CDN Performance Management: bridging the gap between business and technology

www.adolforosas.com adolfomaria.rosasgomez@telefonica.com

Madrid, May 2014

1. Introduction:

In the CDN industry and ecosystem there are many pieces, technologies, companies and roles. One that is of key importance is 'Performance Management'.

After all, CDNs exist as businesses for only one reason: the 'standard' worldwide network (Internet) does not perform well enough in content distribution. All of us dream of a day in which the Internet will handle seamlessly any type of content without interruption, degradation or excessive delay. OK, probably pure CDNs do not dream of that day, as their businesses will change dramatically. But they will adapt.

So CDNs are all about improving performance of the underlying network, and thus it would make sense that anyone running a CDN takes Performance Management *very* seriously.

You will be surprised to know that this is not always the case. It is amazing how many misinterpretations of 'performance' are out there and very specially in the CDN ecosystem. It is very common to find companies offering 'performance data' that in fact do not reveal anything about the performance of a CDN, others offer data that are a collection of 'insights about performance' but cannot connect these insights with any actions that the CDN designers and admins could possibly do, so you can only get anxious about any problems that you discover… some others miss completely the point looking at performance figures not related to content distribution… Most of the times the problem is that some people do not know the right definition of 'performance' for a given system, or which is the right information about performance they should collect in their system or how to handle that information to their advantage.

2. Do people know what 'Performance' is?

You are probably thinking now: 'oh man that is obvious. I've known that for years. Please do not offend me…' Well. I not so long ago did an experiment. I asked a new technology support team supposed to operate a new CDN: what data would you offer to me about the performance of my CDN? They, after a moderately long time, responded with a template from a well-known industry tool, a commercial SW package from a big technology vendor. It was a standard template based on instrumentation agents for standard OSes (windows, Linux…). The template was offering, for ALL machines in the CDN the following information: %CPU occupation, %RAM occupation, %HD occupation. That was all. Every machine was reported in the same way: streamers, web caches, backend servers, routers…

I went back to the team and said: "… hey guys, I've got a problem, probably I did not make myself clear. I want to know as much as possible about the performance of my CDN. What can you offer me?" They started to look suspiciously at me (I started to think suspiciously of them…). They repeated their answer. It was then clear to me I was in front of a 'standard IT/OSS operations team'. They had received the blueprints and manuals of the new CDN (by the way, not a commercial one, so many new things were inside, quite

non-standard for any datacenter except for a CDN datacenter and thus out of standard OSS experience) and they had addressed them, in good faith, as if the CDN were a collection of computing machines in need of processor, RAM, HD, network and power. No less .No more.

It took tremendous effort in terms of time, money and human resources to close the existing gap about 'performance management' in that CDN. But in doing that effort many rewards were received: many SW bugs were found that limited the expected performance, some HW pieces were found working out of specs, some architectural principles had been violated in field deployments, some parts had been badly deployed…

It turned out that despite the existence of many Quality Assurance teams, many test environments and many controls in manufacturing, SW development and deployment, there was no coordination between teams and no one in management was concerned enough of end to end performance of the whole system.

3. Performance: an every day's concept.

Today 'performance' has transformed into a buzzword of the technical world but it is good to remember its everyday language meaning. To perform is to 'act', to do something. Your 'performance' can either be your action itself or, if taken as a reference to continued action, the outstanding properties of your continued action. In brief your performance is an analysis of '*how you do what you do*': how accurate, how costly, how fast, how reliable, how regular, how efficient…

From the logical-philosophical point of view a 'complete' description of even moderately-complex systems performance could be close to impossible. Some properties of your performance will be relevant for some analysts of your action and some other properties will be relevant for others. We will concentrate in a practical view of performance that helps monetize a technological business, in this case a CDN.

4. Performance Management

Performance Management is all about knowing what is important to you from among your 'behaviors' , retrieving all possible information about those relevant behaviors, connecting it to your action planning so you can: 1- avoid situations that you do not want and 2- improve your behaviors through feedback.

You have to think of what is important to know about your CDN as a business… and know even more clearly what is not important, so you do not waste your time and money on it.

Businesses are used to handle KPIs: Key Performance Indicators. They work hard to find the key properties of the business behavior, then look at them (collect series of values) and react according to what they see. Typical KPIs in businesses may be: cost, revenue and derivatives: benefit, EBITDA, efficiencies… Unfortunately CDN machines and SW do not come out of the box with an interface to query them about the whole CDN costs, or the revenue, benefits, etc… Even the smartest off-the-shelf SW suite designed for performance analysis can retrieve just some part of the information and it does not have any clue on which is the influence of that part in the behavior of your very specific business/CDN and thus it does not have the KPIs that you want to build. It is you, the business owner, the CDN owner, the CDN designer at large you design the strategy of the CDN business as well as the architecture), who needs to build KPIs from simpler

pieces of information picked up from here and there. Of course there are tools, including SW suites, that correctly used can help you a lot.

