Improving the performance and *count* accuracy of VertNet search API queries

Author: Brian Stucky, CU
Date: March 26, 2014

The primary goal of this investigation was to determine how to best improve the accuracy of the *count* value returned from queries to the VertNet search API. In the current API implementation, *count* is often incorrect by one or more orders of magnitude, even for relatively small result sets. A secondary goal was to tune key App Engine search parameter values to optimize the performance of VertNet search API queries.

To explore these problems, I developed a scripting framework for automated testing of a large number of API queries and a customized version of the search API that allows for direct manipulation of the App Engine *number_found_accuracy* query parameter and access to the full range of possible App Engine *limit* parameter values. Initial investigations with this testing framework suggested that neither the *count* accuracy problem nor the performance tuning problem was likely to have a simple, obvious solution, so my attempt to find satisfactory answers became rather involved. Hence, I decided it would be most helpful to other members of the VertNet team if I were to fully document my methodology, results, and conclusions, which led to this report.

My approach focused on answering three sequential, smaller questions. First, I wanted to determine how (and if) the two key App Engine parameters *number_found_accuracy* and *limit* affected *count* error. Next, I wanted to figure out how values for the *limit* parameter affected search API performance. Finally, I investigated how values of the *number_found_accuracy* parameter affected search API performance. I will discuss my methods and results for answering each of these questions in turn, and close by offering my recommendations for how to improve the VertNet API. The test query sets and testing scripts I developed are all available in the "test" directory of the "feature/apicnttest" branch of webapp.

1. How do the *limit* and *number_found_accuracy* parameter values affect *count* error?

<u>*Methods:*</u> Preliminary testing revealed that the value of *count* returned by the VertNet search API could vary across repeated requests of the same query, which meant that the error was subject to at least some seemingly random variation. To deal with this, I decided that a statistical approach with a reasonably large sample size was needed to effectively characterize the average behavior of the search API results.

I used a set of 63 test queries taken from the spreadsheet that Dave provided me. Each replicate consisted of running all 63 queries on the VertNet search API with a given value of *limit* and *number_found_accuracy*. For queries where the returned value of *count* was less than 10,000, the true number of records in the result set was counted by repeatedly querying the API using a results cursor. The absolute difference between the true number of records and the value of *count* was calculated and standardized by the value of *count*:

$$error = \frac{|true\_count - count|}{count} \times 100 \ ,$$

and the mean error was calculated across all queries.

The values (20, 160, 600, 1000) were used for the *limit* parameter, and (21, 100, 1000, 10000) were used for the *number_found_accuracy* parameter. These values essentially span the range of allowable values for each parameter, except they do not include the smallest possible values. Each value of *limit* was tested with each value of *number_found_accuracy*. Each *limit*/*number_found_accuracy* combination was tested twice for a total of 32 replicates, and the overall mean error was calculated for each *limit*/*number_found_accuracy* combination.

<u>*Results and discussion:*</u> The results are illustrated as a 3D surface plot in Figure 1. Two general patterns are clear. First, as expected, the value of *number_found_accuracy* has an effect on the *count* accuracy for all values of *limit*. Second, and more surprising, the value of *limit* also has a strong effect on *count* accuracy. For most values of *number_found_accuracy*, larger values of *limit* exhibit lower overall error than smaller values of *limit*. It should also be noted, though, that the maximum value of *number_found_accuracy* (10,000) resulted in accurate *count* values regardless of the value of *limit*.

