Kyle Peltonen, Ramu Movva, Mark Aggar, Alisson Sol and Maria Alvarez.

5. References

[1] comScore, "comScore Releases February 2016 U.S. Desktop Search Engine Rankings". [Online]. Available: https://www.comscore.com/Insights/Rankings/comScore-Releases-February-2016-US-Desktop SearchEngine-Rankings [Accessed: 01-Jun-2016].

[2] Wikipedia, "Available seat miles". [Online]. Available: https://en.wikipedia.org/wiki/Available_seat_miles [Accessed: 10-Dec-2015].