KPIs that make sense are directly related to the business. It does not make sense to monitor behaviours just for the sake of 'knowledge'. You must have a clear purpose. Think of the purpose of your CDN. You as a CDN designer/owner are selling an IMPROVED delivery service over internet, so your customers expect your CDN to behave BETTER than not having a CDN at all (of course). Your customers are concerned with: time, accuracy, reliability and confidentiality of the delivery. You must also be concerned about all these properties of your CDN and you must gather the best possible information about how are you doing in these departments. You want to have KPIs defined about: latency (first byte time, first frame time, DNS resolution time, re-buffering time, re-buffering ratio, re-buffering events per period,…) either average or instant, usable throughput :'goodput' measured in many places, traffic degradation (bytes lost, frames lost, bytes corrupted, artifacts shown, …) measured in many places … and probably many other KPIs that allow you to know which is the REAL value of your CDN for your customers (this is good to help you in pricing your service), KPIs that help you propose SLOs and SLAs or accept the SLAs your customers want to have, KPIs that let you anticipate upcoming facts with enough time in advance (for instance reaching top capacity), KPIs that help you learn about your business dynamics and trends so you can modify your products and/or create new ones, KPIs that let you know which is your operational efficiency ratio (how many dollars does it cost you to generate any new dollar), KPIs that let you discover that you should do some things differently, KPIs that can be shown to your stakeholders to make them confident and proud of your CDN, KPIs that let you compare to your competitors…

5. What performance of a CDN is not.

Performance of a complex information system is NOT about 'just collecting' tons of data about everything. Analyzing your CDN infrastructure as if it were the CERN's Large Hadron Collider does not make sense. For sure you can get tons of data about the behavior of even the simplest computer system. And it would be easy, you just buy a 'performance analysis package', check all monitoring options and 'voilà' you will get more data than you could cope with in ten lifetimes. The lesson is: think first of what you want to discover, and then look for related data. Don't let package vendors tell you what is important for your business.

Any serious performance management approach is a purpose-specific task and it starts from the highest level of your business concepts and goes all the way down to the level of detail that you can afford. You should stop at the point in which you are incurring in disproportionate costs to uncover your behavior…. If the task of analyzing your behavior were for free it would virtually have no limits. You could benefit in so many ways of knowing intricate data about your behavior that the performance management task could be easily more complex than running your own business. (Big Data fans painfully start to notice this …)

Of course many of the KPIs that are legitimate in a CDN (some examples were given in the above paragraph), like time-measures: first byte time, first frame time, etc… are in some way related to computing variables. These 'time-measures' depend on the available RAM, the available cycles on a core on a processor, the available HD throughput, the available network throughput… of some or all the machines in the CDN at the same time. The dependence of business-valuable KPIs (examples above), which usually are end-to-end system-wide measures, to computing variables measured in one or more machines is so complex that it is completely useless to try to establish that relationship analytically. And thus it is also completely useless to present these directly-measured computing properties as KPIs.

To say it in lay words: any graph of your caches %CPU occupation (for instance) will give you a very poor indication of what is going on in your CDN business. Of course knowing that is better than nothing. If you look at such a graph produced let's say every 10 seconds you get a huge amount of 'data' but you only get very little ' information'. In this very specific example, you for sure realize that it is not the same thing to be running at 1% CPU than to be running at 99% CPU. At 1% you may think you are not making much business, though it could be perfectly normal and respectable, and at 99% you may be perfectly OK or maybe you are down due to overload, it depends on your CDN design and many other things. The key point is that %CPU occupation (to name just one variable) is NOT a KPI for you, it carries very little information. It is also information very difficult to act upon. This is also the case with RAM occupation, HD occupation, link occupation, etc…. All these observations carry very little value by themselves. To be useful as KPIs their contribution to at least one business KPI should be established clearly…and in that case what is valuable is the interpretation of the resulting KPI. It is a waste of resources to retrieve process and represent the wrong data.