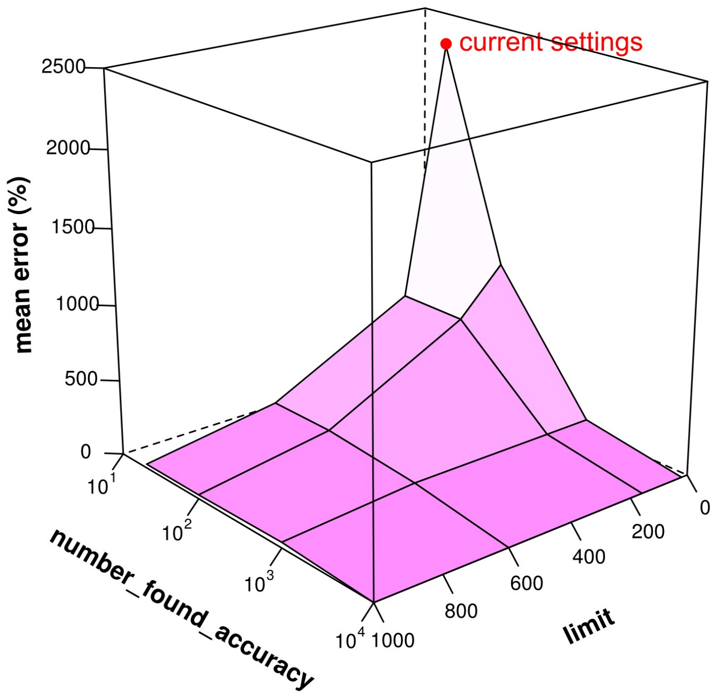

Figure 1. *The effects of the App Engine parameters* number_found_accuracy *and* limit *on the error of the* count *value returned by VertNet search API queries. The red dot indicates the values for these two parameters that are currently used by the search API.*

These results also indicate that the values for *number_found_accuracy* and *limit* currently used by the VertNet search API (*limit* = 20, *number_found_accuracy* = 21) are just about the worst choices possible as far as maximizing *count* accuracy. Increasing the default values of both parameters would dramatically improve the reliability of the returned *count* values. If optimizing *count* accuracy is a primary goal, than *number_found_accuracy*, and, to a lesser extent, *limit*, ought to be as large as possible. Whether or not that is practical leads to the next question.

2. How does the value of the *limit* parameter affect API performance?

Given the results above, I wanted to next determine how the value of the App Engine *limit* parameter influences the time it takes to retrieve data from the VertNet search API. Ideally, we should use the maximum possible value for *limit* (1,000) for the best possible *count* accuracy, but if large values for limit degrade API performance, this might be unacceptable. Furthermore, because *limit* appears to be less important than *number_found_accuracy* in determining *count* accuracy, it makes sense to tune *limit* for optimal query performance as much as possible.

Methods: The VertNet search API can obviously be used in a variety of ways, ranging from relatively lightweight queries that require minimal data movement to full downloads of large result sets. Preliminary investigations suggested that the value of the *limit* parameter affected performance differently under these two extremes. Consequently, I decided that the most robust approach would be to test API performance with a query set that mixed these usage patterns. To accomplish this, I assembled a set of of 38 test queries, approximately 40% of which produced large result sets of greater than 15,000 records, 40% produced medium-sized result sets of between 100 and 15,000 records, and 20% produced small result sets of less than 100 records.

Each replicate consisted of running all 38 queries against the search API with the value of *number_found_accuracy* set to 10,000 and the value of *limit* set to a given test value. For the large result sets, only the first response was examined (to approximate exploratory/inventory usage of the API), and for the small and medium-sized result sets, the entire result set was retrieved (to approximate targeted data retrieval). I ran four replicates for each of 7 test *limit* values (20, 80, 160, 320, 600, 800, 1000) (these values approximately span the range of *limit* values allowed by App Engine) for a total of 28 replicates. The total time to complete each replicate was used as the measure of API performance. The resulting data were visualized and analyzed with R.

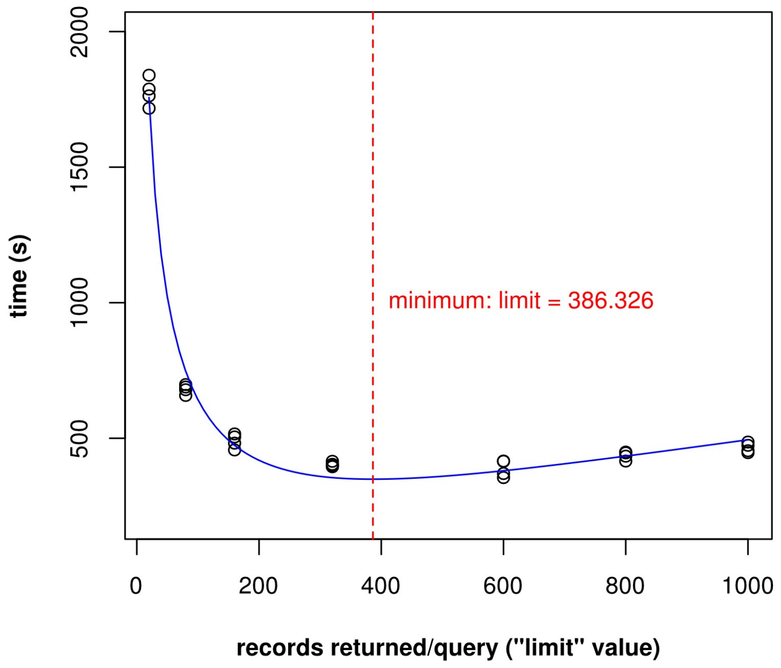

Figure 2. *Scatterplot of the value of the App Engine* limit *parameter and total test query set run time. The estimated log-squared model relating these two variables is illustrated by the solid blue line, and the analytical minimum of the model equation is indicated by the dashed red line.*

