EWT and URS

Universal Routing 8.1.4

4/18/2023

Table of Contents

Expected Wait Time (EWT) in URS - White Paper

This document discusses how Expected Wait Time (EWT) and its related statistics, functions, and methods are utilized in Universal Routing Server (URS). The following sections aim to provide a good understanding of the different variations in calculating EWT and their impacts, from a URS standpoint.

Function: InVQWaitTime (Virtual Queue)

This section lists the variations in usage for the `InVQWaitTime (Virtual Queue)` function.

- When a Virtual Queue is associated with an agent group, then URS takes Stat Server's EWT for the Virtual Queue and adjusts it relative to the call's position in the queue.
- If the Virtual Queue is NOT associated with an agent group, then URS calculates `AverageHandlingTime` for all agents from the Virtual Queue and multiplies it by the call's position in the queue. While calculating `AverageHandlingTime`, URS tries to consider cases of multi-skilled agents and multiple URS instances.

> **Important**
>
> The above applies when using the SData[Virtual Queue, StatExpectedWaitingTime] funtion. URS can count AverageHandlingTime for a Virtual Queue itself only if it has (or had) calls in it.

Web Method: urs/call/connid/lvq/VirtualQueueName?aqt=stat/urs/urs2

This web method provides more information than the `InVQWaitTime` function and also allows you to control exactly how to count EWT. The JSON object returned with this method contains an EWT parameter, which is semantically close to the value returned by `InVQWaitTime`. The `aqt` input parameter allows more control in accounting for EWT stats.

- `stat` – identical to the value returned by the `InVQWaitTime` function.
- `urs2` – similar to the `InVQWaitTime` function, but URS always calculates the `AverageHandlingTime` by itself (regardless of whether the Virtual Queue is configured with agent group).
- `urs` – default value, like `urs2`, but instead of `AverageHandlingTime`, URS uses `AverageQuittingTime` from the Virtual Queue. URS counts this average by itself based on the last **32** calls distributed from

the queue.

Note: The `lvq` method can also be applied to groups of virtual queues. If the virtual queue name ends with `.GQ`, URS calculates EWT and other parameters for the entire collection of Virtual Queues. This and any other web method can be called from a routing strategy with the `RequestRouter` function. For example, `RequestRouter['##SELF', 'urs/call/connid/lvq/VirtualQueueName', 'aqt=stat','', '' ]`.

Web Method: `urs/call/connid/query`

This method is applicable when the use of Virtual Queues are not appropriate or not possible. For example, Virtual Queues that include calls that might wait for totally different targets or have different thresholds, etc. This method uses the Router's Queues (all calls in Router's Queues wait for the same targets and have the same thresholds).

This method internally uses `RvqData[]` URS function to get aggregation information about all Router's Queues that the call is in.

The JSON object returned with this query method contains among other attributes:

- `min_rvq_ewt` – Minimal EWT among all the Router's Queues counted using `AverageHandlingTime` provided by URS.
- `min_ewt` – Minimal EWT among all the Router's Queues counted using `AverageQuittingTime` provided by URS.

> **Important**
>
> `AverageHandlingTime` that URS provides for the Router's Queue does not consider whether agents are multi-skilled or cases of multiple URS instances.

Challenges

Calculating EWT for a call is in itself a complex task and is usually further complicated by the absolute absence of restrictions that strategies might use when placing calls in queues. This results in arbitrary many-to-many relations between calls and agents and significantly erodes the concept of a **call's position in queue**. Another factor that complicates the task is performance. URS must count EWT very quickly so as not to affect its primary task of routing.

The following methods that we use, help address this complexity.

Router's Queues

When processing calls, URS has a lot of objects in memory called internal queues or Router's Queues.

Every time a target `Selection` object/function is executed, URS creates a Router's Queue, if it was not already created. The Router's Queue includes (refers to) every agent with the provided skills and places the call into this queue (according to its priority).

- Every call can be in multiple Router's Queues simultaneously (every next target `Selection` object can place a call into a new Router's Queue) and the positions of the same call in these queues are usually different.
- Any single agent (especially those multi-skilled) also might be included in multiple Router's Queues.
- EWT calculation for a call must consider that every agent that a call is waiting for, through one of the Router's Queues, also has other calls waiting from an arbitrary set of other Router's Queues.