6. Characterization versus Modelling

As I have proposed it is useless to try to derive the true analytical relationship between observable values of computing properties and 'useful business KPIs'. This statement is too general to not be immediately challenged by bold engineers and statisticians coming from the technical field. OK. Let me then say that at least I personally find that to derive analytical relationships between these observable properties and 'end-to-end system-wide properties' is impractical. These functions most of the times do not exist. Sometimes you can only create a multidimensional 'mapping' (new word for 'function' popular today among mathematicians) made of disjoint patches in all the independent variables domains. Essentially you have to pick a finite number N of independent variables (observable computing properties), and try to 'map' an N-vector of values to the value of a system-wide property 'S' (can be another vector or a scalar), so there is a mapping from vector domain [N] to scalar domain S or vector domain [S] .

To start with, you cannot be sure to have chosen all the relevant independent variables in your system, and you cannot be sure they are really mutually-independent. To continue there are many physical complex relationships that link these variables and many times you just have models provided by manufactures and third parties, usually these models are limited and, worse than that, you do not have enough documentation on the theoretical applicability limitations of these models.

Building this analytical mapping is a 'Modelling' process in which we substitute our system (our CDN) by a 'Model' with N inputs and M outputs and we derive the analytical relationship between input values and output values. Modelling seems at least 'impractical' if not impossible. So, which is the alternative?

The alternative is 'Characterization'. It is an old, very practical tool used in engineering. If your system shows an exceedingly complex relationship between inputs and outputs you can at least try to figure out a set of your most relevant inputs, put your system in isolation from any other influence and observe output values evolve while you make your system traverse through all possible input values. You draw the curves stimulus-response. Stimulus is, in general, an N-vector. Response can be also an M-vector, though it is more practical to choose a 1-dimension response domain (for example mapping re-buffering ratio as a function of number of streams and bitrate of streams). You may end up with no possibility of any graphical representation in case N>2 or in case M>1. These cases unfortunately cannot be skipped sometimes and then we will have to live with planar projections or parallel cuts of the 'mathematical multidimensional objects' that we produce.

7. Behavior curves: curves stimulus-response

These 'curves' stimulus-response are non-analytical representations of experiments. To derive confidence in the results, the experiments must be carefully controlled and the curves must be repeated and all trials must show a 'similar' behavior. As we have said, it is impossible to account for all influences in the experiment and thus it is impossible to isolate the system 'freezing the world around' except N input variables… The result of this limitation is that even the most careful lab team will never obtain two identical response curves for the same set of input values (stimuli). It is necessary to repeat the experiment a reasonable number of times (the number depends on the statistical variability that you observe through trials) and instead of a curve you can draw a 'region' in which it is most probable to find the curve. You can imagine how difficult this turns out to be even for a low and 'easy' value of N. For N=1 each 'curve' is a set of coplanar points so it can be drawn in the plane and the set of all curves from different trials concentrates to shape up a region on the plane. Doing several trials turns the output values 'line' (a 2D set that approaches a winding trail) into a 'strip' that encloses all individual curves. In case N=2 you obtain a 3D volume that is the 'envelope' for many very similar but not identical 3D surfaces. Of course in the process you may discover that the 'point sets' that approximate 2D curves (or 3D surfaces) in repeated trials of the same experiment do not come 'close' one to another. In that case you cannot keep the data set, you need to go back to your experiment, improve the isolation between influences and start all over again to collect curves. What you are seeing is that the dominant effect in the output comes from some variable you have not controlled or succeeded to freeze.

When repeating an experiment it is very helpful to draw all the different curves produced by the successive trials in the same plane (If N allows for that), accumulating them, but using different 'colors' or in parallel planes, one besides another adding an extra dimension (if N allows for that). The planar representation has the advantage that it can be observed purposely forgetting which response corresponds to which trial (forgetting about color) so for each value of the independent variable (stimulus) there is now a set of dependent values (responses). This in fact creates a collection of histograms displayed in sequence in front of you. You can graphically observe statistical properties like accumulation points. If you are in need of creating a single collection of 'pairs' stimulus-response, to represent the outcome of all the trials as a single experiment, the best idea is to choose the accumulation point of the responses at each stimulus value.

(P.S.: some people I've known do not notice how important it is to repeat every experiment enough times and look at the accumulation. In case they repeat the experiment blindly a low number of times, say 2 or 3, they are tempted to pick the average response value at each stimulus value. This is a serious mistake. Accumulation points may be very different from an arithmetic average of a few values. They usually show 'gaps' in regions in which for some reason you never find the response to your stimulus. These regions can tell you a lot of things in the course of any further investigation. It is much easier and more accurate to detect an accumulation visually than simply taking an average. At the same time if after M trials you do not see clearly any accumulation you need more trials. If you just take averages you are at risk of stopping too soon your trials. Averages will hide the gaps that can be so informative of behavior. Averages in certain cases may not give you any clue about the statistical accumulation of results. Averages in many cases destroy information instead of providing you with information.)

