Calculation of a host's Physical CPU Usage (phCPU) with the use of New Relic Queries

Table of Contents

Definitions:

- **Physical CPU Usage (phCPU):** The number of physical CPUs used by a host in a given time period, multiplied by that time period. In this document Physical CPU usage is abbreviated as phCPU. Is should be mentioned that PhCPU usage is different from the host's actual CPU usage, or simply CPU usage, on a given time period.
- **CPU Allocation:** The total amount of physical CPU usage a host has been provided with. CPU Allocation can be described in various time units. The ones most frequently used are the CPU days and the CPU hours. It is a measurement of hardware-resource usage a host is provided, for a given time period. Usually it is defined in a contract agreement.
- **Time allocation:** The (maximum) time period a given host can be used. This is usually defined in a contract agreement.

Definitions Example:

The example which will be used to describe the above definitions involves a hypothetical host which uses 3 instances (also known as units, entities, or nodes).

- Hardware Configuration: Each instance has 4 Physical CPUs. For each instance, the Physical CPUs may be arranged in various hardware configurations such as one core with 4 CPUs or two cores with 2 CPUs each. We are interested in the sum of the number of CPUs per instance, regardless of the hardware arrangement/configuration.

- The Case of a Contract Agreement: A contract has been signed so that the host has been provided with a **CPU Allocation** of 10,000 CPU days and a **Time Allocation** of 2 years. The way the CPU Allocation can be used is entirely at the host's team discretion and decision. If the host's team decides to use all three servers per day, then the utilization would be 3 instances x 4 CPUs per day, which means that the server would use 12 CPU days per day. If, however, for whatever reason the host's team decides to use only one instance per day (and keep the other two as reserves), then the server utilization would be 1 instance x 4 CPUs per day, which translates to 4 CPU days per day.
- The case where Allocated CPU days have run out: If the allocated CPU days have been used before the end of the time allocation, the team responsible for the host's usage and maintenance needs to renew/update the contract with a new amount of CPU allocation. If we assume that the host will be continuously running in a 3 x 4 = 12 CPU configuration, then the expected number of days the server is entitled to run with the given CPU allocation of 10,000 CPU days is 10,000/12 = 833 days. We observe that the expected # of days the host is allowed to run based upon its hardware configuration is slightly bigger than its Time Allocation of 2 years. Usually, contract participants consider the most probable hardware configuration to be used throughout the Time Allocation period, so that they can tailor the CPU Allocation operational needs and define them in the contract.
- The case where Time allocation has run out: If the Time Allocation has been reached, the team responsible for the host usage and maintenance has to renew/update the contract. The remaining CPU allocation may or may not be included in the renewed contract.
- Conversions: CPU days can be converted to CPU hours when we use the appropriate conversion factor. 12 CPU days convert to CPU hours by multiplying with the factor 24hr/day. So, 12 CPU days = 12 CPU days x 24hr/day = 576 CPU hours. Using again the case of a Contract Agreement mentioned above, the 3x4 CPUs instances server uses 12 CPU days per day, which converts to 576 CPU hours per day.

Importance:

CPU Allocation defines the hardware resources a host can access. From the starting date where a host is in the state of "Production," certain amounts of hardware resources are used/consumed on a daily basis - or any other time unit of choice. The host's team is responsible for the knowledge of that usage. The exact usage of resources should be available to both the host users (the customers) as well as the company providing the computational resources, especially when a contract renewal is due.

The phCPU usage is perhaps the most important of all hardware resources. It's the most frequently used resource to measure consumption of a server or an individual computer application. It is also the computer resource's type most frequently purchased within contracts. Therefore, phCPU Usage directly affects the operational costs of the host's team. Because of this reason, tools and methods which quickly and correctly calculate phCPU Usage on any required time frames are of utmost importance to both the host's team as well as the host's provider.