Virtual Queues

A set of Router's Queues can be combined in bigger Virtual Queues. Every Router's Queue can be included into only one Virtual Queue, which is specified in the properties of the IRD target `Selection` object creating the Virtual Queue.

For URS, a Virtual Queue is a collection of one or more Router's Queues. It is a logical queue having calls from all involved Router's Queues and also combining all agents from them.

> **Important**
>
> The following statement on Router's Queues in the previous section is valid for Virtual Queues also: EWT calculation for a call must consider that every agent that a call is waiting for, through one of the Router's Queues, also has other calls waiting from an arbitrary set of other Router's Queues.

Lifetime - VQ versus Router's Queue

Router's Queues are highly dynamic objects. They can be created/destroyed on the fly. If the executed target `Selection` object requires a Router's Queue that URS does not have, it will be created and will also be associated with the corresponding Virtual Queue. If a Router's Queue is empty with no call in it for a specific period, the queue will be destroyed. If the queue is required later, it will be recreated.

Virtual Queues are permanent objects in URS memory. Once created, they exist forever even if all Router's Queues referring to it are removed from URS memory.

EWT calculation for a call in URS can be based on either the Router's Queues that the call is in or the Virtual Queues that the call is in. The Router's Queue approach involves a lesser amount of and more relevant calls and agents. But due to the dynamic nature of Router's Queues in general, EWT can provide more volatile data reflecting real time fluctuations/peaks of calls/agents.

Relying on Virtual Queues can provide more stable data, but not necessarily very precise data due to the nature of dynamic events in a contact center. Virtual Queues often used for different business

purposes are sometimes not suitable for EWT calculations. For example, Virtual Queues could include Router's Queues with very different sets of agents (having nothing or little in common) solving completely different tasks.

Stat Server Data

In all of its EWT calculations, URS relies in one or another way on data provided by Stat Server. Stat Server-supplied data used by URS ranges from raw data about agents from Router's Queues to more ready-to-use data about Virtual Queues. URS provides a few different variations of EWT calculation. Each variation uses the Stat Server data differently.

The oldest and simplest method of EWT calculation is by using InVQWaitTime function, which relies on the ExpectedWaitingTime statistic for Virtual Queues provided by Stat Server (StatLoadBalance is used), scaling it to the current call position in some Virtual Queue.

The major point in relying on Stat Server Virtual Queue statistics is that it always requires configuring the association of a Virtual Queue with some agent group. In cases when it is not possible (such as in strategies that use extremely dynamic skills-based routing), this might be a serious factor in the decision to not use Virtual Queue statistics from Stat Server.

Because of that, URS chooses to use a hybrid approach when Stat Server EWT for Virtual Queues is required. It uses a Stat Server statistic where possible. However, in cases when it doesn't work (Stat Server returns a value 10000+ and also the configuration does not have any single agent group referred to in the Virtual Queue), URS silently switches to its own method, that is, it calculates AverageHandlingTime for all agents in the Virtual Queue multiplied by the call position in the queue.

Router's Queues - EWT Methods

In the following description, StatEWT(VQ) refers to the ExpectedWaitingTime statistic as returned by Stat Server for Virtual Queue (VQ).

For every Router's Queue, URS has the following information: - list of all included agents. - sorted by priority array of calls waiting in the queue.

- The `RvqData` function (or the `rvqdata` web method) in the context of a call, provides the following EWT related information (information about the `rvqdata` web method can also be taken directly from URS with the `/urs/help/call/rvqdata` web request):
  - `queue_len = RvqData[rq_id, RVQ_DATA_QUEUE_LEN]` returns the number of calls in the queue.
  - `pos = RvqData[rq_id, RVQ_DATA_POSITION]` returns the position (starting from 1 to `queue_len`) of a call in the queue.
  - `quit_rate = RvqData[rq_id, RVQ_DATA_QUIT_RATE]` returns how quickly calls are distributed from the queue (more exactly, it returns the average number of seconds between 2 subsequent quits of calls from the queue). The quit rate returned here is either for the Virtual Queue associated with the Router's queue (if URS option `wait_time_prediction` is set to virtual) or otherwise for Router's queue itself. (The `wait_time_prediction` option controls which queue URS will use when it needs the average quit time of calls from some internal routing queue. You can specify the value for the option as `internal` (internal queue is used) or `virtual` (virtual queue associated with the internal routing queue is used). The default value is `internal`. The option is specified in the URS

application object and changes take effect immediately.)