8. Performance Management in three steps: Characterization -> Measurement -> Action

Over the years I've used this method for managing performance of complex systems. It is pure common sense.

Step 1: Characterization: (in the labs, before production):

Take each part of your system that can be reasonably identified as an 'independent entity' that is worth of considering isolated from the rest, select your N inputs and M outputs, put the entity in the lab and provide total isolation apart from these N inputs, then prepare experiments that let you traverse the full space defined by these N inputs and carefully annotate the values of all the M outputs for every input combination (an input vector). This step is 'characterization' of your entity. In a CDN a typical decomposition of the whole system into a list of entities may be: video cache, web cache, acquisition server, request router, DNS, live splitter, backend server of application X, backend server of application Y…, aggregation switch, virtual machine X, cluster of virtual machines, virtualization HW enclosure, edge router,…. whatever you think matters for your very specific design.

It is important to note that the characterization process give us documentation about the range in which our entity can work. Of course this range is multidimensional as is the input space.

It is also important to note that in the characterization docs we describe not only the range (i.e.: a couple of values per independent input variable), but we get a curve that tells us the exact behavior that we can expect for each output when the input moves through that range. It is perfectly possible and reasonable that after looking at the curve we take actions to ensure that the system is never out of some sub-range that we consider optimal. (See compensation mechanisms in 'step 3' later in this paragraph.)

At the end of the characterization process you have : a set of documents that describe the behavior of one entity (cache, acquisition server, content router,…) when placed in 'controlled input conditions' : known request arrival rate, known bitrate of videos, known title popularity distribution, known instantaneous number of streams,… . This behavior consists of measurements of 'output values' : time to serve first byte, time to serve first frame, re-buffering ratio, number of buffering events per minute, instantaneous throughput, average throughput per minute,… . If you have done your characterization 'professionally', then any two instances of the characterized entity: any two caches, any two acquisition servers, etc… will have a behavior that falls in the regions delimited by the characterization curves. It is required to randomly pick some instances of the entity from manufacturing plant or from production, take them to the lab and apply a quick subset of your characterization experiments just to see if there is excessive variation from the curves. If there is… bad luck, you have to refine your characterization experiments and recreate your documentation.

Step 2: Measurement: (in the field, continuously):

After you know how your 'entity' reacts to controlled stimuli, you must provide some way to measure the output of this entity while it is at work in the field, in real production. Of course collecting the output must be done without disturbing the behavior of the entity, or at least with the minimum reasonable disturbance.

The collection process must be more robust and refined than the collection performed for characterization so it can be run continuously and unmanned. Note that this is not a lab. You must be prepared to collect data and process them as close as possible to real time, with no human access to the entities, and keep the process running for long periods of time. One of your concerns is to avoid running out of space for collected

data which implies forwarding semi-processed data to other systems for representation, feedback to the CDN compensation mechanisms and long term storage for Business Intelligence trend-spotting and/or audit.

At this stage the response values are much more meaningful that before Characterization. Now you can offer a quasi-real-time plot of each value while showing the proper scale in the background. This 'scale' is the characterization curve. The most informative layout shows at the same time the whole range of the curve (for any given output variable) and plotted OVER that curve the point that the system is outputting at this instant in time.

Let me give an example. Let's say that you did your homework to instrument a video cache by building a SW agent that collects a true business-KPI: 'time to deliver first byte'. You are in production, measuring, and you obtain values: 3ms, 5ms, 3ms, 4ms, 5ms… If you had not gone through Characterization you could only have questions: is this good? Is this bad? How good is this? You are managing a true system-wide KPI, a business KPI that is many times used as SLO in contracts that set a given SLA. You can compare the current instantaneous value of the output to the behavior range that you have discovered in the labs. The best way to it is by displaying the current value over the behavior curve using the curve as a background. For example let's say that you have a planar curve: 'number of concurrent streams – time to first byte'. So you represent first byte delay as a 'function' of number of concurrent streams. The current value you get is: (1000 streams, 5ms). Maybe your curve ranges from <1ms for numbers of streams under 500 all the way up to 10ms for 2000 concurrent streams. Your curve is not a dot trail, it is a 'strip' that has 2ms height and it is centered at 1ms for 500 users and centered at 9ms for 200 users and it resembles a straight band with some slight deformation. This is much more valuable information. Now you know that you are experiencing 5ms latency while having 1000 users, and you know that it is NORMAL, because the real-time 'dot' lies on the behavior curve (within the 'normality strip'). If you were getting 1ms for 1000 users, notice this point lies out of the curve (a pack of curves that forms a strip or band of 2ms height), something is going strange (unexpectedly 'good latency' in this case). You must worry. If you are getting 200ms for 1000 users it is also strange (unexpectedly 'seriously bad latency' in this case). You must worry. Notice that you must worry in BOTH cases because the behavior you see is not normal. Very rarely you will receive a 'gift' of extra performance for free. Chances are that some other things have broken. Apart from getting all this information in a glimpse, you also can see that you are running with 1000 users and that is under the dangerous region of 2000 users that marks the end of the range. If the current value were: (1995 streams, 9ms) the value is NORMAL, because it lies in the curve, but you have to worry, because you are at the brink of overload …and if Nature is repeatable you are about to get unacceptable delay, just as you measured previously in the labs. Not bad. With just a look at a real time plotted dot on top of a pre-computed background curve you know a lot of things and all of them are relevant to your business. And all of them mark your next action if needed.