In many cases, phCPU usage is easily calculated. When the host's hardware configuration in a given time period is known and unchanged, then phCPU usage is simply the total number of the host's CPUs multiplied by the time unit (example days) of choice of this time period. The phCPU calculation becomes a challenge however, when the host undergoes hardware configuration changes during a given time period. For example, the host may need frequent upsizes, i.e., temporary provision of hardware configuration with stronger characteristics to tackle increased traffic/demand. The upsizes may be followed by scheduled downsizes to the original hardware configuration once the need for an upsize has ended. Additionally, the total number of instances may change, either intentionally by the host's admins (shutdown due to maintenance, permanent changes) or unintentionally when an outage occurs. In these examples, one

can still calculate phCPU usage with the methods described above. However, the calculations become complex, with increased likelihood of invalid results.

This article provides methods where the user will be able to calculate Physical CPU usage, in a variety of cases -such as the occurrence of a resize- and time periods of any magnitude: from a mere 1 hour to months. The New Relic Query Environment is used.

The host's scenario:

For all the New Relic queries described in this article, we are going to use a hypothetical host with the following characteristics:

1. The host has defined all environments (for example, Production and Staging environments) with a unique name. To find the host's environment's names, one uses the following query:

   ```
   SELECT uniques(apmApplicationNames) from SystemSample SINCE 20 days ago
   ```

   From the query's result, we pick the name whose environment we wish to study. In this scenario we will use the hypothetical 'prodname' for the host's Production environment. The use of the correct name of the environment one wishes to study is of utmost importance, since this name is used in all concurrent queries.
2. Let's hypothesize that the 'prodname' starting hardware configuration consists of 3 instances, with 16 CPUs in each instance. During the date of January 26, the host undergoes a downsize, i.e., reduces the number of CPUs of each instance, from 16 to 8 CPUs. The downsize occurs on January 26, 2021, between 13:00 – 15:00 UTC. The host's downsize is described in Figure 1.

Figure 1: A merchant whose environment undergoes downsizing. 3 instances downsized from 16 CPUs (before 2:00 PM) into 8 CPUs (after 3:00 PM). During transition (2:00 PM),

the instances report to New Relic a CPU active count of approximately 10.7, 10.7, and 9.33.

3. Before we start with the creation and description of the phCPU calculations queries, let's define the query which will be the workhorse of all our subsequent queries. The below query calculates the value (but not its usage through time) of the average Physical CPU used by the host's environment named 'prodname'. The time frame for the averaging is Jan 26, 2021, 12:00-13:00 (one hour):

```
SELECT average(numeric(processorCount)) as processorCount FROM
SystemSample where apmApplicationNames = 'prodname' facet
entityName SINCE '2021-01-26 12:00:00 UTC' UNTIL '2021-01-26
13:00:00 UTC'
```

The result of the above query, with the instances names covered, looks like the following:

If we use a time frame later than 12:00-13:00, for example from 15:00 and after, the result would be showing the downsized average Physical CPU, which would be [8, 8, 8]. It is very interesting to mention the results of the query when used during the downsizing time frame, which is 14:00 -15:00:

The above values are due to New Relic's rounding of the Physical CPU which is downsized from its original value of 16 to the final value of 8.

Physical CPU (phCPU) usage calculations:

Calculation of phCPU hours usage during 1 hour time frame:

We wrap the workhorse query into a new one by adding a 'timeseries' and sum it over the variable "processorCount", while the time frame is from 12:00 to 13:00. We add the keyword "limit" because if the total number of instances is greater than 5, New Relic does not add them. By increasing the limit, we

make certain that New Relic adds all possible instances. The sum(processorCount) is the phCPU hours of the host, in the time frame of Jan 26, 2021, 12:00-13:00 UTC.

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
hours limit 20) SINCE '2021-01-26 12:00:00 UTC' UNTIL '2021-01-26
13:00:00 UTC'
```

Result:

It is very interesting to mention the value of phCPU hours, during the transition (14:00 – 15:00) time frame. To calculate this value, we use the transition time into our previous query (`SINCE '2021-01-26 14:00:00 UTC' UNTIL '2021-01-26 15:00:00 UTC'`). The result is demonstrated below:

Calculation of phCPU hours usage during many hours time frame:

We use the exact query mentioned on the 1-hour time frame example, and we simply change the time frame into the desired one. For example, if we wish to calculate the phCPU for 12 hours, Jan 26, 2020, 00:00-12:00 UTC, we have:

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
hours limit 20) SINCE '2021-01-26 00:00:00 UTC' UNTIL '2021-01-26
12:00:00 UTC'
```