Quit Rate

For every call, URS tracks the number of times and when the call entered and the call left every Router's Queue and every Virtual Queue.

Using these event counts:

- For every Router's Queue, URS counts the average quitting rate (abandoned calls are not counted; average is counted for last 10 calls that left the Router's Queue).

- For every Virtual Queue, URS counts the average quitting rate; only calls leaving a Virtual Queue because of routing or explicit clearing by a strategy are considered and the average is counted for the last 32 calls that left the Virtual Queue.

- Quitting rate is expected to automatically take into account cases of multiple URS instances. The more the URS instances, the bigger the "quitting rate" that is counted by each URS instance; calls inside every URS will have a lesser chance to be routed when an agent becomes available, increasing the quit rate within every involved URS.

- `AHT = RvqData[rq_id,RVQ_DATA_AHT]` returns the Average Handling Time of one call by agents from this queue. Semantically, this data is the same as that mentioned in the above quit_rate, but it is calculated very differently and is very close to how Stat Server calculates it. URS explicitly goes through every agent from the Router's Queue and gets the Average Handing Time for the agent (the time the agent takes on an average to handle one call).

Agents not logged into the call's media are ignored. If it is Workforce Management targeting a Router's Queue and an agent is not assigned to the required Activity, such an agent is also ignored. Average Handling Time of all remaining agents is combined into their collective Average Handing Time. Once calculated, the Routing Queue AHT will be reused for the next 4 seconds. The fact that agents can be involved in other internal queues is not taken into account assuming agents handle calls only from this Router's Queue.

Note: For every agent and every media, URS counts how long on average the agent processes calls of a particular media. URS tracks the time when an agent starts and stops processing every interaction. The time difference between them is the time the agent handles interactions.

Average Agent Talking Time (ATT) is the average of the last 10 calls of the given media that the agent handles. It is possible to override this ATT value with an explicitly provided value. The `agent_att` URS option can explicitly provide agents' average talking time (ATT) values for voice. Also the `use_agent_att` option controls whether the `agent_att` option will always be used or only until URS collects enough information (10 calls processed by agent) to form its own ATT. The `UseAverageTalkingTime[time in seconds]` strategy function can also set the ATT for the call's media. If nothing is specified and URS does not have any data, then URS uses `300` seconds as the ATT for voice and `600` seconds for multimedia.

- ewt = RvqData[rq_id, RVQ_DATA_EWT] returns only `aht*pos`.

As previously mentioned, a single call can enter a few Router's Queues with every one of them having its own EWT. Some aggregation of multiple EWTs is required to provide a single EWT for the entire call. As such, EWT can be used (usually minimal through all of the Router's Queues). However,

in cases when the next routing queue call placed is not an *extension of* but rather in *addition to* the previous routing queue, another special aggregation is more appropriate. Corresponding calls to the RvqData function return the appropriate aggregative values.

- `min_ewt = RvqData[NA, RVQ_DATA_MIN_EWT]` returns the minimal EWT among all Router's Queues.
- `agr_ewt = RvqData[NA, RVQ_DATA_AGR_EWT]` returns the aggregative (inverse to the sum of inverses) EWT among all Router's Queues.
- The Web API method's query returns, among other things, the EWT information for the requested call:
  - `rwq_ewt` – `RvqData[NA, RVQ_DATA_AGR_EWT]`
  - `min_rvq_ewt` – `RvqData[NA, RVQ_DATA_MIN_EWT]`
  - `min_ewt` – Minimal among all Router's Queues `quit_rate * pos`
  - `max_ewt` – Maximum among all Router's Queues `quit_rate * pos`

Virtual Queue - EWT Methods

Virtual Queues are collections of Router's Queues. As with Router's Queues, URS counts `quit_rate` and AHT for Virtual Queues too. In addition to its own metrics for Virtual Queues, URS can also use metrics provided by Stat Server. As with Router's Queues, EWT can be calculated based either on Virtual Queue `quit_rate` or Virtual Queue AHT.