Step 3: Re-action: (in the field, when needed):

In the above example about interpretation of the real-time measures you have realized the power of having a pre-computed behavior curve as background. But what happens if you see that you are about to surpass any well-known limit? (In our previous example the limit was the max number of allowed concurrent users). Do you have time to react? I would say: no. You should plan for these cases in advance and build mechanisms inside your entities and inside your whole CDN to force your system to go through a known path even in case of unusual input. You cannot change the outside world. You cannot stop your users demanding more and more… but you do not have to die when it happens. You can plan ahead and decide at the design desk what your entities will do when they are forced outside of their limits by the uncontrollable outside world.

In our example above a cache has a known limitation of 2000 concurrent streams. (PS: This is of course an oversimplification, if you use an input vector of <number of streams, bitrate of streams, req arrival rate> you will probably notice that these three variables combine to limit the acceptable functioning range). You know that output will be unacceptable if there are 2001 users in your system so, which is a reasonable behavior when user number 2001 tries to get in? This choice of course depends on the design of your CDN. It may be you have a nice and complex request routing and you can redirect the user to another cache, it may happen that all caches are full or maybe not but the cost of impregnating a new cache with a content it does not have may be too high…it doesn't matter what it is. At some point you have to take some hard choice like in this case: dropping requests. Of course this is just an example. There are many other situations in which real-time measurements and previous characterization info combine to feed 'compensation mechanisms' to keep the CDN well behaved. Here you can see another true power of the characterization process. If you do not allow any component (entity) of your CDN to work out of range, by taking the proper (even hard) choices at the entity level you can control the 'acceptable behavior path' of your entire CDN even in case of very serious massive problems throughout your CDN. (For instance in case of a DDoS attack.)

'Compensation Mechanisms' can be designed per entity and also per system. The best possible design will probably have both types. It happens that there are system-wide properties that may NOT be the sum of the entities properties. A good example is 'goodput'. You may have N caches that work well in a complex range of situations offering a variable 'goodput', well maybe your CDN cannot take the sum of all individual maximum 'goodput' values. In this case you have to notice that the sum of traffics is going to cause problems and react to a high summed value in some way, probably dropping a few requests in some cleverly chosen caches. This kind of compensation mechanism that acts system-wide is the hardest to apply as it requires gathering information from the entire system, reasoning above the consolidated data, acting over many parts of the system (potentially)… and everything must be done in a very quick cycle. For wide very distributed CDNs that reach for the whole planet there is a challenge to simply collect-consolidate-decide-propagate in such a wide network. The cycle may take minutes in a global network with thousands of machines over tens of thousands of Km of cabling. (P.S: the best figure I've seen for a planet-wide CDN cycle is 40s to centrally compute a request routing map and I find it hard to believe. Just looks too good to be true for such a huge network with today's technology, but it is definitely possible if enough resources are put to the task and the implementation is smart.)

9. Practical CDN list of entities with their most relevant 'input variables' and 'observable outputs'.

I will suggest here a practical (minimum) decomposition of a 'generic CDN' in relevant entities and I will propose a list of stimuli/response to watch for each entity. This is a decomposition based on my own experience designing CDN and analyzing performance. There are many other possibilities, but I think it is worth looking at the entities I've selected and the criteria used to look at their performance, as this reveals a lot about the method to link the CDN design to the business objectives. You can discover for example that every time it is possible I select a response space that contains SLOs and it is very easy to interpret the response as real business KPIs.

List of Entities (a minimum proposal):

1. – Streaming Cache: a cache used to serve streaming objects, very big objects delivered using several protocols: MPEG-DASH, HLS, HDS, SS, progressive download… any streaming protocol

Stimuli space: it is made of consumer requests and CDN commands. The latter are completely controlled by the CDN and represent very few transactions so we concentrate in consumer requests or simply 'requests'.