Result:

This result can also be calculated as: phCPU = 3 instances x 16 CPUs x 12 hours = 576 CPU hours

If we use the above query for the entire 24hr time period of Jan 26 (that means we use `SINCE '2021-01-26 00:00:00 UTC' UNTIL '2021-01-27 00:00:00 UTC'` time frame) we will also include the downsize transition. The result is the below:

This value can also be calculated by hand as follows: From Figure 1, we see the entire CPU profile. We observe that during the 24hr time period, the environment is on a 3x16 configuration for 14 hours, downsizes into a 3x8 configuration for 9 hours, and finally has 1 hour transition where the instances have CPU values of (see previous results) 11.4, 10.3, and 9.36 respectively. So, the phCPU hours for this environment for the entire day is 3 instances x 16 CPUs x 14 hours + 11.4 + 10.3 + 9.36 + 3 instances x 8 CPUs x 9 hours = 919.06 phCPU hours.

Calculation of phCPU hours usage during many days time frame

This is the most important case, because the user would like to calculate vCPU hours on a large time period, like for example, 1 month or even an entire year. For this example, we have to change our approach to build the appropriate query.

We do a timeseries sample on a 1-day time frame, instead of 1 hour. The reason for that is that the 'timeseries' keyword can be used with up to 366 time-buckets. So, if/when we use a timeseries of 1 hour, we can sample roughly 366 hours, which is approximately 15 days. If we use a study time period of more than 15-16 days (the "Since" keywords on the queries) on the previous examples, New Relic will produce a 500 error like the one below:

To avoid this error, we switch the timeseries to “1 day” instead of “1 hour”. With this switch, we will be able to sample with the timeseries approximately 366 days (roughly 1 year), instead of 2 weeks.

If we want to calculate CPU days allocation, we keep the variable sum(processorCount) as is. But if we want to calculate CPU hours as we did on all the previous examples, we multiply sum(processorCount) with the number of hours per day, which is 24. In this way, we convert the CPU days which are actually calculated, into vCPU hours, as noted in the “definitions” section of this article. First, let’s see what the result looks like if we use only a 1-day time frame on the query we mentioned in this example:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit 20) SINCE '2021-01-27 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

Result:

For two days, from Jan 26, 2020, 00:00 – Jan 28, 2020:
Result:

Finally, we can use a long time period, and see how much phCPU hours have been used. The below example is for 28 days, Jan 1 to Jan 28, 2020:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit 20) SINCE '2021-01-01 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

And the result is shown below:

Hint: if we want to get the phCPU hours value without rounding “approximations,” we can switch the type of report from the “Chart type” drop-down menu, from “Billboard” into “JSON”. Then we can view the exact value:

Calculation of phCPU days usage during many days time frame

The same query, when used to calculate phCPU days (instead of phCPU hours) is the following:

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit 20) SINCE '2021-01-01 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

In this example, we demonstrate how the results values change when we request phCPU days rather than phCPU hours. The query is identical to the queries of the previous examples, with the only difference that we do not multiply sum(processorCount)with the factor 24 hour/day. The result is demonstrated below:

Chart of phCPU hours usage during many days time frame

If we want to create a chart of the phCPU hours per day an environment has used, we can use the previous query and include an additional ‘timeseries’ keyword, to produce a chart. The final query looks like the one below:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit 20) timeseries 1 day SINCE '2021-01-01 00:00:00 UTC' UNTIL
'2021-02-01 00:00:00 UTC'
```

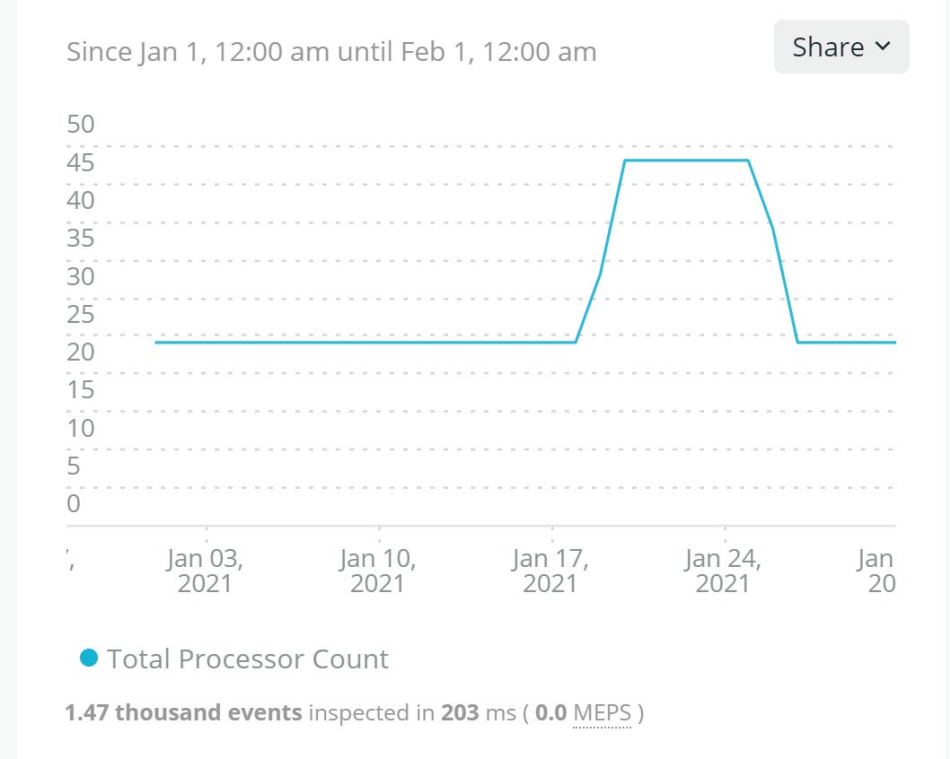
Result: The chart is demonstrated below. The user can check the value of phCPU hours used for any particular date by hovering their mouse over the desired date. Note that on the transition dates, i.e., the dates where a resize is taking place, the values demonstrated in the chart may not be as accurate relative to the total sum calculated with the previous queries. This is due to different rounding when New Relic creates charts.

Chart of phCPU days usage during many days time frame

The same query used to create the phCPU hours chart, can be used to calculate and create the equivalent phCPU days chart. We only need to remove the 24hrs/day factor.

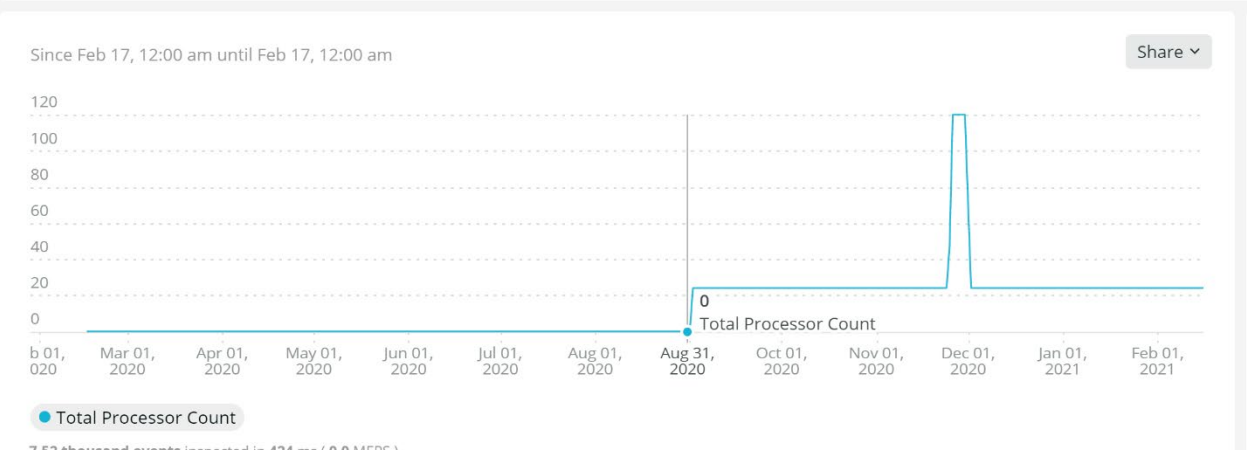
```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit 20) timeseries 1 day SINCE '2021-01-01 00:00:00 UTC' UNTIL
'2021-02-01 00:00:00 UTC'
```

And the result is demonstrated below:



Manual Calculation of phCPU

All the above chapters have described cases where the Application Performance Suit (in this case, New Relic) has been adequately installed, and the environments (defined by the keyword "`apmApplicationNames`") contain the appropriate instances. There are many cases however, where New Relic environments do not tag the appropriate instances