Results and discussion: A scatterplot of total time versus the *limit* value suggested that the relationship could be modeled with a log-squared curve (Figure 2). I used R to fit a general linear model following a log-squared relationship to the data. The resulting estimated equation was

$$t = 6046.762 - 1913.026 \cdot \ln(l) + 160.578 \cdot \ln^2(l) \; ,$$

where t is total time and l is the value of the App Engine *limit* parameter. This model explained the data remarkably well, with adjusted $R^2 = 0.9913$, and the relationship was highly significant ($F_{2,25} = 1530$, $p < 0.00001$), which strongly suggested that the model very nearly captured the true relationship between the value of *limit* and the search API query run time. The run time data and log-squared model are depicted in Figure 2.

To estimate the optimal value of *limit* that minimizes API query run time, I used basic differential calculus to find the location of the exact minimum of the log-squared model equation. The minimum occurs at *limit* = 386.326 (Figure 2).

These results, along with the results of the previous section, indicate that the current VertNet search API default value of *limit* = 20 is a very poor choice, both in terms of *count* accuracy and API query run time. For the query test set used for this analysis, query run time improves rapidly as *limit* increases to about 400, then begins to slowly degrade again as *limit* approaches 1,000 (the maximum value allowed by App Engine). Assuming that the test queries are a reasonable representation of real-world VertNet API usage, 400 would be a good choice for *limit* to improve both *count* accuracy and query run time.

3. How does the value of the *number_found_accuracy* parameter affect API performance?

The analysis for the previous section used a fixed *number_found_accuracy* value of 10,000 to maximize *count* accuracy, but I didn't know if 10,000 is truly the best choice for *number_found_accuracy*. The last important remaining problem, then, was to determine how severely large values of *number_found_accuracy* affected VertNet API query performance.

Methods: I used the same set of test queries as for the previous section, for the same reasons. As before, a replicate consisted of running all 38 queries against the search API, inspecting only the first response for the “large” queries, and downloading the full result set for the “medium” and “small” queries. All replicates were run with *limit* = 400, following the results obtained above. 4 replicates were run for each of 5 *number_found_accuracy* values, (100, 1000, 4000, 7000, 10000), for a total of 20 replicates. The total time to complete each replicate was again used as the measure of API performance, and the resulting data were analyzed with R.

Results and discussion: A scatterplot of the data suggested a linear relationship between *number_found_accuracy* and total query run time (Figure 3). There was one extreme outlier for *number_found_accuracy* = 20 that greatly exceeded all other run times (not depicted in Figure 3); I removed it prior to further analysis. Fitting a simple linear regression model to the data resulted in the equation

$$t = 410.3 + 0.00241\,a \; ,$$

where t is the total query run time and a is the value of *number_found_accuracy*. This model is depicted in Figure 3. Although a linear relationship does seem to be the best choice for these data, the slope of the regression equation was barely significant ($p = 0.0405$) and the model explained relatively little of the variation in run times ($R^2 = 0.224$). Indeed, with the outlier included in the dataset, the regression analysis failed to detect a relationship at all.

The rather poor model fit confirms what is visually obvious in the scatterplot: compared to the variance in run times due to other factors, *number_found_accuracy* has relatively little effect. On average, very low values of *number_found_accuracy* are expected to result in faster run times, but even so, some replicates with *number_found_accuracy* set to 7,000 or 10,000 outperformed replicates with *number_found_accuracy* set to 100 or 1,000.

The model equation estimates that the fastest mean replicate run time, when *number_found_accuracy* = 100, is 410.521 seconds, while the slowest mean run time, when *number_found_accuracy* = 10,000, is 434.366 seconds. This translates to a per-query mean run time increase of about 0.627, barely more than half a second, and this estimate includes test queries with result sets that required several minutes to retrieve. Considering the substantial improvements in *count* accuracy (see above), the small performance penalty incurred by setting *number_found_accuracy* to 10,000 is probably not of much concern.