Properties of the stimulus: (input variables):

-Request arrival rate: number of requests arriving per time unit (units: req/s)

-Concurrent sessions/connections/streams: total number of outgoing streams (units: natural number, non-dimensional)

-Demanded throughput: aggregate throughput demanded by all concurrent sessions (units: bits/s)

-Distribution of requests over content space: the statistical character of the distribution of requests over the 'content space' (the list of titles) is also known as 'popularity distribution' and must be known A-priori.

-Distribution of size and bitrate over content space: the statistical character of the distribution of size and bitrate over the list of titles affects performance and must be known A-priori.

Response space: the response of a Streaming cache is always a stream.

Properties of the response: (output observables):

-Response time: time from request to the 'first signal of service'. This property is usually measured by the consumer but that is not appropriate to characterize a server (cache) so we most commonly will use 'first byte time' which is the time from 'request arrival' to 'first byte of response going out'.

-Quality of streams: 'quality' is not the same as 'performance'. Usually there is a relationship between these two concepts. 'Play quality' must be measured at player. When characterizing the streamer we are interested in keeping 'play quality' intact (whatever it is at player) while streamer performance moves through its allowed range. There is a complex relationship between traffic shaping at the streamer output and play quality. Let's assume that the network in between is 'perfect' : it does not add delay, losses or jitter, in that case a lag in server output may cause re-buffering at player. If we define a 'player model' that buffers N s worth of packets, then by inspecting the traffic shaping at streamer output we can determine how the streamer performance will impact on play quality. Usual observable outputs are: a) number of buffer under-run events per time unit (units: events/minute); b) re-buffer ratio: time spent refilling / total stream play time (units: rational number, non-dimensional)). (P.S: Players today are very complex. This simple model does not account for streamer-player situations in which the players can speedup/throttle the streamer through signaling. Adaptive HTTP streaming and classic streaming protocols deal with these situations effectively decoupling streamer performance from play quality. Increased player buffers also decouple streamer performance from play quality. Anyway it is worth testing the streamer performance (traffic shaping) as a function of load (requests) as this is the 'base' for an understanding of play quality at player. Once the streamer is characterized we can add on top the behavior of player (buffering) and streaming protocols (signaling for speedup/throttling).

-Aggregate throughput: total throughput measured going out of cache (units: bits/s)

Curves stimulus-response:

SET 1: (A 'must have' for any decent CDN.)

Arrival rate (X) –response time(Y)

(NOTE: apply requests following a uniform distribution over a space of content that fits completely in the machines DISK but NOT IN RAM. All items in content space must be EQUAL in size. Suggestion: use files over 20 MBytes and under 1Gbyte. Proposed value: 200 MBytes. Proposed total number of items: 20000. For simplicity: encode files at the same bitrate: suggestion: 1 Mbps.)

Arrival rate (X) – re-buffer ratio (Y)

(NOTE: this curve should be plotted overlaid to the previous curve, sharing the X axis: arrival rate).

We want to trace the above mentioned pair of curves for every 'pure' service that the streamer is capable of: VoD HDS, VoD HLS, VoD SS, Progressive D., Live HDS, Live HLS, Live SS …whatever…

We want to trace the above mentioned pair of curves for 'non pure' service. The 'non pure' services are made of 'service mixes'. Enough service mixes must be considered. A service mix can be defined as: X1% requests come from service 1, X2% requests come from service 2 …XN% requests come from service N. X1+X2+…XN =100. Pure services have a mix in which only one Xi is non-zero.

SET 2: (A 'must have')

Connections open(X)-response time for a single new incoming request (Y)

(NOTE: when X users are in the system and a new request arrives it receives Y response time)

Connections open(X) – re-buffer ratio (Y)

(NOTE: this curve should be plotted overlaid to the previous curve, sharing the X axis: connections).

We want to trace the above mentioned pair of curves for every 'pure' service and for some service mixes. (See above details in SET 1).

SET 3: (Recommended)

Surface: (input X,Y, response Z): arrival rate(X)-Demanded throughput(Y)-response time(Z)

for all pure services and for some selected mixes of service.

(NOTE: apply requests so 'demanded throughput' varies in 10 steps that cover from 10% nominal NIC capacity to 100% nom NIC capacity increasing 10% with each step. This process produces 10 plane curves arrival rate(X)-response time (Z) which are 'cuts' to the surface A.R.(X)-D.T.(Y)-R.T.(Z))

Surface: arrival rate (X) –Demanded throughput(Y)- re-buffer ratio (Z).