correctly into the relevant environments. There are many reasons why this phenomenon occurs, but they are not described in this article. Suffice it to say that when the server instances are not properly tagged, the query when using the keyword "`apmApplicationNames`" and the appropriate environment name, will produce an incorrect result. The chart below demonstrates such a case:



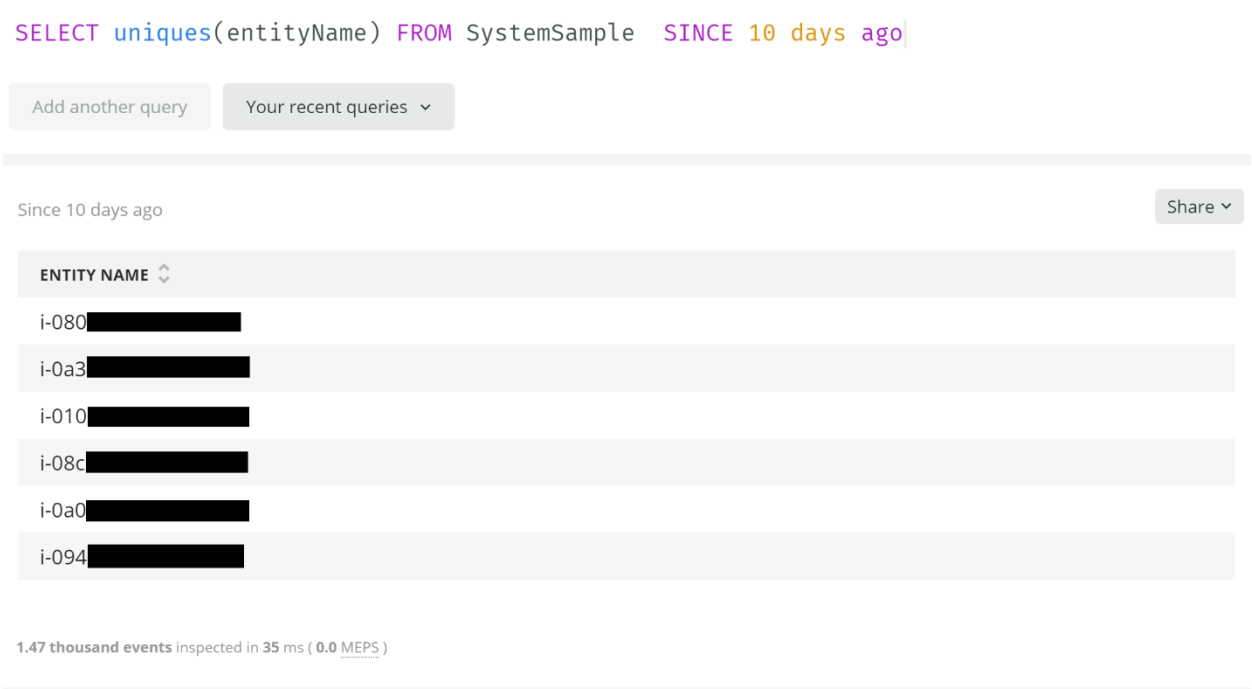
From the above chart we see that from the starting date of Feb 17, 2020, until the end of August, the phCPU per day is calculated as zero. Then at the beginning of September, the chart demonstrates some calculations and an upsize. It completes the calculations until the last requested day which is Feb 17, 2021. This chart demonstrates a typical case where not all instances are tagged into the environment of study, hence the "zero" value for a large portion of the studied time period.

The only way we can avoid such incorrect calculations is to manually find which instances belong to the environment of study (for example, the site's Production), and then create a New Relic query which specifically targets these instances and calculates phCPU. The steps for this approach are as follows:

1. Find all instances of the site. To accomplish this, one can use the following query:

   ```
   SELECT uniques(entityName) FROM SystemSample  SINCE 10 days ago
   ```

   Again, the time frame of 10 days is purely empirical and up to the user. The result will look similar to the below screenshot:



2. Manually find which of the instances belong to the environment the user wishes to study. There are many different ways to accomplish this, but they will not be discussed in this article. From the example above, let's assume that the instances belonging to Production are the following: i-080, i-010, and i-094.