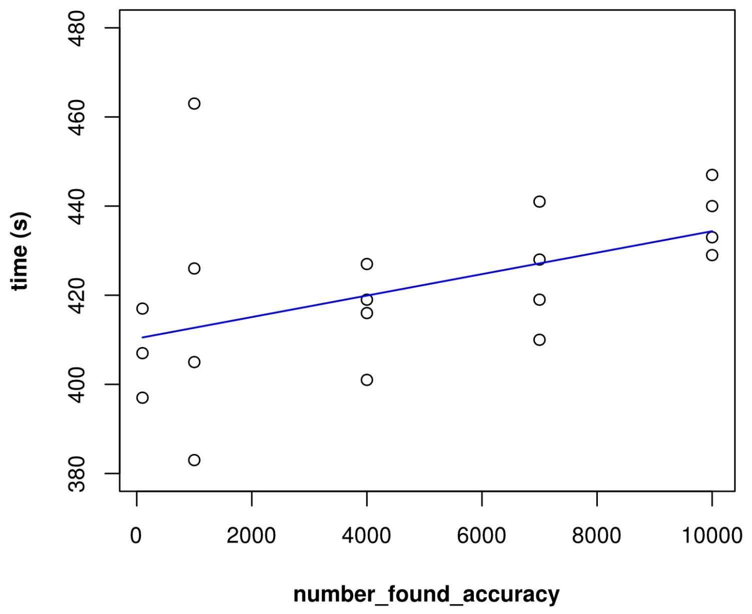

Figure 3. Scatterplot of the value of the App Engine number_found_accuracy *parameter and total test query set run time. The estimated simple linear regression model relating these two variables is illustrated by the solid blue line.*

4. Conclusions and recommendations

Improving count *accuracy:* The analyses above suggest that 10,000 should be used as the default value of *number_found_accuracy* for all queries. Across all test replicates, this always resulted in *count* values that were correct for all queries with result sets of 10,000 records or less.

The value of the *limit* parameter also influences *count* accuracy, although less so than *number_found_accuracy*. The VertNet API's current default value of 20 for *limit* performs especially poorly, and increasing *limit* to a value of at least several hundred would be prudent. If *number_found_accuracy* is set at 10,000, though, there appears to be little reason to set *limit* to an especially high value.

Optimizing query performance: Between the App Engine parameters *limit* and *number_found_accuracy*, *limit* has by far the biggest impact on VertNet API query and data retrieval speed. For the set of test queries and data retrieval methods used in this investigation, setting the value of *limit* to 400 would be expected to result in approximately optimal query performance. The log-squared model relating query run time to the value of *limit* estimates that if *limit* were to be increased from its current value of 20 to 400, mean query run times would be just over 5 times faster, or approximately 37 seconds faster per query.

By comparison, *number_found_accuracy* has an almost negligible effect on query speed. The large performance boost achieved by increasing *limit* from 20 to 400 should more than compensate for the minor performance penalty caused by increasing *number_found_accuracy* to 10,000.

Implementation details: If the recommendations above are implemented, then I expect that the *count* values for all queries returning about 10,000 records or less will be completely reliable. I have done limited testing of the accuracy of *count* for even larger result sets, up to about 30,000 records. Unfortunately, even with *number_found_accuracy* set to 10,000 and a relatively large value of *limit*, *count* accuracy declines as result sets approach tens of thousands of records. I suspect this is simply a limit of the App Engine data storage architecture.

To deal with this, I suggest that we do two things. First, we should expand the VertNet API documentation to clearly explain the expected reliability pattern for reported *count* values. Second, we should consider modifying the value of *count* returned for large result sets to clearly indicate that it is an estimate only. For example, for result sets of more than 10,000 records, we might return *count* = ">10000" or append a tilde ("~") in front of *count* to mark it as an approximation.

Future work: To some extent, the accuracy of this analysis hinges on the test query set and test methods I developed and whether they accurately represent "real life" usage of the VertNet search API. I hope they at least get us close. Right now, the API only receives limited traffic, much of which likely originates from the VertNet team, so realistic usage patterns are probably difficult to assess. In the future, as the search API begins to see more widespread use, it might be worthwhile to use API request logging to develop a more realistic test query set. The new test query set could then be used to repeat these analyses and assess whether the App Engine parameters need further tuning.