(NOTE: we want the slice curves of this surface to be plotted overlaid to the previous slice curves, sharing the X-Y planes: arrival rate-demanded throughput).

SET 4: (optional)

Surface: (input X,Y, response Z): connections open(X)-Demanded throughput(Y)-response time(Z) for all pure services and for some selected mixes of service. (See above NOTE).

Surface: connections open(X) – Demanded Throughput(Y)-re-buffer ratio (Z).

(NOTE: we want the slice curves of this surface to be plotted overlaid to the previous slice curves, sharing the X-Y planes: arrival rate-demanded throughput).

SERIES OF SETS 5: (optional)

Repeat SETS 1,2,3,4 using a non-uniform distribution of requests over content space. As the non-uniform distributions are parametric (Zipf skew factors, variance in normal distributions, etc…) a family of curves and a family of surfaces will result. To get useful data use at least 3 different skew factors. This series of experiments will easily explode your data and are time consuming. In case you go for this you will obtain a parametric family of surfaces that will cost you much effort. The surfaces will be useful only in case your streaming cache is designed to be very sensible to request distribution. Unfortunately this is the case for most caches in the world. Notice that uniform distribution of requests over title space is the worst case while Zipf is the best. Notice that Normal distribution of size over title space is close to the real world.)

2. – Small object cache (web cache): a cache used to serve small objects: small web files

Stimuli Space: the stimulus to the cache is always a consumer request.

Properties of the stimulus: (input variables)

-Request arrival rate: the number of requests arriving per time unit. (Units: r/s).

-Distribution of requests over content space: object popularity distribution.

-Distribution of Object size over content space: object size distribution.

-Distribution of Object duration over content space: distribution of object duration (how many objects are replaced by others and how often) impacts greatly on cache performance

Response space: the response of a web cache is always a web object: small file.

Properties of the response: (output observables):

Response time: (see above definition of response time for Streaming cache)

Aggregate throughput: (see above definition of throughput for Streaming cache)

Curves stimulus-response:

Arrival rate (X) –response time(Y)

Trace a family of curves varying:

Distribution of requests over content space: select some small object (40 Kbyte). Create 10^9 renamed copies: ~40 TB worth of objects. Use several Zipf distributions to generate requests changing skew. For each distribution plot the complete curve moving the value: req/s. Plot no less that 5 curves and mark them using the skew parameter.

Distribution of size over content space: select some small object (40 KB). Create 10^9 renamed copies. In the copy process modify the object size following a normal distribution. Do this several times changing parameters in the normal distribution: mean and standard deviation. Use as mean: 4 KB, 40KB, 400 KB, 4MB. Use as variance: 0.1, 1, 5. You obtain 12 distributions. Plot the curve: A.rate(x)-R.time(Y) for each combination of the 5 previous Zipf distributions of requests and the 12 Normal distributions of size. You will obtain a total of 60 curves.

Distribution of duration: Select 3 Zipf distributions with different skew factors. Apply these distributions to object duration (to the total amount of 10^9 objects). Obtain 60x3= 180 curves.

Surface: Arrival rate(X)-Demanded throughput(Y)-response time (Z)

(See Streaming cache SET 3).

Apply different distributions of requests (over object space), of size (over object space) and of duration (over object space) to generate a family of families of surfaces (120 surfaces).

3. – Live Acquisition server: stream oriented (splitter) for live signals.

Stimuli Space: the stimulus is the number of incoming streams and the demand of 'splits': outgoing streams

Properties of stimulus:

Total number of input + output streams: see streaming cache: connections.

Stream bitrate: See streaming cache: demanded throughput.

Distribution of stream bitrate over stream space (over all connections)

Response space: outgoing streams or 'splits'.

Properties of response: ability to sustain bitrate and quality of all demanded streams

Total input + output throughput: sum of bitrates of all streams: in + out.

Output quality: see Endpoint stream quality.

Curves stimulus-response:

Connections(X)-throughput in + out(Y)

Connections(X)-Quality(Y)

Surface: Connections(X)-Demanded Throughput(Y)-Quality (Z)

(NOTE: See streaming cache SET 4-B. Repeat surface for several Normal distributions of connections of varying BITRATE. Distribute the BITRATE following the Normal distribution.)

4. – File Acquisition server: file oriented. Is the entry point for files (no matter these files will be distributed as streams later). It usually serves also as an 'internal origin' for the CDN caches.

Stimuli Space: requests to read & write from/to files.

Properties of stimulus:

Writes rate: (Units: writes per second)

Reads rate: (Units: reads per second)

Concurrent users: total connections (writing + reading)

Demanded due size: concurrent reads * size of demanded objects.