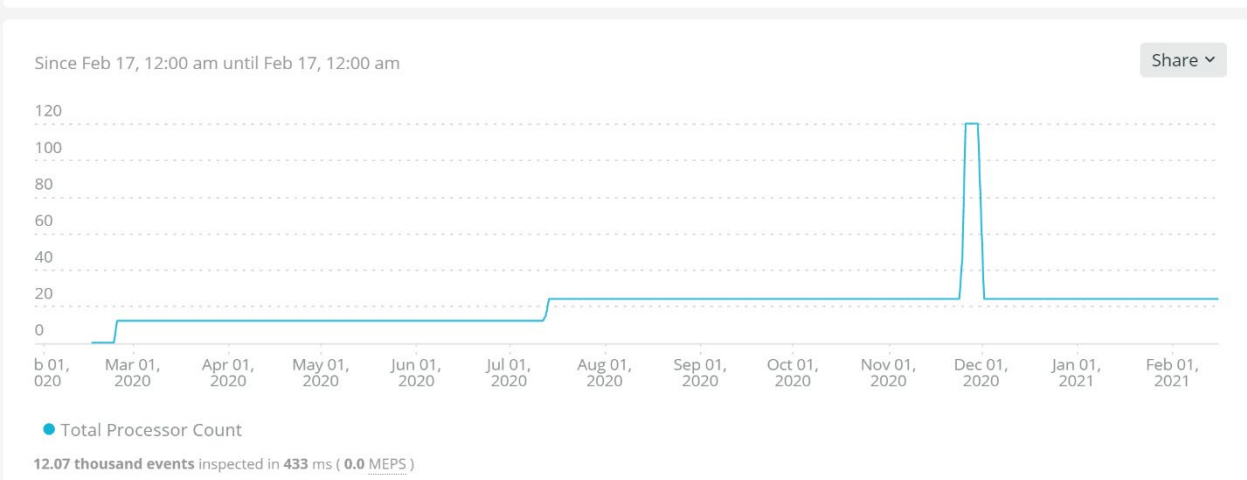
3. Once the user knows, or otherwise has decided which instances to include in the study, they may use the below query:

   ```
   SELECT sum(processorCount) FROM (SELECT
   average(numeric(processorCount)) as processorCount FROM
   SystemSample facet `entityName` where (`entityName` in ('i-
   ```

```
080','i-010','i-094') ) timeseries 1 day ) timeseries 1 day SINCE
'2020-02-17 00:00:00 UTC' UNTIL '2021-02-17 00:00:00 UTC'
```

In the above query, we use the keyword `` `entityName` `` which targets the instances as a "facet", which we later on constrain to receive only the three values of i-080, i-010, and i-094 by using the "where" expression. Note that the keyword `` `entityName` `` has to be used exactly as indicated on the above query. Any other keyword or variable name of the user's choice will lead to "4XX" New Relic errors, or entirely incorrect results.

The result of the above query targets the same site and the same time frame of the chart demonstrated at the beginning of this chapter. The revised query result is demonstrated on the chart below:



From the above chart, we observe that New Relic correctly depicts the time period of Feb 17, 2020, up to mid-July 2020 with a PhCPU value of 12. Then, the chart describes an upsize in approximately mid-July 2020 to a value of 24, another brief upsize to a value of 120, and finally a downsize back to the value of 24 for the rest of the time period. The only "zero" values observed are at the very beginning of the chart, where we reach the threshold of New Relic's storage range.