URS counts `AverageHandlingTime` (AHT) for Virtual Queues in a similar manner, but in a more complex way than for Router's queues. Specifically, URS tries to take into account the possibility of an agent being shared by multiple Virtual Queues, and also by multiple URS instances. However, it only works when URS has *material* to work with, that is, at least one Router's Queue referring to the Virtual Queue must be processed with URS (so that URS will be able to go through the path - *Virtual Queue > Router's Queues(s) > Agents*. In other words, it is better to first place a call in a Virtual Queue (to guarantee the existence of such a path) and then request for AHT-based EWT.

Virtual Queue Average Handling Time

- URS explicitly goes through all Router's Queues associated with a Virtual Queue and inside every one of them, it goes through all agents.
- Agents not logged into the media associated with the call are ignored.
- If it is Workforce Management targeting a Router's Queue and the agent is not assigned to the required `Activity`, such an agent is also ignored.

URS calculates the Agent Average Talking Time (ATT) (how long it takes agents, on average, to handle one interaction). See note on AHT calculation for internal Router's Queues.

- If an agent is involved in multiple Virtual Queues, the agent's average talking time is increased proportionally.
- If other URS instances also have calls in the same Virtual Queue, the agent's average talking time again is increased proportionally (Routers from self-awareness clusters share information about which Virtual Queue each of them is currently working with (has call waiting in these Virtual Queues).

Scaled Average Handling Time of agents is combined into their collective Average Handling Time (inverse to the sum of inverses).

If a Virtual Queue has no associated Router's Queues, then URS instead tries to use the agent group/skill expression presented for this Virtual Queue. For every Virtual Queue, URS remembers which targets were used in the past for a Virtual Queue and stores one of them as presenting this Virtual Queue. If defined, then URS will go through the agents from this presenting group and perform similar calculations.

- If the Virtual Queue has no presenting group defined, then URS will use the `unknown_aht` option as a ready-to-use value. (The `unknown_aht` option provides the average handling time for virtual queues that URS will use in cases when there is not enough information to calculate the actual EWT. The default value of this option is *9.999 secs*. You can specify the option in the URS application object and changes takes effect after a restart.
- Once calculated, URS will reuse the Virtual Queue AHT value for the next 10 seconds.

Once counted, the AHT of the Virtual Queue is used:

- For calculating `InVQWaitTime` in cases when the Virtual Queue is not configured with an agent group.
- For answering the `lvq` web method (or `LvqData` function). The `lvq` web method (or `LvqData` function) can also be directly obtained from URS with a `urs/help/call/lvq` web request.

EWT Evaluation

EWT evaluation is based on the following:

- **`aqt=stat`**: quit rate (time per call) is provided by Stat Server. It is calculated as `ewt/ciq` (both from Stat Server). URS multiplies it by the number of calls (in its memory) and by the scale factor (to account for multiple URS instances).

Note: Prior to v8.1.400.37, the scale factor was not used. It resulted in the final EWT value not being scalable for the multiple URS instances case. Multiple URS instances safe variation was provided by the `InVQWaitTime` function (on the assumption that all the URS instances are loaded with a similar calls pattern).

- **`aqt=urs`**: quit rate is provided by URS. It is calculated as the average time between each `EventDiverted` event from the Virtual Queue. URS multiplies it by the number of calls in its memory. Such a quit rate is already scaled between multiple URS instances and adding more URS instances will result in increasing this time for every separate URS instance. As a result EWT is scalable for the multiple URS instances case (on the assumption that all the URS instances are loaded with a similar calls pattern).
- **`aqt=urs2`**: quit rate is provided by URS. It is calculated as described above in the *Virtual Queue Average Handling Time* section. URS multiplies it by the number of calls in its memory and by the scale factor (to account for multiple URS instances).