Distribution of size over content space

Response space: reads & writes over files.

Properties of response

Response time (write): first byte: See streaming cache.

Response time (read): first byte: See streaming cache.

Write throughput: for a single connection & aggregate (all connections)

Read throughput: for a single connection & aggregate (all connections)

Curves stimulus-response:

Connections(X)-Read Throughput(Y)

Connections(X)-Write Throughput(Y)

Demanded due size(X)-Read Throughput(Y)

Connections(X)-Demanded due size(Y)-Read Throughput (Z)

5. - Request Router: very dependent on CDN design, maybe part of DNS implementation

Stimuli Space: consumer requests in need of routing. The router will respond causing redirection.

Properties of stimulus:

Request rate

Request distribution (over content space)

Request distribution (over consumer-IP space)

Response space: responses to routing queries.

Properties of response:

Response time

Curves stimulus-response:

Request arrival rate(X)-Response time(Y)

(NOTE: Repeat the above curve for various values of 'popularity' of URLS: distribute requests over URLs (content space) using a Zipf distribution.)

6. - Hierarchical DNS: I assume your CDN will use some form of modified DNS. In any case you will need at least to have a regular DNS to interface to the outside world and its performance always matters.

Stimuli Space: DNS requests.

Stimulus properties:

Request rate

Response space: DNS responses.

Response properties:

Response time

Curves stimulus-response:

Request rate(X)-Response time(Y)

10. Conclusions about CDN Performance Management : Concepts & Tasks

We have seen that there are misconceptions about the goal and the method to analyze performance.

We have seen that having the right knowledge about a CDN performance is a powerful business tool that has a direct impact in service monetization, service maintenance, problem solving, and proper design-implementation and as a bonus it helps in trend spotting and evolution of our business.

We have stated that analytical modelling of a CDN is impractical.

We have proposed Characterization as the right tool to help in performance analysis.

We have introduced behavior curves.

We have proposed a method in three steps: Characterization (a priori in labs) -> Measurement (continuous, in the field) -> Action (when required in the field, through compensation mechanisms).

We have suggested a simple decomposition of a modern CDN suitable to apply Performance Management

We have suggested a reasonable set of inputs/outputs and behavior curves for each entity in the CDN.

All these focus in performance management and all these concepts usually lead to carrying out some tasks in our CDN business. Here is my suggestion for a healthy CDN life-cycle: (list of tasks):

-Characterization Lab: for every entity in the CDN the lab provides the behavior curves: stimulus/response. The Lab must be available to re-characterize any production item when the specs are changed (i.e.: a new processor or mother board is used in a cache, a new NIC is used, RAM technology improves, HD technology improves, etc…)

-Soft Real-time Dashboard: a central tool that allows looking at every KPI defined for the CDN in soft real-time. It involves instrumenting all the entities (running collecting agents per entity), pre-computing behavior curves and then graphically displaying real-time measured values of KPIs aggregated or from individual entities over behavior curves. Today the dashboard is typically a graphic web tool.

-Deployment & sizing guide: a high level outcome of the characterization is a 'rule-of-thumb' to dimension the max capacity that is usable from current CDN parts. This 'rule-of-thumb' is an easy and quick chart that can be used to match a CDN deployment (number and type of caches, links, ports, infrastructure…) to known input (distribution of demand, types of content, bitrates, concurrency…). This is an invaluable document for pre-sales when you meet with your prospective partners/customers and they ask for on-net services that would require an ad-hoc deployment. Having your quick chart at hand will allow you to provide immediately a crude estimation of cost and deployment time in direct comparison to demand. If you just have regular 'pure CDN' customers your sizing guide will help you in talking to your engineering and deployment to discuss costs of a capacity increase or renewal of your CDN edge.

-Compensation & safety mechanisms: must exist per entity and per CDN. These mechanisms can be really sophisticated but as the very minimum they must ensure that the CDN still works while it is under pressure: too many requests coming, requests coming at too fast rate... These mechanisms should include 'levers' to adjust global behavior, redundancy mechanisms, failover mechanisms, etc…. Many of the most interesting CDN patents and many of the smartest contributions of CDNs to the state of the art in distributed systems are 'global compensation mechanisms'.

As an ending note I'd say that performance analysis is an endless world that joins together the deepest technology knowledge with the most modern business management techniques. Today Big Data approach to business is starting to nail the surface of instrumentation in many businesses. Technological businesses like CDNs have the advantage of being born technological so it is inherent to them to be instrumented. Remember that it is always better to recall the true meaning of performance and do not be confused by the buzzwords of technical environment.